The Timex/Sinclair 2068 ROM Disassembly

David Anderson

Acknowledgements

This book would not be possible without the valuable contributions of the designers, programmers, engineers and other staff at the Timex Computer Corporation and all those who've done the hard work of decoding the ZX Spectrum ROM and the Timex/Sinclair 2068 ROMs.

This work draws from efforts by Michael H. Branigin, Wes Brzozowski, Virginia Carole Corcoran, Lloyd Dreger, Ian Logan, Frank O'Hara, Robert Orrfelt, Nazir Pashtoon, William Pederson and many others.

Table of Contents

Introduction

Timex Computer Corporation announced their version of the Sinclair ZX Spectrum at the 1983 Winter Consumer Electronics Show, in early January.

The specifications for the computer included a sound generator chip, 16K ROM and the cartridge port, despite the fact that the prototype was still in a Spectrum case.

Unlike their work on the Timex/Sinclair 1000, which was largely cosmetic, this re-imagined computer required its own teams of managers, engineers and programmers. They came from other computer industry companies like Burroughs and Texas Instruments.

Over the next several months, Timex added memory bank switching and support for cartridges and devices beyond those imagined by Sinclair. The intended expansion unit would have supported disk drives, modems, local area networks and extended memory, up to 16mb.

Timex engineers replaced the Spectrum ULA with a Standard-Cell Logic Device (SCLD) from NCR and added the sound generator chip, cartridge port, two joystick ports and a power switch.

The two ROM chips containing the operating system are socketed, suggesting Timex intended to offer a replacement in the future. The ROM has a special memory location to indicate the version, which would support upgraded ROMs from Timex.

Timex programmers re-engineered the Spectrum ROM to accommodate the new features in the TS 2068. The TS 2068 ROM contains many new ideas not present in the Spectrum, including a function dispatcher meant to give programmers a consistent access method to routines across changes in the ROM.

By the time the computer was available in stores, in November, it was largely incompatible with the Spectrum.

Naming Conventions

The Timex programmers renamed sections of the ROM and routines within those sections.

This book prefers Spectrum section and routine names throughout. Where Timex programmers gave it a different name, it's noted in parenthesis at the section or routine location.

HOME ROM

The HOME ROM is where most of the action is with the Timex Sinclair 2068 operating system and BASIC. The vast majority of the original ZX Spectrum ROM code lives in the HOME ROM, albeit often in different locations.

Timex moved some routines from the HOME ROM to clear space for code to handle cartridge-based programs and accommodate its planned memory and extended feature Bus Expansion Unit.

Not all of the new code works and code that does may be slightly wonky. I've called out those issues as best as possible in the comments.

HOME ROM

Restart Routines And Tables

The Z80 has eight special shortcut calls to specific addresses at the beginning of memory. These calls use only one byte, versus three bytes for a standard call.

The ZX Spectrum and TS2068 use these special locations to handle startup processing and other frequently used routines.

Start

On power up, Interrupt Mode 0 is enabled in the Z80. This maskable interrupt is disabled and the DE register pair set to hold the 'top of possible RAM'. This routine can be used to reset the computer.

START (RST0)

```
M0000   DI              Disable interrupts.
        XOR A           +00 for start (but +FF for 'NEW').
        LD DE, $FFFF    Set pointer the top of possible
        JP START-NEW    RAM. Continue with initialization.
```

Error

Error handler. Top of machine stack has the address of the error identifier. Can be used to support new commands or variants by hijacking the error interpreter. The ZX Spectrum Interface 1 and the Timex Portugal floppy disk drive systems (FDD-3 and FDD-3000) use this vector to add their commands to Sinclair BASIC.

ERROR-1 (RST8)

```
M0008   LD HL, (CH_ADD)     Load HL with CH_ADD, the current address
                            reached by the interpreter.
        LD (X_PTR), HL      Store the character after ? mark in X_PTR.
        JR ERROR-2          Continue at the ERROR-2 routine.
```

Print Character

Writes the code in A to the current output channel.

PRINT-A-1 (WRCH)

```
M0010   JP PRINT-A-2        Jump to stream output routine.
```

SYSTEM-VERSION

```
M0013   DEFB $FF            Memory location that identifies the  revision level
                            of the system software. Would have counted
                            down as versions incremented.
```

Get Character

Fetch the contents in the location pointed to by CH_ADD. Returns if the value is a printable character, otherwise CH_ADD is incremented.

GET-CHAR (CURCH)

```
M0018   LD HL, (CH_ADD)     Get contents of location in CH_ADD.
        LD A, HL            Get the current character.
```

TEST-CHAR

```
M001C   CALL SKIP-OVER      Test if it's printable.
        RET NC              Return if not a token or a space.
```

Get Next Character

As a BASIC line is interpreted, this routine is called to step along the line. If the character is a space or a token with a value less than $1F, it gets the next character.

NEXT-CHAR

```
M0020   CALL CH_ADD+1       Get the next character.
        JR TEST-CHAR        Test the character.
```

Call Calculator

Jumps to the floating point calculator interpreter.

```
M0028   JP CALCULATE        Jump to calculator.
```

Make BC Spaces

This routine creates free locations in the work space. The number of locations is set by BC.

BC-SPACES (ALLOCBC)

```
M0030   PUSH BC             Save the number.
        LD HL, (WORKSP)     Get the address of the start of the
        PUSH HL             workspace and save it
        JP RESERVE          before proceeding.
```

Maskable Interrupt

The real time clock is incremented and the keyboard scanned whenever a maskable interrupt occurs.

MASK-INT

```
M0038   PUSH AF             Save the current values held in
        PUSH HL             these registers.
        LD HL,(FRAMES)      The lower two bytes of the frame counter
                            (FRAMES) are incremented
        INC HL              every 17 ms.
        LD (FRAMES),HL      The highest byte of the frame
        LD A,H              counter is only incremented when
        OR L                the value of the lower two bytes is
```

```
        JR NZ,KEY-INT           zero.
        INC (IY+OFRAMES2)       Increment high byte of FRAMES (FRAMES2).
```

KEY-INT

```
M0048   PUSH BC                 Save these registers.
        PUSH DE
        CALL KEYBOARD           Scan the keyboard.
        POP DE                  Restore registers.
        POP BC
        POP HL
        POP AF
        EI                      Enable interrupts.
        RET
```

Error-2

The return address to the interpreter points to the DEFB that signifies which error has occurred. This DEFB is fetched and transferred to ERR_NR. The machine stack is cleared before jumping forward to clear the calculator stack.

ERROR-2 (GETERNR)

```
M0053   POP HL                  Get the error address.
        LD L,(HL)               Put the error number in L.
```

ERROR-3 (LE3)

```
M0055   LD (IY+OERRNR),L        Save error number in ERR_NR.
        LD SP,(ERRSP)           Load SP with the error routine stack pointer.
        JP SET-STK              Reset the calculator stack and memory pointers.
```

NMI

This routine is not used in the standard TS 2068 but the code allows for a system reset to occur following activation of the NMI line. NMIADD ($5CB0) has to have the value zero for the reset to occur.

```
M0066   PUSH AF                 Save registers.
        PUSH HL
        LD HL, (NMIADD)         Get NMIADD, check to see if
        LD A,H                  both are zero.
        OR L
        JR NZ, NO-RESET         This is a bug. Should be JP Z.
        JP (HL)                 Jump to START.
```

NO-RESET

```
M0070   POP HL                  Restore registers.
        POP AF
        RETN
```

CH_ADD+1 Subroutine

The address held in CH_ADD is fetched, incremented and restored. The contents of the location now addressed by CH_ADD is fetched. The entry points of TEMP-PTR1 and TEMP-PTR2 are used to temporarily set CH_ADD.

CH_ADD+1 (NEXTCH)

```
M0074   LD HL,(CH_ADD)          Get CH_ADD.
```

TEMP-PTR1 (NC_HL)

Bump the current character pointer by 1.

```
M0077   INC HL                  Move to next character.
```

TEMP-PTR2 (TC_HL)

Update the character pointer with HL.

```
M0078   LD (CH_ADD),HL          Save HL to CH_ADD.
        LD A,(HL)               Get the character.
        RET
```

Skip-Over Subroutine

This subroutine skips over white-space and other characters irrelevant to the parsing of a BASIC line. The A register holds the character to be tested and HL holds its address.

CH_ADD is advanced twice if the character is any of the display control tokens (INK, PAPER, FLASH, BRIGHT, INVERSE, or OVER).

SKIP-OVER (TEST_CH)

```
M007D   CP $21                  Return if '!' or later with no carry.
        RET NC
        CP $0D                  Exit if end-of-line has been
        RET Z                   reached.
        CP $0C                  Exit if DELETE.
        RET Z
        CP $10                  Exit if less than $10 but
        RET C                   with carry set.
        CP $18                  Exit if greater than $17.
        CCF
        RET C
```

At this point HL is pointing at a character in the range $10 to $17, which are various control tokens.

```
        INC HL                  Advance to next character.
        CP $16                  If the token is a display command,
        JR C, SKIPS             (INK to OVER) jump ahead.
        INC HL                  Advance to next character (AT & TAB).
```

SKIPS

```
M0093   SCF                     Set carry.
        LD (CH_ADD),HL          Save new character pointer in CH_ADD.
        RET
```

TOKENS

The tokenized characters 134 (RND) to 255 (COPY) are expanded using this table. The last byte of a token is inverted to denote the end of the word. The first entry is an inverted step-over byte.

```
M0098   DEFB ('?'+$80)
        DEFM "RN"&('D'+$80)
        DEFM "INKEY"&('$'+$80)
        DEFM "P"&('I'+$80)
        DEFM "F"&('N'+$80)
        DEFM "POIN"&('T'+$80)
        DEFM "SCREEN"&('$'+$80)
        DEFM "ATT"&('R'+$80)
        DEFM "A"&('T'+$80)
        DEFM "TA"&('B'+$80)
        DEFM "VAL"&('$'+$80)
        DEFM "COD"&('E'+$80)
        DEFM "VA"&('L'+$80)
        DEFM "LE"&('N'+$80)
        DEFM "SI"&('N'+$80)
        DEFM "CO"&('S'+$80)
        DEFM "TA"&('N'+$80)
        DEFM "AS"&('N'+$80)
        DEFM "AC"&('S'+$80)
        DEFM "AT"&('N'+$80)
        DEFM "L"&('N'+$80)
        DEFM "EX"&('P'+$80)
        DEFM "IN"&('T'+$80)
        DEFM "SQ"&('R'+$80)
        DEFM "SG"&('N'+$80)
        DEFM "AB"&('S'+$80)
        DEFM "PEE"&('K'+$80)
        DEFM "I"&('N'+$80)
        DEFM "US"&('R'+$80)
        DEFM "STR"&('$'+$80)
        DEFM "CHR"&('$'+$80)
        DEFM "NO"&('T'+$80)
        DEFM “BI”&('N'+$80)
```

The previous 32 function-type words are printed without a leading space. The following have a leading space if they begin with a letter.

```
M0103   DEFM "O"&('R'+$80)
        DEFM "AN"&('D'+$80)
```

```
DEFM "<"&('='+$80)
DEFM ">"&('='+$80)
DEFM "<"&('>'+$80)
DEFM "LIN"&('E'+$80)
DEFM "THE"&('N'+$80)
DEFM "T"&('O'+$80)
DEFM "STE"&('P'+$80)
DEFM "DEF F"&('N'+$80)
DEFM "CA"&('T'+$80)
DEFM "FORMA"&('T'+$80)
DEFM "MOV"&('E'+$80)
DEFM "ERAS"&('E'+$80)
DEFM "OPEN "&('#'+$80)
DEFM "CLOSE "&('#'+$80)
DEFM "MERG"&('E'+$80)
DEFM "VERIF"&('Y'+$80)
DEFM "BEE"&('P'+$80)
DEFM "CIRCL"&('E'+$80)
DEFM "IN"&('K'+$80)
DEFM "PAPE"&('R'+$80)
DEFM "FLAS"&('H'+$80)
DEFM "BRIGH"&('T'+$80)
DEFM "INVERS"&('E'+$80)
DEFM "OVE"&('R'+$80)
DEFM "OU"&('T'+$80)
DEFM "LPRIN"&('T'+$80)
DEFM "LLIS"&('T'+$80)
DEFM "STO"&('P'+$80)
DEFM "REA"&('D'+$80)
DEFM "DAT"&('A'+$80)
DEFM "RESTOR"&('E'+$80)
DEFM "NE"&('W'+$80)
DEFM "BORDE"&('R'+$80)
DEFM "CONTINU"&('E'+$80)
DEFM "DI"&('M'+$80)
DEFM "RE"&('M'+$80)
DEFM "FO"&('R'+$80)
DEFM "GO T"&('O'+$80)
DEFM "GO SU"&('B'+$80)
DEFM "INPU"&('T'+$80)
DEFM "LOA"&('D'+$80)
DEFM "LIS"&('T'+$80)
DEFM "LE"&('T'+$80)
DEFM "PAUS"&('E'+$80)
DEFM "NEX"&('T'+$80)
DEFM "POK"&('E'+$80)
DEFM "PRIN"&('T'+$80)
DEFM "PLO"&('T'+$80)
DEFM "RU"&('N'+$80)
```

```
DEFM "SAV"&('E'+$80)
DEFM "RANDOMIZ"&('E'+$80)
DEFM "I"&('F'+$80)
DEFM "CL"&('S'+$80)
DEFM "DRA"&('W'+$80)
DEFM "CLEA"&('R'+$80)
DEFM "RETUR"&('N'+$80)
DEFM "COP"&('Y'+$80)
DEFM "DELET"&('E'+$80)
DEFM "ON ER"&('R'+$80)
DEFM "STIC"&('K'+$80)
DEFM "SOUN"&('D'+$80)
DEFM "FRE"&('E'+$80)
DEFM "RESE"&('T'+$80)
```

Keyboard Tables and Scanning

These six look-up tables are used by the keyboard reading routine to decode the key values.

The first table contains the maps for the 39 keys of the standard TS 2068 keyboard. The remaining key (SHIFT, $27) is read directly.

The keys consist of the 26 upper-case alphabetic characters, the 10 digit keys and the space, ENTER and symbol shift key. Unshifted alphabetic keys have $20 added to the value. The keywords for the main alphabetic keys are obtained by adding $A5 to the values obtained from this table.

Each of these tables reads left-to-right.

Main Keys Table (LCKEYS)

```
M0227   DEFB $42,$48,$59,$36,$35,$54,$47,$56   B,H,Y,6,5,T,G,V
        DEFB $4E,$4A,$55,$37,$34,$52,$46,$43   N,J,U,7,4,R,F,C
        DEFB $4D,$4B,$49,$38,$33,$45,$44,$58   M,K,I,8,3,E,D,X
        DEFB $0E,$4C,$4F,$39,$32,$57,$53,$5A   SYM SFT,L,O,9,2,W,S,Z
        DEFB $20,$0D,$50,$30,$31,$51,$41       SPACE,ENTER,P,0,1,Q,A
```

Extended Mode (EKEYS)

Unshifted extended mode keys for the alphabetic characters.

```
M024E   DEFB $E3    READ        DEFB $C4    BIN
        DEFB $E0    LPRINT      DEFB $E4    DATA
        DEFB $B4    TAN         DEFB $BC    SGN
        DEFB $BD    ABS         DEFB $BB    SQR
        DEFB $AF    CODE        DEFB $B0    VAL
        DEFB $B1    LEN         DEFB $C0    USR
        DEFB $A7    PI          DEFB $A6    INKEY$
```

```
        DEFB $BE    PEEK            DEFB $AD    TAB
        DEFB $B2    SIN             DEFB $BA    INT
        DEFB $E5    RESTORE         DEFB $A5    RND
        DEFB $C2    CHR$            DEFB $E1    LLIST
        DEFB $B3    COS             DEFB $B9    EXP
        DEFB $C1    STR$            DEFB $B8    LN
```

Shifted Extended Mode (SEKEYS)

Letter and shift.

```
M0268   DEFB $7E    FREE            DEFB $DC    BRIGHT
        DEFB $DA    PAPER           DEFB $5C    \
        DEFB $B7    ATN             DEFB $7B    { (ONERR)
        DEFB $7D    } (SOUND)       DEFB $D8    CIRCLE
        DEFB $BF    IN              DEFB $AE    VAL$
        DEFB $AA    SCREEN$         DEFB $AB    ATTR
        DEFB $DD    INVERSE         DEFB $DE    OVER
        DEFB $DF    OUT             DEFB $7F    © (RESET)
        DEFB $B5    ASN             DEFB $D6    VERIFY
        DEFB $7C    STICK           DEFB $D5    MERGE
        DEFB $5D    ]               DEFB $DB    FLASH
        DEFB $B6    ACS             DEFB $D9    INK
        DEFB $5B    [               DEFB $D7    BEEP
```

Control Codes (NUMFNTBL)

Digit keys and CAPS SHIFT.

```
M0282   DEFB $0C    DELETE          DEFB $07    EDIT
        DEFB $06    CAPS LOCK       DEFB $04    TRUE VIDEO
        DEFB $05    INV VIDEO       DEFB $08    CUR L
        DEFB $0A    CUR D           DEFB $0B    CUR U
        DEFB $09    CUR R           DEFB $0F    GRAPHICS
```

Symbol Keys (KKEYS)

Letter and Symbol shift.

```
M028C   DEFB $E2    STOP            DEFB $2A    *
        DEFB $3F    ?               DEFB $CD    STEP
        DEFB $C8    >=              DEFB $CC    TO
        DEFB $CB    THEN            DEFB $5E    ^
        DEFB $AC    AT              DEFB $2D    -
        DEFB $2B    +               DEFB $3D    =
        DEFB $2E    .               DEFB $2C    ,
        DEFB $3B    ;DEFB $22       "
        DEFB $C7    <=              DEFB $3C    <
        DEFB $C3    NOT             DEFB $3E    >
        DEFB $C5    OR              DEFB $2F    /
```

```
        DEFB $C9    <>              DEFB $60    £
        DEFB $C6    AND             DEFB $3A    :
```

Extended Mode (SSKEYS)

Symbol shift with digit keys.

```
M02A6   DEFB $D0    FORMAT          DEFB $CE    DEFFN
        DEFB $A8    FN              DEFB $CA    LINE
        DEFB $D3    OPEN #          DEFB $D4    CLOSE #
        DEFB $D1    MOVE            DEFB $D2    ERASE
        DEFB $A9    POINT           DEFB $CF    CAT
```

Keyboard Scanning Routine

This subroutine is called by the main keyboard subroutine and the INKEY$ routine (in SCANNING).

In all instances the E register is returned with a value in the range of $0 to $27 or $FF for no key pressed.

The D register is returned with a value that indicates which single shift key is being pressed.

If both shift keys are being pressed then the D and E registers are returned with the values for the CAPS SHIFT and SYMBOL SHIFT keys respectively.

If no keys is being pressed then the DE register pair is returned holding $FFFF.

The zero flag is returned reset if more than two keys are being pressed, or neither key of a pair of keys is a shift key.

KEY-SCAN (K_SCAN)

```
M02B0   LD L,$2F            Initialize position code.
        LD DE,$FFFF         Initialize to "no key" found.
        LD BC,$FEFE         C=keyboard port, B=scan group to test. The '0'
                            bit is the key group that is scanned.
```

Loop over keyboard. Eight passes are made. Each pass has a different initial key value and scans a different line of five keys. The first line is CAPS SHIFT, Z, X, C, V.

KEY-LINE

```
M02B8   IN A,(C)            Get key information; low=key pressed.
        CPL                 High is now a pressed key.
        AND $1F             Allow only keyboard bits.
        JR Z, KEY-DONE      Jump if no keys pressed.
```

At least one key has been pressed.

```
        LD H,A              Save the key value.
        LD A,L              Save position code.
```

KEY-3KEYS

```
M02C1   INC D              Return if too many (>2) keys pressed.
        RET NZ
```

KEY-BITS

```
M02C3   SUB $08            Repeatedly subtract 8 from the
        SRL H              preset key value until a key-bit is found.
        JR NC, KEY-BITS    Jump until there is a match.
        LD D,E             Put previous key value into D.
        LD E,A             Put present key value into E.
        JR NZ, KEY-3KEYS   Check for another key.
```

KEY-DONE

```
M02CD   DEC L              Line has been scanned; reduce
        RLC B              initial key value for the next pass.
        JR C, KEY-LINE     Shift counter and jump if more groups need to be
                           scanned.
```

Having scanned all key groups and not returned; there must have been only one key pressed, shift and key pressed or no keys pressed.

```
        LD A,D             Get previous key.
        INC A              Return if only one key or no keys
        RET Z              pressed.
        CP $28             Return if caps shift and another
        RET Z              key.
        CP $19             Return if symbol shift and another
        RET Z              key.
        LD A,E
        LD E,D             Swap D and E.
        LD D,A
        CP $18             Check for symbol shift.
        RET                ZF = 1 if there is a symbol shift.
                           ZF = 0 if two character keys.
```

Keyboard Subroutine

Called every time a maskable interrupt happens, once every 17 ms in normal operation. Scans the keyboard and decodes the key value. The code produced will, if the 'repeat' status allows it, be passed to the system variable LAST-K. When a code is put into this system variable bit 5 of FLAGS is set to show that a 'new' key has been pressed.

KEYBOARD (UPD_K)

```
M02E1   CALL KEY-SCAN      Scan the keyboard.
        RET NZ             Return immediately if too many keys pressed.
```

A double system of KSTATE system variables (KS_A1, KS_C1, KS_D1, KS_B1 and KS_A2, KS_C2, KS_D2, KS_B2) is used below.

The two sets allow for the detection of a new key being pressed (using one set) while still within the repeat period of the previous key pressed (details in the other set).

A set will only become free to handle a new key if the key is held down for about 1/10th of a second. i.e. five calls to KEYBOARD.

```
        LD HL,KS_A1             Get key state.

K-ST-LOOP
M02E8   BIT 7,(HL)              Jump if no key stored.
        JR NZ, K-CH-SET
        INC HL                  If the set is not free,
        DEC (HL)                decrement its "5 call counter".
        DEC HL
        JR NZ, K-CH-SET         When it reaches zero, signal.
        LD (HL),$FF             the set as free.

K-CH-SET
M02F3   LD A,L                  Save lower byte of key buffer.
        LD HL,KS_A2             "Next key" buffer address.
        CP L                    Jump if we are looking at different
        JR NZ, K-ST-LOOP        buffers.
        CALL K-TEST             Look for key code.
        RET NC                  Return if no character key detected.
```

A key stroke that is being repeated (held down) is now separated from a new key stroke.

```
        RES DELREP,(IY+OFLAGS2) Reset repeat deletion flag in FLAGS2.
        LD HL,KS_A1
        CP (HL)                 Jump to handle repeat counters if
        JR Z,K-REPEAT           KEY0 = KEY2.
        EX DE,HL                Save address KS_A1 in DE.
        LD HL,KS_A2             Examine second KSTATE.
        CP (HL)                 Jump to handle repeat counters if
        JR Z,K-REPEAT           KEY0 = KEY2.
```

But a new key will not be accepted unless one of the sets of KSTATE system variables is 'free'.

```
        BIT 7,(HL)              Jump if KS_A2 indicates no key pressed
        JR NZ,K-NEW             (saves A into KS_A2).
        EX DE,HL                Point to KS_A1.
        BIT 7,(HL)              Return if KS_A1 has key hit.
        RET Z
```

The new key is to be accepted. But before the system variable LASTK can be filled, the KSTATE system variables, of the set being used, have to be initialized to handle any repeats and the key's code has to be decoded.

```
K-NEW
M0317   LD E,A                  Pass code to E
        LD (HL),A               and to KS_A1 or KS_A2.
        INC HL                  Initialize the debounce counter.
```

```
        LD (HL),$05
        INC HL
        LD A,(REPDEL)           Initialize the repeat counter.
        LD (HL),A
        INC HL                  Point to KSTATE 3/7.
```

The decoding of a main code depends upon the present state of MODE, bit 3 of FLAGS and the shift byte.

```
        LD C,(IY+OMODE)         Fetch MODE.
        LD D,(IY+OFLAGS)        Fetch FLAGS.
        PUSH HL                 Save the character pointer.
        CALL K-DECODE
        POP HL
        LD (HL),A               Final state saved in KSTATE 3/7.
```

K-END

```
M032E   LD (LAST-K),A           Store the keystroke in LAST-K
        SET KEYHIT,(IY+OFLAGS)  Signal a "new key".
        RET
```

Repeat Key Subroutine

A possible repeat has been identified. HL addresses the raw key. The last location of the key map holds the decoded key from the first context. This could be a keyword and, with the exception of NOT, a repeat is syntactically incorrect and not really desirable.

K-REPEAT

```
M0336   INC HL                  Initialize the debounce counter.
        LD (HL),$05
        INC HL                  Point to repeat counter.
        LD A,(LAST-K)
        CP $CE                  Return if last key value is a token.
        RET NC
        DEC (HL)                Decrement repeat counter.
        RET NZ                  Return if not time to repeat yet.
        LD A,(REPPER)           Initialize repeat counter.
        LD (HL),A
        INC HL                  Point to character.
        LD A,(HL)               Get character.
        CP $0C                  Jump if character code is not DELETE.
        JR NZ,K-END
        SET DELREP,(IY+OFLAGS2) Set delete key repeat on.
        PUSH AF                 Save AF.
        LD BC,$4E20
```

K-REP-DELAY

```
M0354   DEC BC                  Delay loop.
        LD A,C                  Put lower byte in C.
        OR B                    Clear B.
```

```
              JR NZ, K-REP-DELAY      Loop until zero.
              POP AF
              JR K-END
```

K-TEST Subroutine

The key value is tested and returns immediately if 'no-key' or 'shift-only'. Otherwise, the main code for that key is found.

K-TEST (K_BASE)

```
M035C         LD B,D                  Last key to B.
              LD D,$00                Clear D for later.
              LD A,E                  Key number to A.
              CP $27                  Return if key is greater than
              RET NC                  CAPS SHIFT.
              CP $18                  Jump if not SYMBOL SHIFT
              JR NZ, K-MAIN
              BIT 7,B                 Return if this only one key
              RET NZ                  has been pressed.
```

K-MAIN

```
M036A         LD HL, LCKEYS           Offset to table 3.
              ADD HL,DE               Retrieve the key code from
              LD A,(HL)               LCKEYS table.
              SCF                     Indicate key was found.
              RET
```

Keyboard Decoding Subroutine

Entered with the main code in the E register, the value of FLAGS in the D register, the value of MODE in the C register and the shift byte in the B register.

Evaluating these four values and referring, as necessary, to the six key tables a final code is produced and returned in the A register.

K-DECODE (CHCODE)

```
M0371         LD A,E                  Get key code to A.
              CP $3A                  Jump if key code is a digit key,
              JR C,K-DIGIT            space, ENTER or shift (<$3A).
              DEC C                   Decrement the MODE to remove K/L.
              JP M,K-KLC-LET          Jump forward, as needed, for modes
              JR Z,K-E-LET            K, L, C and E.
```

Only graphics mode is left and the final code for letter keys in graphics mode is computed from the main code.

```
              ADD A,$4F               Add offset to make graphics character.
              RET
```

K-E-LET
Letters in extended mode are considered next.

```
M037F   LD HL, EKEYS-'A'        Offset to base of "E" table.
        INC B                   Jump if shifted character
        JR Z, K-LOOK-UP
        LD HL, LCKEYS           unshifted "E" mode table
```

K-LOOK-UP
Key tables ‘b-f’ are served by this look-up routine. In all cases a 'final code' is found and returned.

```
M0388   LD D,$00                Get "E" mode character from
        ADD HL,DE               selected table.
        LD A,(HL)
        RET
```

K-KLC-LET
Letter keys in 'K', 'L' or 'C' modes are now considered. But first the special SYMBOL SHIFT codes have to be dealt with.

```
M038D   LD HL,$024B             Load table 5 offset.
        BIT 0,B                 Jump if symbol shift.
        JR Z, K-LOOK-UP
        BIT 3,D                 Jump if "K" mode.
        JR Z, K-TOKENS
        BIT CAPS_L,(IY+OFLAGS2) Return if CAPS LOCK in FLAGS2
        RET NZ                  or CAPS SHIFT.
        INC B                   Return if upper case (B=$FF).
        RET NZ
        ADD A,$20               Convert to lower case.
        RET
```

K-TOKENS
The 'final code' values for tokens are found by adding +A5 to the 'main code’.

```
M03A2   ADD A,$A5               Convert to keyword.
        RET
```

K-DIGIT
Key code is <$3A, so it’s a digit key, SPACE, ENTER or SHIFT.

```
M03A5   CP $30                  Return if key <$30 (unprintable).
        RET C
        DEC C
        JP M ,K-KLC-LET         Jump with 'K', 'L' & 'C' modes;
        JR NZ,K-GRA-DGT         and also with 'G' mode.
        LD HL,$0276             Get "E" mode key table.
        BIT 5,B                 Jump if symbol shift and a digit
        JR Z,K-LOOK-UP          key in extended mode.
```

```
        CP $38                  Jump if digit is 8 or 9.
        JR NC,K-8-&-9
```

Digit key 0-7 in extended mode are either PAPER or INK colors.

```
        SUB $20                 Reduce the range to $10 - $17.
        INC B                   Return with this PAPER color code
        RET Z                   if CAPS SHIFT is not being used.
        ADD A,$08               CAPS SHIFT is used; convert to $18
        RET                     $18 - $1F for INK color code.
```

K-8-&-9
The keys 8 and 9 in extended mode are BRIGHT & FLASH codes.

```
M03C0   SUB $36                 Convert to BRIGHT/UNBRIGHT.
        INC B                   Return if no SHIFT.
        RET Z
        ADD A,$FE               Convert to FLASH/UNFLASH.
        RET
```

K-GRA-DGT
The digit keys in graphics mode are the block graphic characters (+80 to +8F), GRAPHICS code (+0F) and DELETE code (+0C).

```
M03C7   LD HL,$0252             Get "G" mode key table.
        CP $39
        JR Z,K-LOOK-UP          Jump if character is "0" or "9"
        CP $30
        JR Z,K-LOOK-UP
        AND $07
        ADD A,$80               Make graphics mosaic patterns.
        INC B                   Return if no SHIFT.
        RET Z
        XOR $0F                 Invert pattern if SHIFT.
        RET
```

K-KLC-DGT
Last, evaluate the digit keys in 'K', 'L' & 'C' modes.

```
M03DB   INC B                   Return if no SHIFT.
        RET Z
        BIT 5,B                 Is it SYMBOL SHIFT?
        LD HL,$0252             Jump if not SYMBOL SHIFT.
        JR NZ,K-LOOK-UP
        SUB $10                 Convert to symbol.
        CP $22                  Jump if ampersand "@".
        JR Z,K-@-CHAR
        CP $20                  Return for space.
        RET NZ
        LD A,$5F                Make underline "_".
```

```
        RET
```

K-@-CHAR

```
M03F0   LD A,$40                Return ampersand "@".
        RET
```

Speaker Routines

In the ZX Spectrum, sound is produced by alternating the value of bit 4 on port FE. These routines support the BEEP command and its options. The SOUND command is in the **BASIC Line and Command Interpretation** section.

Beeper Subroutine

DE contains number of cycles - 1. HL contains the waveform period (8*HL+236 to 8*HL+246 t-states).

BEEPER (PARP)

```
M03F3   DI                      Disable the interrupt during 'beep'.
        LD A,L                  Save L.
        SRL L                   Divide L by 4.
        SRL L
        CPL                     Initialize fine tune for the
        AND $03                 requested period.
        LD C,A
        LD B,$00
        LD IX, $040F            The base address for the timing loop.
        ADD IX,BC               Alter the length of the timing loop.
        LD A,(BORDCR)           Fetch the border color.
        AND $38                 Make it compatible with the port at $FE.
        RRCA
        RRCA
        RRCA
        OR $08                  Set the MIC output to 'off.'
```

A holds a template value to be output to the speaker port. Bit 4 (value $10) is toggled to make the noise.

The NOPs below allow fine tuning of the period.

BE-IX+3

```
M040F   NOP
```

BE-IX+2

```
        NOP
```

BE-IX+1

```
        NOP
```

```
BE-IX+0
          INC B
          INC C

BE-H&L-LP
M0414     DEC C                 Timing loop based on C.
          JR NZ,BE-H&L-LP
          LD C,$3F
          DEC B                 Timing loop based on B.
          JP NZ,BE-H&L-LP
          XOR $10               Toggle the speaker output (bit 4).
M041F     OUT ($FE),A           Perform OUT leaving BORDER unchanged.
          LD B,H                Reset the B register.
          LD C,A                Save the A register.
          BIT 4,A               Jump if the speaker bit is 1.
          JR NZ,BE-AGAIN
          LD A,D
          OR E                  Exit if cycle counter is zero.
          JR Z,BE-END
          LD A,C                Fetch saved value for C.
          LD C,L                Reset the C register.
          DEC DE                Decrement the loop counter.
          JP (IX)               Back around for more cycles.

BE-AGAIN
M0430     LD C,L                BC contains the period value
          INC C                 so loop back for more cycles.
          JP (IX)

BE-END
M0434     EI                    Enable interrupts.
          RET
```

BEEP Command Routine

Two numbers on the calculator stack represent the pitch of the note (top number) and the duration (second number).

```
BEEP
M0436     RST $28               Call the floating point calculator.
          DEFB $31              Duplicate pitch value.
          DEFB $27              Find the integer portion of the note.
          DEFB $C0              T -> MEM 0 copy to MEM 0.
          DEFB $03              SUB calc now has the fractional portion of the
                                note value.
          DEFB $34              Stack the decimal value
          DEFB $EC              0.0577622606.
          DEFB $6C
          DEFB $98
          DEFB $1F
```

```
                DEFB $F5
                DEFB $04            Multiply T.
                DEFB $A1            Stack one.
                DEFB $0F            Add.
                DEFB $38            End calc. Calc now has a factor that represents
                                    the ratio of the requested frequency to a full step
                                    in frequency.
                LD HL,MEMBOT        Get the calculator's memory (MEM 0).
                LD A,(HL)           Fetch the exponent of integer part of note.
                AND A               Jump if the number is not an integer.
                JR NZ,REPORT-B
                INC HL              Get the sign byte.
                LD C,(HL)
                INC HL              Get the low byte of the number.
                LD B,(HL)
                LD A,B
                RLA                 Check to see if the number is C is less than
                SBC A,A             128 and zero the accumulator.
                CP C
                JR NZ,REPORT-B
                INC HL
                CP (HL)             Jump if high byte is not zero.
                JR NZ,REPORT-B
                LD A,B              A now has the note code.
                ADD A,$3C           Jump if the note code is 68 or less.
                JP P, BE-I-OK
                JP PO, REPORT-B     Jump if overflow, A must be < -60.
```

The note number (-60 to +69 has now been shifted to $0 to $81). Next, find the correct pitch.

BE-I-OK

```
M0463   LD B,$FA            Start 6 octaves below middle C.
```

BE-OCTAVE

```
M0465   INC B               Reduce I to find the correct octave.
        SUB $0C
        JR NC,BE-OCTAVE
        ADD A,$0C           A now contains the note within the octave stored
                            in B.
        PUSH BC             Save the octave number.
        LD HL,$04AC         Base of the note table.
        CALL LOC-MEM        HL = HL + 5 * A
        CALL STACK-NUM      Put the number on the calc stack.
        RST $28             Call the floating point calculator.
        DEFB $04            Multiply the base frequency by the factor
                            computed above. This gives the final frequency.
        DEFB $38            Exit calculator.
        POP AF              Get the octave number.
        ADD A,(HL)          2^N*note_number.
```

```
LD (HL),A
RST $28          Call the floating point calculator.
DEFB $C0         T -> MEM 0. MEM 0 now contains the final
                 frequency for the desired note.
DEFB $02         Remove the frequency from the FP stack.
DEFB $31         Duplicate the duration value.
DEFB $38         Exit calculator.
CALL FIND-INT1   Convert the FP top of stack to a value in A
CP $0B
JR NC,REPORT-B
```

Translate the duration value to the equivalent number of cycles for PARP.

```
RST $28          Call the floating point calculator.
DEFB $E0         MEM 0 -> T. Get the desired frequency.
DEFB $04         Multiply by required the duration to give the # of
                 cycles.
DEFB $E0         MEM 0 -> T again, get the frequency.
DEFB $34
DEFB $80
DEFB $43
DEFB $55,$9F,$80 f*t ,f, 437500
DEFB $01         SWAP
DEFB $05         Convert the frequency into a period value for
                 PARP.
DEFB $34,$35,$71 f*t, 30.125
DEFB $03         Subtract
DEFB $38         Exit calculator.
```

Note: The value 437,500/f gives the half-cycle length of the note and reducing it by 30.125 allows for 120.5 T states in which to actually produce the note and adjust the counters.

```
CALL FIND-INT2   Period on FP stack to BC.
PUSH BC          Save frequency.
CALL FIND-INT2   Convert duration to BC.
POP HL           Restore the frequency.
LD D,B           Move the duration to DE.
LD E,C
```

Before making the beep, test the value f*t.

```
LD A,D           Return if f*t results in no cycles.
OR E
RET Z
DEC DE           Decrease the duration for PARP.
JP BEEPER        Make the noise.
```

REPORT-B

```
M04AA   RST $08          Error: Integer out of range.
        DEFB $0A
```

Semi-tone Table

Holds the frequencies of the twelve semi-tones in an octave.

```
M04AC   DEFB $89,$02,$D0,$12,$86    261.63 Hz C
        DEFB $89,$0A,$97,$60,$75    277.18 Hz C#
        DEFB $89,$12,$D5,$17,$1F    293.66 Hz D
        DEFB $89,$1B,$90,$41,$02    311.12 Hz D#
        DEFB $89,$24,$D0,$53,$CA    329.63 Hz E
        DEFB $89,$2E,$9D,$36,$B1    349.23 Hz F
        DEFB $89,$38,$FF,$49,$3E    369.99 Hz F#
        DEFB $89,$43,$FF,$6A,$73    3912 Hz G
        DEFB $89,$4F,$A7,$00,$54    415.30 Hz G
        DEFB $89,$5C,$00,$00,$00    440 Hz A
        DEFB $89,$69,$14,$F6,$24    466.16 Hz A#
        DEFB $89,$76,$F1,$10,$05    493.88 Hz B
```

ZX81 Program Name Subroutine

Subroutine used by the ZX81; was not removed when ROM was rewritten for the Spectrum.

```
M04E8   CALL SCANNING       Evaluate the current expression
        LD A,(FLAGS)        Jump if the expression is numerical
        ADD A,A
        JP M, REPORT-C      Error - BAD BASIC
        POP HL              Restore HL
        RET NC              Return if syntax checking
        PUSH HL             Save HL
        CALL STK-FETCH      Get the string descriptor
        LD H,D              String location to DE
        LD L,E
        DEC C               Return if the string length is zero
        RET M
        ADD HL,BC           Point to last character of string
        SET 7,(HL)          Set bit 7
        RET
```

Screen/Printer Handling Routines

These routines handle all of printing to the main part of the screen, the lower part of the screen and the printer.

Input/output routines of the 2068 are vectored through the channel & stream information areas. On the stock 2068, input is only possible from the keyboard but output can be directed to the printer, the upper part of the display or the lower part of the display.

The major input routine in this part is the EDITOR that allows the user to enter characters into the lower part of the display.

The PRINT-OUT routine is a rather slow routine as it is used for all means of updating screen data. For example, adding of a single byte to the display area involves considering the status of OVER and INVERSE each time.

This section begins a major departure from the original Spectrum ROM. In the Spectrum, the cassette I/O routines follow the speaker routines.

PRINT-OUT Routines

All printing to the upper and lower portions of the screen, as well as to the printer, is handled by these routines. PRINT-OUT is entered with the code for a control character, a printable character or a token in the A register.

PRINT-OUT (SENDTV)

```
M0500   CALL PO-FETCH           Update DFCC and DFLCC.
        CP $20                  Jump if code is not a control
        JP NC,PO-CHAR           character.
        CP $0C                  Jump if character is not DELETE.
        JR NZ, PO-1
        BIT TOKEN,(IY+OFLAGS)   Is TOKEN on or off?
        JP Z, PO-CHAR           Jump if TOKEN is set.
```

PO-1

```
M0513   CP $06                  Jump if char is less than COMMA.
        JR C,PO-QUEST           All of these are not used.
        CP $18                  Jump if char is greater than TAB.
        JR NC,PO-QUEST
```

Printing a control character, get the jump address for the code to handle it.

```
        LD HL,CONTRO - 6        Base of jump table - 6.
        LD E,A
        LD D,$00                Form the address of the jump table entry.
        ADD HL,DE
        LD E,(HL)               Get the code offset.
        ADD HL,DE               Make the address of the code handler.
        PUSH HL                 Save it.
```

```
              JP PO-FETCH          Jump to the print routine. This is effectively a call
                                   to the stacked address after DFCC and DFLCC
                                   have been set up.
```

Control Character Jump Table (Characters 6-24)

Offsets to routines that handle the control characters.

```
CONTRO
M0528   DEFB $4E                   Comma (P_COMMA)
        DEFB $57                   Edit (PO-QUEST)
        DEFB $10                   Cursor left (P_LFT)
        DEFB $29                   Cursor right (P_RT)
        DEFB $54                   Cursor down (PO-QUEST)
        DEFB $53                   Cursor up (PO-QUEST)
        DEFB $52                   Delete (PO-QUEST)
        DEFB $37                   Enter
        DEFB $50                   Not used (PO-QUEST)
        DEFB $4F                   Not used (PO-QUEST)
        DEFB $5F                   Ink (PO-TV-1)
        DEFB $5E                   Paper (PO-TV-1)
        DEFB $5D                   Flash  (PO-TV-1)
        DEFB $5C                   Bright (PO-TV-1)
        DEFB $5B                   Inverse (PO-TV-1)
        DEFB $5A                   Over (PO-TV-1)
        DEFB $54                   At (PO-2-OPER)
        DEFB $53                   Tab (PO-2-OPER)
```

Cursor Left Subroutine

B register holds the current line number and C register the current column number.

```
PO-BACK-1 (P_LFT)
M053A   INC C
        LD A,$22                   Move left one column.
        CP C                       Unless against the left side of the screen.
        JR NZ, PO-BACK-3
        BIT PR,(IY+OFLAGS)         Jump forward if printing to the printer.
        JR NZ, PO-BACK-2
        INC B                      Against the left, go up a line.
        LD C,$02                   Set column value.
        LD A,$19                   At the top line?
        CP B
        JR NZ, PO-BACK-3           Nope, good.
        DEC B                      Too far, back down a line.

PO-BACK-2 (PR-CUR-L-2)
M054F   LD C,$21                   Set to left column.

PO-BACK-3 (PR-CUR-L-3)
M0551   JP CL-SET                  Return via CL-SET.
```

Cursor Right Subroutine

B register holds the current line number and C register the current column number. Performs the equivalent of PRINT OVER 1; CHR$ 32; - in BASIC.

PO-RIGHT (P_RT)

```
M0554   LD A,(PFLAG)          Get printing flags and
        PUSH AF               save them.
        LD (IY+OPFLAG),01     Set print flag to OVER 1.
        LD A, $20             Print a space to the TV.
        CALL PO-CHAR
        POP AF                Restore the print flags.
        LD (PFLAG),A          Restore PLFAG.
        RET
```

Carriage Return Subroutine

If the output is going to the printer, then empty printer buffer. If the printing to the screen, test for 'scroll?' before decrementing the line number. Number of lines in B.

PO-ENTER (P_NL)

```
M0566   BIT PR,(IY+OFLAGS)    Jump forward if printing to printer.
        JP NZ, COPY-BUFF
        LD C, $21             Set left column.
        CALL PO-SCR           Return here if the screen is not full.
        DEC B                 Move down a line.
        JP CL-SET
```

Print Comma Subroutine

Current column value is manipulated and the A register set to $00 (for TAB 0) or $10 (for TAB 16).

PO-COMMA (P_COMMA)

```
M0576   CALL PO-FETCH         Get display file pointer (DE is the
        LD A,C                important one).
        DEC A                 Move right two columns.
        DEC A
        AND $10
        JR PO-FILL
```

Print a Question Mark Subroutine

Print question mark when asked to print an unprintable code.

PO-QUEST

```
M0580   LD A,'?'              Output a question mark bit pattern.
        JR PO-CHAR
```

Control Characters With Operands

The control characters from INK to OVER require a single operand. AT and TAB are followed by two operands.

PO-TV-2

```
M0584   LD DE, $059E            Save the first operand in TVDATA-hi
        LD (TVDATA+1),A         and change the address of the output
        JR PO-CHANGE            routine to PO-CONT.
```

PO-2-OPER

Enter here when handling the AT and TAB characters (two operands).

```
M058C   LD DE, $0584            Change address of output routine to
        JR PO-TV-1              PO-TV-2.
```

PO-1-OPER

Enter when handling INK to OVER.

```
M0591   LD DE, $059E            Change output routine to PO-CONT.
```

PO-TV-1

```
M0594   LD (TVDATA),A           Save the control code to TVDATA-lo.
```

PO-CHANGE (CHANGE)

Temporarily change output routine address.

```
M0597   LD HL,(CURCHL)          Put address of the output channel in HL.
        LD (HL),E               Enter the new output routine address and
        INC HL                  force the next character code to be
        LD (HL),D               treated as an operand.
        RET
```

PO-CONT

All necessary data collected, continue.

```
M059E   LD DE, PRINT-OUT        Restore the original address for
        CALL PO-CHANGE          PRINT-OUT.
        LD HL,(TVDATA)          Fetch the control code and first operand.
        LD D,A                  The last operand and the control
        LD A,L                  code are moved.
        CP $16                  Jump forward if handling INK to OVER.
        JP C, CO-TEMPS
        JR NZ, PO-TAB           Jump forward if handling TAB.
```

SET_AT

Handle the AT code.

```
M05B0   LD B,H                  Line number.
        LD C,D                  Column number.
        LD A,$1F                Reverse the column number.
        SUB C
```

```
                JR C, PO-AT-ERR         Jump if not in range.
                ADD A,$02               Add offset to make C $21 - $22
                LD C,A
                BIT PR,(IY+OFLAGS)      Jump if printing to printer
                JR NZ, PO-AT-SET
                LD A,$16                Reverse the line number.
                SUB B
```

PO-AT-ERR

```
M05C3   JP C, REPORT-B          Jump forward if range ok.
        INC A                   Adjust the range.
        LD B,A
        INC B
        BIT LHS,(IY+OTVFLAG)    If printing to the lower part of the screen,
        JP NZ, PO-SCR           consider whether scroll is needed.
        CP (IY+ODFSZ)
        JP C, REPORT-5          Out of screen, report error 5.
```

PO-AT-SET

```
M05D6   JP CL-SET
```

PO-TAB

Handle TAB.

```
M05D9   LD A,H                  Fetch the first operand.
```

PO-FILL

```
M05DA   CALL PO-FETCH           The current print position.
        ADD A,C                 Add the current column value.
        DEC A                   Find out how many 'spaces', MOD 32,
        AND $1F                 are required and return if the result is zero.
        RET Z
        LD D,A                  Use 0 as the counter.
        SET SPC,(IY+OFLAGS)     Suppress 'leading space'.
```

PO-SPACE

```
M05E7   LD A,$20                Print number of spaces specified by D.
        CALL PO-SAVE
        DEC D
        JR NZ, PO-SPACE
        RET
```

Printable Character Codes

Character(s) are printed by PO-ANY and flow continues to PO-STORE.

PO-ABLE (PO-CHAR)

```
M05F0   CALL PO-ANY          Print the characters and continue below.
```

Position Store Subroutine

Set the new column position and display file pointers for the TV based on various flags

PO-STORE (STTVCU)

```
M05F3   BIT PR,(IY+OFLAGS)       Jump if printing to printer.
        JR NZ, PO-ST-PR
        BIT LHS,(IY+OTVFLAG)     Jump if printing to the
        JR NZ, PO-ST-E           lower half of the screen.
        LD (SPOSN),BC            Update the upper screen column pointer.
        LD (DFCC),HL             Update upper display file character
        RET                      pointer.
```

PO-ST-E

Save the lower screen parameters.

```
M0607   LD (SPOSNLCOL),BC        Update lower screen col pointer.
        LD (ECHOE),BC            Update keyboard buffer pointer.
        LD (DFCCL),HL            Update lower display file character pointer.
        RET
```

PO-ST-PR

```
M0613   LD (IY+OPPOSN),C         Update print position.
        LD (PRCC),HL             Update printer buffer character pointer.
        RET
```

PO-FETCH

Load registers with display information. HL contains display file position for a character.

BC loaded with the LIN (B) and COL (C) of the current screen position.

PO-FETCH (LDTVCU)

```
M061A   BIT PR,(IY+OFLAGS)       Jump if printing to printer.
        JR NZ, PO-F-PR
        LD BC,(SPOSNCOL)         Get current print column position.
        LD HL,(DFCC)             Load into the display file column.
        BIT LHS,(IY+OTVFLAG)     Return if printing to the upper half
        RET Z                    of the screen.
        LD BC,(SPOSNLCOL)        Put the lower screen print column position
        LD HL,(DFCCL)            into the display column pointer.
        RET
```

PO-F-PR

```
M0634   LD C,(IY+OPPOSN)         Get the print buffer position.
```

```
        LD HL,(PRCC)
        RET
```

Print Any Character(s) Subroutine

Put a character on the screen. A contains the character, DE contains the character position in the display file.

PO-ANY
Handle the printable characters.

```
M063B   CP $0C                  Jump if char is not DELETE
        JR NZ, PO-ANY-NOT-DEL
        LD A,$7A
        JR PO-T                 Jump to print.
```

PO-ANY-NOT-DEL

```
M0643   CP $7C                  Jump if char is STICK.
        JR Z, PO-T
        CP $7E                  Jump if char is FREE.
        JR Z, PO-T
        CP $7B                  Jump if char is printable.
        JR C, PO-ANY-C
        CP $80                  Jump if char is block graphics 0.
        JR NC, PO-ANY-C
        BIT TOKEN,(IY+OFLAGS)   Is TOKEN on?
        JR Z, PO-T              Yes, print the token.
```

PO-ANY-C
Do some more tests, then get the character.

```
M0659   CP $80                  Jump forward if it's an ordinary print code.
        JR C, PO-CHAR
        CP $90                  Jump if is a UDG or token.
        JR NC, PO-TOKUDG
        LD B,A                  Graphic code, load the character in B.
        CALL PO-GR-1            Generate a block graphic of the character code in
                                B.
        CALL PO-FETCH           HL now has the display file position.
        LD DE, MEMBOT           point to the block graphic bit map buffer
        JR PR-ALL
```

PO-GR-1 (MKBLKGR)
Dynamically generate the block graphics characters.

```
M066D   LD HL,MEMBOT            Point to free space, provides a buffer for the
                                character bit map.
        CALL PO-GR-2            Generate the first four scan lines
                                (bits 0 and 1).
```

PO-GR-2

```
M0673   RR B            Determine bits 0 and 2 of graphic code.
        SBC A,A         If LSB is 1, A=$FF else A=$00.
        AND $0F         Keep only the bits needed for the character.
        LD C,A          Save the character.
        RR B            Check bit 1 (and bit 3).
        SBC A,A         Same as above.
        AND $F0
        OR C            Combine the results.
        LD C,$04
```

PO-GR-3

```
M0681   LD (HL),A       A holds upper half of character,
        INC HL          duplicate it four times.
        DEC C
        JR NZ, PO-GR-3  Then do the lower half of the character.
        RET
```

PO-T&UDG (PO-TOKUDG)

Handle the user defined graphics and tokens.

```
M0687   SUB $A5         Jump if a token, not a UDG.
        JR NC,PO-T
        ADD A,$15       Adjust value for UDG bit map table.
        PUSH BC         Save current position.
        LD BC,(UDG)     Load BC with the base of the UDG table.
        JR PO-CHAR-2
```

PO-T

```
M0694   CALL PO-TOKENS  Print the token in A.
        JP PO-FETCH     HL points to the display file.
```

PO-CHAR

```
M069A   PUSH BC         Save the current position.
        LD BC,(CHARS)   Get the base address of the character
                        bit map table.
```

PO-CHAR-2

```
M069F   EX DE,HL        Save the print address.
        LD HL,FLAGS     Force no printing leading spaces.
        RES SPC,(HL)
        CP $20          Jump if the character is not a space.
        JR NZ,PO-CHAR-3
        SET SPC,(HL)    Force printing leading spaces.
```

PO-CHAR-3

```
M06AB   LD H,$00        Put character code in HL.
        LD L,A
        ADD HL,HL       Multiply by 8.
        ADD HL,HL
```

```
        ADD HL,HL
        ADD HL,BC       HL contains the bit map for the character.
        POP BC          Restore current position.
        EX DE,HL        Store bit map location in DE.
```

Print All Characters Subroutine

Print all 8*8 bit characters. DE points to the character bit map, HL points to the display file or printer buffer. BC contains line and column.

```
PR-ALL
M06B4   LD A,C              Put column position to A.
        DEC A               Move right one column.
        LD A,$21            Jump forward if new line.
        JR NZ, PR-ALL-1
        DEC B               Move down one line.
        LD C,A
        BIT PR,(IY+OFLAGS)  Jump if printing to TV.
        JR Z, PR-ALL-1
        PUSH DE             Save the bit map buffer address.
        CALL COPY-BUFF      Flush the printer buffer to the printer.
        POP DE              Restore the buffer address.
        LD A,C              Get the new column number.

PR-ALL-1
M06C8   CP C                Test whether a new line is needed.
        PUSH DE             Save the bit map buffer address.
        CALL Z ,PO-SCR      Return here if display is not full.
        POP DE              Restore the bit map buffer address.
        PUSH BC             Save current position and
        PUSH HL             display file to the stack.
        LD A,(PFLAG)        Get printing flags and read bit 0.
        LD B,$FF            Put OVER mask in B.
        RRA                 Jump if OVERing new and
        JR C, PR-ALL-2      old characters.
        INC B

PR-ALL-2
M06D9   RRA                 Check bit 2 of PFLAG and put
        RRA                 INVERSE mask in C register.
        SBC A,A             A=$FF if INVERSE, else A=0.
        LD C,A              Put the value in C.
        LD A,$08            Set A to hold pixel-line counter and
        AND A               clear carry flag.
        BIT PR,(IY+OFLAGS)  Jump if printing to TV.
        JR Z, PR-ALL-3
        SET PRLEFT,(IY+OFLAGS2) Force print buffer to not empty.
        SCF                 Set carry flag to show printer being used.
```

PR-ALL-3

```
M06EB   EX DE,HL            HL contains the bit map address,
                            DE contains display file pointer.
```

PR-ALL-4

Character can now be printed. Eight passes are make through the loop, one for each pixel line.

```
M06EC   EX AF,AF'           Carry flag set when using the printer. Save flag to
                            F'.
        LD A,(DE)           Get the first byte of display file character.
        AND B               Use the OVER mask then
        XOR (HL)            XOR the bit map.
        XOR C               If INVERSE, do that.
        LD (DE),A           Save back to display file.
        EX AF,AF'           Fetch printer flag and jump forward
        JR C, PR-ALL-6      if necessary.
        INC D               Point to next byte in the display file.
```

PR-ALL-5

```
M06F6   INC HL              Point to next bit map character.
        DEC A               Decrement byte counter.
        JR NZ, PR-ALL-4     Jump back until all eight lines are done.
        EX DE,HL            Put high-address for character area in H.
        DEC H
        BIT PR,(IY+OFLAGS)  Set attribute byte for this character, only
        CALL Z, PO-ATTR     for the screen.
        POP HL              Restore destination and position values.
        POP BC
        DEC C               Decrease the column number and
        INC HL              increase destination address.
        RET
```

PR-ALL-6

When sending to the printer, the destination has to be updated by $20.

```
M0708   EX AF,AF'           Save the printer flag.
        LD A,$20            Update the destination address by $20
        ADD A,E             for print buffer rather than the display file.
        LD E,A
        EX AF,AF'           Fetch printer flag.
        JR PR-ALL-5         Go back to the loop.
```

Set Attribute Byte Subroutine

Identify and fetch the attribute byte. The new attribute value is set from the old value, ATTRT, MASKT and PFLAG.

PO-ATTR (ATTBYT)

```
M0710   LD A,H              Get the high byte of the display file address
        RRCA                Divide by 8 then AND with $03 to
```

```
            RRCA                    determine which third of the screen is
            RRCA                    being addressed.
            AND $03
            OR $58                  Generate the high byte of the
            LD H,A                  attribute address.
            LD DE,(ATTRT)           D holds ATTRT, E holds MASKT
            LD A,(HL)               Get the current attribute.
            XOR E                   For every bit in D that is a 0,
            AND D                   the corresponding bit in A will
            XOR E                   come from E, else it will come from A.
            BIT F_CB, (IY+OPFLAG)   If the foreground is complement of
            JR Z,PO-ATTR-1          background, jump ahead.
            AND $C7                 Keep flash, bright and foreground colors.
            BIT 2,A                 Jump if green is active.
            JR NZ,PO-ATTR-1
            XOR $38                 Invert foreground colors.
```

PO-ATTR-1

```
M072F       BIT F_CB,(IY+OPFLAG)    If the foreground is complement of
            JR Z,PO-ATTR-2          background, jump ahead.
            AND $F8                 Keep flash, bright and background colors.
            BIT 5,A                 Jump if green is active
            JR NZ, PO-ATTR-2
            XOR $07                 Invert background colors.
```

PO-ATTR-2

```
M073D       LD (HL),A               Store the new attribute value.
            RET
```

Message Printing Subroutine

Print messages and tokens. Register A holds the message or token entry number in a table. DE register pair holds the base address of the table.

PO-MSG (PUTMES)

```
M073F       PUSH HL                 The high byte of the last entry
            LD H,$00                on the machine stack is made zero
            EX (SP),HL              to suppress trailing spaces.
            JR PO-TABLE             Go find string in buffer.
```

PO-TOKENS (PRTTOK)

Print the token represented by value is in A.

```
M0745       LD DE,$0098             Base of token table.
            CP $5B                  Adjust token number, if necessary (range is
            JR C,M074E              $00 - $5A).
            SUB $1F
M074E       PUSH AF                 Save the code on the stack.
```

PO-TABLE

```
M074F   CALL PO-SEARCH          Find the message number A in table at (DE).
        JR C, PO-EACH           No leading space needed, so jump ahead.
        LD A,$20                Print a leading space, if needed, before
        BIT SPC,(IY+OFLAGS)     the message/token.
        CALL Z, PO-SAVE
```

PO-EACH
Characters of the message/token are printed.

```
M075D   LD A,(DE)               Get the character.
        AND $7F                 Cancel any 'inverted' bit.
        CALL PO-SAVE            Put the char to the current stream.
        LD A,(DE)               Re-get the character.
        INC DE                  Advance the message pointer.
        ADD A,A                 Back around for more if the
        JR NC,PO-EACH           high bit of the character is not set.
        POP DE                  Is a trailing space required?
        CP $48                  Jump if the character is "$"
        JR Z,PO-TR-SP
        CP $82                  Return if char is less than "A"
        RET C
```

PO-TR-SP (PO-TRSP)
Evaluate for printing a trailing space.

```
M0770   LD A,D                  Examine value in D; return if
        CP $03                  it is a message, RND, INKEY$
        RET C                   or PI.
        LD A,$20                Output a space.
```

PO-SAVE Subroutine

Subroutine allows for characters to be printed recursively. Registers are saved while PRINT-OUT is called.

PO-SAVE (PR_TV2)

```
M0776   PUSH DE                 Save character pointer (DE).
        EXX                     Save HL & BC.
        RST $10                 Output the character in A to the current stream.
        EXX                     Restore HL & BC.
        POP DE                  Restore character pointer (DE).
        RET
```

Table Search Subroutine

Find the message with index A located in table at DE. Carry flag is reset if a leading space should be printed.

PO-SEARCH (FINDMSG)

```
M077C   PUSH AF             Save message number index.
        EX DE,HL            Put base address in HL.
        INC A               Bump message number.
```

PO-STEP

```
M077F   BIT 7,(HL)          Skip over characters until an inverted
        INC HL              character is found.
        JR Z, PO-STEP
        DEC A               Decrement msg number, count through
        JR NZ, PO-STEP      until the correct message is found.
        EX DE,HL            DE points to the initial character.
        POP AF              Restore the entry number and return with
        CP $20              carry set for the first thirty two entries.
        RET C
        LD A,(DE)           If the initial character is a letter then a
        SUB $41             leading space may be needed.
        RET
```

Test for Scroll Subroutine

Called whenever there might be the need to scroll the display. This occurs when handling a carriage return; when using AT in an INPUT line or when the current line is full and the next line has to be used. B holds the line number under test. If the screen is full, exit via ERROR 5 - OUT OF SCREEN.

PO-SCR (TVFUL)

```
M0790   BIT PR,(IY+OFLAGS)  Return if printing to printer.
        RET NZ
        LD DE,CL-SET        Load the machine stack with the address
        PUSH DE             of CL-SET.
        LD A,B              Put line number in A.
        BIT LHS,(IY+OTVFLAG) Jump if 'INPUT … AT' in the lower
        JP NZ, PO-SCR-4     half of the screen.
        CP (IY+ODFSZ)       Return if the line number is greater than
        JR C, REPORT-5      the value of DFSZ. Exit via error 5
        RET NZ              if we are not on the last line.
        BIT TVLIST,(IY+OTVFLAG) Jump if not an automatic listing.
        JR Z,PO-SCR-2
        LD E,(IY+OBREG)     Fetch the line counter.
        DEC E               Decrease the counter.
        JR Z,PO-SCR-3       Jump forward if the listing it to be scrolled.
        LD A,$00            Otherwise, open channel K,
        CALL CHAN-OPEN      restore the stack pointer,
        LD SP,(LISTSP)      flag that the automatic listing has finished
        RES TVLIST,(IY+OTVFLAG) and return.
```

```
        RET
```

REPORT-5 (ERROR-5)

```
M07C1   RST $08                 Error: Out of screen.
        DEFB $04
```

PO-SCR-2

Determine if the "Scroll?" prompt is needed. B contains the number of lines to scroll.

```
M07C3   DEC (IY+OSCRCT)         Decrement scroll count.
        JR NZ,PO-SCR-3          Jump if scroll count not zero (proceed)
        LD A,$18                Reset counter.
        SUB B                   Subtract screen line number.
        LD (SCRCT),A            Store the number of lines below B back
        LD HL,(ATTRT)           into scroll count. Get the temporary
        PUSH HL                 printing attributes and save them.
        LD A,(PFLAG)            Save the printing flags.
        PUSH AF
        LD A,$FD                Stream -3 (keyboard/lower screen)
        CALL CHAN-OPEN
        XOR A
        LD DE, SCROLLMSG        Print the "SCROLL?" message.
        CALL PO-MSG
        SET CLHS,(IY+OTVFLAG)   Force clearing the lower half screen.
        LD HL,FLAGS
        SET LMODE2,(HL)         Force "L" mode at cursor.
        RES KEYHIT,(HL)         Force no keyboard input.
        EXX
        CALL WAIT-KEY           Read the keyboard.
        EXX
        CP $20                  Jump if a space.
        JR Z,REPORT-D
        CP $E2                  Jump if "STOP" token.
        JR Z,REPORT-D
        OR $20                  Force lower case.
        CP $6E                  Jump to BREAK if "n" is input.
        JR Z,REPORT-D
        LD A,$FE                Open channel 'S' (main screen).
        CALL CHAN-OPEN
        POP AF                  Restore the printing flags/
        LD (PFLAG),A
        POP HL                  Restore the temporary attributes
        LD (ATTRT),HL
```

PO-SCR-3

Scroll the display.

```
M080D   CALL CL-SC-ALL          Scroll the screen one line.
        LD B,(IY+ODFSZ)         The line and column numbers for the
        INC B                   start of the line above the lower part of
```

```
              LD C,$21          the display are found and saved.
              PUSH BC
              CALL CL-ADDR      Get the display file address for line "B".
              LD A,H            HL holds the address of the byte.
              RRCA
              RRCA
              RRCA
              AND $03
              OR $58
              LD H,A
```

Line in question has 'lower part' attribute values and the new line at the bottom of the display may have 'ATTR-P' values so the attribute values are exchanged.

```
              LD DE,$5AE0       Line 23 attribute address.
              LD A,(DE)         A = line 23 attribute.
              LD C,(HL)         C = line "B" attribute.
              LD B,$20          There are 32 bytes.
              EX DE,HL          HL -> line 23, DE -> line "B".
```

PO-SCR-3A

```
M082B         LD (DE),A         Make the first exchange and proceed
              LD (HL),C         to use the same value for the thirty two
              INC DE            attribute bytes of the two lines being
              INC HL            handled.
              DJNZ PO-SCR-3A
              POP BC            Restore line and column numbers of upper
              RET               part before returning.
```

SCROLLMSG

```
M0833         DEFB $80
              DEFM "scroll"&('?'+$80)
```

REPORT-D

```
M083B         RST $08           Error: Break - Cont repeats.
              DEFB $0C
```

PO-SCR-4

Handle the lower part of the display.

```
M083D         CP $02            The 'out of screen' error is given if
              JR C,REPORT-5     the lower part is going to be 'too large'
              ADD A,(IY+ODFSZ)  and return if scrolling is not necessary.
              SUB $19
              RET NC
              NEG               A now holds the number of scrolls.
              PUSH BC           Save line and column numbers.
              LD B,A            Get scroll number.
              LD HL,(ATTRT)     Save MASKT and PLFAG.
              PUSH HL
```

```
        LD HL,(PFLAG)
        PUSH HL
        CALL TEMPS
        LD A,B              Get scroll number.
```

PO-SCR-4A
The lower part of the screen is scrolled A number of times.

```
M0857   PUSH AF             Save the number.
        LD HL,DFSZ          Get DFSZ.
        LD B,(HL)           Put in B.
        LD A,B              Copy to A.
        INC A               Increment A.
        LD (HL),A           Put at in DFSZ.
        LD HL,SPOSNLIN
        CP (HL)
        JR C,PO-SCR-4B      Jump if only lower part is scrolled.
        INC (HL)            Otherwise, inc SPOSNLIN and scroll
        LD B,$18            entire display.
```

PO-SCR-4B

```
M0868   CALL CL-SCROLL
        POP AF              Fetch and decrement the scroll number.
        DEC A
        JR NZ,PO-SCR-4A     Jump back until finished.
        POP HL              Restore value of PFLAG.
        LD (IY+OPFLAG),L
        POP HL              Restore values of ATTRT
        LD (ATTRT),HL       and MASKT.
        LD BC,(SPOSNCOL)    If SPOSNCOL has been changed,
        RES LHS,(IY+OTVFLAG) give matching value to DFCC.
        CALL CL-SET
        SET LHS,(IY+OTVFLAG) Reset TVFLAG to indicate lower part
        POP BC              has been handled, restore line and col.
        RET
```

Temporary Color Items Subroutine

If printing to the upper screen, zero MASKT and transfer the border color to ATTRT, else transfer ATTRP to ATTRT. Also, if printing to the upper screen, transfer the perm attributes to the temp attributes via TEMPS-1. If printing to the lower half of the screen, put the temporary attributed into the permanent attributes.

TEMPS (DO_ATTS)

```
M0888   XOR A               Zero A.
        LD HL,(ATTRP)       Get the permanent attributes and mask.
        BIT LHS,(IY+OTVFLAG) Jump if not printing to lower half.
        JR Z,TEMPS-1
        LD H,A              Load attribute mask (take attrib from ATTRP).
        LD L,(IY+OBORDCR)   Get the border color.
```

TEMPS-1

```
M0896   LD (ATTRT),HL          Store into the temp attributes.
```

If zero flag=1, transfer the permanent attributes to the temporary attributes. If zero flag=0, update the temporary attributes. Enter with new attribute flags in A.

```
M0899   LD HL,PFLAG            Get the print flag address.
        JR NZ,TEMPS-2          Jump forward if dealing with lower part of screen.
        LD A,(HL)              Fetch the print attribute flags and
        RRCA                   make the perm flags the temp flags.
```

TEMPS-2

```
M08A0   XOR (HL)               Copy the even bits of A to PFLAG.
        AND $55
        XOR (HL)
        LD (HL),A
        RET
```

CLS Command

In the first instance, the whole display is cleared: the pixels are all reset and the attribute bytes are set to equal the value in ATTRP, then the lower part of the display is reset.

CLS (K_CLS)

```
M08A6   CALL CL-ALL            Clear the entire screen.
```

CLS-LOWER (CLLHS)

```
M08A9   LD HL,TVFLAG
        RES CLHS,(HL)          Force no clear LHS when key pressed.
        SET LHS,(HL)           Force printing to LHS.
        CALL TEMPS             Set up the attributes. Effectively transfers the
                               permanent attributes to the temporary attributes.
        LD B,(IY+ODFSZ)        Get the size of the lower screen half.
        CALL CL-LINE           Clear the lower screen half.
        LD HL,$5AC0            Address of end of lower screen attributes+1
        LD A,(ATTRP)           Get the permanent attributes
        DEC B                  Normalize line counter for 1 base
        JR CLS-3
```

CLS-1

```
M08C2   LD C,$20               32 characters per line.
```

CLS-2

```
M08C4   DEC HL                 Go back to the line setting
        LD (HL),A              the attribute bytes.
        DEC C                  Jump until all 32 characters
        JR NZ,CLS              have been cleared.
```

CLS-3

```
M08C9   DJNZ CLS-1             Jump back until finished.
        LD (IY+ODFSZ),02       Force 2 lines for the LHS
```

CL-CHAN

```
M08CF   LD A,$FD              Open channel K (keyboard/lower screen)
        CALL CHAN-OPEN
        LD HL,(CURCHL)        Get the address of current channel and set
        LD DE,PRINT-OUT       output address.
        AND A                 reset the carry flag
```

CL-CHAN-A

```
M08DB   LD (HL),E             Update the current channel output
        INC HL                routine address,
        LD (HL),D             then the input routine address.
        INC HL
        LD DE, KEY-INPUT
        CCF                   Loop counter.
        JR C,CL-CHAN-A
        LD BC,$1721           Line 2, start of line.
        JR CL-SET             Update the printing parameters.
```

Clear the Whole Display Area Subroutine

Called from the CLS command routine, the main execution routine and the automatic listing routine.

CL-ALL (CLS)

```
M08EA   LD HL,$0000           Zero the X and Y print
        LD (XCOORD),HL        coordinates.
        RES ALOS,(IY+OFLAGS2) Signal screen is clear.
        CALL CL-CHAN
        LD A,$FE              Open channel 'S' (main screen).
        CALL CHAN-OPEN
        CALL TEMPS            Update the attributes.
        LD B,$18              Clear 24 lines of the display.
        CALL CL-LINE
        LD HL,(CURCHL)        Get the address of the current channel
        LD DE,PRINT-OUT       output routine.
        LD (HL),E             Make the current output channel
        INC HL                the TV.
        LD (HL),D
        LD (IY+OSCRCT),01     Reset scroll counter.
        LD BC,$1821           Set the cursor pointers to the top left corner of
                              the screen.
```

CL-SET Subroutine

BC holds the line and column numbers of character areas or the C register holds the column number within the printer buffer. The appropriate address of the first character bit is then found. Returns via PO-STORE to store all the values in the required system variables.

CL-SET (STTVC)

```
M0914   LD HL,PRBUF           Point to printer buffer.
```

```
            BIT PR,(IY+OFLAGS)      Jump if printing to the printer.
            JR NZ,CL-SET-2
            LD A,B                  Get the line number into A.
            BIT LHS,(IY+OTVFLAG)    Jump if not printing to
            JR Z,CL-SET-1           the lower half of the screen.
            ADD A,(IY+ODFSZ)        Add the number of lines in the lower
            SUB $18                 screen.
```

CL-SET-1

```
M0929       PUSH BC                 Briefly save line and column numbers.
            LD B,A                  Put line number in A.
            CALL CL-ADDR            Find the display file address for the
                                    screen line in B.
            POP BC                  Restore line and column numbers.
```

CL-SET-2

```
M092F       LD A,$21                Reverse column number and transfer
            SUB C                   the char position in C in to DE.
            LD E,A
            LD D,$00                Display file address for the char in
            ADD HL,DE               C.
            JP PO-STORE             Update character pointers.
```

Scrolling Subroutine

Number of lines to scrolled is in the B register.

CL-SC-ALL (SCRL1)
Entry point to scroll one line and after "scroll?"

```
M0939       LD B,$17
```

CL-SCROLL (SCRLB)
Entry point to scroll "B" lines.

```
M093B       CALL CL-ADDR            Get the line's address in Display File 1.
            LD C,$08                8 scan ('pixel') lines per character line.
```

CL-SCR-1
Main scrolling loop. B register holds the number of the top line to be scrolled, the HL register pair the starting address of this line in the display file and C register the pixel line counter.

```
M0940       PUSH BC                 Save scan line counter.
            PUSH HL                 Save the display file "FROM" address.
            LD A,B                  Save character lines and
            AND $07                 test if this is the last scan line in the character line.
            LD A,B                  Save the character line counter.
            JR NZ,CL-SCR-3          Jump if the not the last scan line in the character
                                    line.
```

CL-SCR-2

The pixel lines of the top lines of the thirds of the display have to be moved across the 2K boundaries. (Each third = 2K.)

```
M0948   EX DE,HL          Make DE point to the previous
        LD HL,$F8E0       scan line.
        ADD HL,DE
        EX DE,HL          HL now points to the line that number  was in B.
                          DE points to the line before it.
        LD BC,$0020       32 characters.
        DEC A             Decrement the character line counter.
        LDIR              Move the 32 bytes.
```

CL-SCR-3

The pixel lines within the 'thirds' can now be scrolled. The A register holds, on the first pass, +01 - +07, +09 - +0F or +11 - +17.

```
M0954   EX DE,HL          Make DE point to required destination.
        LD HL,$FFE0
        ADD HL,DE         Move HL to point back one character line.
        EX DE,HL
        LD B,A            Save the line number in B.
        AND $07           Find out how many characters are
        RRCA              remaining in the 'third'.
        RRCA
        RRCA
        LD C,A            Pass character total to C.
        LD A,B            Fetch line number.
        LD B,$00          BC holds 'character total' and a pixel line
        LDIR              from each of the characters is scrolled.
        LD B,$07          Prepare to increment the address to jump
        ADD HL,BC         a 'third' boundary.
        AND $F8           Jump back if there are any 'thirds' left.
        JR NZ,CL-SCR-2
        POP HL            Fetch original address.
        INC H             Address the next pixel line.
        POP BC            Fetch counters.
        DEC C             Decrease pixel line counter.
        JR NZ,CL-SCR-1    Jump back unless eight lines have been moved.
```

Scroll the attribute bytes, too.

```
        CALL CL-ATTR      Form the address for the attributes.
        LD HL,$FFE0       Subtract 32 from DE to get to location
        ADD HL,DE         for attribute bytes.
        EX DE,HL
        LDIR              Scroll attribute bytes.
        LD B,$01          Clear the bottom line of the display.
```

Clear Lines Subroutine

Clear lower B lines of the display.

CL-LINE (CLS_B)

```
M097F   PUSH BC             Save the line number to clear.
        CALL CL-ADDR        HL=location in Display File 1 where the line in B
                            resides.
        LD C,$08            8 scan lines per character line.
```

CL-LINE-1

```
M0985   PUSH BC             Save the scan line counter.
        PUSH HL             Save the scan line address.
        LD A,B              Get the character line number.
```

CL-LINE-2

```
M0988   AND $07             Number of character lines to clear in this
        RRCA                block.
        RRCA
        RRCA
        LD C,A              C holds the result; 0 for a third.
        LD A,B              Fetch the line number.
        LD B,$00            Make BC pair hold one less than
        DEC C               number of characters.
        LD D,H              Copy the first location to clear
        LD E,L              to DE.
        LD (HL),$00         Clear the pixel-byte of the first char.
        INC DE              Make DE point to the second character.
        LDIR                Clear the scan line.
        LD DE,$0701         Point to the start of the next
        ADD HL,DE           scan line in the next block.
        DEC A               Decrement the character line counter.
        AND $F8             Discard any extra lines and
        LD B,A              pass the 'third' count to B.
        JR NZ,CL-LINE-2     Jump back if there are still 'thirds' to address.
        POP HL              Update the address for each pixel line.
        INC H
        POP BC              Fetch scan line counter.
        DEC C               Decrease scan line counter.
        JR NZ,CL-LINE-1     Jump back unless finished.
```

Scroll the attribute bytes, too.

```
        CALL CL-ATTR        Get the start address of the attributes.
        LD H,D              Copy the attribute start address to HL.
        LD L,E
        INC DE              Point to second attribute byte.
        LD A,(ATTRP)        Get the permanent attributes.
        BIT LHS,(IY+OTVFLAG) Jump if doing the upper half screen.
        JR Z,CL-LINE-3
```

```
              LD A,(BORDCR)       Use border color is used for the lower
                                  screen half.
```

CL-LINE-3

```
M09BB         LD (HL),A           Set attribute byte.
              DEC BC              Adjust counter.
              LDIR                Copy the value all the attributes.
              POP BC              Restore line number.
              LD C,$21            Set the column number to left
              RET                 column and return.
```

CL-ATTR Subroutine

This subroutine has two functions:

- For a given display area address, the appropriate attribute address is returned in the DE register. Note that the value on entry points to the 'ninth' line of a character.
- For a given line number, in the B register, the number of character areas in the display from the start of that line onwards is returned in the BC register pair.

CL-ATTR

```
M09C3         LD A,H              Fetch the high byte.
              RRCA                Multiply by 32.
              RRCA
              RRCA
              DEC A               Go back to the 'eight' line.
              OR $50              Address the attribute area.
              LD H,A              Restore the high byte and transfer
              EX DE,HL            address to DE.
              LD H,C              This is always zero.
              LD L,B              Get the number of character lines.
              ADD HL,HL           Multiply by 32.
              ADD HL,HL
              ADD HL,HL
              ADD HL,HL
              ADD HL,HL
              LD B,H              number of attribute bytes
              LD C,L              to change
              RET
```

CL-ADDR Subroutine

Form the base address in Display File 1 of the line number in B. On exit, HL points to the location in Display File 1 where the line resides. Perform the calculation $4000+ $100*INT(line#/2)+(32*(line# AND 7)), where line # = 24-B.

CL-ADDR

```
M09D6         LD A,$18            Reverse the line number.
              SUB B
              LD D,A              Save result to D.
              RRCA                (A mod 8) *32.
              RRCA
```

```
         RRCA
         AND $E0
         LD L,A              Low byte in A.
         LD A,D              True line number fetched.
         AND $18
         OR $40
         LD H,A
         RET
```

Scroll Wait Subroutine

Wait for a key, then clear the lower half screen.

```
SCR-WAIT
M09E7    PUSH AF             Save registers.
         PUSH BC
         PUSH DE
         LD BC,$9C40         40,000

SCR-WAIT-LOOP
M09ED    DEC BC              Count down to 0
         LD A,C
         OR B
         JR NZ,SCR-WAIT-LOOP

SCR-KEY-PRESS
M09F2    XOR A               Clear A and carry flag.
         IN A,($FE)          Loop until no keys are pressed
         AND $1F
         CP $1F
         JR Z, SCR-KEY-PRESS
         CALL CLS-LOWER      Clear the lower half screen.
         POP DE              Restore registers.
         POP BC
         POP AF
         RET
```

COPY Command

Dump the screen to the printer.

```
COPY (K_DUMP)
M0A02    DI                  Disable interrupts.
         LD B,$B0            22 character lines * 8 scan lines per character.
         LD HL,DF1           HL points to display file #1.

COPY-1
M0A08    PUSH HL             Save the base address and number of line.
         PUSH BC
         CALL COPY-LINE      Output (HL) to the printer.
         POP BC              Fetch line number and base address.
```

```
            POP HL
            INC H           Point to the next scan line
            LD A,H          Jump if we have not finished
            AND $07         8 scan lines.
            JR NZ, COPY-2
```

Update the base address for each new line of characters.

```
            LD A,L          Bump to the next character
            ADD A,$20       line.
            LD L,A          Carry flag will be reset when 'within thirds' of
                            display.
            CCF             Change carry flag.
            SBC A,A         A=$F8 if in the same DF segment or A=$00 if
                            going to the next segment.
            AND $F8         -8
            ADD A,H         Subtract 8 to put H back to where
            LD H,A          it should be.
```

COPY-2

```
M0A1F       DJNZ COPY-1     Jump back until all scan lines have printed.
            JR COPY-END     Jump forward to end routine.
```

COPY-BUFF Subroutine

Called when the printer buffer contents are passed to the printer.

COPY-BUFF (DUMPPTR)

```
M0A23       DI              Disable interrupts.
            LD HL,PRBUF     Point to the printer buffer.
            LD B,$08        Eight scan lines.
```

COPY-3

```
M0A29       PUSH BC         Save scan line counter.
            CALL COPY-LINE  Output the buffer to the printer.
            POP BC          Restore the scan line counter.
            DJNZ COPY-3     Back around for all of the scan lines.
```

COPY-END

```
M0A30       LD A,$04        Stop printer motor off.
            OUT ($FB),A
            EI              Interrupts back on.
```

Clear Printer Buffer Subroutine

CLEAR-PRB (CLEAR-PRB)

```
M0A35       LD HL,PRBUF     Point to the printer buffer.
            LD (IY+OPRCC),L The printer cursor now points to the first
                            location of the printer buffer.
            XOR A           Clear A.
```

```
         LD B,A                    Set B to 256 bytes.

PRB-BYTES
M0A3D    LD (HL),A                 The 256 bytes of the buffer
         INC HL                    are cleared.
         DJNZ PRB-BYTES
         RES PRLEFT,(IY+OFLAGS2)   Signal buffer is empty.
         LD C,$21                  Set printer position and
         JP CL-SET                 return.
```

COPY-LINE Subroutine

Output the bytes at (HL) to the printer as one scan line of character data. HL points to a 32 byte buffer; B contains scan line number.

```
COPY-LINE (PRSCAN)
M0A4A    LD A,B                    Copy the scan line number.
         CP $03                    A will hold 0 until last two scan lines are
         SBC A,A                   being handled.
         AND $02
         OUT ($FB),A               Slow the printer motor for the last two
         LD D,A                    scan lines. D will hold either 0 or 2.

COPY-L-1
M0A53    CALL BREAK-KEY            Jump if break key not pressed.
         JR C,COPY-L-2
         LD A,$04                  Turn printer motor off.
         OUT ($FB),A
         EI                        Enable interrupts.
         CALL CLEAR-PRB            Clear the printer buffer and exit via
         RST $08                   Error: Break - Cont repeats.
         DEFB $0C

COPY-L-2
M0A62    IN A,($FB)                Read printer port.
         ADD A,A                   Return if printer "NOT CONFIGURED".
         RET M
         JR NC,COPY-L-1            Jump if not "START OF PAPER".
         LD C,$20                  32 characters.

COPY-L-3
M0A6A    LD E,(HL)                 Get the first byte to output.
         INC HL                    Point to the next.
         LD B,$08                  8 bits of data.

COPY-L-4
M0A6E    RL D                      Rotate D left.
         RL E                      Move each bit of E in to carry.
         RR D                      Move D right, pick up carry.
```

COPY-L-5

```
M0A74   IN A,($FB)          Wait until printer is ready
        RRA                 for the next pixel.
        JR NC,COPY-L-5
        LD A,D              Output the pixel to the
        OUT ($FB),A         printer.
        DJNZ COPY-L-4       Loop until the bit counter is zero.
        DEC C               Loop until the byte counter is zero.
        JR NZ,COPY-L-3
        RET
```

Editor Routines

The editor is called by the main execution routine, so that the user can enter a BASIC line into the system, and the INPUT command routine.

EDITOR (EDIT_K)

```
M0A82   LD HL,(ERRSP)       Push the error stack pointer onto
        PUSH HL             the stack.
```

ED-AGAIN

```
M0A86   LD HL,$0BE5         Address of ED-ERROR.
        PUSH HL             Any event that leads to the error handling
        LD (ERRSP),SP       will go to ED-ERROR.
```

ED-LOOP

Loop to handle each keystroke.

```
M0A8E   CALL WAIT-KEY       Get a character from the current channel.
        PUSH AF             Save the character.
        LD D,$00            Get duration of keyboard click.
        LD E, (IY+OPIP)
        LD HL,$00C8         And pitch.
        CALL BEEPER         Make the sound.
        POP AF              Get the character.
        LD HL,$0A8E         Push ED-LOOP address to stack.
        PUSH HL
```

Handle some exceptions.

```
        CP $0C              Jump if not DELETE.
        JR NZ, ED-LOOP-1
        BIT DELREP,(IY+OFLAGS2)  Jump if DELETE key is
        JR NZ, ED-LOOP-1    held down.
        BIT LMODE2,(IY+OFLAGS)   Jump if not "L" mode.
        JR Z, ADD-CHAR
```

ED-LOOP-1
Analyze the code.

```
M0AB2   CP $18                  Jump if potentially a letter or token.
        JR NC, ADD-CHAR
        CP $07                  Accept a comma.
        JR C, ADD-CHAR
        CP $10                  Jump if an editing key.
        JR C, ED-KEYS
        LD BC,$0002             INK to TAB considered.
        LD D,A                  Copy code to D.
        CP $16                  Jump forward with INK and PAPER.
        JR C, ED-CONTR
        INC BC                  Handle AT, TAB.
        BIT LINPLN,(IY+OFLAGX)  Jump forward unless dealing with
        JP Z, ED-IGNORE         INPUT LINE.
        CALL WAIT-KEY           Get second code and put it in E.
        LD E,A
```

ED-CONTR
INK, PAPER, FLASH, BRIGHT, INVERT, OVER

```
M0AD2   CALL WAIT-KEY           Get another code.
        PUSH DE                 Save the previous codes.
        LD HL,(KCUR)            Fetch KCUR.
        RES 0,(IY+OMODE)        Signal K mode.
        CALL MAKE-ROOM          Make two or three spaces.
        POP BC                  Restore the previous codes.
        INC HL                  Point to the first location.
        LD (HL),B               Enter first code.
        INC HL                  Then the second code, which will be
        LD (HL),C               overwritten if INK or PAPER.
        JR ADD-CH-1
```

ADDCHAR Subroutine

Add the code to the current EDIT or INPUT line.

ADD-CHAR (INSA)

```
M0AE7   RES 0,(IY+OMODE)        Force K mode.
        LD HL,(KCUR)            Fetch cursor position.
        CALL ONE-SPACE          Make a single space.
```

ADD-CH-1

```
M0AF1   LD (DE),A               Enter the code into the space and
        INC DE                  signal that the cursor goes to location
        LD (KCUR),D             after.
        RET                     Return to ED-LOOP.
```

ED-KEYS
EDIT, CSR_LT, CSR_RT, CSR_DN, CSR_UP, DELETE, CR, SYM-SHFT, GRAPHICS

```
M0AF8   LD E,A              Move code to DE pair.
        LD D,$00
        LD HL,M0B06         Base address of editing key table.
        ADD HL,DE           Get entry and fetch in to E.
        LD E,(HL)
        ADD HL,DE           Save address of edit key handler
        PUSH HL             to the stack.
        LD HL,(KCUR)        Set HL pair and
        RET                 jump to the edit key handler.
```

EDITING KEYS TABLE

```
M0B06   DEFB EDIT $09
        DEFB cursor left $66
        DEFB cursor right $6A
        DEFB cursor down $50
        DEFB cursor up $B5
        DEFB DELETE $70
        DEFB ENTER $7E
        DEFB SYMBOL SHIFT $CF
        DEFB GRAPHICS $D4
```

Edit Key Subroutine

When in editing mode, pressing the EDIT key will bring the current BASIC line to the editor. In INPUT mode, the EDIT key clears the current entry.

ED-EDIT (EDITCMD)

```
M0B0F   LD HL,(EPPC)            Get the current line number.
        BIT INPLN,(IY+OFLAGX)   Jump forward if in 'INPUT mode.'
        JP NZ,CLEAR-SP
        CALL LINE-ADDR          Find the address of the start of the current
        CALL LINE-NO            line and its number.
        LD A,D                  If the line number returned is zero
        OR E                    then clear the editing area.
        JP Z, CLEAR-SP
        PUSH HL                 Save the address of the line.
        INC HL                  Move on to collect the length of the line.
        LD C,(HL)
        INC HL
        LD B,(HL)
        LD HL,$000A             Add $0A to the length and test that there
        ADD HL,BC               is sufficient room for a copy of the line.
        LD B,H
        LD C,L
        CALL TEST-ROOM
        CALL CLEAR-SP           Clear the editing area.
        LD HL,(CURCHL)          Fetch the current channel address and
```

```
        EX (SP),HL          exchange it for the address of the line
        PUSH HL             Save it temporarily.
        LD A,$FF            Open channel 'R' so line will be copied
        CALL CHAN-OPEN      to the editing area.
        POP HL              Get the address of the line.
        DEC HL              Go to before the line.
        DEC (IY+OEPPC)      Decrement current line number to avoid
        CALL LPO            printing the cursor, print the line.
        INC (IY+OEPPC)      Increment current line number.
        LD HL,(ELINE)       Fetch start of the line in the editing area
        INC HL              and step past the line number and the
        INC HL              length to find the address for KCUR.
        INC HL
        INC HL
        LD (KCUR),HL
        POP HL              Fetch the former channel address and
        CALL CHAN-FLAG      set the appropriate flags before returning
        RET                 to ED-LOOP.
```

Cursor Down Editing Subroutine

```
ED-DOWN (CSR_DNCMD)
M0B59   BIT INPLN,(IY+OFLAGX)   Jump if in 'INPUT mode.'
        JR NZ,ED-STOP
        LD HL,EPPC              Find next line number and
        CALL LN-FETCH           produce automatic listing.
        JR ED-LIST

ED-STOP
M0B67 LD (IY+OERRNR),$10        'STOP in INPUT' report.
        JR ED-ENTER
```

Cursor Left Editing Subroutine

```
ED-LEFT (CSR_LTCMD)
M0B6D   CALL ED-EDGE            Move cursor.
        JR ED-CUR               Jump forward.
```

Cursor Right Editing Subroutine

```
ED-RIGHT (CSR_RTCMD)
M0B72   LD A,(HL)               Is the current character a carriage return?
        CP $0D
        RET Z                   Yes, return.
        INC HL                  Otherwise, put cursor after character.

ED-CUR
M0B77   LD (KCUR),HL            Update KCUR
        RET
```

Delete Editing Subroutine (DELSYM)

HL contains the address of the item to be deleted.

ED-DELETE (DELETECMD)

```
M0B7B   CALL ED-EDGE        Move cursor left.
        LD BC,$0001         Reclaim the current character.
        JP RECLAIM-2
```

ED-IGNORE Subroutine

ED-IGNORE

```
M0B84   CALL WAIT-KEY       Next two code from key-input
        CALL WAIT-KEY       are ignored.
```

Enter Editing Subroutine

ED-ENTER (CRCMD)

```
M0B8A   POP HL              Discard addresses of ED-LOOP and
        POP HL              ED-ERROR.
```

ED-END

```
M0B8C   POP HL              Restore old value of ERR-SP.
        LD (ERRSP),HL
        BIT 7,(IY+OERRNR)   Return if no errors.
        RET NZ
        LD SP,HL            Otherwise, indirect jump to error
        RET                 routine.
```

ED-EDGE Subroutine

The address of the cursor is in the HL register pair and will be decremented unless the cursor is already at the start of the line. Care is taken not to put the cursor between control characters and parameters.

ED-EDGE

```
M0B97   SCF                 DE with either hold E-LINE (for editing)
        CALL SET-DE         or WORKSP (for 'INPUTing').
        SBC HL,DE           Carry flag will be set if cursor is already
        ADD HL,DE           at start of the line.
        INC HL              Correct for subtraction.
        POP BC              Drop the return address.
        RET C               Return via ED-LOOP if carry flag is set.
        PUSH BC             Restore the current address.
        LD B,H              Move the current address of the cursor
        LD C,L              to BC.
```

ED-EDGE-1
Loop to check that control characters are not split from their parameters.

```
M0BA4   LD H,D              HL will point to the character in the line
        LD L,E              after that addressed by DE.
        INC HL
        LD A,(DE)           Fetch character code.
        AND $F0             Jump forward if the code does
        CP $10              not represent INK to TAB.
        JR NZ, ED-EDGE-2
        INC HL              Allow for one parameter.
        LD A,(DE)           Fetch the code again.
        SUB $17             Carry is reset for TAB.
        ADC A,$00           Differentiate between AT and TAB.
        JR NZ, ED-EDGE-2    Jump forward unless AT or TAB.
        INC HL              AT and TAB have two params, so do that.
```

ED-EDGE-2

```
M0BB7   AND A               Prepare for true subtraction.
        SBC HL,BC           Carry flag will be reset when 'updated
        ADD HL,BC           pointer' reaches KCUR.
        EX DE,HL            For the next loop use 'updated pointer'
        JR C, ED-EDGE-1     but if exiting, use 'present pointer'
        RET                 for KCUR.
```

Cursor Up Editing Subroutine

ED-UP (CSR_UPCMD)

```
M0BBF   BIT INPLN,(IY+OFLAGX)   Return if in 'INPUT mode.'
        RET NZ
        LD HL,(EPPC)            Get listing current line number
        CALL LINE-ADDR          and its address.
        EX DE,HL                HL now points to previous line.
        CALL LINE-NO            DE=line number
        LD HL,EPPC+1
        CALL LN-STORE           Store the line number.
```

ED-LIST

```
M0BD4   CALL AUTO-LIST      List the line and re-open channel 'K'
        LD A,$00            (lower screen) before returning to
        JP CHAN-OPEN        ED-LOOP.
```

ED-SYMBOL Subroutine

Handle SYMBOL and GRAPHIC codes.

ED-SYMBOL

```
M0BDC   BIT LINPLN,(IY+OFLAGX)  Test if in INPUT LINE mode.
        JR Z, ED-ENTER
```

ED-GRAPH (ED-GRAPHIC)
M0BE2 JP ADD-CHAR

ED-ERROR Subroutine

Come here if there's been an error.

ED-ERROR

```
M0BE5   BIT RETPOS,(IY+OFLAGS2)   Jump back if using channel other
        JR Z,ED-END               than 'K'.
        LD (IY+OERRNR),$FF        Cancel error number and
        LD D,$00                  make a noise before
        LD E,(IY+ORASP)           going around the editor
        LD HL,$1A90               again.
        CALL BEEPER
        JP ED-AGAIN
```

CLEAR-SP Routine

Clear editing area or workspace as directed.

CLEAR-SP (DEL_C)

```
M0BFD   PUSH HL                   Save pointer to the space.
        CALL SET-HL               DE will point to the first character
        DEC HL                    and HL to the last.
        CALL RECLAIM-1            Deletes the bytes HL and DE.
        LD (KCUR),HL              System variables KCUR and MODE
        LD (IY+OMODE),00          are initialized before
        POP HL                    restoring the pointer and
        RET                       returning.
```

Keyboard Input Subroutine

Returns the code of the last key pressed. CAPS LOCK, mode changing and color control parameters are handled here, too.

KEY-INPUT (IN-K)

```
M0C0E   BIT ECHREQ,(IY+OTVFLAG)   Call if echo to screen requested.
        CALL NZ,ED-COPY           INPUT-line to the screen if mode has changed.
        AND A                     Return with both carry and zero flags reset
        BIT KEYHIT,(IY+OFLAGS)    if no new key has been pressed.
        RET Z
        LD A,(LASTK)              Otherwise, get the code and signal that it
        RES KEYHIT,(IY+OFLAGS)    has been taken.
        PUSH AF                   Save code temporarily.
        BIT CLHS,(IY+OTVFLAG)     Clear lower part of the display if necessary.
        CALL NZ,CLS-LOWER
        POP AF                    Fetch the code.
        CP $20                    Accept all characters and tokens.
        JR NC,KEY-DONE
        CP $10                    Jump forward with most control
```

```
              JR NC,KEY-CONTR       character codes.
              CP $06                Jump forward with the 'mode.'
              JR NC,KEY-M&CL
              LD B,A                Deal with FLASH, BRIGHT and INVERSE.
              AND $01               Keep only bit 0.
              LD C,A                C holds 0 (OFF) or 1 (ON).
              LD A,B                Fetch the code.
              RRA                   Rotate to lose bit 0.
              ADD A,$12             Increase by $12 to mark as FLASH,
              JR KEY-DATA           BRIGHT, and INVERSE.
```

KEY-M&CL

```
M0C41         JR NZ, KEY-MODE       Jump forward for 'mode' codes.
              LD HL,FLAGS2
              LD A,$08              Flip bit 3 (CAPS LOCK) of FLAGS2.
              XOR (HL)
              LD (HL),A
              JR KEY-FLAG
```

KEY-MODE

```
M0C4C         CP $0E                Check lower limit.
              RET C
              SUB $0D               Reduce the range.
              LD HL,MODE
              CP (HL)               Has mode been changed?
              LD (HL),A             Enter new mode code.
              JR NZ, KEY-FLAG       Jump if it has changed.
              LD (HL),$00           Otherwise, make it L mode.
```

KEY-FLAG

```
M0C5A         SET ECHREQ,(IY+OTVFLAG) Signal mode might have changed.
              CP A                  Reset carry flag.
              RET
```

KEY-CONTR

Control codes (except FLASH, BRIGHT, INVERSE) adjusted.

```
M0C60         LD B,A                Save the code.
              AND $07               Make the C register hold
              LD C,A                the parameter.
              LD A,$10              A now holds INK code.
              BIT 3,B               But if the code was an unshifted
              JR NZ,KEY-DATA        code then make A hold PAPER.
              INC A
```

KEY-DATA
Parameter is saved in KDATA and the channel address changed from KEY-INPUT to KEY-NEXT.

```
M0C6B   LD (IY+(-$2D)),C        Save the parameter.
        LD DE,$0C73             Address of KEY-NEXT.
        JR KEY-CHAN
```

KEY-NEXT
On the first pass entering at KEY-INPUT, A register is returned holding a control code. On the next pass, entering at KEY-NEXT, the parameter is returned.

```
M0C73   LD A,(KDATA)            Fetch the parameter.
        LD DE,$0C0E             Address of KEY-INPUT.
```

KEY-CHAN
Set the input address in the first channel area.

```
M0C79   LD HL,(CHANS)           Fetch channel address.
        INC HL
        INC HL
        LD (HL),E               Set the input address.
        INC HL
        LD (HL),D
```

KEY-DONE
Exit with the required code in A.

```
M0C81   SCF                     Show a code has been found
        RET                     and return.
```

Lower Screen Copying Subroutine

Called when the line in the editing area or the INPUT area is to be printed in the lower part of the screen.

ED-COPY

```
M0C83   CALL TEMPS              Use the permanent colors.
        RES ECHREQ,(IY+OTVFLAG) Signal that the mode is unchanged;
        RES CLHS,(IY+OTVFLAG)   lower screen does not need clearing.
        LD HL,(SPOSNLCOL)       Save the print line and column.
        PUSH HL
        LD HL,(ERRSP)           Save the error stack pointer
        PUSH HL
        LD HL, $0CCD            Address of local error handler.
        PUSH HL                 Push to stack to make ED-FULL entry point
        LD (ERRSP),SP           for errors.
        LD HL,(ECHOE)           Push the value of ECHO-E to the stack.
        PUSH HL                 Make HL point to start of space and DE to
        SCF                     end.
        CALL SET-DE             Returns with DE pointing to the WORKSP
```

```
              EX DE,HL                  or ELINE if expecting user input.
              CALL OUT-LINE2            Print the line.
              EX DE,HL                  Exchange pointers and print cursor.
              CALL OUT-CURS
              LD HL,(SPOSNLCOL)         Fetch current value of SPOSNLCOL and
              EX (SP),HL                exchange with ECHOE.
              EX DE,HL                  Pass ECHOE to DE.
              CALL TEMPS                Fetch the permanent colors.
```

ED-BLANK
Remainder of any line that has been started is completed with spaces printed with the 'permanent' PAPER color.

```
M0CB6   LD A,(SPOSNLLIN)          Fetch current line number and
        SUB D                     subtract the old line number.
        JR C, ED-C-DONE           Jump forward if no 'blanking' is required.
        JR NZ, ED-SPACES          Jump forward if not on the same line.
        LD A,E                    Get old line number and subtract the new
        SUB (IY+OSPOSNLCOL)       column number.
        JR NC, ED-C-DONE          Jump if no spaces are required.
```

ED-SPACES

```
M0CC4   LD A,$20                  A space code.
        PUSH DE                   Save the old values.
        CALL PRINT-OUT            Print it.
        POP DE                    Restore the old values.
        JR ED-BLANK               Back again.
```

ED-FULL
Deal with any errors.

```
M0CCD   LD D,$00                  Make a noise.
        LD E,(IY+ORASP)
        LD HL,$1A90
        CALL BEEPER
        LD (IY+OERRNR),$FF        Cancel the error number.
        LD DE,(SPOSNLCOL)         Fetch current value and jump
        JR ED-C-END               forward.
```

ED-C-DONE
Normal exit upon completion of copying the INPUT line.

```
M0CE2   POP DE                    New position value.
        POP HL                    Error address.
```

ED-C-END

```
M0CE4   POP HL                    Restore old value of ERRSP.
        LD (ERRSP),HL
        POP BC                    Fetch old value of SPOSNL.
        PUSH DE                   Save new position values.
        CALL CL-SET               Set system variables.
```

```
        POP HL                  Old value of SPOSNL
        LD (ECHOE),HL           goes into ECHOE.
        LD (IY+(OXPTR+1)),00    X_PTR is cleared.
        RET
```

SET-HL and SET-DE Subroutines

These subroutines return with HL pointing to the first location and DE the 'last' location of either the editing area or the work space.

SET-HL

```
M0CF6   LD HL,(WORKSP)          Point to the last location of the editing
        DEC HL                  buffer.
        AND A                   Clear the carry flag.
```

SET-DE

```
M0CFB   LD DE,(ELINE)           Point to start of the edit buffer and
        BIT INPLN,(IY+OFLAGX)   return if in 'editing mode'.
        RET Z
        LD DE,(WORKSP)          Otherwise, change DE.
        RET C                   Return if intended.
        LD HL,(STKBOT)          Fetch STKBOT and then
        RET                     return.
```

REMOVE-FP Subroutine

Removes the hidden floating-point characters (slugs) in a BASIC line.

REMOVE-FP (DESLUG)

```
M0D0D   LD A,(HL)               Get a character.
        CP $0E                  Is it a number marker (slug)?
        LD BC,$0006             If yes, it occupies 6 locations.
        CALL Z,RECLAIM-2        Delete the floating-point number,
        LD A,(HL)               Get the character again.
        INC HL                  Update the pointer.
        CP $0D                  End of line?
        JR NZ, REMOVE-FP        Go back around until the end of line.
        RET
```

Executive Routines

This section includes the initialization procedure and the main execution loop for the BASIC interpreter.

In the TS 2068, BASIC lines entered by the user are checked for correct syntax before they are saved in the program (if they start with a line number) or before they are executed (for immediate commands).

Initialization Routine

Main entry to this routine is at START/NEW (M0D31). When entered from START (0000), as when the computer is turned on, A=0 and DE=FFFF. Main entry point is also reached after executing the NEW command routine.

NEW (K_NEW)

```
M0D1D   DI              Disable interrupts.
        LD A,$FF        NEW flag.
        LD DE,(RAMTOP)  Preserve existing value of RAMTOP.
        EXX             Load alternate registers
        LD BC,(PRAMT)   with the following system
        LD DE,(RASP)    variables, all of which will be
        LD HL,(UDG)     preserved.
        EXX
```

INIT

System initialization. Enter with A=0 and DE=FFFF and interrupts disabled.

START-NEW

```
M0D31   LD B,A          Save flag for later.
        LD A,$07        Make the border white.
        OUT ($FE),A
        LD A,$3F        Initialize I
        LD I,A          and set A to bottom of RAM (3FFF).
        NOP             Wait 24 T states.
        NOP
        NOP
        NOP
        NOP
        NOP
```

RAMCHECK
Check the memory.

```
M0D40   LD H,D          Transfer value in DE (START=FFFF,
        LD L,E          NEW=RAMTOP).
```

RAM-FILL

```
M0D42   LD (HL),$02     Enter $02 into every location above
        DEC HL          $3FFF.
        CP H
        JR NZ,RAM-FILL
```

RAM-READ

```
M0D48   AND A           Reset carry for subtraction.
        SBC HL,DE       Carry flag will reset when the
        ADD HL,DE       top is reached.
        INC HL          Update pointer
        JR NC, RAM-DONE Jump when at the top.
        DEC (HL)        Decrement memory location
        JR Z, RAM-DONE  If 0, RAM is faulty. Set RAMTOP and
        DEC (HL)        decrement memory again.
        JR Z, RAM-READ  Loop until it fails.
```

RAM-DONE

```
M0D55   DEC HL          HL points to the last valid memory address.
```

Restore the 'preserved' systems variables. Meaningless when coming from START.

```
        EXX             Swap to the alternate registers
        LD (PRAMT),BC   Restore the system variables.
        LD (RASP),DE
        LD (UDG),HL
        EXX
        INC B           Test START/NEW flag,
        JR Z, RAM-SET   jump forward if NEW.
```

Overwrite the system variables when coming from START and initialize the user-defined graphics area.

```
        LD (PRAMT),HL   Set P_RAMTOP to discovered value.
        LD DE,$3EAF     Initialize the UDG area.
        LD BC,$00A8     Transfer the character data at
        EX DE,HL        $3EAF - $3F57 to the top of RAM.
        LDDR
        EX DE,HL        Set the pointers back.
        INC HL          Point to the first byte.
        LD (UDG),HL     Set UDG to point to transferred data.
        DEC HL          Joint just below UDG.
        LD BC,$0040     Set the system variables.
        LD (RASP),BC    Set RASP=40, PIP=0.
```

RAM-SET
Remainder of this routine is common to START and NEW.

```
M0D7F   LD (RAMTOP),HL          Set BASIC's RAMTOP to just below the UDG area.
        LD HL,$3C00             Initialize system variable CHARS.
        LD (CHARS),HL
```

Set up the machine stack.

```
        LD HL,$6200             Set MSTBOT, the address of
        LD (MSTBOT),HL          location above machine stack.
        DEC HL                  Point to machine stack,
        LD (HL),$3E             initialize to $3E.
        DEC HL
        LD SP,HL
        DEC HL                  Step down two locations to
        DEC HL                  find the correct value for
        LD (ERRSP),HL           ERRSP.
        IM 1                    TS uses interrupt mode 1.
        NOP                     Wait.
        LD IY,ERR_NR            Initialize IY to address of ERR_NR.
        LD HL,$6840             Set CHANS to point to
        LD (CHANS),HL           channels buffer.
        LD DE,$11AA             Transfer initial channel data
        LD BC,$0015             to the RAM buffer.
        EX DE,HL
        LDIR
        EX DE,HL
        LD A,$38
        LD (ATTRP),A            Set ATTR_P.
        LD (ATTRT),A            Set ATTR_T.
        LD (BORDCR),A           Set BORDCR (border color).
        LD HL,$0523
        LD (REPDEL),HL          Set REPDEL.
        DEC (IY+(-$3A))         Set KSTATE (KS_A1 ) to FF.
        DEC (IY+(-$36))         Set KS_A2 to FF.
        LD HL,$11C1             Move the initial stream data
        LD DE,STRMS             from its table in ROM to the
        LD BC,$000E             streams area in RAM.
        LDIR
        XOR A                   1. enable dock bank selection
        OUT (DECR),A            2. enable 17mS interrupt
                                3. set normal video mode
                                4. set attrib to black on white
        SET PR,(IY+OFLAGS)      Signal 'printer in use' and
        CALL CLEAR-PRB          clear printer buffer.
        LD (IY+ODFSZ),02        Set size of the lower part of the screen and
        CALL CLS                clear the screen.
        XOR A                   Zero A.
```

```
SET TOKEN,(IY+OFLAGS)   Set token mode in FLAGS.
LD DE,$1117             Address of the banner messages.
CALL PO-MSG             Print banner messages.
SET CLHS,(IY+OTVFLAG)   Signal 'clear the lower part' in TV_FLAG.
LD HL,$0E0B
LD DE,$6000             Transfer the routine that
LD BC,$001D             moves the function dispatcher
LDIR                    code from EXROM to RAM.
CALL M6000              Call the transfer routine.
LD HL,$65CE             Initialize the bank stack pointer
LD ($65CE),HL           in the function dispatcher.
LD HL,$08E7             Jumps to $08E7 in EXROM.
CALL M6815              Return address is popped into IX and call routine
                        to jump to INIT2.
```

These instructions are transferred to 6000h in RAM by init at M0DED then called. This routine transfers the function dispatcher from the EXROM to its initial RAM location.

```
LD A,$01                Enable the EXROM bank
OUT (HSR),A             in the Horizontal Select Register (HSR).
IN A,(DECR)             Get the value of the display enhancement control
                        register.
SET 7,A                 Enable EXROM.
OUT (DECR),A            Write to the register.
LD HL,$1000             Origin in EXROM for the code.
LD DE,$6200             Transfer the function
LD BC,$0630             dispatcher code.
LDIR
RES 7,A                 Disable the EXROM.
OUT (DECR),A
XOR A                   Back to home bank.
OUT (HSR),A             Write to the HSR.
RET
```

Main Execution Loop

Main loop; controls the editing mode, executing direct commands and generating reports.

MAIN-EXEC

```
M0E28   LD (IY+ODFSZ),02    Set lower part of screen to two lines.
        CALL AUTO-LIST
```

MAIN-1 (LED18)

```
M0E2F   CALL SET-MIN        Initialize ELINE, KCUR, and WORKSP pointers.
```

MAIN-2

```
M0E32   LD A,$00            Open channel 'K' before calling EDITOR.
        CALL CHAN-OPEN
        CALL EDITOR         Call to allow user to build BASIC line.
        CALL LINE-SCAN      Scan the current line for correct syntax.
```

```
                BIT 7,(IY+OERRNR)       Jump forward if syntax is correct.
                JR NZ, MAIN-3
                BIT RETPOS,(IY+OFLAGS2) Jump forward if retype possible after
                JR Z, MAIN-4            syntax error is set.
                LD HL,(ELINE)           Point to the start of the line with the error.
                CALL REMOVE-FP          Remove the floating point from this line.
                LD (IY+OERRNR),$FF      Reset ERR_NR and jump back to MAIN-2,
                JR MAIN-2               leaving listing unchanged.
```

MAIN-3
Edit-line has passed syntax and the three types of line that are possible have to be distinguished from each other.

```
M0E55   LD HL,(ELINE)           Point to the start of the line.
        LD (CH_ADD),HL          Set CH_ADD to the start, too,
        CALL E-LINE-NO          Fetch any line number in to BC.
        LD A,B                  Is the line number valid?
        OR C
        JP NZ,MAIN-ADD          Jump if so and add the new line to the
                                existing program.
        RST $18                 Get the first character of the line
        CP $0D                  and test if it is only a carriage return.
        JR Z,MAIN-EXEC          Jump back if true.
```

Edit-line starts with a direct BASIC command, so interpret it.

```
        BIT ALOS,(IY+OFLAGS2)   Clear display unless this flag
        CALL NZ,CL-ALL          indicates it's not required.
        CALL CLS-LOWER          Clear the lower part.
        LD A,$19                Set the appropriate value for the
        SUB (IY+OSPOSNLIN)      scroll counter.
        LD (SCRCT),A
        SET INTPT,(IY+OFLAGS)   Signal 'line execution'.
        LD (IY+OERRNR),$FF      Ensure ERR_NR is correct.
        LD (IY+ONSPPC),01       Handle the first statement in the line.
        LD (IY+OERRLN),00       Set ERRLN to 0 for report.
        CALL LINE-RUN           Interpret and execute the line.
```

MAIN-4
The line has been interpreted and all the actions have been completed, return to MAIN-4 and report back to user.

```
M0E8D   HALT                    Enable maskable interrupt.
        LD A,(IY+OERRNR)        Get the error number.
        CP $FF                  Is it FF?
        JR Z, SOUND-RASP        An error, let the user know.
        BIT 7,(IY+(OERRLN+1))   Error, line number high.
        JR Z, SOUND-RASP        Let the user know.
        SET 6,(IY+(OERRLN+1))   Error line HI plus 64.
        INC A
```

```
              LD (ERRT),A              Put A in ERRT.
              LD (IY+OERRNR),$FF       Clear the error number.
              LD HL,(PPC)              Put the line number in HL.
              LD (ERRC),HL             Put the line number in ERRC.
              LD A,(SUBPPC)            Get the statement number within the line.
              LD (ERRS),A              Put that in the error statement number.
              LD HL,(ERRLN)            Line number for ON ERROR
              RES 7,H                  Make sure new line is less than 10000.
              RES 6,H
              LD (NEWPPC),HL           Put it in the the line to be jumped to.
              LD (IY+ONSPPC),01        Sub-line to jump to.
              LD HL,$0E8D              Put address of this routine in HL.
              PUSH HL                  Push HL to the stack.
              JP STMT-RET              END statement.
```

SOUND-RASP

```
M0EC8  LD A,$07                        Mixer control/IO enable port of AY-3-8910.
       OUT ($F5),A                     Select the port.
       LD A,$FF                        Set all flags to 1 (IOA output, disable all
       OUT ($F6),A                     sound and noise channels).
       RES ECHREQ,(IY+OTVFLAG)         Echo off.
       RES KEYHIT,(IY+OFLAGS)          Signal 'ready for new key'.
       BIT PRLEFT,(IY+OFLAGS2)         Empty printer buffer if it has been used.
       CALL NZ,COPY-BUFF
       LD A,(ERR_NR)                   Fetch the error number and
       INC A                           increment it.
```

MAIN-G

```
M0EE3  PUSH AF                         Save the error number.
       LD HL,$0000                     Set these system variables to $0000.
       LD (IY+OFLAGX),H                FLAGX
       LD (IY+(OXPTR+1)),H             PTR hi
       LD (DEFADD),HL                  DEFADD
       LD HL,$0001                     Ensure stream $00 points to channel K.
       LD (CH_0),HL
       CALL SET-MIN                    Clear all work areas and calculator stack.
       RES INPLN,(IY+OFLAGX)           Signal 'editing mode.'
       CALL CLS-LOWER                  Clear the lower screen.
       SET CLHS,(IY+OTVFLAG)           Signal 'lower screen will need to be cleared.'
       POP AF                          Fetch the report value.
       LD B,A                          Make a copy in B.
       CP $0A                          Jump forward with report numbers
       JR C, MAIN-5                    0 to 9.
       ADD A,$07                       Add the ASCII letter offset value.
```

MAIN-5

```
M0F0C  CALL OUT-CODE                   Print the report code,
       LD A,$20                        followed by a space.
       RST $10
```

```
              LD A,B                  Get the report value and use
              LD DE,$0F65             it to identify the required report message.
              CALL PO-MSG
              XOR A                   Print the message and follow with comma
              LD DE,$1115             and space.
              CALL PO-MSG
              LD BC,(PPC)             Fetch current line number
              CALL OUT-NUM-1          and print it.
              LD A,$3A                Follow it with a colon (:).
              RST $10
              LD C,(IY+OSUBPPC)       Get the current statement number and
              LD B,$00                put it in BC.
              CALL OUT-NUM-1          Print value in BC pair.
              CALL CLEAR-SP           Clear the editing area.
              LD A,(ERR_NR)           Get the ERR_NR again.
              INC A                   Increment it.
              JR Z, MAIN-9            If the program completed ok, there's no CONT so
                                      jump ahead.
              CP $09                  If the program halted with
              JR Z,MAIN-6             STOP or 'BREAK into program',
              CP $15                  CONT will be from next statement.
              JR NZ,MAIN-7

MAIN-6
M0F43         INC (IY+OSUBPPC)        Otherwise SUBPPC is unchanged.

MAIN-7
M0F46         LD BC,$0003             System variables OSPPC and NSPPC set
              LD DE,OSPCC             with CONT line and statement numbers.
              LD HL,NSPPC             Values used will those in PPC and SUBPPC
              BIT 7,(HL)              unless NSPPC indicates the break came
              JR Z, MAIN-8            before a jump.
              ADD HL,BC

MAIN-8
M0F54         LDDR

MAIN-9
M0F56         LD (IY+ONSPPC),$FF      NSPPC reset to indicate 'no jump.'
              RES LMODE2,(IY+OFLAGS)  'K mode' selected.
              RES ECHREQ,(IY+OTVFLAG) Echo off.
              JP MAIN-2               Jump back but no program listing appear
                                      until requested.
```

Report Message Table

Each message has the last character inverted (+$80).

ERRMSGS

```
DEFB $80          Initial marker
DEFB $4F, $CB     0, OK
DEFM              1, NEXT without FOR
DEFM              2, Variable not found
DEFM              3, Subscript wrong
DEFM              4, Out of memory
DEFM              5, Out of screen
DEFM              6, Number too big
DEFM              7, RETURN without GOSUB
DEFM              8, End of file
DEFM              9, STOP statement
DEFM              A, Invalid argument
DEFM              B, Integer out of range
DEFM              C, Nonsense in BASIC
DEFM              D, BREAK - CONT repeats
DEFM              E, Out of DATA
DEFM              F, Invalid file name
DEFM              G, No room for line
DEFM              H, STOP in INPUT
DEFM              I, FOR without NEXT
DEFM              J, Invalid I/O device
DEFM              K, Invalid color
DEFM              L, BREAK into program
DEFM              M, RAMTOP no good
DEFM              N, Statement lost
DEFM              O, Invalid stream
DEFM              P, FN without DEF
DEFM              Q, Parameter error
DEFM              R, Tape loading error
DEFM              S, Missing LROS
```

And start-up messages.

```
DEFM              ',' & (' '+$80)
DEFM              $7F &" 1982 Sinclair Research Ltd"& $0D & $0D
DEFM              $7F &" 1983 Timex Computer Corp"
```

REPORT-G (UNKEDIT)
No room for line.

```
M1150   LD A,$10
        LD BC,$0000
        JP MAIN-G
```

MAIN-ADD Subroutine

Add a new BASIC line to the existing BASIC program in the program area. If a line has both an old and a new version then the old one is reclaimed. A new line that consists of only a line number does not go into the program area.

MAIN-ADD

```
M1158   LD (EPPC),BC        Make new line number the 'current line'.
        LD HL,(CH_ADD)      Fetch CH_ADD and save the address to
        EX DE,HL            DE.
        LD HL,$1150         Push address of REPORT-G on to the stack.
        PUSH HL             ERRSP will now point to REPORT-G.
        LD HL,(WORKSP)      Fetch WORKSP.
        SCF                 Find the length of the line from the line
        SBC HL,DE           number to the carriage return.
        PUSH HL             Save the length.
        LD H,B              Move the line number to the HL pair.
        LD L,C
        CALL LINE-ADDR      Is there an existing line with this number?
        JR NZ, MAIN-ADD1    Jump if not.
        CALL NEXT-ONE       Find the length of the 'old' line
        CALL RECLAIM-2      and reclaim it.
```

MAIN-ADD1

```
M1178   POP BC              Fetch the length of the new line
        LD A,C              and jump forward if it is only a line
        DEC A               number and carriage return.
        OR B
        JR Z, MAIN-ADD2
        PUSH BC             Save the length.
        INC BC              Add four extra locations:
        INC BC              two for the number and two
        INC BC              for the length.
        INC BC              Make HL point to the location before the
        DEC HL              destination.
        LD DE,(PROG)        Save the current value of PROF to avoid
        PUSH DE             corruption when making space for new line.
        CALL MAKE-ROOM      Create space for the new line.
        POP HL              Recall old value of PROG.
        LD (PROG),HL        Restore.
        POP BC              Copy of line length.
        PUSH BC
        INC DE              Make DE point to end of location in new
        LD HL,(WORKSP)      area and HL to the carriage return
        DEC HL              character of new line in the editing area.
        DEC HL
        LDDR                Copy the line to the program.
        LD HL,(EPPC)        Fetch the line number.
        EX DE,HL            Destination in HL, number in DE.
        POP BC              Get new line's length.
        LD (HL),B           High length byte.
        DEC HL
        LD (HL),C           Low length byte.
        DEC HL
        LD (HL),E           Low line number byte.
```

```
        DEC HL
        LD (HL),D           High line number byte.

MAIN-ADD2
M11A6   POP AF              Drop address of REPORT-G.
        JP MAIN-EXEC
```

Initial Channel Information

There are four channels - 'K', 'S', 'R', & 'P' - for communicating with the 'keyboard', 'screen', 'work space' and 'printer'. Each channel is arranged as: output routine address, input routine address and the channel code.

```
M11AA   DEFB $00, $05       PRINT-OUT
        DEFB $0E, $0C       KEY-INPUT
        DEFB $4B            K (Keyboard)
        DEFB $00, $05       PRINT-OUT
        DEFB $BF, $11       REPORT J
        DEFB $53            S (Screen)
        DEFB $E7, $0A       ADD-CHAR
        DEFB $BF, $11       REPORT J
        DEFB $52            R (Workspace)
        DEFB $00, $05       PRINT-OUT
        DEFB $BF, $11       REPORT J
        DEFB $50            P (Printer)
        DEFB $80            End of table

REPORT J (INVI)
M11BF   RST $08             Error: Invalid IO device.
        DEFB 12H
```

Initial Stream Data

There are 7 built-in streams.

```
        DEFB $01 $00        Stream $FD (-3): Channel 'K'
        DEFB $06 $00        Stream $FE (-2): Channel 'S'
        DEFB $0B $00        Stream $FF (-1): Channel 'R'
        DEFB $01 $00        Stream 0: Channel 'K'
        DEFB $01 $00        Stream 1: Channel 'K'
        DEFB $06 $00        Stream 2: Channel 'S'
        DEFB $10 $00        Stream 3: Channel 'P'
```

WAIT-KEY Routine

Read a character from the current channel, deposit it in A.

```
WAIT-KEY (RDCH)
M11CF   BIT CLHS,(IY+OTVFLAG)   Jump if not clearing LHS
        JR NZ, WAIT-KEY1        on key press.
        SET ECHREQ,(IY+OTVFLAG) Force echo keyboard input.
```

WAIT-KEY1

```
M11D9   CALL INPUT-AD       Get a character from the current channel.
        RET C               Return with the character
        JR Z, WAIT-KEY1     Back around for the next character.
```

REPORT-8

```
M11DF   RST $08             Error: End of File.
        DEFB $07
```

INPUT-AD Subroutine

Get address of character from the current channel in CURCHL without waiting. Returns C if the character is ok. Returns NC, NZ if end of file (for a finite device like a disk drive). Returns NC, Z if no character was available.

INPUT-AD (INCH)

```
M11E1   EXX                 Save the registers.
        PUSH HL
        LD HL,(CURCHL)      Get current channel routine address.
        INC HL              Point to the input routine.
        INC HL
        JR CALL-SUB         Jump forward.
```

Main Printing Subroutine

Called with either an absolute value or a character code in A. Timex extended this routine to send the value in the A register to a routine in any expansion bank. CURCBN (current channel bank number) is checked to provide support for expansion banks. For HOME bank, it's set to 0. The DOCK and EXROM banks are not supported by mechanism. If it weren't for a bug, it could be used to INPUT a character through a routine in an expansion bank numbered 2 or higher.

OUT-CODE (PUTDIG)

Send a decimal digit whose binary value is in A to the current channel.

```
M11EA   LD E,$30            Make the digit ASCII.
        ADD A,E
```

PRINT-A-2 (SENDCH)

Send character in A to the current stream.

```
M11ED   EXX                 Save the registers.
        PUSH HL             Save HL.
        LD HL,(CURCHL)      Get the base address for the current channel. It
                            will point to an output address.
```

CALL-SUB

HL points to the output or input address.

```
M11F2   EX AF, AF'          Save AF.
        LD A, (CRCBN)       Get the current channel bank number.
        CP $02              Is the current channel bank number <= 2?
```

```
        JR NC, CKEXPBNK       Yes, check if an expansion bank is active.
        EX AF, AF'            Restore AF.
        LD E,(HL)             Load DE with
        INC HL                output routine address
        LD D,(HL)             for the current channel.
        EX DE,HL              Output routine address to HL, CURCHL to
        CALL CALL-JUMP        DE.
        POP HL                Restore HL.
        EXX                   Back to the normal regs.
        RET
```

CKEXPBNK

```
M1205   EX AF,AF'             Restore AF.
        LD HL,(CURCHL)        Get the current channel pointer.
        LD B,(HL)
        LD C,$88
        LD A,(ARSFLG)         Get cart flags.
        BIT BANKCHN,A
        JR NZ, BANK-STREAM    Jump: output to an expansion bank.
        INC HL
        INC HL
```

BANK-STREAM

Send this data to an expansion bank.

```
M1215   LD A,(STRMN)          Get current stream number
        LD E,A                Put in E to save it.
        LD D,$00
        PUSH DE               Save the stream pointer.
        LD DE,$0007
        ADD HL,DE             Address of the stream handler in the bank.
        PUSH HL               Call address.
        PUSH BC               B = bank, C=Horizontal Select.
        LD BC,$0002           Two parameters out.
        PUSH BC
        LD BC,$0000           No parameters in.
        PUSH BC
        CALL CALL_BANK
        POP HL                Restore the current stream number.
        EXX                   Restore normal registers.
        RET
```

Channel Open Subroutine (SELECT)

Select a stream, stream number in A. Depending on the stream data, a particular channel will be made the current channel.

CHAN-OPEN (SELECT)

```
M1230   ADD A,A               Double stream number,
        ADD A,$16             add offset into stream table and
```

```
                LD L,A                  move address to L.
                LD H,$5C                High byte of stream base address.
                LD E,(HL)               Get pointer to the stream
                INC HL                  routine.
                LD D,(HL)
                LD A,D                  Is entry zero?
                OR E
                JR NZ,SKIP-ERR          Jump if stream entry is not zero.
```

REPORT-O (ERRO)

```
M123D   RST $08                 Error: Invalid stream.
        DEFB $17
```

SKIP-ERR

```
M123F   CP $80                  Jump if stream number is negative.
        JR NC, EXPBK-CHAN
```

CHAN-OP-1

```
M1243   DEC DE                  Reduce the stream data.
        LD HL,(CHANS)           Get CHANS
        ADD HL,DE               Make the required address.
```

CHAN-FLAG Subroutine

Set the appropriate flags for the different channels.

CHAN-FLAG (SET_HL)

```
M1248   LD (CURCHL),HL          Make the channel address in HL the current
                                channel.
        LD A,$00                Force bank number of current channel
        LD (CRCBN),A            to be zero.
        RES RETPOS,(IY+OFLAGS2) Reset 'retype possible after syntax error.'
        INC HL                  Step past the output
        INC HL                  and the input addresses and
        INC HL                  make HL point to the
        INC HL                  channel code.
        LD C,(HL)               Get channel type (K,S,P).
        LD HL, KLOOK            Find in the table below.
        CALL INDEXER            Index into the table, find the offset.
        RET NC                  Return if not found (not K,S,P).
        LD D,$00                Pass offset to the
        LD E,(HL)               DE register pair.
        ADD HL,DE               Jump forward to the appropriate
```

CALL-JUMP

```
M1264   JP (HL)                 Output routine.
```

EXPBK-CHAN

```
M1265   LD HL,(SYSCON)          Get address of system config table.
        SUB $80                 Make stream positive.
        LD D,A                  Save in D.
```

```
                ADD HL,DE                   Add DE to SYSCON address to HL.
                LD (CURCHL),HL              Make the channel address in HL the current
                                            channel.
                LD A,(HL)                   Put bank number for channel in A.
                LD (CRCBN),A                Set CURCBN.
                RES RETPOS,(IY+OFLAGS2)     Reset 'retype possible after syntax error.'
                INC HL                      Step past the output
                INC HL                      and the input addresses and
                INC HL                      make HL point to
                INC HL                      the SELECT routine.
                INC HL
                INC HL
                LD A,(CRCBN)                Get current channel bank number.
                LD B,A                      Save in B.
                LD C,$88
                LD D,(HL)                   High byte address of SELECT routine.
                INC HL
                LD E,(HL)                   Low byte address of SELECT routine.
                LD H,D
                LD L,E
                PUSH HL                     Call address.
                PUSH BC                     C=Horizontal Select, B=bank.
                LD BC,$0000                 No params.
                PUSH BC
                PUSH BC
                CALL CALL_BANK
                RET
```

Channel Code Look-Up Table

KLOOK

```
                                    Chan  Offset   Address
M1293   DEFM $4B, $06,              "K"   $06      M129A
        DEFM $53, $12               "S"   $12      M12A8
        DEFM $50, $1B               "P"   $1B      M12AC
        DEFB 0                      End of table
```

Channel 'K' Flag Subroutine

CHAN-K (KANALK)

```
M129A   SET LHS,(IY+OTVFLAG)        Print to lower half of screen.
        RES KEYHIT,(IY+OFLAGS)      Signal 'ready for a key'.
        SET RETPOS,(IY+OFLAGS2)     Retype possible after syntax error.
        JR CHAN-S-1                 Jump forward.
```

Channel 'S' Flag Subroutine

CHAN-S (KANALS)

```
M12A8   RES LHS,(IY+OTVFLAG)    Print to the top half of the screen.
```

CHAN-S-1 (KANS1)

```
M12AC   RES PR,(IY+OFLAGS)      Reset printer flag, print to TV.
        JP TEMPS                jump to set the attributes.
```

Channel 'P' Flag Subroutine

CHAN-P (KANALP)

```
M12B3   SET PR,(IY+OFLAGS)      Force print to printer.
        RET
```

MAKE-ROOM Subroutine

Called on to open up an area in RAM. HL points to the location after the place where room is required and BC holds the amount of space needed. When only a single space only is required, subroutine is entered at ONE-SPACE.

ONE-SPACE (INSI)

```
M12B8   LD BC,$0001             Insert 1 byte at (HL).
```

MAKE-ROOM (INSERT)

```
M12BB   PUSH HL                 Save insertion address.
        CALL TEST-ROOM          Check for sufficient RAM; error 4 if not.
        POP HL                  Restore insertion address.
        CALL POINTERS           Adjust various pointers in the system variables.
```

DE now points to the old STKEND (just before the free RAM) and HL points to the fence value (will be discarded). BC will contain the length of the memory block to be moved. This will usually be different from the entry value.

```
        LD HL,(STKEND)          Get STKEND (start of spare space).
        EX DE,HL                HL = entry value, DE = HL + BC.
        LDDR                    Move the bytes up to open buffer.
        RET
```

POINTERS Subroutine

Whenever memory space has to be created or reclaimed, the system variables that address locations beyond the position of the change have to be amended as required. On entry, BC holds the number of bytes involved and HL addresses the location before the position.

POINTERS (REMGSZ)

```
M12CA   PUSH AF                 Save registers.
        PUSH HL
```

This section of code adjusts the pointer to the AROS buffer stored in ARSBUF, if needed.

RESET-CART-POINTERS

```
M12CC   LD HL,ARSBUF        Get AROS buffer pointer.
        LD E,(HL)           Copy to DE, DE now points to the
        INC HL              AROS buffer.
        LD D,(HL)
        EX (SP),HL          Save ARSBUF and get old HL.
        AND A               Set carry if HL<DE.
        SBC HL,DE           If address in HL is lower than the
        ADD HL,DE           AROS buffer, do CP HL,DE.
        EX (SP),HL          Get ARSBUF back and save HL.
        JR NC, NOT-AROS     Jump if the address in HL is after AROS.
        EX DE,HL            DE now has ARSBUF and HL has the address of
                            the AROS buffer.
        ADD HL,BC           Adjust the address of the AROS buffer by
                            BC bytes.
        EX DE,HL            HL now has ARSBUF and DE has the new
        LD (HL),D           address of the AROS buffer.
        DEC HL              Store the value in ARSBUF.
        LD (HL),E
```

NOT-AROS

Spectrum POINTERS routine picks up here; this section adjusts the system variables VARS through STKEND.

```
M12E0   LD HL,VARS          Get the address for VARS.
        LD A,$0E            Number of variables to update.
```

NEXT-POINTER

```
M12E5   CP $09              Test NXTLIN against ARSFLG
        JR Z, 8-OR-9        because it may be in the AROS buffer.
        CP $08              DATADD: data statement address,
        JR NZ, PTR-NEXT     same situation as NXTLIN.
```

8-OR-9

Test the AROS present flag. If an AROS is present, skip the pointer update.

```
M12ED   PUSH HL             Briefly save HL.
        LD HL,ARSFLG        Load ARSFLG
        LD L,(HL)           Get the AROS flag and test
        BIT AROS,L          to see if an AROS is present.
        POP HL              Restore HL.
        JR Z, PTR-NEXT      Jump if AROS not present.
        INC HL              Point to the high byte of the pointer,
        JR PTR-DONE         skip the addition routine.
```

PTR-NEXT
Update a pointer by determining if it points above the fence address in HL. If it does, add BC to the pointer and then stuff it back into the system variables.

```
M12FA   LD E,(HL)           Get address pointer at (HL) into DE.
        INC HL
        LD D,(HL)           Exchange the system variable with the
        EX (SP),HL          address of the position.
        AND A               Set carry flag.
        SBC HL,DE           Jump if HL >= (POINTER).
        ADD HL,DE           Pointer doesn't need updating since we
        EX (SP),HL          are moving stuff above where it points.
        JR NC, PTR-DONE
```

Adjust the pointer by adding BC to it, then save it back to the system variables.

```
        PUSH DE             Save the old value.
        EX DE,HL            Add the value in BC to old value.
        ADD HL,BC
        EX DE,HL            Save new value to the system value,
        LD (HL),D           high byte first.
        DEC HL
        LD (HL),E
        INC HL              Point again to the high byte.
        POP DE              Get the old value.
```

PTR-DONE
Now look at the next pointer and decrement the counter in A. That tells us how many pointers are left to update. If there are more, loop back again.

```
M130E   INC HL              Pointer to next system variable
        DEC A               and jump back until all 14 have been
        JR NZ,NEXT-POINTER  updated.
```

Find the size of the block to be moved.

```
        EX DE,HL            Put old value of STKEND in HL
        POP DE              and restore other registers.
        POP AF
        AND A
        SBC HL,DE           Find the number of bytes between the
        LD B,H              fence and where old STKEND pointed and
        LD C,L              copy it to BC. This sets the size of the memory
                            block to be moved.
        INC BC              Add 1 for the inclusive byte.
        ADD HL,DE           Restore old value of STKEND and pass to
        EX DE,HL            DE before returning.
        RET
```

Collect a Line Number Subroutine

On entry, HL points to the location under consideration. If the location holds a value that constitutes a suitable high byte for a line number, then the line number is returned in DE. However if this is not, the location addressed by DE is tried instead. If that is also unsuccessful, line number zero is returned.

LINE-ZERO (DUMMYLINE)

```
M131E   DEFW $0000          Line number zero.
```

LINE-NO-A

```
M1320   EX DE,HL            Point to the previous line.
        LD DE,LINE-ZERO     Allow an exit by pointing to line zero.
```

LINE-NO (GET_LN)

Get the line number into DE from (HL). On entry, HL points to the program line number; DE points to the previous program line number. On exit, HL points to the low byte of the line number, DE contains the line number.

```
M1324   LD A,(HL)           Get the high byte of the line number.
        AND $C0             Jump if not a line number.
        JR NZ, LINE-NO-A
        LD D,(HL)           Load the line number into DE.
        INC HL
        LD E,(HL)
        RET
```

Reserve Subroutine

On entry, stack holds the WORKSP pointer and value for BC. On exit, DE points to the first byte of of the reserved area. Makes room between WORKSP and the calculator stack.

RESERVE (LCU2)

```
M132D   LD HL,(STKBOT)      Get STKBOT and decrement it to
        DEC HL              point to address just below calc stack.
        CALL MAKE-ROOM      Insert BC items at (HL), updating system
                            variables.
        INC HL              Point to the first news space
        INC HL              and then the second.
        POP BC              Get saved value for WORKSP and restore
        LD (WORKSP),BC      it.
        POP BC              Restore BC, the number of items to insert.
        EX DE,HL            Switch pointers.
        INC HL              HL = 2 bytes into the moved memory block. DE =
                            the first byte of the new block.
        RET
```

Set Minimum Workspace Subroutine

This subroutine resets the editing area and the areas after it to their minimum sizes. Clears those areas.

SET-MIN

```
M133F   LD HL,(ELINE)       Get ELINE.
        LD (HL),$0D         Make the editing area hold only
        LD (KCUR),HL        'carriage return'.
        INC HL              Put sentinel byte in ELINE buffer.
        LD (HL),$80
        INC HL              Initialize the workspace pointer
        LD (WORKSP),HL
```

SET-WORK (X_CALC)

Entering here 'clears' the work space and the calculator stack.

```
M134E   LD HL,(WORKSP)      Fetch WORKSP
        LD (STKBOT),HL      Clear work space.
```

SET-STK (RESET)

Reset the floating point stack and floating point memory pointers. EXIT with STKBOT in HL.

```
M1354   LD HL,(STKBOT)      Get STKBOT.
        LD (STKEND),HL      Clear the stack.
```

Make MEM address the calculator memory area.

```
M1354   PUSH HL             Save STKBOT.
        LD HL,MEMBOT        Base of memory area.
        LD (MEM),HL         Reset MEM memory pointer.
        POP HL              Restore STKBOT
        RET
```

Reclaim the Edit-Line Subroutine (X_T_HL)

REC-EDIT (X_T_HL)

```
M1363   LD DE,(ELINE)       Get ELINE.
        JP RECLAIM-1        Reclaim the memory.
```

Indexer Subroutine

Searches for an item in C starting at (HL). Terminates when item found or when 0 found. On exit HL points one past the item if found or to the 0 byte if 0 was found. CF=0 if zero found, else CF=1 if item in C was found.

INDEXER-1

```
M136A   INC HL              Move on to consider next pair of entries.
```

INDEXER (SEARCH)

```
M136B   LD A,(HL)          Fetch first pair of entries,
        AND A              but return if it is zero (the end marker).
        RET Z
        CP C               Compare with the supplied code.
        INC HL             Point to next entry
        JR NZ, INDEXER-1   Did not find code, go back around.
        SCF                Found the byte, set C.
        RET
```

Search Expansion Bank System Configuration

Search system configuration table for expansion bank channel. This code is invoked in two other routines that are not used.

SRCHSC

```
M1374   LD HL,(SYSCON)     Get SYSCON.
        LD DE,$000C        Step past the AROS and LROS
        ADD HL,DE          entries.
```

SR-CH-LOOP

```
M137B   LD A,(HL)          Get SYSCON table item.
        CP $80             End of table?
        JR Z,SR-CH-2
        INC HL             Skip past next two bytes.
        INC HL
        CP $01             Bank not in use, so check.
        JR NZ, SR-CH-1
        LD A,(HL)          Check for bank signature,
        CP C               If found, exit routine.
        JR Z, SR-CH-OUT
```

SR-CH-1

```
M138A   PUSH HL            Save pointer.
        EX DE,HL
        LD DE,$0018        Skip to next SYSCON table
        ADD HL,DE          entry+2.
        EX DE,HL           DE points to new entry.
        POP HL             Get old pointer.
        PUSH DE            Save new pointer.
        LD DE,$0016        Make old pointer point
        ADD HL,DE          to new SYSCON entry.
        POP DE             Restore DE.
        JR SR-CH-LOOP      Keep searching.
```

SR-CH-2

SYSCON DOCK.

```
M139A   AND A              Clear carry.
        RET                Exit.
```

SR-CH-OUT
SYSCON XBANK.

```
M139C   DEC HL
        SCF             Set the carry.
        RET
```

CLOSE # Command Routine

Allows the user to CLOSE streams. Streams 0 to 3 are the 'initial' streams: data is restored and these streams cannot be CLOSEd.

CLOSE

```
M139F   CALL STR-DATA   Get the stream at the top of the calculator
        LD A,B          stack.
        OR C            Return if the stream routine
        RET Z           address is zero (invalid stream).
        CALL CLOSE-2
```

RESTORE-STREAM (RSTSTR)

```
M13A8   LD BC,$0000     Prepare to make stream data zero
        LD DE,$A3E2     Setup to identify use of streams 0 to 3.
        EX DE,HL
        ADD HL,DE       Carry flag set for stream 4 to 15.
        JR C,CLOSE-1    Jump forward for streams 4 to 15.
        LD BC,$11CF     Address of RD_CH.
        ADD HL,BC       Find the correct entry in the 'initial stream
        LD C,(HL)       data' table and get it for streams 0 to 3.
        INC HL
        LD B,(HL)
```

CLOSE-1

```
M13B9   EX DE,HL        Enter the data: either zeros or the initial
        LD (HL),C       values.
        INC HL
        LD (HL),B
        RET
```

CLOSE-2 Subroutine

Close a K, S or P channel. HL is stream address pointer, BC is the address of the stream routine.

CLOSE-2 (CLCHAN)

```
M13BE   PUSH HL         Save stream address pointer.
        LD A,B          Jump if the stream is negative.
        CP $80
        JR NC,SYSCON-CK
        LD HL,(CHANS)   Get the address of the CHANS area
        ADD HL,BC       and find the channel data for the
        INC HL          stream being closed. Skip past subroutine
```

```
        INC HL          addresses and pick up code fro the
        INC HL          channel.
        LD C,(HL)       Get the stream type: "K","S"or "P"
        EX DE,HL        Save the pointer in HL.
        LD HL,CSTRTA    Point to the close stream table.
        CALL INDEXER    Search for the stream type.
        LD C,(HL)       Get the offset and add it to
        LD B,$00        HL to give the address of the
        ADD HL,BC       close stream routine.
        JP (HL)         Close the stream.
```

SYSCON-CK
System configuration check for close routine.

```
M13D8   SUB $80         Normalize the number.
        LD B,A
        LD HL,(SYSCON)
        ADD HL,BC
        LD A,(HL)
        CP $00          Is it zero?
        RET Z           Yes, return.
        CP $80          End marker?
        RET Z           Yes, return.
        INC HL          Skip past bank number.
        LD B,(HL)       Get channel specifier.
        INC HL          Skip subroutine addresses.
        INC HL
        INC HL
        INC HL
        LD E,(HL)       Get SELECT low byte.
        INC HL
        LD D,(HL)       Get SELECT high byte.
        LD H,D
        LD L,E
        LD A,(STRMN)
        LD E,A
        LD D,$00
        PUSH DE         Return address.
        PUSH HL
        PUSH BC         B= bank, C=Horizontal Select.
        LD BC,$0002     PRM_OUT
        PUSH BC
        LD BC,$0000     No PRM_IN.
        PUSH BC
        CALL CALL_BANK  Call expansion bank SELECT routine.
        POP HL
        RET
```

Close Stream Look-Up Table

CSTRTA

```
M1407   DEFB $4B, $05          "K"
        DEFB $53, $03          "S"
        DEFB $50, $01          "P"
```

CLOSE-STR (CLOSTR)
Close stream.

```
M140D   POP HL                 Get the channel information pointer and
        RET                    return.
```

Stream Data Subroutine

Returns the stream data for a given stream in the BC register pair.

STR-DATA

```
M140F   CALL FIND-INT1         Convert the top of the calculator stack
                               to a byte in A. If the number is more
                               than 255, we won't return here.
        LD (STRMN),A           Update current stream number.
        CP $10                 Jump if stream number is valid (less than
        JR C, STR-DATA1        $10).
```

REPORT-O

```
M1419   RST $08                Error: Invalid stream.
        DEFB $17
```

STR-DATA1
Continue with valid stream numbers.

```
M141B   ADD A,$03              Adjust range to $3 to $12;
        RLCA                   then to $6 to $24.
        LD HL,STRMS            Point to entry in streams table.
        LD C,A                 Move stream code to BC.
        LD B,$00
        ADD HL,BC              Index into the data area.
        LD C,(HL)              Get the address for the
        INC HL                 stream routine to BC.
        LD B,(HL)
        DEC HL                 Make pointer address first of data bytes
        RET                    before returning.
```

OPEN # Command Routine

OPEN streams. Channel code must be supplied and it must be 'K', 'k', 'S', 's', 'P', or 'p'. No attempt is made to give streams 0 to 3 their initial data.

IS-COMMA

```
M142A   CP ','                 Jump if current character is a comma.
```

```
            JR Z, GET-NEXT-ARG
            CALL CHECK-END      Will return if interpreting, else
            JR OPEN             will exit via ENDTEM.
```

GET-NEXT-ARG

```
M1433       CALL SYNTAX-Z       Jump if interpreting
            JR NZ, OPEN
            CALL SKIPIT         skip to the end of a statement.
            CALL CHECK-END
```

OPEN

```
M143E       RST $28             Use the calculator to
            DEFB $01            exchange the stream number and
            DEFB $38            the channel code.
            CALL STR-DATA       Fetch the data for the stream.
            LD A,B              Jump forward if both bytes are zero
            OR C                (the stream was in a closed state).
            JR Z, OPEN-1
            EX DE,HL            Save DE.
            LD HL,(CHANS)       Fetch CHANS, base address of the
            ADD HL,BC           channel information and find the code
            INC HL              of the channel associated with the stream
            INC HL              to be OPENed.
            INC HL
            LD A,(HL)
            EX DE,HL            Restore DE.
            CP $4B              The code fetched from channel
            JR Z, OPEN-1        information area must be 'K', 'S' or 'P'.
            CP $53              Is it 'S'?
            JR Z, OPEN-1
            CP $50              Is it 'P'?
            JR NZ, REPORT-O     Nope, report error.
```

OPEN-1 (OPEN-IT)

```
M145E       CALL OPEN-2         Get the appropriate data in DE.
            LD (HL),E           Enter stream data in to the two bytes
            INC HL              in the stream information area.
            LD (HL),D
            RET
```

OPEN-2 Subroutine

Get stream data bytes for the channel that is associated with the stream being OPENed.

OPEN-2 (OPCHAN)

```
M1465       PUSH HL             Save HL.
            CALL STK-FETCH      Get a parameters of the channel code.
            DEC BC
            LD A,B              Jump if the expression is null.
            OR C
```

```
         JR Z, OPEN-3
```

REPORT-J

```
M146E    RST $08          Error: Invalid IO device.
         DEFB $12
```

REPORT-F

```
M1470    RST $08          Error: Invalid save name.
         DEFB $0E         No name or name more than 10 chars.
```

OPEN-3

```
M1472    INC BC           Restore channel designator string length.
         PUSH BC          Save it.
         LD A,(DE)        Get the channel designator character,
         AND $DF          force to upper case.
         LD C,A           Move code to C.
         LD HL,OPTAB      Base addr of OPEN stream look-up table.
         CALL INDEXER     Index into table and find required offset.
         JR NC, INVALID-IO Not in the table, jump to INVALID IO DEVICE.
         LD C,(HL)        Pass offset to BC
         LD B,$00         register pair.
         ADD HL,BC        Make HL point to start of the routine.
         POP BC           Restore the string length.
         JP (HL)          Jump to the open routine.
```

INVALID-IO

```
M1486    JR REPORT-J
```

CALL-EXPBANK

Would have allowed the execution of expansion bank code to OPEN a stream to an expansion bank channel.

```
M1488    CALL SRCHSC
         JR NC, REPORT-J
         POP BC           Get BC off the stack.
         DEC BC           Decrease the length of the expression
         LD A,B           and report an error if it was not
         OR C             a single character.
         JR NZ, REPORT-J
         PUSH DE
         EX DE,HL
         CALL PASSEM
         EX DE,HL
         LD B,(HL)        Get bank number.
         LD C,$88         All chunks but 0 assigned to new bank.
         INC HL
         INC HL
         LD E,(HL)
         INC HL
         LD D,(HL)
```

```
        LD H,D
        LD L,E
        LD A,(STRMN)
        LD E,A
        LD D,$00
        PUSH DE
        PUSH HL
        PUSH BC
        LD HL,(STKEND)
        LD C,(HL)
        DEC HL
        LD (STKEND),HL
        LD B,$00
        INC BC
        INC BC
        PUSH BC
        LD BC,$0000
        PUSH BC
        CALL CALL_BANK
        POP DE
        LD A,D
        ADD A,$80
        LD D,A
        POP HL
        RET
```

Open Stream Look-Up Table

OPTAB

```
M14C7   DEFB $4B, $06       "K", offset +$06
        DEFB $53, $08       "S", offset +$08
        DEFB $50, $0A       "P", offset +$0A
        DEFB $00            End marker
```

OPEN-K Subroutine

OPEN-K

```
M14CE   LD E,1              Data bytes are 01 and 00.
        JR OPEN-END
```

OPEN-S Subroutine

OPEN-S

```
M14D2   LD E,6              Data bytes are 06 and 00.
        JR OPEN-END
```

OPEN-P Subroutine

OPEN-P

```
M14D6   LD E,10H            Data byte are 10 and 00.
```

OPEN-END

```
M14D8   DEC BC              Decrease length of expression
        LD A,B              and report an error if it was not
        OR C                a single character.
        JP NZ, REPORT-F
        LD D,A              Otherwise, clear D,
        POP HL              fetch HL and return.
        RET
```

LIST & LLIST Command Routines

These produce listings of the current BASIC program. Each line has its line number evaluated, tokens expanded and appropriate cursors positioned. AUTO-LIST is used by both the MAIN EXECUTION routine and the EDITOR to produce a single page of the listing.

AUTO-LIST (TSLIST)

```
M14E1   LD (LISTSP),SP          Save the stack pointer for the listing.
        LD (IY+OTVFLAG),$10     Signal 'automatic listing' in main screen.
        CALL CL-ALL             Clear the upper screen.
        SET LHS,(IY+OTVFLAG)    Set printing to lower half of screen.
        LD B,(IY+ODFSZ)         Get lower screen size.
        CALL CL-LINE            Clear the lower screen for B lines.
        RES LHS,(IY+OTVFLAG)    Release writing to the lower screen
        SET ALOS,(IY+OFLAGS2)   Set auto listing on screen.
        LD HL,(EPPC)            Get the current listing line number.
        LD DE,(STOP)            Get the line listed at top of screen.
        AND A
        SBC HL,DE
        ADD HL,DE               Jump if top line is greater than
        JR C, AUTO-L-2          the current listing line.
        PUSH DE                 Save the top line.
        CALL LINE-ADDR          Find the current line and
        LD DE,$02C0             produce an address roughly
        EX DE,HL                a 'screen before it'
        SBC HL,DE               (negated).
        EX (SP),HL              Save result on the machine stack.
        CALL LINE-ADDR          Top line goes in HL.
        POP BC                  Result goes in BC.
```

AUTO-L-1

```
M151A   PUSH BC             Save result.
        CALL NEXT-ONE       Find address of the start of the line after the
                            present line (in DE).
        POP BC              Save that.
```

```
              ADD HL,BC               Add and jump
              JR C, AUTO-L-3          forward if finished.
              EX DE,HL                Move the next line's address to HL
              LD D,(HL)               and get its number.
              INC HL
              LD E,(HL)
              DEC HL
              LD (STOP),DE            Update STOP and test with
              JR AUTO-L-1             new line.

AUTO-L-2
M152D         LD (STOP),HL            When EPPC is less than STOP.

AUTO-L-3
M1530         LD HL,(STOP)            Get the top line's number and
              CALL LINE-ADDR          its address.
              JR Z, AUTO-L-4          If the line cannot be found,
              EX DE,HL                use DE instead.

AUTO-L-4
M1539         CALL LIST-ALL           Make the listing.
              RES TVLIST,(IY+OTVFLAG) Return here unless scrolling was
              RET                     needed to show current line.
```

LLIST Entry Point

Open the printer channel, then list.

```
LLIST (K_LLST)
M1541         LD A,$03                Use stream $03.
              JR LIST-1
```

LIST Entry Point

Open the main screen channel, then list.

```
LIST (K_LIST)
M1545         LD A,$02                Use stream $02.

LIST-1
M1547         LD (IY+OTVFLAG),00      Printing to upper half of screen, not outputting
                                      line for edit or number for string, no keyboard
                                      echo, no automatic listing, do not clear lower half
                                      screen when key pressed.
              CALL SYNTAX-Z           Select stream 2 if interpreting.
              CALL NZ,CHAN-OPEN
              RST $18                 Get current character.
              CALL STR-ALTER          Update the current stream for "bank" devices.
              JR C, LIST-4            Jump if not valid instruction.
              RST $18                 Get current character.
              CP ';'                  Is it a semi-colon?
```

```
            JR Z, LIST-2            Jump if it is.
            CP ','                  Is it a comma?
            JR NZ, LIST-3           Jump if not.

LIST-2
M1560       RST $20                 A numeric expression must follow the comma.
                                    Get it.
            CALL EXPT-1NUM          Get the first line to be listed.
            JR LIST-5

LIST-3
M1566       CALL USE-ZERO           Otherwise, use a zero and
            JR LIST-5               jump forward.

LIST-4
M156B       CALL FETCH-NUM          Get any line or zero.

LIST-5
M156E       CALL CHECK-END          If checking the syntax of edit line move on next
                                    statement.
            CALL FIND-INT2          Put the first line number to be listed in BC.
            LD A,B                  Force the number to be a valid
            AND $3F                 line number.
            LD H,A                  load the line number into HL.
            LD L,C
            LD (EPPC),HL            Set EPPC (current line number) in
            CALL LINE-ADDR          the listing and find the first line after it.

LIST-ALL
M157F       LD E,$01                Flag 'before the current line'.

LIST-ALL-1
M1581       CALL OUT-LINE           Print the entire line.
            RST $10                 And a carriage return.
            BIT TVLIST,(IY+OTVFLAG) Jump back unless dealing with
            JR Z, LIST-ALL-1        and automatic listing.
            LD A,(DFSZ)             Also jump back if there is still
            SUB (IY+OSPOSNLIN)      space on the main screen that
            JR NZ, LIST-ALL-1       can be used.
            XOR E                   Return here if the screen is full and
            RET Z                   the current line has been printed.
            PUSH HL                 If the current line is missing from
            PUSH DE                 the listing, then STOP has to be
            LD HL,STOP              updated and another line printed
            CALL LN-FETCH           (using scrolling).
            POP DE
            POP HL
            JR LIST-ALL-1
```

Print A Whole Basic Line Subroutine

HL register pair points to the start of the line, the location holding the high byte of the line number. Line number is tested to determine whether it comes before the 'current' line, is the 'current' line or comes after and then printed.

OUT-LINE (PUT_SR)

```
M15A1   LD BC,(EPPC)          Get the 'current line' number
        CALL CP_LINES         and compare it.
        LD D,$3E              Pre-load D with the current line
        JR Z,OUT-LINE1        cursor.
```

LPO

```
M15AC   LD DE,$0000           Load D with zero and set E to hold
        RL E                  1 if the line is before the current line and 0 if after
                              it.
```

OUT-LINE1

```
M15B1   LD (IY+OBREG),E       Save line marker.
        LD A,(HL)             Get the high byte of the line
        CP $40                number and make a full return if the
        POP BC                listing is finished.
        RET NC
        PUSH BC
        CALL OUT-NUM-2        Line number can be printed with
        INC HL                leading spaces. Move pointer to
        INC HL                the address of first command code
        INC HL                in the line.
        RES SPC,(IY+OFLAGS)   Signal 'leading space allowed'.
        LD A,D                Get current cursor code and
        AND A                 jump forward unless it the cursor
        JR Z, OUT-LINE3       should be printed.
        RST $10               Print the cursor.
```

OUT-LINE2 (PUT)

```
M15C9   SET SPC,(IY+OFLAGS)   Set 'no leading space'.
```

OUT-LINE3

```
M15CD   PUSH DE               Save the registers.
        EX DE,HL              Move pointer to DE.
        RES L_STR,(IY+OFLAGS2) Signal 'not in quotes'.
        LD HL,FLAGS
        RES LMODE1,(HL)       Signal 'print in K-mode'.
        BIT INPLN,(IY+OFLAGX) Jump forward unless in
        JR Z, OUT-LINE4       INPUT mode.
        SET LMODE1,(HL)       Signal 'print in L-mode'.
```

OUT-LINE4
Loop to print all the codes in the rest of the BASIC line, jumping over floating-point numbers as necessary.

```
M15E0   LD HL,(X_PTR)       Get the syntax error point and jump
        AND A               forward unless it is time to print the
        SUB HL,DE           error marker.
        SBC HL,DE
        JR NZ, OUT-LINE5
        LD A,$3F            Print the error marker,
        CALL OUT-FLASH      a flashing '?'.
```

OUT-LINE5

```
M15ED   CALL OUT-CURS       Consider whether to point the cursor.
        EX DE,HL            Move pointer to HL.
        LD A,(HL)           Get each character in turn.
        CALL NUMBER         If the character is a 'number marker', then skip
                            over the hidden floating-point char.
        INC HL              Update pointer for next pass.
        CP $0D              Is this a carriage return?
        JR Z, OUT-LINE6     Jump if it is.
        EX DE,HL            Switch the pointer to DE.
        CALL OUT-CHAR       Print the character.
        JR OUT-LINE4        Make another pass through the loop.
```

OUT-LINE6

```
M1600   POP DE              Restore DE and
        RET                 return.
```

NUMBER Subroutine

If the A register holds the number marker, then HL is advanced past the floating-point form.

NUMBER

```
M1602   CP $0E              Is the character a 'number marker'?
        RET NZ              Return if not.
        INC HL              Advance pointer six times
        INC HL              to past the number marker
        INC HL              and characters holding
        INC HL              the floating point number.
        INC HL
        INC HL
        LD A,(HL)           Get the current code and
        RET                 return.
```

Print A Flashing Character Subroutine

Print the cursor used by the error and mode cursors.

OUT-FLASH (FLASHA)

```
M160D   EXX                     Save current register.
        LD HL,(ATTRT)           Save ATTR and MASKT
        PUSH HL                 to the machine stack.
        RES 7,H                 Set FLASH to active.
        SET 7,L
        LD (ATTRT),HL           Use the modified values.
        LD HL,PFLAG             Save PFLAG to machine stack.
        LD D,(HL)
        PUSH DE
        LD (HL),$00             Insure INVERSE 0, OVER 0 and not
                                PAPER 9 nor INK 9.
        CALL PRINT-OUT          Print the character.
        POP HL                  Restore PFLAG.
        LD (IY+OPFLAG),H
        POP HL                  Restore ATTR-T and MASK-T.
        LD (ATTRT),HL
        EXX
        RET
```

Print The Cursor Subroutine

Return if it is not the correct place to print the cursor. If it is, then either 'C', 'E', 'G', 'K' or 'L' will be printed.

OUT-CURS (PR_CUR)

```
M162D   LD HL,(KCUR)            Get the address of the cursor,
        AND A                   return if this is not the correct place
        SBC HL,DE               to print a cursor.
        RET NZ
        LD A,(MODE)             Get current value of MODE
        RLC A                   and double it.
        JR Z, OUT-C-1           Jump forward unless dealing with
        ADD A,$43               Extended mode or Graphics.
        JR OUT-C-2
```

OUT-C-1

```
M163F   LD HL,FLAGS             Put FLAGS in HL.
        RES LMODE2,(HL)         Signal 'K-mode'.
        LD A,$4B                Character is 'K'.
        BIT LMODE1,(HL)         Jump forward to print K.
        JR Z, OUT-C-2
        SET LMODE2,(HL)         Signal 'L-mode'.
        INC A                   Set cursor to L.
        BIT CAPS_L,(IY+OFLAGS2) Jump forward if not in 'C-mode'.
        JR Z, OUT-C-2
```

```
        LD A,$43                Set cursor to C.
```

OUT-C-2

```
M1655   PUSH DE                 Save DE.
        CALL OUT-FLASH          While the cursor is printed.
        POP DE
        RET
```

LN-FETCH Subroutine

Returns the line number of the line that follows the line pointed to in HL.

LD-FETCH (NEXT_L)

```
M165B   LD E,(HL)               Get the number of the line held by
        INC HL                  the system variable (STOP or EPPC)
        LD D,(HL)               that HL points to.
        PUSH HL                 Save the pointer.
        EX DE,HL                Move line number to be found to HL.
        INC HL                  Look for the next line number.
        CALL LINE-ADDR          Find the address of line number.
        CALL LINE-NO            Put the line number into DE.
        POP HL                  Restore the program line number.
```

LN-STORE (DE_HL)

This entry point is used by the editor. On entry, HL points to the line number location, DE contains the line number. On exit, HL points to the low byte of the program line number, DE contains the line number.

```
M1668   BIT INPLN,(IY+OFLAGX)   Return if 'INPUT mode';
        RET NZ                  otherwise proceed to
        LD (HL),D               store the line number in
        DEC HL                  its location.
        LD (HL),E
        RET
```

Printing Characters In A Basic Line Subroutine

All of the character/token codes in a BASIC line are printed by repeatedly calling this subroutine. The entry point OUT-SP-NO is used when printing line numbers which may require leading spaces.

OUT-SP-2

```
M1671   LD A,E                  A holds $20 (space) or $FF (no space).
        AND A                   Test and return if there is not to be a
        RET M                   space.
        JR OUT-CHAR
```

OUT-SP-NO

```
M1676   XOR A                   Clear A.
```

OUT-SP-1
HL register pair holds the line number and the BC register the value for repeated subtraction. (BC holds -1000, -100 or -10.)

```
M1677   ADD HL,BC               Start subtraction.
        INC A                   Count each attempt.
        JR C, OUT-SP-1          Jump back until done.
        SBC HL,BC               Restore last subtraction
        DEC A                   and adjust the counter.
        JR Z, OUT-SP-2          If no subtractions, jump back and see if a space
                                should be printed.
        JP OUT-CODE             Otherwise, print digit.
```

OUT-CHAR

```
M1683   RES TOKEN,(IY+OFLAGS)   Token off.
        BIT LMODE1,(IY+OFLAGS)  Is 'next character is printed in L-mode' set?
        JR Z, OUT-CHAR-1        No, jump forward to print the character.
        SET TOKEN,(IY+OFLAGS)   Yes, turn token on.
```

OUT-CHAR-1

```
M1691   CALL NUMERIC            Return with carry reset if this is numeric.
        JR NC, OUT-CH-3
        CP $0C                  Is it Delete?
        JR Z, OUT-CH-2          Yes, print in L-mode.
        CP $21                  Space or lower?
        JR C, OUT-CH-3          Print character.
        RES LMODE1,(IY+OFLAGS)  Signal 'K-mode'.
        CP $7B                  Is it ON ERR?
        JR NZ, OUT-CHAR-2
        BIT TOKEN,(IY+OFLAGS)   Is token mode on?
        JR Z, OUT-CH-3
```

OUT-CHAR-2

```
M16AC   CP $CB                  Is it THEN?
        JR Z, OUT-CH-3
        CP $3A                  A colon?
        JR NZ, OUT-CH-1
        BIT INPLN,(IY+OFLAGX)   Is this 'INPUT mode'?
        JR NZ, OUT-CH-2
        BIT L_STR,(IY+OFLAGS2)  Inside of a string?
        JR Z, OUT-CH-3
        JR OUT-CH-2
```

OUT-CH-1

```
M16C2   CP $22                  Is it a quote (")?
        JR NZ, OUT-CH-2
        PUSH AF                 Save the character code while changing to quote
                                mode.
        LD A,(FLAGS2)           Get FLAGS2 and
        XOR $04                 flip bit 2.
```

```
         LD (FLAGS2),A          Put FLAGS2 back.
         POP AF                 Restore character code.
```

OUT-CH-2

```
M16D0    SET LMODE1,(IY+OFLAGS) Signal 'next character is printed in L-mode'.
```

OUT-CH-3

```
M16D4    RST $10                Print the character.
         RET
```

LINE-ADDR Subroutine

Returns the starting address of the line in HL or the 'first line after' and the start of the previous line in DE.

If the line number is being used the zero flag will be set. However if the 'first line after' is substituted then the zero flag is returned reset.

LINE-ADDR (FIND_L)

```
M16D6    PUSH HL                Save the line number.
         LD HL,(PROG)           Point to the start of the program,
         LD D,H                 copy it to DE.
         LD E,L
```

LINE-AD-1

```
M16DC    POP BC                 Get the line number.
         CALL CP-LINES          Compare the line number in BC to (HL).
         RET NC                 Return if the current line number is greater than or
                                equal to the number in BC.
         PUSH BC                Save the desired line number.
         CALL NEXT-ONE          Get the length of the record.
         EX DE,HL               HL now points to the next line,
                                DE points to the previous line.
         JR LINE-AD-1           Back around for the next line.
```

Compare Line Numbers Subroutine

Line number in BC is compared with the addressed line number at (HL). Zero flag set if a match, carry flag set if BC < (HL).

CP-LINES (CP_BC)

```
M16E8    LD A,(HL)              Compare the first byte of the
         CP B                   line number to the desired line number.
         RET NZ                 Return if no match.
         INC HL                 Get the next byte of the line number.
         LD A,(HL)
         DEC HL                 Restore the pointer.
         CP C                   Compare to the low byte of the requested
         RET                    number.
```

Find Each Statement Subroutine

Find the statement within a line. On entry, D has the statement number (starting with 1). Returns with HL addressing the location before the start of the statement and the zero flag set. It can also find a statement, if any, that starts with a given token code (in the E register).

SUBLIN

```
M16F0   INC HL          Not used.
        INC HL
        INC HL
```

EACH-STMT (SUBLIN1)

```
M16F3   LD (CH_ADD),HL  Set CH_ADD to the current byte.
        LD C,$00        Set the 'quotes off' flag.
```

EACH-S-1

Loop to handle each statement in the BASIC line.

```
M16F8   DEC D           Decrease D and return if the
        RET Z           required statement has been
        RST $20         found. Get next character code and
        CP E            jump if it does not match the given token code.
        JR NZ, EACH-S-3
        AND A           It's a match. Return with carry and
        RET             zero flags reset.
```

EACH-S-2

```
M1700   INC HL          Update the pointer and fetch the
        LD A,(HL)       next character code.
```

EACH-S-3

```
M1702   CALL NUMBER     Step over any number and get the
        LD (CH_ADD),HL  new code.
        CP $22          Jump forward if the character is not
        JR NZ, EACH-S-4 a quote (").
        DEC C           Otherwise set the quotes flag.
```

EACH-S-4

```
M170D   CP $3A          Jump forward if the character is a colon (:).
        JR Z, EACH-S-5
        CP $CB          Jump forward unless the code token is
        JR NZ, EACH-S-6 'THEN'.
```

EACH-S-5

```
M1715   BIT 0,C         Read the quotes flag and jump back at the end of
                        each
        JR Z, EACH-S-1  statement (incl after THEN).
```

```
EACH-S-6
M1719   CP $0D          Jump back unless at end of BASIC line.
        JR NZ, EACH-S-2
        DEC D           Decrease the statement counter and set the carry
                        flag
        SCF             before returning.
        RET
```

NEXT-ONE Subroutine

Used to find the next line in the program area or the next variable in the variables area. On entry HL has the point from which to start looking. On exit, HL is the same as entry, DE holds address of next record and BC is length record.

```
NEXT-ONE (RECLEN)
M1720   PUSH HL         Save the address of current line or
        LD A,(HL)       variable and get the first byte.
        CP $40          Jump if this is a BASIC line
        JR C, NEXT-O-3  (line numbers are less than $40)
        BIT 5,A         Jump if an array or a string.
        JR Z, NEXT-O-4
        ADD A,A         Jump if simple number or
        JP M, NEXT-O-1  FOR-NEXT variable.
        CCF             Left with long name numeric variables.

NEXT-O-1
M172F   LD BC,$0005     A numeric variable uses five locations but a FOR-
                        NEXT
        JR NC, NEXT-O-2 control variable
        LD C,$12        uses eighteen locations.

NEXT-O-2
M1736   RLA             The carry flag is reset only for long name
                        variables. Loop until the
        INC HL          entire variable name is found.
        LD A,(HL)       Get the next code.
        JR NC, NEXT-O-2 Jump back unless code was last one in the
                        variable name.
        JR NEXT-O-5     Found all of the name, move on to adjust memory
                        pointer.

NEXT-O-3
M173D   INC HL          Bump for a BASIC line: step past low byte of line
                        number.
```

NEXT-O-4

```
M173E   INC HL          2nd bump for BASIC line or bump to access
                        length for number arrays
        LD C,(HL)       or string entities.
        INC HL          Put the length into BC.
        LD B,(HL)
        INC HL          Allow for the inclusive byte.
```

NEXT-O-5

```
M1743   ADD HL,BC       Point to the first byte of the next line or variable,
                        restore
        POP DE          the character pointer.
```

Difference Subroutine

The length between two starts is put in BC. The pointers are restored but returned exchanged.

DIFFER

```
M1745   AND A           Prepare for subtraction.
        SBC HL,DE       Find the length from one start to
        LD B,H          the next and put the result in BC.
        LD C,L
        ADD HL,DE       Restore the address and exchange
        EX DE,HL        before returning.
        RET
```

Reclaiming Subroutine

The entry point RECLAIM-1 is used when the address of the first location to be reclaimed is in the DE register pair and the address of the first location to be left alone is in the HL register pair. The entry point RECLAIM-2 is used when the HL register pair points to the first location to be reclaimed and the BC register pair holds the number of the bytes that are to be reclaimed.

RECLAIM-1 (DEL_DE)

```
M174D   CALL DIFFER     Use difference to find appropriate values.
```

RECLAIM-2 (DELREC)

```
M1750   PUSH BC         Save number of bytes to be reclaimed.
        LD A,B          All the system variable pointes above the area
                        have to be
        CPL             adjusted by BC, so
        LD B,A          this number is the 2's complement
        LD A,C          of BC before the pointers are altered.
        CPL
        LD C,A
        INC BC
        PUSH BC         Save BC.
        CALL POINTERS   Adjust system variable pointers
        EX (SP),HL      down BC bytes. Get BC back and
```

```
        ADD HL,BC           save HL.
        LD C,L
        LD B,H
        POP DE
        POP HL
        ADD HL,DE
        PUSH DE
        LDIR                Do the actual reclaiming.
        POP HL
        RET
```

E-LINE-NO Subroutine

Reads the line number of the line in the editing area. If there is no line number, i.e. a direct BASIC line, then the line number is considered to be zero. In all cases the line number is returned in BC.

E-LINE-NO (LINNG)

```
M1768   LD HL,(ELINE)       Get pointer to the edit line.
        DEC HL              Set CH_ADD to the location before
        LD (CH_ADD),HL      any number.
        RST $20             Pass the first code to the A register.
        LD HL,MEMBOT        Before evaluating the code, make
        LD (STKEND),HL      the calculator memory a temporary
        CALL INT-TO-FP      calculator stack. Next, read the
        CALL FP-TO-BC       digits of the line number. Compress line number
                            into BC.
        JR C, E-L-1         Jump forward if number is >65536.
        LD HL,$D8F0         Test if number is >10000.
        ADD HL,BC
```

E-L-1

```
M1782   JP C ,REPORT-C      Report C if over 9999.
        JP SET-STK          Return via calculator stack reset.
```

Report And Line Number Printing Subroutine

OUT-NUM-1 will print the number in BC. Values over 9999 will not print correctly.

OUT-NUM-2 will print the number indirectly addressed by HL, with any necessary leading spaces. OUT-NUM-2 is also will not print numbers over 9999.

OUT-NUM-1 (PUT_BC)

```
M1788   PUSH DE            Save the other registers.
        PUSH HL
        XOR A               Clear A.
        BIT 7,B             Jump forward to print a zero rather than -2
        JR NZ,OUT-NUM-4     when reporting on the edit line.
        LD H,B              Move the number in to HL.
        LD L,C
        LD E,$FF            Flag 'no leading spaces'.
```

```
        JR OUT-NUM-3        Jump forward to print the number.

OUT-NUM-2 (PUT_LN)
M1795   PUSH DE             Save DE.
        LD D,(HL)           Put number in DE
        INC HL              and save the updated pointer.
        LD E,(HL)
        PUSH HL
        EX DE,HL            Move number to HL and flag
        LD E,$20            for printing leading spaces.

OUT-NUM-3
M179D   LD BC,$FC18         This is -1000.
        CALL OUT-SP-NO      Print first digit.
        LD BC,$FF9C         This is -100.
        CALL OUT-SP-NO      Print second digit.
        LD C,$F6            This is -10.
        CALL OUT-SP-NO      Print third digit.
        LD A,L              Move remaining part to A.

OUT-NUM-4
M17AF   CALL OUT-CODE       Print the digit.
        POP HL              Restore the registers.
        POP DE
        RET
```

Cartridge-Based BASIC Routines

One of the most unique features of the TS 2068 is the ability to run BASIC programs from a cartridge.

A number of computers (Atari, Commodore, Texas Instruments) could run machine-language programs from cartridges, like the TS 2068, but only the TI 99/4A and the TS 2068 could run BASIC code from a cartridge ROM.

There are limitations: FOR-NEXT loops can only count up, for instance. But the advantages are tremendous: the entirety of the TS 2068 RAM is available for program data. The cartridge version of Pro/File 2068, for instance, could store 10K more data than the cassette version.

These routines work by copying individual program lines to a special buffer in RAM, where they are then executed.

AROS-CART

```
M17B5   PUSH BC             Save BC to the stack. BC contains number of
                            spaces in RAM to create.
        LD BC,$FF00         Put bank # 255 in B.
        CALL BANK_ENABLE    Bank enable (in RAM).
        POP BC              Restore BC.
        CALL MAKE-ROOM      Insert BC spaces, starting at location in HL.
        LD HL,(SYSCON)
        LD DE,$0004         Skip to the memory chunk specification
        ADD HL,DE           for an AROS.
        LD A,(HL)           Get the memory chunk specification.
        LD B,$00            Specify DOCK bank.
        LD C,A
        CALL BANK_ENABLE    Enable the banks requested by the AROS.
        RET
```

GET-A-LINE (GETAL)

Find line number BC in the AROS cartridge. On exit, HL points to the first byte of the line number.

```
M17CF   LD HL,(SYSCON)      Get the address of the
        INC HL              code in the AROS
        INC HL              cartridge.
        LD E,(HL)           LSB of the address in E.
        INC HL              DE points to the code
        LD D,(HL)           in the cart.
        EX DE,HL            Move to HL.
```

A-FIND-LN

Search for the AROS line number.

```
M17D8   LD A,(HL)           Check if this is the line number,
        CP B                move on.
```

```
         JR NZ, A-LN-NT-FND
         INC HL              Bump the pointer,
         LD A,(HL)           get the next byte.
         DEC HL              Point back to where we were.
         CP C
```

A-LN-NT-FND

```
M17E0    RET NC              Return if we have passed the required line
                             number - that exact number does not exist, but
                             we are past it.
         INC HL              Get previous the line number.
         INC HL
         LD E,(HL)
         INC HL              DE now has the line length.
         LD D,(HL)           Point HL to the first byte of code in the
         INC HL              line, then update HL to
         ADD HL,DE           points to the start of the next line.
         JR A-FIND-LN        Back around for the next line.
```

AROS-LINE (AR_LN)

```
M17EA    PUSH HL             Save the line pointer.
         LD HL,(SYSCON)      Put location of SYSCON in HL.
         LD DE,$0004         Skip past first 4 bytes
         ADD HL,DE           to
         LD A,(HL)           get the chunk map.
         LD C,A              Pass to C.
         LD B,$00            Specify dock bank.
         CALL BANK_ENABLE    Enable the chunks.
         POP BC              Restore line pointer to BC.
         CALL GET-A-LINE     Find line number.
         JR AROS-NEXT-1
```

AROS-NEXT (AR_NXT)

```
M17FF    CALL SYNTAX-Z       Return if syntax checking, otherwise
         RET Z               fetch the next line address from DOCK.
         LD HL,(SYSCON)      Get the address of the SYSCON table.
         LD DE,$0004         Skip ahead to the chunk map.
         ADD HL,DE
         LD A,(HL)           Get the chunk map.
         LD C,A              Pass to C.
         LD B,$00            Set up for DOCK bank.
         CALL BANK_ENABLE    Switch on DOCK bank.
         LD HL,(NXTLIN)      Get address of the next line from it.
         LD (IY+ONSPPC),00   Reset NSPPC.
```

AROS-NEXT-1

```
M1818    LD A,(HL)           Get first byte of the line.
         AND $C0             Clear bits 0 to 6.
         JR Z, AROS-NEXT-2   If not EOL, jump ahead.
         LD BC,$FF00         Enable HOME bank, all chunks.
```

```
            CALL BANK_ENABLE
            RET
```

AROS-NEXT-2

```
M1824       LD D,(HL)               Get the high byte of the line address.
            INC HL                  Move to the low byte.
            LD E,(HL)               Get the low byte.
            LD (PPC),DE             Put it in the variable for the current line.
            INC HL                  Move to the next byte.
            LD E,(HL)
            INC HL
            LD D,(HL)
            INC HL
            PUSH HL
            ADD HL,DE
            LD (NXTLIN),HL          Set the address of the next line in the program.
            PUSH DE
            LD HL,(CHANS)           Get the address of CHANS.
            DEC HL                  Go one lower in memory.
            LD DE,(ARSBUF)          Get the address of the AROS line buffer.
            AND A                   Clear A.
            SBC HL,DE               Subtract ARSBUF from CHANS.
            LD DE,$00D0             Normal size of ARSBUF.
            EX DE,HL                Swap.
            AND A                   Clear A.
            SBC HL,DE               Subtract that to find out if ARSBUF is set.
            JR NC, AROS-NEXT-3      Jump if ARSBUF is available.
            LD A,L                  Put LSB in A.
            CPL                     Flip the bits.
            LD C,A                  Move A to C.
            LD A,H                  Put H in A.
            CPL                     Flip the bits.
            LD B,A                  Move A to B.
            INC BC                  Jump forward two bytes.
            INC BC
            LD HL,(ARSBUF)          Address of ARSBUF in HL.
            PUSH BC
            LD BC,$FF00             Set up to enable HOME bank, all chunks
            CALL BANK_ENABLE
            POP BC
            CALL RECLAIM-2          Clean up memory.
            LD HL,(SYSCON)          Point to the SYSCON table.
            LD DE,$0004             Offset to chunk map.
            ADD HL,DE               Add to get there.
            LD A,(HL)               Get the chunk map.
            LD B,$00                Set the DOCK bank.
            LD C,A                  Chunk map to C.
            CALL BANK_ENABLE        Enable the cartridge.
```

AROS-NEXT-3

```
M186E   POP HL              Get HL value off the stack.
        PUSH HL             Put it back.
        LD DE,$00CF         Size of ARSBUF - 1.
        DEC HL              And decrement HL.
        AND A               Clear A.
        SBC HL,DE           Subtract.
        JR C, AROS-NEXT-4   Carry is set, jump forward.
        LD C,L              Move L to C.
        LD B,H              Move H to B.
        INC BC              Increment to start of ARSBUF.
        LD HL,(CHANS)       Put address of CHANS in HL.
        DEC HL              Decrement it.
        CALL AROS-CART      Make some room.
```

AROS-NEXT-4

```
M1883   POP BC              Pop these values off the stack.
        POP DE
        LD HL,$00FF         And 255 in HL.
        PUSH HL             Put that on the stack.
        PUSH DE
        LD HL,(ARSBUF)      Get the address of ARSBUF.
        LD (HL),$0D         Put 13 at that location.
        LD (CH_ADD),HL      Move it to CH_ADD.
        INC HL
        PUSH HL
        PUSH BC
        LD BC,$0001
        PUSH BC
        CALL XFER_BYTES     Transfer bytes.
        LD A,(IY+ONSPPC)    Get ONSPPC.
        LD (IY+ONSPPC),$FF  Set ONSPPC to $FF.
        CP $01              Set the flags if we subtracted 1.
        ADC A,$00           Add, set the carry flag.
        DEC A               Count down one.
        PUSH AF             Put it on the stack.
        LD (SUBPPC),A       Set the SUBPPC to value of A.
        LD (IY+OERRNR),$FF  Set OERRNR to $FF.
        LD BC,$FF00         HOME bank, all chunks.
        CALL BANK_ENABLE    Enable.
        POP AF              Get A and flags back.
        JP Z, LS4           If zero is set, jump to LS4.
        INC A               Increment A.
        LD D,A              Move to D.
        LD E,$00            Set LSB to 0.
        CALL SUBLIN1        Get the sub-line.
        JP Z, STMT-NEXT     If zero, go to STMT-NEXT.
```

REPORT-N

```
        RST $08          Error: Statement lost.
        DEFB $16
```

AROS

Sets up the 2068 for working with an AROS cartridge.

```
M18C6   LD HL,ARSFLG     Get AROS flags address.
        LD (HL),$80      Set ARSFLG to AROS present.
        LD BC,$00D0      Set up how much space to set aside.
        LD HL,$6840      ARSBUF will be at $6840.
        DEC HL
        CALL MAKE-ROOM   Create space for the line.
        LD HL,$6840      Set the location of ARSBUF.
        LD (ARSBUF),HL
        LD HL,(SYSCON)   Get the address of the SYSCON table.
        LD DE,$0006      Offset to amount of RAM to reserve
        ADD HL,DE        for machine code variables.
        LD C,(HL)        And set
        INC HL           the value
        LD B,(HL)        for inserting
        LD HL,$6840      some more space.
        DEC HL
        CALL MAKE-ROOM   Create the space.
        LD HL,(SYSCON)   Get the address of SYSCON table again.
        LD DE,$0004      Set the offset for memory chunk specification.
        ADD HL,DE
        LD A,(HL)        Get the memory chunk spec.
        LD B,$00         Clear B.
        LD C,A           Set BC to memory chunk spec.
        CALL BANK_ENABLE And enable the DOCK bank(s).
        LD HL,(SYSCON)   Get the address of SYSCON table yet again.
        INC HL           Skip ahead two bytes to the starting address
        INC HL           of the AROS, address of the first program line.
        LD E,(HL)        Get the LSB of the address.
        INC HL           Move to the next byte.
        LD D,(HL)        Get the MSB of the address.
        EX DE,HL         Swap DE and HL.
        LD D,(HL)        Get the MSB of the first line.
        INC HL
        LD E,(HL)        Get the LSB of the first line.
        LD BC,$FF00
        CALL BANK_ENABLE Enable the HOME bank.
        LD HL,(SYSCON)   Get the address of SYSCON table, yet again.
        LD BC,$0005      Skip ahead to autostart specification.
        ADD HL,BC        Set address.
        LD A,(HL)        And get the autostart value.
        CP $00           Is it an autostart AROS?
```

```
            JR Z, AROS-END          Not autostart, so finish processing.
            LD (NEWPPC),DE          Set the line number to go to.
            CALL CLS                Clear the screen.
            LD HL,(SYSCON)          And get the pointer to SYSCON table.
            INC HL                  Skip ahead two bytes
            INC HL                  to the starting address.
            LD E,(HL)               Get the starting
            INC HL                  address.
            LD D,(HL)
            EX DE,HL                Swap and then
            DEC HL                  go back one.
            LD (DATADD),HL          Set the data list pointer.
            LD (IY+OERRNR),$FF      Reset ON ERR.
            SET INTPT,(IY+OFLAGS)   Set up FLAGS.
            LD (IY+ONSPPC),00       Set up NSPCC.
            LD HL,$0E8D             Address for MAIN-4.
            PUSH HL
            LD HL,$1AB9             Address for STMT-RET.
            EI                      Re-enable interrupts.
            JP (HL)                 Jump to STMT-RET.

AROS-END
M1941       EI                      Re-enable interrupts.
            JP MAIN-1               Back to the main execution loop.
```

BASIC Line and Command Interpretation

This part of the ROM considers a BASIC line as a set of statements. Each line is scanned and each command is executed in turn as it encountered.

Syntax Tables

OFFSET TABLE (KEWTBL)
Offset values for each of the BASIC commands. For example, the first command is DEF FN and its offset is B5. Add this to the start address to get M19FA, the start of the DEF FN code. This is class 05 (at M1B74) and then a jump to M201D, the DEF FN routine.

KEWTBL

```
M1945   DEFB $B5    DEF FN      M19FA
        DEFB $D0    CAT         M1A16
        DEFB $C0    FORMAT      M1A07
        DEFB $C4    MOVE        M1A0C
        DEFB $C8    ERASE       M1A11
        DEFB $B3    OPEN #      M19FD
        DEFB $B8    CLOSE #     M1A03
        DEFB $97    MERGE       M19E3
        DEFB $95    VERIFY      M19E2
        DEFB $96    BEEP        M19E4
        DEFB $99    CIRCLE      M19E8
        DEFB $9C    INK         M19EC
        DEFB $9C    PAPER       M19ED
        DEFB $9C    FLASH       M19EE
        DEFB $9C    BRIGHT      M19EF
        DEFB $9C    INVERSE     M19F0
        DEFB $9C    OVER        M19F1
        DEFB $9C    OUT         M19F2
        DEFB $83    LPRINT      M19DA
        DEFB $85    LLIST       M19DD
        DEFB $32    STOP        M198B
        DEFB $70    READ        M19CA
        DEFB $72    DATA        M19CD
        DEFB $74    RESTORE     M19D0
        DEFB $4C    NEW         M19A9
        DEFB $98    BORDER      M19F6
        DEFB $5C    CONT        M1989
        DEFB $43    DIM         M19A3
        DEFB $45    REM         M19A6
        DEFB $2F    FOR         M1991
        DEFB $1B    GOTO        M197E
        DEFB $23    GOSUB       M1987
        DEFB $3B    INPUT       M19A0
        DEFB $7B    LOAD        M19E1
```

```
        DEFB $48    LIST          M19AF
        DEFB $13    LET           M197B
        DEFB $5D    PAUSE         M19C6
        DEFB $2F    NEXT          M1999
        DEFB $47    POKE          M19B2
        DEFB $31    PRINT         M199D
        DEFB $55    PLOT          M19C2
        DEFB $3E    RUN           M19AC
        DEFB $71    SAVE          M19E0
        DEFB $46    RANDOMIZE     M19B6
        DEFB $11    IF            M1982
        DEFB $4D    CLS           M19BF
        DEFB $60    DRAW          M19D3
        DEFB $48    CLEAR         M19BC
        DEFB $19    RETURN        M198E
        DEFB $61    COPY          M19D7
```

KEWTBL2

```
M1977   DEFB $A4    DELETE        M1A1B
        DEFB $A6    ON ERR        M1A1E
        DEFB $A8    RESET         M1A21
        DEFB $AA    SOUND         M1A24
```

PARAMETER TABLE

For each of the BASIC commands there are up to eight entries in the parameter table. These entries comprise command class details, required separators and, where appropriate, command routine addresses.

```
M197B   P-LET       DEFB $01      CLASS-01
                    DEFB $3D      "="
                    DEFB $02      CLASS-02
        P-GOTO      DEFB $06      CLASS-06
                    DEFB $00      CLASS-00
                    DEFB $F1,$1E  GO-TO, M1EF1
        P-IF        DEFB $06      CLASS-06
                    DEFB CB       "THEN"
                    DEFB $05      CLASS-05
                    DEFB $5B,$1C  IF, M1C5B
        P-GOSUB     DEFB $06      CLASS-06
                    DEFB $00      CLASS-00
                    DEFB $99,$1F  GOSUB, M1F99
        P-STOP      DEFB $00      CLASS-00
                    DEFB $59,$1C  STOP, M1C99
        P-RETURN    DEFB $00      CLASS-00
                    DEFB $D4,$1F  RETURN, M1FD4
        P-FOR       DEFB $04      CLASS-04
                    DEFB $3D      "="
                    DEFB $06      CLASS-06
                    DEFB $CC      "TO"
                    DEFB $06      CLASS-06
```

```
                DEFB $05        CLASS-05
                DEFB $78,$1C    FOR, M1C78
P-NEXT          DEFB $04        CLASS-04
                DEFB $00        CLASS-00
                DEFB $55,$1D    NEXT, M1D55
P-PRINT         DEFB $05        CLASS-05
                DEFB $59,$21    PRINT, M2159
P-INPUT         DEFB $05        CLASS-05
                DEFB $2B,$22    INPUT, M222B
P-DIM           DEFB $05        CLASS-05
                DEFB $C0,$2F    DIM, M2FC0
P-REM           DEFB $05        CLASS-05
                DEFB $00,$1B    REM,M1B00
P-NEW           DEFB $00        CLASS-0
                DEFB $0D,$1D    NEW, M1D0D
P-RUN           DEFB $03        CLASS-03
                DEFB $2B,$1F    RUN, M1F2B
P-LIST          DEFB $05        CLASS-05
                DEFB $45,$15    LIST, M1545
P-POKE          DEFB $08        CLASS-08
                DEFB $00        CLASS-00
                DEFB $0A,$1F    POKE, M1F0A
P-RANDOMIZE
                DEFB $03        CLASS-03
                DEFB $D4,$1E    RANDOMIZE, M1ED4
P-CONTINUE      DEFB $00        CLASS-00
                DEFB $E4,$1E    CONTINUE, M1EE4
P-CLEAR         DEFB $03        CLASS-03
                DEFB $36,$1F    CLEAR, M1F36
P-CLS           DEFB $00        CLASS-00
                DEFB $A6,$08    CLS, M08A6
P-PLOT          DEFB $09        CLASS-09
                DEFB $00        CLASS-00
                DEFB $35,$26    PLOT, M2635
P-PAUSE         DEFB $06        CLASS-06
                DEFB $00        CLASS-00
                DEFB $EB,$1F    PAUSE, M1FEB
P-READ          DEFB $05        CLASS-05
                DEFB $97,$1D    READ, M1D79
P-DATA          DEFB $05        CLASS-05
                DEFB $82,$1E    DATA, M1E82
P-RESTORE       DEFB $03        CLASS-03
                DEFB $9D,$1E    RESTORE, M1E9D
P-DRAW          DEFB $09        CLASS-09
                DEFB $05        CLASS-05
                DEFB $DB,$26    DRAW, M26DB
P-COPY          DEFB $00        CLASS-00
                DEFB $02,$0A    COPY, M0A02
P-LPRINT        DEFB $05        CLASS-05
```

```
                DEFB $55,$21    LPRINT, M2155
P-LLIST         DEFB $05        CLASS-05
                DEFB $41,$15    LLIST, M1541
P-SAVE          DEFB $0B        CLASS-0B
P-LOAD          DEFB $0B        CLASS-0B
P-VERIFY        DEFB $0B        CLASS-0B
P-MERGE         DEFB $0B        CLASS-0B
P-BEEP          DEFB $08        CLASS-08
                DEFB $00        CLASS-00
                DEFB $36,$04    BEEP, M0436
P-CIRCLE        DEFB $09        CLASS-09
                DEFB $05        CLASS-05
                DEFB $79,$26    CIRCLE, M2679
P-INK           DEFB $07        CLASS-07
P-PAPER         DEFB $07        CLASS-07
P-FLASH         DEFB $07        CLASS-07
P-BRIGHT        DEFB $07        CLASS-07
P-INVERT        DEFB $07        CLASS-07
P-OVER          DEFB $07        CLASS-07
P-OUT           DEFB $08        CLASS-08
                DEFB $00        CLASS-00
                DEFB $04,$1F    OUT, M1F04
P-BORDER        DEFB $06        CLASS-06
                DEFB $00        CLASS-00
                DEFB $3E,$24    BORDER, M243E
P-DEF FN        DEFB $05        CLASS-05
                DEFB $1D,$20    DEF FN, M201D
P-OPEN          DEFB $06        CLASS-06
                DEFB $2C        ','
                DEFB $0A        CLASS-0A
                DEFB $05        CLASS-05
                DEFB $2A,$14    OPEN, M142A
P-CLOSE         DEFB $06        CLASS-06
                DEFB $00        CLASS-00
                DEFB $9F,$13    CLOSE, M139F
P-FORMAT        DEFB $0A        CLASS-0A
                DEFB $2C        ','
                DEFB $05        CLASS-05
                DEFB $CC,$25    FORMAT, M25CC
P-MOVE          DEFB $0A        CLASS-0A
                DEFB $2C        ','
                DEFB $05        CLASS-05
                DEFB $D0,$25    MOVE, M25D0
P-ERASE         DEFB $0A        CLASS-0A
                DEFB $2C        ','
                DEFB $05        CLASS-05
                DEFB $D4,$25    ERASE, M25D4
P-CAT           DEFB $0A        CLASS-0A
                DEFB $2C        ','
```

```
                DEFB $05        CLASS-05
                DEFB $C8,$25    CAT, M25C8
P-DEL           DEFB $05        CLASS-05
                DEFB $D1,$20    DELETE, M20D1
P-ON ERR        DEFB $05        CLASS-05
                DEFB $80,$20    ONERR, M2080
P-RESET         DEFB $05        CLASS-05
                DEFB $54,$24    RESET, M2454
P-SOUND         DEFB $05        CLASS-05
                DEFB $28,$21    SOUND, M2128
```

Note: The requirements for the different command classes are as follows:

CLASS-00: No further operands.
CLASS-01: Used in LET. A variable is required.
CLASS-02: Used in LET. An expression, numeric or string, must follow.
CLASS-03: A numeric expression may follow. Zero to be used in case of default.
CLASS-04: A single character variable must follow.
CLASS-05: A set of items may be given.
CLASS-06: A numeric expression must follow.
CLASS-07: Handles color items.
CLASS-08: Two numeric expressions, separated by a comma, must follow.
CLASS-09: As for CLASS-08 but color items may precede the expressions.
CLASS-0A: A string expression must follow.
CLASS-0B: Handles cassette routines.

Main Parser Of BASIC Interpreter Syntax

The parsing routine of the BASIC interpreter is entered at LINE-SCAN when syntax is being checked and at LINE-RUN when a BASIC program of one or more statements is to be executed. Each statement is considered in turn and the system variable CH_ADD is used to point to each code of the statement as it occurs in the program area or the editing area.

LINE-SCAN

```
M1A27   RES INTPT,(IY+OFLAGS)   Signal that syntax checking is on.
        CALL E-LINE-NO          Make CH_ADD point to first code after line
                                number.
        LD A,B                  Put B in A to test if
        OR C                    BC = 0?
        JR Z, LINE-SCAN-1       If line number is 0, skip check for cartridge.
```

CART-CHECK

```
M1A32   LD A, (ARSFLG)          Get AROS flags.
        BIT AROS,A              Is an AROS cartridge present?
        JP NZ, REPORT-C         Report error: BAD BASIC. Should not be syntax
                                checking programs running from a cartridge.
```

LINE-SCAN-1

```
M1A3A   XOR A               Initialize A to 0.
        LD (SUBPPC),A       Set SUBPPC to 0.
        DEC A               Set A to FF.
        LD (ERR_NR),A       Set ERR_NR to FF.
        JR STMT-L-1         Jump forward to consider first statement in the
                            line.
```

STMT-LOOP (LS4)

Each statement is evaluated until the end of the line is reached.

```
M1A44   RST $20             Advance CH_ADD along the line.
```

STMT-L-1

```
M1A45   CALL SET-WORK       Clear the work space.
        INC (IY+OSUBPPC)    Bump the sub-statement and jump if there are
                            more than 127 in the line.
        JP M, REPORT-C      Report error: BAD BASIC.
        RST $18             Get current character.
        LD B,$00            Set up B for later.
        CP $0D              Is character a carriage return?
        JP Z, LINE-END      Jump to LINE-END if yes.
        CP ':'              Is this a colon?
        JR Z, STMT-LOOP     Go back around to start of loop.
```

Found a statement, do some tests before evaluating it.

```
        LD HL,$1AB9         Put return address of STMT-RET on stack.
        PUSH HL
        LD C,A              Save command temporarily in C.
        RST $20             Advance CH_ADD.
        LD A,C              Restore last command.
        CP $0C              Jump if token is DELETE.
        JR Z, TKN-IS-DEL
        CP $7B              Jump if token is ON ERR.
        JR C, GET-TOKEN-OFFSET
        CP $80              Is it a graphic character?
        JR NC, GET-TOKEN-OFFSET
        BIT 0,A             Jump if token is for keywords
        JR NZ,TKN-IS-DEL    ONERR, SOUND, RESET.
```

GET-TOKEN-OFFSET

```
M1A71   SUB $CE             Reduce the command code to the range of $0 -
                            $31.
        JP C, REPORT-C      If this is not a command code, do error - BAD
                            BASIC.
        LD C,A              Move the offset value into BC.
        LD HL, KEWTBL       Set HL to base address of the keyword syntax
                            offset table.
```

FIND-OFFSET-ADDR

```
M1A7A   ADD HL,BC          Pass required offset.
        LD C,(HL)          Compute the base address of the
        ADD HL,BC          parameter table.
        JR GET-PARAM
```

TKN-IS-DEL

```
M1A7F   CP $0C             Jump if token is not DELETE.
        JR NZ,TKN-IS-OTHER
        LD A,$00           Force A to zero and bypass
        JR CHK-KEWTBL2     next test.
```

TKN-IS-OTHER

```
M1A87   SUB $7A            A=1 for ON ERR, 3 for SOUND, 5 for RESET.
        CP $05             Jump if token is not RESET.
        JR NZ, CHK-KEWTBL2
        LD A,$02           Force A (RESET) to 2.
```

CHK-KEWTBL2

```
M1A8F   LD HL, KEWTBL2     Point to second half of keyword table.
        LD C,A             Put the massaged token in C.
        JR FIND-OFFSET-ADDR Look up in the table.
```

SCAN-LOOP

```
M1A95   LD HL, (TADDR)     The class handler returns here from the address
                           pushed below.
```

GET-PARAM

```
M1A98   LD A,(HL)          Get the first (or next) byte of the syntax table - the
                           instruction's class (first byte) or the required token
                           or class (succeeding bytes).
        INC HL             Point to the next byte in the syntax table.
        LD (TADDR),HL      Store its address in TADDR.
        LD BC, SCAN-LOOP   Return address from class handler.
        PUSH BC
        LD C,A             Save the “class".
        CP $20             Jump if the "class" is not in the range $00..$11.
        JR NC, SEPARATOR   This means it could be the next value in the
                           syntax table.
        LD HL, CLASTBL     Put base address of command class table in HL.
        LD B,$00           Clear B.
        ADD HL,BC          Combine to make index into the table.
        LD C,(HL)          Get offset from the table
        ADD HL,BC          and form address of the actual "class"
                           handler ...
        PUSH HL            ... and push it onto the stack.
        RST $18            Get current character.
        DEC B              B=$FF
        RET                Indirect jump to the command class routine.
```

Separator Subroutine

The report 'Nonsense in BASIC' is given if the required separator is not present. When syntax is being checked, the actual report does not appear on the screen, only the 'error marker'.

SEPARATOR

```
M1AB2   RST $18          Get current character.
        CP C             Check the current character in A against the value
                         from the syntax table. Declare an error if they
                         don't match - the syntax is wrong.
        JP NZ,REPORT-C   Error - BAD BASIC
        RST $20          Step past a correct character
        RET              and return.
```

STMT-RET Subroutine

After the correct interpretation of a statement, a return is made to this entry point.

STMT-RET (ENDBTT)

```
M1AB9   CALL BREAK-KEY   BREAK key is tested after every statement.
        JR C, STMT-R-1   Jump forward if not pressed.
```

REPORT-L

```
        RST $08          Error: Break.
        DEFB $14
```

STMT-R-1

```
M1AC0   BIT 7,(IY+ONSPPC)   Jump forward if there is not a
        JP NZ, STMT-NEXT    "jump" to be made.
        LD HL,(NEWPPC)      Get the new line number and jump
        BIT 7,H             forward unless handling another statement
        JR NZ,LINE-RUN      in the editing area.
        LD A,(ARSFLG)       Put the AROS flags in A.
        BIT 7,A             Is this program running in AROS?
        JP NZ, AROS-LINE    Yes, get the next line from AROS.
        JR LINE-NEW         Nope, continue as normal.
```

LINE-RUN Entry Point

Used when a line in the editing area is to be 'run'. The syntax/run flag (bit 7 of FLAGS) will be set. Also used in the syntax checking a line in the editing area that has more than one statement (bit 7 of FLAGS will be reset).

LINE-RUN (EXECUTE)

```
M1AD8   LD HL,$FFFE      Line in the editing area is considered as
        LD (PPC),HL      line number -2.
        LD HL,(WORKSP)   Make HL point to the end marker of the
        DEC HL           editing area and DE point to the location
        LD DE,(ELINE)    before the start of that area.
        DEC DE
        LD A,(NSPPC)     Fetch the number of the next statement.
```

```
            JR NEXT-LINE
```

LINE-NEW

```
M1AEC   CALL LINE-ADDR      Find the starting address of the line or
        LD A,(NSPPC)        first line after and move to A.
        JR Z, LINE-USE      Jump forward if the line was found,
        AND A               otherwise, check validity of statement number.
        JR NZ, REPORT-N     If it's not zero, error to Report N.
        LD B,A              And check that the 'first line after' is not
        LD A,(HL)           after the end of program.
        AND $C0
        LD A,B
        JR Z, LINE-USE      Jump forward with valid address or signal done
                            with 0, OK.
```

REPORT-0

```
        RST $08             Report: 0, OK.
        RST $38             FF
```

REM Command Routine

Return address to STMT-RET is dropped which has the effect of forcing the rest of the line to be ignored.

```
M1B00   POP BC              Drop the address.
        LD A,(ARSFLG)       Put the AROS flags in A.
        BIT AROS,A          Is this program running in AROS?
        JP NZ, AROS-NEXT    Yes, handle with AROS version of LINE-END.
```

LINE-END Routine

If checking syntax a simple return is made but when 'running' the address held by NXTLIN has to be checked before it can be used.

LINE-END

```
M1B09   CALL SYNTAX-Z       Return if syntax checking.
        RET Z
        LD HL,(NXTLIN)      Otherwise, get the address of next line.
        LD A,$C0            Return if the address is after
        AND (HL)            the end of the program:
        RET NZ              the 'run' has finished.
        XOR A               Signal 'statement zero' before proceeding.
```

LINE-USE Routine

This short routine has three functions: change statement zero to statement '1'; find the number of the new line and enter it into PPC; form the address of the start of the line after.

```
LINE-USE
M1B15   CP $01              If A is zero, make it 1.
        ADC A,$00
        LD D,(HL)
        INC HL              Put the new line number into the
        LD E,(HL)           "line currently being interpreted"
        LD (PPC),DE         pointer.
        INC HL
        LD E,(HL)           Get the line length.
        INC HL
        LD D,(HL)
        EX DE,HL            Point to the next line and put the pointer
        ADD HL,DE           into the "next line to be interpreted"
        INC HL              pointer.
```

NEXT-LINE Routine

On entry, HL points to the location after the end of the 'next' line to be handled and the DE register pair to the location before the first character of the line. This applies to lines in the program area and also to a line in the editing area - where the next line will be the same line again whilst there are still statements to be interpreted.

```
NEXT-LINE
M1B27   LD (NXTLIN),HL      Set NXTLIN for use once the current line has been
                            completed.
        EX DE,HL            Point CH_ADD to the location before
        LD (CH_ADD),HL      the first character.
        LD D,A              Save statement number in D.
        LD E,$00            Clear E register in case EACH-STMT is used.
        LD (IY+ONSPPC),$FF  Signal 'no jump'.
        DEC D               Statement number minus one
        LD (IY+OSUBPPC),D   goes in SUBPPC.
        JP Z, STMT-LOOP     First statement can be evaluated.
        INC D               For later statements, the starting address
        CALL EACH-STMT      has to be found.
        JR Z, STMT-NEXT     Jump forward unless statement does not exist.
```

REPORT-N

```
M1B42   RST $08          Error: Statement lost.
        DEFB $16
```

CHECK-END Routine

Checks the interpret flag (bit 7 of FLAGS) and gives an error report if the end of a statement has not been reached. Moves on to STMT-NEXT if the syntax is correct.

CHECK-END (ENDQ)

```
M1B44   CALL SYNTAX-Z    Do not continue unless checking
        RET NZ           syntax.
        POP BC           Drop addresses of SCAN-LOOP and
        POP BC           STMT-RET before continuing.
```

STMT-NEXT Routine

If the next character is a carriage return, the next statement is on the next line. If it's a colon, it's on the same line. If any other character is found, there's an error in syntax.

STMT-NEXT (ENDTEM)

```
M1B4A   RST $18          Get current character.
        CP $0D           Jump if not at the end of a line.
        JR NZ, SUB-LINE
        LD HL,(NXTLIN)   Get the pointer to the next BASIC line.
        LD A,(ARSFLG)    Put AROS flags in A.
        BIT AROS,A       Is this code being run from the DOCK?
        JP NZ, AROS-NEXT Handle in AROS section.
        JR LINE-END      Continue normal line execution.
```

SUB-LINE

Test for line end.

```
M1B5C   CP ':'           Is this a colon?
        JP Z, STMT-LOOP  No, keep executing statements.
        JP REPORT-C      Error - BAD BASIC.
```

COMMAND CLASS TABLE (CLASTBL)

```
M1B64   DEFB $0F   CLASS-00   M1B73
        DEFB $1D   CLASS-01   M1B82
        DEFB $4B   CLASS-02   M1BB1
        DEFB $D9   CLASS-03   M1B70
        DEFB $67   CLASS-04   M1BCF
        DEFB $0B   CLASS-05   M1B74
        DEFB $7B   CLASS-06   M1BE5
        DEFB $8E   CLASS-07   M1BF9
        DEFB $71   CLASS-08   M1BDD
        DEFB $BC   CLASS-09   M1C29
        DEFB $81   CLASS-0A   M1BEF
        DEFB $D7   CLASS-0B   M1C46
```

Command Classes 00, 03 and 05

Commands in CLASS-03 might be followed by a number. e.g. RUN or RUN 200.

CLASS-03 (TEM3)

```
M1B70   CALL FETCH-NUM          See if there's a number. Zero is provided if none.
```

CLASS-00 (TEM0)

CLASS-00 commands have no operands. Ex: COPY or CONTINUE.

```
M1B73   CP A                    Set the zero flag for later.
```

CLASS 05 (TEM5)

CLASS-05 commands may be followed by a set of items. Ex: PRINT or PRINT "222".

```
M1B74   POP BC                  In all cases drop the address SCAN-LOOP.
        CALL Z ,CHECK-END       If handling classes 0 or 3 and syntax is being
                                checked, move on to next statement.
        EX DE,HL                Save the line pointer to DE.
```

JUMP-C-R Routine

After the command class entries and the separator entries in the parameter table have been processed, jump to the appropriate command routine.

JUMP-C-R

```
M1B79   LD HL,(TADDR)           Get the point to the
        LD C,(HL)               entries in the parameter table
        INC HL                  and fetch the address of the
        LD B,(HL)               command routine.
        EX DE,HL                Exchange pointers back and
        PUSH BC                 make an indirect jump to the
        RET                     command routine.
```

Command Classes 01, 02 and 04

These three command classes are used by the variable handling commands (LET, FOR & NEXT) and indirectly by READ & INPUT. Command CLASS-01 is concerned with the identification of the variable in a LET, READ or INPUT statement.

CLASS-01 (TEM1)

```
M1B82   CALL LOOK-VARS
```

Variable in Assignment Subroutine

This subroutine gets the appropriate values for the system variables DEST & STRLEN.

VAR-A-1

```
M1B85   LD (IY+OFLAGX),00       Initialize FLAGX to 0.
        JR NC, VAR-A-2          Jump forward if variable has been used before.
        SET FLEX,(IY+OFLAGX)    Signal 'flexible length variable.'
```

```
              JR NZ, VAR-A-3          Report error if trying to used undimensioned
                                      array.
```

REPORT-2

```
M1B91         RST $08                 Error: Variable not found.
              DEFB $01
```

VAR-A-2

```
M1B93         CALL Z,STK-VAR          Pass parameters of string and array variables to
                                      the calculator stack.
              BIT NUM,(IY+OFLAGS)     Jump forward if this is a numeric variable.
              JR NZ, VAR-A-3
              XOR A                   Clear A.
              CALL SYNTAX-Z           Get parameters of string or string array
              CALL NZ,STK-FETCH       unless syntax is being checked.
              LD HL,FLAGX             Put FLAGX in HL.
              OR (HL)                 Set bit 0 when handling complete
              LD (HL),A               simple strings, signaling 'deleted old copy.'
              EX DE,HL                HL points to string or element in array.
```

VAR-A-3

The data now come together to set STRLEN & DEST as required. For all numeric variables and new string & string array variables, STRLEN-lo holds the letter of the variable's name. But for old string & string array variables whether sliced or complete, it holds the length in assignment.

VAR-A-3

```
M1BA9         LD (STRLEN),BC          Set STRLEN.
              LD (DEST),HL            DEST holds address for the destination of
                                      the old variable but is also source for new
                                      variable.
              RET
```

CLASS-02 (TEM2)

Class 02 calculates the value to be assigned in a LET statement.

```
M1BB1         POP BC                  Drop SCAN-LOOP address from stack.
              CALL VAL-FET-1          Assign the value.
              CALL CHECK-END          Move on to next statement or STMT-RET.
              RET
```

Fetch a Value Subroutine

This subroutine is used by LET, READ and INPUT statements to first evaluate and then assign values to the previously designated variable. The entry point VAL-FET-1 is used by LET & READ and considers FLAGS whereas the entry point VAL-FET-2 is used by INPUT and considers FLAGX.

VAL-FET-1

```
M1BB9         LD A,(FLAGS)            Get FLAGS.
```

VAL-FET-2 (LT22)

```
M1BBC   PUSH AF              Save FLAGS.
        CALL SCANNING        Evaluate the next expression.
        POP AF               Get FLAGS back.
        LD D,(IY+OFLAGS)     Get the new FLAGS.
        XOR D                Determine if the type (numeric or string)
        AND $40              of variable and expression match.
        JR NZ,REPORT-C       Error - BAD BASIC.
        BIT INTPT,D          Jump forward to make assignment unless
        JP NZ, LET           checking syntax, then return.
        RET
```

Command Class 04 Routine

Class 04 is used by FOR & NEXT statements.

CLASS-04 (TEM4)

```
M1BCF   CALL LOOK-VARS       Look in variables area for the variable.
        PUSH AF              Save AF.
        LD A,C               Test to see if this variable is a FOR-NEXT
        OR $9F               control variable.
        INC A                If not, report
        JR NZ,REPORT-C       Error - BAD BASIC
        POP AF               Restore AF.
        JR VAR-A-1
```

Expect Numeric/String Expressions Subroutine

Series of short subroutines that fetch the result of evaluating the next expression. The result from a single expression is returned as a 'last value' on the calculator stack. NEXT-2NUM is used when CH_ADD needs updating to point to the start of the first expression.

NEXT-2NUM (DYADIC)

```
M1BDC   RST $20              Move CH_ADD forward.
```

EXPT-2NUM (TEM8/CLASS-08)

```
M1BCF   CALL EXPT-1NUM       Evaluate the next expression,
        CP ','               jump if another argument is
        JR NZ,REPORT-C       not available.
        RST $20              Get next char, fall through to
                             evaluate the next expression.
```

EXPT-1NUM (TEM6, CLASS-06)

```
M1BE5   CALL SCANNING        Evaluate the next expression.
        BIT NUM,(IY+OFLAGS)  Return if the expression was a number.
        RET NZ
```

REPORT-C

```
M1BED   RST $08              Error: Nonsense in basic.
        DEFB $0B
```

EXPT-EXP (TEM10, CLASS-0A)

```
M1BEF   CALL SCANNING          Evaluate the next expression.
        BIT NUM,(IY+OFLAGS)    Return if the last expression
        RET Z                  was a string (not a number).
        JR REPORT-C            Error - BAD BASIC.
```

Set Permanent Colors Subroutine (CLASS-07)

Make the temporary colors permanent. As command class 07, it is the command routine for INK, PAPER, FLASH, BRIGHT, INVERSE and OVER.

PERMS (TEM7)

```
M1BF9   BIT INTPT,(IY+OFLAGS)  Check the syntax/run flag.
        RES LHS,(IY+OTVFLAG)   Signal 'upper screen'.
        CALL NZ,TEMPS          If this is a 'run', call TEMPS to ensure temporary
                               colors are the main screen colors.
        POP AF                 Drop the return address (SCAN-LOOP).
        LD A,(TADDR)           Put the low byte of TADDR in A.
        LD HL,(TADDR)          Put value of TADDR in HL.
        LD DE,$1914            This is some kind of offset constant?
        AND A                  Clear flags.
        SBC HL,DE              Get the difference of (TADDR) - $1914.
        LD A,L                 Put the low byte from TADDR in L.
        CALL CO-TEMP-4         Jump forward to change temporary colors as
                               directed by BASIC statement.
        CALL CHECK-END         Move to the next statement if checking syntax.
        LD HL,(ATTRT)          Make temporary color values permanent.
        LD (ATTRP),HL
        LD HL,PFLAG            Check the P-FLAG.
        LD A,(HL)
```

The following instructions make the permanent bits the same as the temporary by copying event bits of the supplied by to the odd bits.

```
        RLCA                   Move the mask to the left.
        XOR (HL)               Set up the mask
        AND $AA                so it considers only even bits
        XOR (HL)               of the other byte.
        LD (HL),A
        RET
```

Command Class 09 Routine

Used by PLOT, DRAW and CIRCLE to specify default conditions of 'FLASH 8; BRIGHT 8; PAPER 8'. Set up before any embedded color items are considered.

CLASS-09 (TEM9)

```
M1C29   CALL SYNTAX-Z          Jump forward if checking syntax.
        JR Z, CL-09-1
        RES LHS,(IY+OTVFLAG)   Signal 'upper screen.'
        CALL TEMPS             Set the temporary color for the upper screen.
```

```
              LD HL,MASKT              Put MASKT in HL.
              LD A,(HL)                Get the current value but
              OR $F8                   keep on the INK part (via a mask).
              LD (HL),A                Restore value which is now FLASH 8; BRIGHT 8;
                                       PAPER 8.
              RES B_CF,(IY+OPFLAG)     Ensure setting is not PAPER 9.
              RST $18                  Get current character before going to deal
                                       with embedded color items.
```

CL-09-1 (STK_O)

```
M1C41         CALL CO-TEMP-2           Handle the local color items.
              JR EXPT-2NUM             Get the next two operands for PLOT, DRAW or
                                       CIRCLE.
```

Command Class 0B Routine

Routine used by SAVE, LOAD, VERIFY and MERGE statements.

CLASS-0B (TEM11)

```
M1C46         JP NEWDEV
```

Fetch A Number Subroutine

Gets a number for evaluation. Put a zero on FP stack if at the end of a line or statement or if there is no expression.

FETCH-NUM (OPTNO)

```
M1C49         CP $0D                   Jump if the end of line.
              JR Z, USE-ZERO
              CP ':'                   Jump to evaluate a numeric expression
              JR NZ,EXPT-1NUM          if not the end of statement.
```

USE-ZERO (STKFP0)

```
M1C51         CALL SYNTAX-Z            Return if checking syntax.
              RET Z
              RST $28                  Otherwise, put a zero on the calculator stack.
              DEFB $A0                 CONST 0 (0)
              DEFB $38                 QUIT
              RET
```

STOP Command Routine

STOP is a call to the error handling routine.

STOP (TSSTOP)

```
M1C59         RST $08                  Error: Stop.
              DEFB $08
```

IF Command Routine

On entry the value of the expression between the IF and the THEN is the last value on the calculator stack. If this is logically true then the next statement is considered; otherwise the line is considered to have been finished.

IF (TSIF)

```
M1C5B   POP BC              Drop the current return address (STMT-RET).
        CALL SYNTAX-Z       Jump forward if syntax checking.
        JR Z, IF-1
        RST $28             Use the calculator to delete the last value
        DEFB $02            (DROP) on the calculator stack but leave DE.
        DEFB $38            (QUIT) addressing first byte of the value.
        EX DE,HL            Make HL point to the first byte and
        CALL TEST-ZERO      call TEST-ZERO.
        JR NC, IF-1         Not zero, so jump forward.
        LD A, (ARSFLG)      Put ARSFLG in A.
        BIT AROS,A          Is there an AROS present?
        JP NZ, AROS-NEXT    Yes, go handle in AROS section.
        JP LINE-END         It was a zero, jump to the next line.
```

IF-1

```
M1C75   JP STMT-L-1         Jump to the next statement (after THEN).
```

FOR Command Routine

Entered with the VALUE and the LIMIT of the FOR statement on the top of the calculator stack.

FOR

```
M1C78   CP $CD              Jump forward unless a STEP value is provided.
        JR NZ, F-USE-1
        RST $20             Advance CH_ADD and
        CALL EXPT-1NUM      get the value for STEP.
        CALL CHECK-END      Move to next statement if checking syntax,
        JR F-REORDER        else jump forward.
```

F-USE-1

```
M1C85   CALL CHECK-END      Move to the next statement if checking syntax.
        RST $28             Set the unspecified STEP value to 1.
        DEFB $A1            CONST 1 (256)
        DEFB $38            QUIT
```

F-REORDER

The three values on the calculator stack are the VALUE (v), the LIMIT (l) and the STEP (s). They need to be re-ordered.

```
M1C8B   RST $28             Call calculator. Values are v, l, s
        DEFB $C0            Put s in mem-0
        DEFB $02            Drop s
        DEFB $01            Swap l, v
```

```
        DEFB $E0            Recall mem-0 (s): l, v, s
        DEFB $01            Exchange to l, s, v.
        DEFB $38            QUIT
        CALL LET            Find the variable or create if necessary.
        LD (MEM),HL         Make it a "memory" area.
```

The variable that has been found may be a simple numeric variable using only six locations. If so, it needs to be extended.

```
        DEC HL              Get the variable's single character
        LD A,(HL)           name.
        SET 7,(HL)          Check whether bit 7 is set.
        LD BC,$0006         It will have six locations at least.
        ADD HL,BC           Make HL point after them,
        RLCA                Rotate the name and jump if
        JR C, F-L&S         it was already a FOR variable.
        LD C,$0D            Otherwise, expand to 13 characters.
        CALL MAKE-ROOM
        INC HL              Make HL point to the LIMIT.
```

F-L&S

```
M1CA9   PUSH HL             Save the pointer.
        RST $28             Call calculator with l, s still on calc stack,
        DEFB $02            Drop l.
        DEFB $02            Drop s.
        DEFB $38            Quit. DE still points to 'l'.
        POP HL              Restore the pointer
        EX DE,HL            and exchange them.
        LD C,$0A            Ten bytes of the LIMIT and STEP are
        LDIR                moved.
        LD HL,(PPC)         Get the current line number.
        EX DE,HL            Exchange the registers before
        LD (HL),E           adding the line number to the
        INC HL              FOR control variable.
        LD (HL),D
        LD D,(IY+OSUBPPC)   Looping statement is always the next
        INC D               statement, whether it exists
        INC HL              or not.
        LD (HL),D
```

Call NEXT-LOOP to test the possibility of a 'pass' and a return is made if one is possible. Otherwise the statement after for FOR - NEXT loop has to be identified.

```
        CALL NEXT-LOOP      Is a 'pass' possible?
        RET NC              Return now if it is.
        LD HL,(PPC)         Copy the current line number
        LD (NEWPPC),HL      to NEWPCC.
        LD A,(SUBPPC)       Get the current statement number
        NEG                 and twos complement it.
        LD D,A              Transfer result to D.
```

```
        LD HL,(SYSCON)          Put location of SYSCON in HL
        INC HL                  Add one to get to cartridge type.
        LD A,(HL)               Get the kind of cartridge.
        CP $02                  Is it an AROS?
        JR NZ, F-L&S-1          No, keep processing the FOR loop.
        INC HL                  Go past cartridge type.
        INC HL                  Skip starting address (two bytes).
        INC HL
        LD A,(HL)               Get the chunk map.
        AND $0F                 Accept only the lower half of the chunk.
        LD C,A                  Move to C.
        LD B,$00                Specify dock bank.
        CALL BANK_ENABLE        Switch the bank on.
        LD BC, (PPC)            Put the current line number in BC.
        CALL GET-A-LINE         Go fetch the line.
        LD H,B                  And put that line number in HL.
        LD L,C
        DEC HL                  Go back one.
        JR F-L&S-2              Continue processing the FOR loop.
```

F-L&S-1

```
M1CF2   LD HL,(CH_ADD)
```

F-L&S-2

```
M1CF5   LD E,$F3                Search for NEXT.
```

F-LOOP

Search in the program area, from this point forward, for NEXT followed by the correct variable.

```
M1CF7   LD BC,(NXTLIN)          Get the current value of NXTLIN.
        CALL LOOK-PROG          Search the program area. BC will change as each
                                new line is examined.
        LD (NXTLIN),BC          Save the pointer.
        LD B,(IY+OSTRLEN)       Put STRLEN in B.
        JR C, REPORT-I          If Carry is set, REPORT I.
        RST $20                 Go past the NEXT that was found.
        OR $20                  Allow for upper and lower case letters
        CP B                    before testing the variable name.
        JR Z, F-FOUND           Variable name matched, jump ahead.
        RST $20                 Not the correct variable name, so
        JR F-LOOP               keep looking.
```

F-FOUND

```
M1D10   RST $20                 Advance CH_ADD.
        LD A,$01                The statement counter in D counted statements
        SUB D                   back from zero so it has to be subtracted from 1.
        LD (NSPPC),A            Store the result.
        LD HL,ARSFLG            Put the AROS flag in HL.
        LD L,(HL)               Only interested in L.
        BIT AROS,L              Is there an AROS present?
```

```
              JR Z, F-FOUND-RET       Yes, jump ahead to return.
              LD BC,$FF00             Enable home bank, all chunks.
              CALL BANK_ENABLE
```

F-FOUND-RET

```
M1D25  RET                            Return to STMT-RET.
```

REPORT-I

```
M1D26  RST $08                        Error: FOR without NEXT.
       DEFB $11
```

LOOK-PROG Subroutine

Finds occurrences of DATA, DEF FN or NEXT. On entry, the appropriate token code is in the E register and the HL register pair points to the start of the search area.

LOOK-PROG (SKIP)

```
M1D28  LD A,(HL)                      Get the current character.
       CP $3A                         Jump forward if it is a colon; there are
       JR Z, LOOK-P-2                 more statements on this line.
```

LOOK-P-1

Loop over and examine each line after "FOR" line.

```
M1D2D  INC HL                         Get the high byte of the line number
       LD A,(HL)                      and return with carry set if there are
       AND $C0                        no more lines in the program.
       SCF
       RET NZ
       LD A,E                         Put the token in A.
       CP $E4                         Is it 'DATA'?
       JR NZ, LOOK-P-1A               No, keep looking through the line.
       LD (ADATLN),HL                 Put HL in the pointer to the start of the current
                                      DATA line in AROS.
```

LOOK-P-1A

```
M1D3B  LD B,(HL)                      Get the line number
       INC HL                         and pass it to NEWPPC.
       LD C,(HL)
       LD (NEWPPC),BC
       INC HL                         Then get the length of the line.
       LD C,(HL)
       INC HL
       LD B,(HL)
       PUSH HL                        Pointer is saved while the address at the
       ADD HL,BC                      end of the line is combined with BC.
       LD B,H
       LD C,L
       POP HL                         Pointer is restored.
       LD D,$00                       Set statement counter to zero.
```

LOOK-P-2

```
M1D4D   PUSH BC             End-of-line pointer is saved while
        CALL EACH-STMT      statements on the line are examined.
        POP BC              Return if there was a match; otherwise
        RET NC              continue to the next line.
        JR LOOK-P-1
```

NEXT Command Routine

The variable in assignment has already been determined (see CLASS-04, M1BCF); and it remains to change the VALUE as required.

NEXT

```
M1D55   BIT UNFND, (IY+OFLAGX)  Jump to error if the variable was not
        JP NZ, REPORT-2         found.
        LD HL,(DEST)            Get the address of the variable
        BIT 7,(HL)              and test the name.
        JR Z, REPORT-1          Error if it does not match the FOR variable.
```

Manipulate the VALUE and STEP with the calculator.

```
        INC HL              Step past the variable name.
        LD (MEM),HL         Put the variable in a calculator memory area.
        RST $28             Call calculator.
        DEFB $E0            MEM 0 -> T (v)
        DEFB $E2            MEM 2 -> T (v, s)
        DEFB $0F            ADD  (v+s)
        DEFB $C0            T -> MEM 0 (v+s)
        DEFB $02            DROP
        DEFB $38            QUIT
```

Result of adding VALUE and STEP is now tested against LIMIT by calling NEXT-LOOP.

```
        CALL NEXT-LOOP      Test new VALUE against LIMIT.
        RET C               Return if FOR-NEXT loop has been completed.
```

Otherwise, get the 'looping' line number and statement.

```
        LD HL,(MEM)         Find the address of the low byte of the
        LD DE,$000F         looping line number.
        ADD HL,DE
        LD E,(HL)           Get the line number.
        INC HL
        LD D,(HL)
        INC HL
        LD H,(HL)           Followed by the statement number.
        EX DE,HL            Exchange the numbers before jumping forward
        JP GO-TO-2          to treat them as the destination of a GO TO.
```

REPORT-1

```
M1D82   RST $08          Error: NEXT without FOR.
        DEFB $00
```

NEXT-LOOP Subroutine

Tests whether LIMIT has been exceeded by the present VALUE. Pays attention to sign of STEP except for programs running from AROS. Returns the carry flag set if LIMIT is exceeded.

NEXT-LOOP

```
M1D84   RST $28          Call calculator.
        DEFB $E1         MEM 1 -> T (l)
        DEFB $E0         MEM 0 -> T (l, v)
        DEFB $E2         MEM 2 -> T (l, v, s)
        DEFB $36         MINUSQ (l, v, 1/0)
        DEFB $00,$02     IFJUMP NEXT-1
        DEFB $01         SWAP (v, l)
```

NEXT-1

```
M1D8C   DEFB $03         SUB (v-l or l-v)
        DEFB $37         PLUSQ (1/0)
        DEFB $00,$04     IFJUMP 1D93
        DEFB $38         QUIT
        AND A            Clear the carry flag and return:
        RET              loop is possible.
```

NEXT-2

```
M1D93   DEFB $38         QUIT: loop is impossible,
        SCF              set the carry flag and return.
        RET
```

READ Command Routine

The READ command reads from a DATA list. It has an effect similar to a series of LET statements. Each assignment within a single READ statement processed in turn. X_PTR holds the pointer to the READ statement; CH_ADD steps along the DATA list.

READ-3

```
M1D96   RST $20          Come here on each pass (after the first) to
                         move the READ statement along.
```

READ

```
M1D97   CALL CLASS-01    Check if variable been used? Find the variable.
        CALL SYNTAX-Z    Jump if syntax checking.
        JP Z, READ-2
        RST $18
        LD (X_PTR),HL    Save the current pointer (CH_ADD) in X_PTR.
        LD HL,ARSFLG     Get the AROS flag to check
        LD L,(HL)        whether this program
        BIT AROS,L       is running in AROS.
```

```
        JP Z, READ-N            Not running in AROS, continue with normal
                                READ.
```

GET-CART-DATA
Set up to read the data from the cartridge.

```
M1DAD   LD HL,(SYSCON)          Set the base address.
        LD DE,$0004             And the offset
        ADD HL,DE               to get to the chunk map.
        LD A,(HL)               Get the chunk map address.
        AND $0F                 Only accept the lower half of the chunk map.
        LD B,$00                DOCK bank.
        LD C,A                  Chunk map to C.
        CALL BANK_ENABLE        Switch in the cart to examine the program.
        LD HL,(DATADD)          Get the DATA list pointer
        LD A,(HL)               and jump forward
```

READ-CART-DATA

```
M1DC1   CP $2C                  Unless a new DATA statement
        JR Z, UPD-CART-DATA     can be found.
        LD E,$E4                Look for next DATA line.
        CALL LOOK-PROG
        JR NC, NEW-DATA-ADDR    If a new DATA line is found, jump ahead.
```

AROS-OUT-OF-DATA
No more data, restore the system and report the error.

```
M1DCC   LD BC,$FF00             Set home bank, all chunks.
        CALL BANK_ENABLE
        JP REPORT-E
```

NEW-DATA-ADDR

```
M1DD5   LD (DATADD),HL          Store the new DATA list pointer.
```

UPD-CART-DATA

```
M1DD8   LD HL,(ADATLN)          Put the data address in HL.
        INC HL                  Skip line number.
        INC HL
        LD C,(HL)               Length to C.
        INC HL
        LD B,(HL)
        LD (DTLNLN+1),BC        Update cart data length.
        LD BC,$FF00             Set to home bank, all chunks.
        CALL BANK_ENABLE        Switch back to home.
        LD BC,(DTLNLN+1)        Put the data length back in BC.
        LD HL,(CHANS)           And save CHANS.
        PUSH HL
        DEC HL                  Go down one then,
        CALL MAKE-ROOM          Open some space.
        POP DE                  Put CHANS in DE.
```

```
LD HL,$00FF          Put $FF in HL to
PUSH HL              get it into
LD HL,(ADATLN)       ADATLN.
INC HL               Skip the line number
INC HL               and line length.
INC HL
INC HL
PUSH HL              Save these
PUSH DE              vars for later.
LD BC,(DTLNLN+1)     Put the data length in BC again.
PUSH BC              And save that.
LD BC,$0001          Next, put a $1 on the stack.
PUSH BC
CALL XFER_BYTES      Transfer the data statement from cartridge to
                     HOME RAM.
LD HL,(ADATLN)       DATA statement address back to HL.
LD DE,(DTLNLN+1)     DATA statement length to DE.
ADD HL,DE            Set HL to end of data length.
LD DE,$0004
ADD HL,DE            Add $04 to the address.
LD BC,(DATADD)       Put DATA list pointer into BC.
AND A                Clear A.
SBC HL,BC            Subtract the DATA list pointer from HL.
LD B,H               Save HL for a bit.
LD C,L
LD HL,(CHANS)        Save CHANS.
AND A                Clear A.
SBC HL,BC            Subtract the DATA list pointer from CHANS.
PUSH HL              Save it on the stack.
INC HL               Bump it forward one byte.
LD (CH_ADD),HL       Store the value in CH_ADD.
CALL VAL-FET-1       Get the first value in the DATA statement.
POP DE               Restore temporary CHANS value.
LD HL,(CH_ADD)       Update CH_ADD.
AND A                Clear A.
SBC HL,DE            Subtract CHANS from CH_ADD.
LD DE,(DATADD)       Store DATA list pointer in DE.
ADD HL,DE            Add that to modified CH_ADD.
LD (DATADD),HL       Update DATA list pointer.
LD HL,(CHANS)        Put CHANS in HL.
LD BC,(DTLNLN+1)     Put DTLNLN + 1 in BC.
AND A                Clear A.
SBC HL,BC            Subtract DTLNLN from CHANS.
CALL RECLAIM-2       Remove the DATA line from workspace.
JP READ-1A           Move on to next READ statement.
```

READ-N
Read DATA as normal, from RAM.

```
M1E52   LD HL,(DATADD)      Get the current DATA list pointer
        LD A,(HL)           and jump forward unless a new
        CP $2C              DATA statement has to be found.
        JP Z, READ-1
        LD E,$E4            Search for 'DATA'.
        CALL LOOK-PROG      Jump forward if the search
        JR NC, READ-1       is successful.
```

REPORT-E
```
M1E62   RST $08             Error: Out of data.
        DEFB $0D
```

Continue - picking up a value from the DATA list.

READ-1
```
M1E64   CALL TEMP-PTR1      Advance the pointer along the DATA list
                            and set CH_ADD.
        CALL VAL-FET-1      Get the value and assign to the variable.
        RST $18             Get current value of CH_ADD
        LD (DATADD),HL      and store it in DATADD.
```

READ-1A
```
M1E6E   LD HL,(X_PTR)       Get the pointer to the READ statement.
        LD (IY+(OXPTR+1)),00 and clear X_PTR.
        CALL TEMP-PTR2      Make CH_ADD point to the READ statement.
```

READ-2
```
M1E78   RST $18             Get the current character and see if it
        CP $2C              is a comma.
        JP Z, READ-3        If it is, go back and keep READing.
        CALL CHECK-END      Otherwise, return via CHECK-END (if checking
        RET                 syntax) or the RET instruction (to STMT-RET).
```

DATA Command Routine

DATA statement syntax is checked when entered to ensure it contains a series of valid expressions, separated by commas. At run-time, the statement is passed by.

DATA
```
M1E82   CALL SYNTAX-Z       Skip ahead if not interpreting.
        JR NZ, DATA-2
```

DATA-1
```
M1E87   CALL SCANNING       Scan for the next expression.
        CP $2C              Confirm comma is separator.
        CALL NZ,CHECK-END   Move to next statement if not a comma.
        RST $20             As long as there are still expressions to evaluate
        JR DATA-1           keep looping.
```

DATA-2

```
M1E92   LD A,$E4          Bypass DATA statement during runtime.
```

Pass-By Subroutine

On entry the A register will hold either 'DATA' or 'DEF FN' token, depending on the type of statement that is being 'passed-by'.

PASS-BY

```
M1E94   LD B,A            Make BC hold a very high number.
        CPDR              Look back along statement for token.
        LD DE,$0200       Look along the line for the statement after
        JP EACH-STMT      (the D-1th statement from the current position).
```

RESTORE Command Routine

The operand for a RESTORE command is taken as a line number, zero being used if no operand is given. The REST-RUN entry point is used by the RUN command routine.

RESTORE

```
M1E9D   CALL FIND-INT2
        LD HL,(SYSCON)
        INC HL
        LD A,(HL)
        CP $02            Is it RAM?
        JR NZ, REST-RUN
        INC HL
        INC HL
        INC HL
        LD A,(HL)
        AND $0F
        PUSH BC
        LD C,A            Chunk map to C.
        LD B,$00          Expansion bank.
        CALL BANK_ENABLE
        POP BC
        CALL GET-A-LINE   AROS line retrieval routine.
        LD BC,$FF00       Home bank, all chunks.
        CALL BANK_ENABLE  Restore to home bank.
        JR REST-DATADD
        CALL LINE-ADDR
```

REST-DATADD

```
M1EC5   DEC HL            Make DATADD point to the location
        LD (DATADD),HL    before.
        RET
```

REST-RUN (RESTBC)

```
M1ECA   LD H,B            Operand is in BC. Transfer to HL.
        LD L,C
        CALL LINE-ADDR    Find the address of the line or the first line after.
```

```
        DEC HL              Make DATADD point to the location
        LD (DATADD),HL      before.
        RET
```

RANDOMIZE Command Routine (RAND)

Once again the operand is compressed into the BC register pair and transferred to the required system variable. However if the operand is zero the value in FRAMES1 and FRAMES2 is used instead.

RANDOMIZE (RAND)

```
M1ED4   CALL FIND-INT2      Get the operand.
        LD A,B              Jump forward unless the value
        OR C                of the operand is zero.
        JR NZ, RAND-1
        LD BC, (FRAMES)     Get the two low order bytes of FRAMES.
```

RAND-1

```
M1EDF   LD (SEED),BC        Enter the result into SEED.
        RET
```

CONTINUE Command Routine (CONT)

The required line number and statement number within that line are made the object of a jump.

CONTINUE (CONT)

```
M1EE4   LD HL, (OLDPPC)     Get the line number.
        INC H               Add one to the high byte.
        JP Z, REPORT-N      If zero flag is set, REPORT N (statement lost).
        DEC H               Line number is ok, reset H.
        LD D,(IY+OOSPCC)    Put the statement number in D.
        JR GO-TO-2          Jump forward.
```

GO TO Command Routine

The operand of a GO TO should be a line number in the range 1 to 9999. Actual test is against an upper value of 61439.

GO-TO (JUMP)

```
M1EF1   CALL FIND-INT2      Get the operand and s
        LD H,B              transfer it to BC.
        LD L,C
        LD D,$00            Set statement number to 0.
        LD A,H              Jump if line number is
        CP $F0              too large.
        JR NC, REPORT-B
```

GO-TO-2

```
M1EFD   LD (NEWPPC),HL          Put new line number into BASIC's program jump
                                instruction pointer
        LD (IY+ONSPPC),D        Set statement pointer to 0.
        RET                     Return to the interpreter
```

OUT Command Routine

The two parameters for the OUT instruction are fetched from the calculator stack and used as directed.

OUT (K_OUTPUT)

```
M1F04   CALL TWO-PARAM
        OUT (C),A
        RET
```

POKE Command Routine

POKE is essentially the same as OUT.

POKE

```
M1F0A   CALL TWO-PARAM
        LD (BC),A
        RET
```

TWO-PARAM Subroutine

The topmost parameter on the calculator stack must be compressible into a single register. It is two's complemented if it is negative. The second parameter must be compressible into a register pair.

TWO-PARAM

```
M1F0F   CALL FP-TO-A            Parameter is fetch from calculator stack.
        JR C,REPORT-B           If number is too high, report an error.
        JR Z, TWO-P-1           Jump forward for positive numbers.
        NEG                     Make negative numbers twos-complement.
```

TWO-P-1

```
M1F18   PUSH AF                 Save first parameter.
        CALL FIND-INT2          Get second parameter.
        POP AF                  Restore first parameter
        RET                     then return.
```

Find Integers Subroutine

The 'last value' on the calculator stack is fetched and compressed into a single register or a register pair by entering at FIND-INT1 AND FIND-INT2 respectively.

FIND-INT1 (INS_U1)

```
M1F1E   CALL FP-TO-A            Top of FP stack to A.
        JR FIND-I-1             to error checking
```

FIND-INT2 (FIX_U)

```
M1F23   CALL FP-TO-BC       top of calculator stack to BC
```

FIND-I-1

```
M1F26   JR C, REPORT-B
        RET Z
```

REPORT-B

```
M1F29   RST $08             Error: Integer out of range.
        DEFB $0A
```

RUN Command Routine

Any RUN parameter is passed to NEWPPC by calling GO TO. RESTORE 0 and CLEAR 0 are performed before returning.

RUN

```
M1F2B   CALL GO-TO          Call GO TO (JUMP) to set NEWPPC.
        LD BC,$0000         Perform RESTORE 0.
        CALL REST-RUN
        JR CLEAR-1          Exit via CLEAR command.
```

CLEAR Command Routine

This routine allows for the variables area to be cleared, the display area cleared and RAMTOP moved. In consequence of the last operation the machine stack is rebuilt thereby having the effect of also clearing the GO SUB stack.

CLEAR

```
M1F36   CALL FIND-INT2      Get the operand; use 0 by default.
```

CLEAR-RUN (CLR_BC)

```
M1F39   LD A,B              Jump forward if the operand is
        OR C                anything other than 0. When called from
        JR NZ, CLEAR-1      RUN there is no jump.
        LD BC, (RAMTOP)     If 0, use the existing value in RAMTOP.
```

CLEAR-1

```
M1F41   PUSH BC             Save the value.
        LD DE,(VARS)        Reclaim all the bytes of the current
        LD HL,(ELINE)       variables area.
        DEC HL
        CALL RECLAIM-1      Clear the variables.
        CALL CLS            Clear the display area.
        LD HL,ARSFLG        Get the AROS flag.
        LD L,(HL)
        BIT AROS,L          Is this program running from a cartridge?
        JR Z, NORMAL-CLEAR  No, do a normal clear.
        LD HL,(SYSCON)      Get the address of the SYSCON table.
        INC HL              Skip past first two entries.
        INC HL
        LD E,(HL)           Put chunks available in E.
```

```
        INC HL
        LD D,(HL)
        EX DE,HL
        DEC HL
        LD (DATADD),HL
        JR CLEAR-1A
```

NORMAL-CLEAR

```
M1F67   LD HL, (PROG)
        DEC HL
        LD (DATADD), HL
```

CLEAR-1A

This is where the original CLEAR-1 code picks back up. Value in the BC, which will be used as RAMTOP, is tested to ensure it is neither too low nor too high.

```
M1F6E   LD HL,(STKEND)          Get the current value of STKEND.
        LD DE,$0032             Increase it by 50 before testing.
        ADD HL,DE               This is the lower limit.
        POP DE
        SBC HL,DE
        JR NC, REPORT-M         RAMPTOP will be too low.
        LD HL,(PRAMT)           Get PRAMT to test for upper limit.
        AND A
        SBC HL,DE
        JR NC, CLEAR-2          Jump forward if it's good.
```

REPORT-M

```
M1F82   RST $08                 Error: RAMTOP no good.
        DEFB $15
```

CLEAR-2

```
M1F84   EX DE,HL                Pass the value to RAMTOP.
        LD (RAMTOP),HL
        POP DE                  Get the STMT-RET address.
        POP BC                  Get the 'error address'.
        LD HL,(MSTBOT)          Get the machine stack bottom.
        DEC HL                  Leave a blank space.
        LD (HL),$3E             Enter a GO SUB stack end marker.
        DEC HL                  Leave another blank space.
        LD SP,HL                Make the stack pointer point
                                to an empty GO SUB stack.
        PUSH BC                 Pass the 'error address' to the stack
        LD (ERRSP),SP           and save its address in ERR-SP.
        EX DE,HL                Make an indirect return
        JP (HL)                 to STMT-RET.
```

GO SUB Command Routine

This routine saves PPC and SUBPPC on the GO SUB stack, then uses GO TO to put the target address into NEWPPC and NSPPC.

GO-SUB (GO_SUB)

```
M1F99   POP DE                  Save the address (STMT-RET).
        LD H,(IY+OSUBPPC)       Get statement number that is executing.
        INC H                   Point to next statement.
        EX (SP),HL              Exchange the 'error address' with the
                                statement number.
        INC SP                  (8 bit PUSH)
        LD BC,(PPC)             Get current line number.
        PUSH BC                 Save it.
        PUSH HL                 Put the 'error address' back on the stack.
        LD (ERRSP),SP           and reset ERRSP to point to it.
        PUSH DE                 Return the address STMT-RET.
        CALL GO-TO-1            Call GO TO to set NEWPPC and NSPPC.
        LD HL,(MSTBOT)          Get address above machine stack.
        DEC H                   Go one page down.
        LD DE,$0010             Need 16 bytes.
        ADD HL,DE               Point to address.
        SBC HL,SP               If stack pointer is still above the
        RET C                   required end address, proceed.
        JR REPORT-4             Out of space, give Report 4.
```

TEST-ROOM Subroutine

Checks to see if sufficient RAM is available below RAMTOP to give the caller BC bytes of room. If not, exits to error 4.

TEST-ROOM (CHK_SZ)

```
M1FBB   LD HL,(STKEND)          Get STKEND (start of spare space).
        ADD HL,BC               Add the number of bytes requested.
        JR C, REPORT-4          If carry is set, not enough memory.
        EX DE,HL                DE points to final address.
        LD HL,$0050             Add a safety margin for machine stack.
        ADD HL,DE
        JR C, REPORT-4          If carry is set, not enough memory.
        LD DE,(RAMTOP)          Get RAMTOP (top of BASIC memory).
        SBC HL,DE
        RET C                   Exit if requested < RAMTOP.
```

REPORT-4 (ERR4)

```
M1FCF   LD L,$03                Out of memory. This is a 'run-time' error
        JP ERROR-3              and the error marker is not used.
```

RETURN Command Routine

Removes the line number and statement number from the GO SUB stack then jumps into the GOTO routine to put them into NEWPPC and NSPPC.

RETURN

```
M1FD4   POP BC              Get STMT-RET.
        POP HL              Get the 'error address'.
        POP DE              Get the last entry on the GO SUB stack.
        LD A,D              Test to see if it is the GO SUB stack end
        CP $3E              marker.
        JR Z, REPORT-7      Jump if it is not found.
        DEC SP              Adjust stack pointer to get statement number,
        EX (SP),HL          get it and save HL.
        EX DE,HL            Put statement number in DE, line number in HL.
        LD (ERRSP),SP       Reset error pointer.
        PUSH BC             Replace STMT-RET.
        JP GO-TO2           GOTO the GOSUB return address.
```

REPORT-7

```
M1FE7   PUSH DE             Replace the end marker and
        PUSH HL             the 'error address.'
        RST $08             Error: RETURN without GOSUB.
        DEFB $06
```

PAUSE Command Routine

Period of PAUSE is determined by counting the number of maskable interrupts as they occur every 1/60th of a second. PAUSE is finished either after the appropriate number of interrupts or by the system Variable FLAGS indicating that a key has been pressed.

PAUSE

```
M1FEB   RES KEYHIT,(IY+OFLAGS)  Reset key hit flag.
        CALL FIND-INT2          Get the PAUSE operand.
```

PAUSE-1

```
M1FF2   HALT                Wait for maskable interrupt (IM1).
        DEC BC              Decrement the PAUSE counter.
        LD A,B              If the counter is reduced to 0,
        OR C                PAUSE timed out.
        JR Z, PAUSE-END
        LD A,B              If the operand was 0, BC now has $FFFF.
        AND C               If BC <> $FFFF jump to examine
        INC A               the key hit.
        JR NZ, PAUSE-2
        INC BC              BC = $FFFF, therefore no timeout.
```

PAUSE-2

```
M1FFE   BIT KEYHIT,(IY+OFLAGS)  Jump back to wait if a key has
        JR Z, PAUSE-1           not been hit.
```

PAUSE-END

```
M2004   RES KEYHIT,(IY+OFLAGS)    Force no key hit.
        RET
```

Break-Key Subroutine

Test BREAK key. Carry flag is returned reset if SHIFT and BREAK are pressed together.

BREAK-KEY (BREAK)

```
M2009   LD A,$7F                  Create the port address and
        IN A,($FE)                read the keyboard.
        RRA                       Examine only bit 0 by shifting it to carry.
        RET C                     Return if BREAK not pressed.
        BIT 6,(IY+(OERRLN+1))     If we running a program,
        JR Z, BR-KEY-1            check if the CAPS SHIFT key is on.
        SCF                       Force no BREAK.
        RET
```

BR-KEY-1

```
M2017   LD A,$FE                  Carry will be set if
        IN A,($FE)                CAPS SHIFT is pressed.
        RRA
        RET
```

DEF FN Command Routine

DEF FN statement is checked when entered to ensure that it has the correct form. Space is also made available for the result of evaluating the function. At run-time, a DEF FN statement is bypassed.

DEF-FN (DEF)

```
M201D   CALL SYNTAX-Z             Jump forward if checking syntax.
        JR Z, DEF-FN-1
        LD A,$CE                  Load A with DEF FN token
        JP PASS-BY                Bypass if doing syntax.
```

DEF-FN-1

```
M2027   SET NUM,(IY+OFLAGS)       Force a numerical result.
        CALL ALPHA                Check if the current code is a letter.
        JR NC, DEF-FN-4           Jump forward if not.
        RST $20                   Get next character.
        CP $24                    Jump forward unless it is $.
        JR NZ, DEF-FN-2
        RES NUM,(IY+OFLAGS)       Reset NUM; this is a string variable.
        RST $20                   Get next character.
```

DEF-FN-2

```
M203A   CP $28              A left parenthesis must follow the variable name.
        JR NZ, DEF-FN-7     Report an error if not a (.
        RST $20             Get next character.
        CP $29              Compare to right parenthesis.
        JR Z, DEF-FN-6      Jump forward as there are no parameters to this
                            function.
```

DEF-FN-3

Loop through the parameters.

```
M2043   CALL ALPHA          The current character must be a letter.
```

DEF-FN-4

```
M2046   JP NC,REPORT-C      Not a character? Error - BAD BASIC.
        EX DE,HL            Save the pointer in DE.
        RST $20             Get next character.
        CP $24              Jump forward unless it is a $.
        JR NZ, DEF-FN-5
        EX DE,HL            Otherwise, save new pointer to DE.
        RST $20             Get next character.
```

DEF-FN-5

```
M2051   EX DE,HL            Move the pointer to the last character
                            of the name to HL.
        LD BC,$0006         Make six spaces after the last character
        CALL MAKE-ROOM      and enter a 'number marker' into the
        INC HL              first of the new locations.
        INC HL
        LD (HL),$0E
        CP $2C              Is the current character is a comma?
        JR NZ, DEF-FN-6     No, jump out of the loop.
        RST $20             Get the next parameter and
        JR DEF-FN-3         continue processing.
```

DEF-FN-6

Process the definition of the function.

```
M2063   CP $29              Check that right parenthesis exists.
        JR NZ, DEF-FN-7     No? Report an error.
        RST $20             Get next character.
        CP $3D              It must be an equal sign ("-").
        JR NZ, DEF-FN-7     No? Report an error.
        RST $20             Get next character.
        LD A,(FLAGS)        Save the nature (numeric, string) of variable.
        PUSH AF
        CALL SCANNING       Process the definition as an expression.
        POP AF              Retrieve nature of variable and check
        XOR (IY+OFLAGS)     that it is the same type as found for the
        AND $40             definition.
```

DEF-FN-7

```
M207A   JP NZ,REPORT-C          Error - BAD BASIC
        CALL CHECK-END          Exit via the CHECK-END subroutine.
```

ON ERR Command Routine

This statement allows the programmer to disable automatic program termination upon encountering an error condition. Line number and statement number that caused the error are available at ERRC ($5CB8) and ERRS($5CBA).

```
M2080   RST $18                 Get current character.
        CP $7F                  Is this RESET?
        JR Z, ON-ERR-RESET      Yes, go to the RESET routine.
        CP $EC                  Is it GO TO?
        JR Z, ON-ERR-GO-TO      Yes, go to the GO TO routine.
        CP $E8                  Is is CONTINUE?
        JP NZ,REPORT-C          No, error - BAD BASIC.
```

ON-ERR-CONT

```
        RST $20                 Get next character.
        CALL CHECK-END          Move to the next statement if checking syntax.
        BIT 7,(IY+(OERRLN+1))   Is the high bit of the error line set?
        RET Z                   No, return.
        LD HL,(ERRC)            Put the line number in HL.
        LD (NEWPPC),HL          And set NEWPPC to that line number.
        LD A,(ERRS)             Put the statement number in A.
        LD (NSPPC),A            And set NSPCC to the statement number.
        RES 6,(IY+(OERRLN+1))   Reset the error flag.
```

ON-ERR-RETURN

```
M20A7   POP HL                  Get STMT-RET address from stack.
        LD DE,$0007             Add 7 bytes to skip past
        ADD HL,DE               BREAK key checking.
        PUSH HL                 Put it back on the stack.
        RET                     And return from the ERROR.
```

ON-ERR-RESET

```
M20AE   RST $20                 Get next character
        CALL CHECK-END          Move to the next statement if checking syntax.
        RES 7,(IY+(OERRLN+1))   Reset the error codes.
        RES 6,(IY+(OERRLN+1))
        JR ON-ERR-RETURN
```

ON-ERR-GO-TO

```
M20BC   RST $20                 Get next character.
        CALL EXPT-1NUM
        CALL CHECK-END          Move to the next statement if checking syntax.
        CALL FP2BC              Convert floating point number, put in BC.
        LD A,B                  Move MSB to A.
        AND $3F                 Clear bits 6 and 7.
```

```
         OR $80              See if the high bit is set.
         LD B,A              Move it back to B.
         LD (ERRLN),BC       Put that into ERRLN.
         RET
```

DELETE Command Routine

DELETE

```
M20D1    RST $18             Get current character
         CP $2C              Is it a comma?
         JR NZ, DELETE-LINES No, deleting a range.
         CALL SYNTAX-Z       Jump if syntax checking.
         JR Z, DELETE-LINES-1
         RST $28             Call calculator.
         DEFB $A1            CONST 1 (256)
         DEFB $38            QUIT
         JR DELETE-LINES-1
```

DELETE-LINES

```
M20E0    CALL EXPT-1NUM      Get the value.
         CP $2C              Is it a comma?
         JR NZ, DELETE-ERROR
```

DELETE-LINES-1

```
M20E7    RST $20             Get next character
         CP $0D              Is it ENTER?
         JR Z,DEL-SORT
         CP $3A              Colon?
         JR Z,DEL-SORT
         CALL EXPT-1NUM
         JR DEL-SORT-1
```

DEL-SORT

Sort delete parameters.

```
M20F5    LD BC,$270F         Put highest line number (9999) in BC.
         CALL SYNTAX-Z       Call if interpreting.
         CALL NZ,STACK-BC    Push value to BC.
```

DEL-SORT-1

```
M20FE    CALL CHECK-END      Move to the next statement if checking syntax.
         CALL DELETE-FP2BC
         INC HL              Increment the line number.
         CALL LINE-ADDR      Is there an existing line with this number?
         PUSH HL             Push the line number to the stack.
         CALL DELETE-FP2BC   Convert to BC.
         CALL LINE-ADDR      Is there an existing line with this number?
         EX DE,HL            Swap DE and HL.
         POP HL              Get HL from the stack.
         PUSH HL             Put it on the stack.
```

```
        SCF                     Clear the carry flag.
        SBC HL,DE               Subtract DE from HL.
        JR C, DELETE-ERROR      If carry is set, crash out.
        POP HL                  Get HL back.
        CALL RECLAIM-1          Remove the line.
        RET                     Done.
```

DELETE-ERROR
Error in the command.

```
M211C   RST $08                 Error: Nonsense in BASIC.
        DEFB $0B
```

DELETE-FP2BC

```
M211E   CALL FP2BC              Convert number from floating point, put in BC.
        LD A,B                  Move MSB to A.
        AND $3F                 Clear bits 6 and 7.
        LD H,A                  Move to HL.
        LD L,C
        RET
M2126   RST $20                 Get next character.
```

SOUND Command Routine

The SOUND command writes the first parameter (register number) to port $F5 (address in the Programmable Sound Generator) and second parameter to port $F6 (data in the PSG). The program line is scanned for multiple parameter pairs and continues writing address/data pairs until the end of the statement is reached.

SOUND

```
M2127   CALL EXPT-2NUM          Evaluate the next two comma-separated
                                expressions.
        CALL SYNTAX-Z           Jump to test if syntax checking.
        JR Z, SOUND-RD          If not, read the register/data pair.
        CALL FP-TO-A            Move the data value to A
        PUSH AF                 and save the result.
        CALL FP-TO-A            Move the register number to A
        CP $11                  Test if the register number is too large.
        JP NC, REPORT-C         Error if too high.
        DEC A                   Test bit 7.
        INC A
        JP M, REPORT-C          Error, A = 0.
        OUT ($F5),A             Output register number to PSG address
                                register.
        POP AF                  Get the data value.
        OUT ($F6),A             Output to the PSG data register.
```

SOUND-RD

```
M2146   RST $18             Get current character.
        CP ';'              If it's a semi-colon, there are more
                            register/data pairs.
        JR Z, SOUND         Keep processing pairs.
        CALL CHECK-END      See if there's anything else on the
                            line to process.
        RET                 Nope, return.
```

UNSTACK-Z Subroutine

Called to 'return early' from a subroutine when checking syntax. Avoids printing characters or passing values to/from the calculator stack.

UNSTACK-Z

```
M214F   CALL SYNTAX-Z       Check mode (syntax checking or interpreting).
        POP HL              Get the return address but ignore and
        RET Z               return if syntax checking.
        JP (HL)             During run-time, return to the calling routine.
```

LPRINT & PRINT Command Routines

Appropriate channel is opened as necessary and the items to be printed are considered in turn.

LPRINT (K_LPR)

```
M2155   LD A,$03            Prepare to open channel P.
        JR PRINT-1
```

PRINT (K_PRIN)

```
M2159   LD A,(ARSFLG)       Get the AROS flag.
        RES BANKCHN,A       Reset bank channel flag, output to normal
                            channel.
        LD (ARSFLG),A       Store back in AROS flag.
M2161   LD A,$02            Prepare to open channel S.
```

PRINT-1

```
M2163   CALL SYNTAX-Z       Is syntax being checked?
        CALL NZ, CHAN-OPEN  No, open the proper channel.
        CALL SYNTAX-Z       Check if syntax is being checked, again.
        CALL NZ, PRINT-1A   No, go set the token flag.
        CALL TEMPS          Set the color attributes.
        CALL PRINT-2        Call the print control subroutine.
        CALL CHECK-END      Consider the next statement.
        RET
```

PRINT-1A

```
M2179   SET TOKEN,(IY+OFLAGS)   Set token mode on.
        RET
```

PRINT-2 (P_SEQ)
The print control subroutine is called by PRINT, LPRINT and INPUT.

```
M217E   RST $18          Get current character.
        CALL PR-END-Z    Jump forward if already at end of the item list.
        JR Z, PRINT-4
```

PRINT-3
Loop to deal with 'position controllers' and the print items.

```
M2184   CALL PR-POSN-1   Handle any consecutive position
        JR Z, PRINT-3    controllers.
        CALL PR-ITEM-1   Handle a single print item.
        CALL PR-POSN-1   Check for more position controllers and
        JR Z, PRINT-3    print items until there are none left.
```

PRINT-4

```
M2191   CP $29           Return if the current character is a right
        RET Z            parenthesis.
```

Print a Carriage Return Subroutine

PRINT-CR

```
M2194   CALL UNSTACK-Z   Return if changing syntax.
        LD A,$0D         Print a carriage return character
        RST $10          and return.
        RET
```

Print Items Subroutine

Called from the PRINT, LPRINT and INPUT command routines. The various print items are identified and printed.

PR-ITEM-1

```
M219B   RST $18          Get current character.
        CP $AC           Jump forward if it is not AT.
        JR NZ, PR-ITEM-2
        CALL NEXT-2NUM   Transfer the two parameters to the calculator
                         stack.
        CALL UNSTACK-Z   Return if checking syntax.
        CALL STK-TO-BC   Compress parameters in to BC.
        LD A,$16         Put AT control character in A.
        JR PR-AT-TAB
```

PR-ITEM-2

```
M21AD   CP $AD           Jump forward if character is not TAB.
        JR NZ, PR-ITEM-3
        RST $20          Get the next character.
        CALL EXPT-1NUM   Transfer one parameter to the calculator
                         stack.
        CALL UNSTACK-Z   Return if checking syntax.
```

```
         CALL FIND-INT2          Compress the value into BC.
         LD A,$17                Put the TAB control character in A.
```

PR-AT-TAB
AT and TAB are printed by making three calls to PRINT-OUT.

```
M21BD    RST $10                 Print the control character.
         LD A,C                  Follow it with the first attribute.
         RST $10                 Print the second value.
         LD A,B
         RST $10
         RET
```

PR-ITEM-3
Handle embedded color.

```
M21C3    CALL CO-TEMP-3          Return with carry reset if color items were found.
         RET NC                  Continue if none were found.
         CALL STR-ALTER          Check if the stream is to be changed.
         RET NC                  Continue unless it was changed.
         CALL SCANNING           Evaluate the expression but return
         CALL UNSTACK-Z          if checking syntax.
         BIT NUM,(IY+OFLAGS)     Test for the nature of the expression.
         CALL Z ,STK-FETCH       If it is a string, get necessary parameters.
         JP NZ, PRINT-FP         If numeric, exit via PRINT-FP.
```

PR-STRING
Handle each character in a string.

```
M21DB    LD A,B                  Return now if there are no characters
         OR C                    remaining in the string; otherwise
         DEC BC                  decrease the counter.
         RET Z
         LD A,(DE)               Get the code and
         INC DE                  increment the pointer.
         RST $10                 Print the code and jump back
         JR PR-STRING            to continue with the string.
```

End Of Printing Subroutine

Checks for termination of a line. If end of statement (close parenthesis, newline or colon), return with zero flag=1.

PR-END-Z

```
M21E4    CP $29                  Return if close parenthesis.
         RET Z
```

PR-ST-END (TERMQ)

```
M21E7   CP $0D          Return if end of line (carriage return).
        RET Z
        CP $3A          Return if colon.
        RET
```

Print Position Subroutine

Position controlling characters are handled by this subroutine.

PR-POSN-1

```
M21ED   RST $18         Get current character.
        CP $3B          Is it a semi-colon?
        JR Z, PR-POSN-3 Yes, jump forward.
        CP $2C          Is it a comma?
        JR NZ, PR-POSN-2 Yes, jump forward.
        CALL SYNTAX-Z   Is syntax being checked?
        JR Z, PR-POSN-3 Yes, but do not print character.
        LD A,$06        Put comma character in A.
        RST $10         Print the comma.
        JR PR-POSN-3
```

PR-POSN-2

```
M2200   CP $27          Is it a question mark?
        RET NZ          No, return.
        CALL PR-CR      Print carriage return.
```

PR-POSN-3

```
M2206   RST $20         Get next character.
        CALL PR-END-Z   Check for end-of-line characters.
        JR NZ, PR-POSN-4 Not end of line, keep going.
        POP BC          Prepare to return to calling routine.
```

PR-POSN-4

```
M220D   CP A            Clear zero flag.
        RET
```

ALTER STREAM Subroutine

Called to check if the output stream should be updated.

STR-ALTER (STRITO)

```
M220F   CP '#'          Is the character a number sign?
        SCF             Set the carry flag and
        RET NZ          return if not #.
        RST $20         Get next character.
        CALL EXPT-1NUM  Pass the parameter to the calculator stack.
        AND A           Clear the carry flag.
        CALL UNSTACK-Z  Return if checking syntax.
        CALL FIND-INT1  Pass the value to A.
        LD (STRMN),A    Put into stream number for "bank" devices.
```

```
        CP $10                  Jump to invalid stream if
        JP NC, REPORT-O         requested stream # > 16.
        CALL CHAN-OPEN          Select the stream in A.
        AND A                   Clear carry flag.
        RET
```

INPUT Command Routine

Assigns values entered from the keyboard to variables. Also prints text embedded in the INPUT statement to the lower part of the display.

INPUT

```
M222B   LD A, (ARSFLG)          Get the AROS flags.
        SET BANKCHN,A           Reset the bank channel flag, input from a normal
                                stream.
        LD (ARSFLG), A          Put the flag back.
        CALL SYNTAX-Z           Jump forward if checking syntax.
        JR Z,INPUT-1
        LD A,$01                Open channel 'K'.
        CALL CHAN-OPEN
        CALL CLS-LOWER          Clear the lower portion of the screen.
```

INPUT-1

```
M2240   LD (IY+OTVFLAG),01      Signal that the lower part of the screen is being
                                used. Reset all other bits.
        CALL IN-ITEM-1          Call subroutine to handle input items.
        CALL CHECK-END          Continue with next statement if checking syntax.
        LD BC,(SPOSNCOL)        Get the current print position.
        LD A,(DFSZ)             Jump forward if the current position is
        CP B                    above the lower half of the screen.
        JR C, INPUT-2
        LD C,$21                Otherwise, set print position to top of
        LD B,A                  lower half of screen.
```

INPUT-2

```
M2257   LD (SPOSNCOL),BC        Set SPOSNCOL with correct values.
        LD A,$19                Set the scroll counter.
        SUB B
        LD (SCRCT),A
        RES LHS,(IY+OTVFLAG)    Signal 'main screen'.
        CALL CL-SET             Set the system variables and
        JP CLS-LOWER            exit via CLS-LOWER.
```

IN-ITEM-1 (I_SEQ)

Handle INPUT and embedded PRINT.

```
M226B   CALL PR-POSN-1          Handle position control characters.
        JR Z, IN-ITEM-1
        CP $28                  Jump forward if current character is not '('.
        JR NZ, IN-ITEM-2
        RST $20                 Get next character.
```

```
         CALL PRINT-2          Call PRINT command to handle items inside
                               parenthesis.
         RST $18               Get current character.
         CP $29                If it's not a right parenthesis, give
         JP NZ,REPORT-C        error - BAD BASIC.
         RST $20               Get next character.
         JP IN-NEXT-2          Jump forward to see if there's any more INPUT to
                               handle.
```

IN-ITEM-2
Handle INPUT LINE.

```
M2282    CP $CA                Jump forward if character is not 'LINE'.
         JR NZ, IN-ITEM-3
         RST $20               Get next character.
         CALL CLASS-01         Determine where to store the variable.
         SET LINPLN,(IY+OFLAGX) Signal 'using INPUT LINE'.
         BIT NUM,(IY+OFLAGS)   If we're using a non-string variable,
         JP NZ,REPORT-C        exit via Report C.
         JR IN-PROMPT
```

IN-ITEM-3
Handle simple INPUT variables.

```
M2297    CALL ALPHA            If the current character is not a letter,
         JP NC, IN-NEXT-1      jump forward to the end of input loop routine.
         CALL CLASS-01         Determine where to store the variable.
         RES LINPLN,(IY+OFLAGX) Signal 'not INPUT LINE'.
```

IN-PROMPT
Create the prompt message.

```
M22A4    CALL SYNTAX-Z         Jump forward if syntax checking.
         JP Z, IN-NEXT-2
         CALL SET-WORK         Set workspace to null.
         LD HL,FLAGX           Put FLAGX in HL.
         RES NO,(HL)           Signal 'string result'.
         SET INPLN,(HL)        Signal 'INPUT mode'.
         LD BC,$0001           Set prompt message to single location.
         BIT LINPLN,(HL)       Jump forward if using 'LINE'.
         JR NZ, IN-PR-2
         LD A,(FLAGS)          Jump forward if expecting a
         AND $40               numeric entry.
         JR NZ, IN-PR-1
         LD C, $03             A string will need three bytes.
```

IN-PR-1

```
M22C4    OR (HL)               Set bit 6 of FLAGX for numeric entry.
         LD (HL),A
```

IN-PR-2

```
M22C6   RST $30              Make required number of bytes available.
        LD (HL),$0D          Put a carriage return in the last location.
        LD A,C               Test bit 6 of the C register
        RRCA                 and jump forward if
        RRCA                 only one byte was required.
        JR NC, IN-PR-3
        LD A,$22             Put double-quote in first and
        LD (DE),A            second locations.
        DEC HL
        LD (HL),A
```

IN-PR-3

```
M22D3   LD (KCUR),HL         Save cursor position.
        BIT LINPLN,(IY+OFLAGX) Jump forward if INPUT LINE.
        JR NZ, IN-VAR-3
        LD HL,(CH_ADD)       Save current values of CH_ADD
        PUSH HL              and ERRSP to machine stack.
        LD HL,(ERRSP)
        PUSH HL
```

IN-VAR-1

```
M22E4   LD HL,$22E4          Set 'return point' for errors.
        PUSH HL
        BIT RETPOS,(IY+OFLAGS2) Change the error stack pointer if retype
        JR Z, IN-VAR-2       possible after syntax error is possible.
        LD (ERRSP),SP
```

IN-VAR-2

```
M22F2   LD HL,(WORKSP)       Set HL to the start of the INPUT line
        CALL REMOVE-FP       and remove and floating-point 'slugs'.
        LD (IY+OERRNR),$FF   Signal 'no error yet'.
        CALL EDITOR          Get the input and set FLAGS to check
        RES INTPT,(IY+OFLAGS) syntax. Check input for errors; jump if ok,
        CALL IN-ASSIGN       return to IN-VAR-1 if not.
        JR IN-VAR-4
```

IN-VAR-3

```
M2308   CALL EDITOR          Get a 'LINE'.
```

IN-VAR-4

```
M230B   LD (IY+(OKCUR+1)),00 Reset the cursor address.
        CALL IN-CHAN-K       Jump if using channel other than 'K'.
        JR NZ, IN-VAR-5
        CALL ED-COPY         Copy line to display and
        LD BC,(ECHOE)        point ECHOE to current position in lower screen.
        CALL CL-SET          Set system variables.
```

IN-VAR-5

```
M231E   LD HL,FLAGX              Put FLAGX in HL.
        RES INPLN,(HL)           Signal 'edit mode'.
        BIT LINPLN,(HL)          Jump forward if handling an
        RES LINPLN,(HL)          INPUT LINE.
        JR NZ, IN-VAR-6
        POP HL                   Drop the address for IN-VAR-1.
        POP HL                   Reset ERRSP to its
        LD (ERRSP),HL            original address.
        POP HL                   Save the original CH_ADD address
        LD (X_PTR),HL            in X_PTR.
        SET INTPT,(IY+OFLAGS)    Set syntax/run flag to run and
        CALL IN-ASSIGN           make the assignment.
        LD HL,(X_PTR)            Restore the original address
        LD (IY+(OXPTR+1)),00     to CH_ADD and clear X_PTR.
        LD (CH_ADD),HL
        JR IN-NEXT-2             Jump forward to see if there's more.
```

IN-VAR-6

```
M2345   LD HL,(STKBOT)           Get the length of the LINE in
        LD DE,(WORKSP)           workspace.
        SCF
        SBC HL,DE
        LD B,H                   DE points to start and BC
        LD C,L                   holds the length.
        CALL STK-ST-$            Stack the parameters and
        CALL LET                 assign the value to the variable.
        JR IN-NEXT-2             Jump forward to see if there's more.
```

IN-NEXT-1

```
M2359   CALL PR-ITEM-1           Handle any print items.
```

IN-NEXT-2

```
M235C   CALL PR-POSN-1           Handle any print position controllers.
        JP Z, IN-ITEM-1          Go around the loop again if there is
        RET                      more to handle, else return.
```

In-Assign Subroutine

Called twice for each INPUT value. Once with the syntax/run flag reset (syntax) and once with it set (run).

IN-ASSIGN

```
M2363   LD HL,(WORKSP)           Set CH_ADD to point to the first
        LD (CH_ADD),HL           location in the workspace and
        RST $18                  get the current character.
        CP $E2                   Is it STOP?
        JR Z, IN-STOP            Yes, exit out.
        LD A,(FLAGX)             Otherwise, assign the value
        CALL VAL-FET-2           to the variable.
```

```
        RST $18         Get current character.
        CP $0D          Is it a carriage return?
        RET Z           Yes, return.
```

REPORT-C

```
        RST $08         Error: Nonsense in BASIC.
        DEFB $0B
```

IN-STOP

```
M237A   CALL SYNTAX-Z   Return if syntax checking but no error message.
        RET Z
```

REPORT-H (ERRH)

```
        RST $08         Error: STOP in INPUT.
        DEFB $10
```

IN-CHAN-K Subroutine

Returns with the zero flag reset only if channel 'K' is being used.

IN-CHAN-K (NOTKBQ)

```
M2380   LD HL, (CURCHL)     Get the base address of the
        INC HL              channel information for the
        INC HL              current channel and compare
        INC HL              the channel code to the
        INC HL              character 'K'.
        LD A, (HL)
        CP $4B
        RET
```

Color Item Routines

These routines handle embedded color item and the color system variable. Embedded color items are handled by calling PRINT-OUT. The entry point is CO-TEMP-2.

CO-TEMP-1

```
M238B   RST $20             Get next character.
```

CO-TEMP-1 (GR_COL)

```
M238C   CALL CO-TEMP-3      Jump forward to check if the code is a
                            temporary, embedded color. Return carry
        RET C               set if is not a color item.
        RST $18             Get current character.
        CP $2C              Jump back if it is a comma
        JR Z, CO-TEMP-1
        CP $3B              or semi-colon.
        JR Z, CO-TEMP-1
        JP REPORT-C         Otherwise, there's been an error.
```

CO-TEMP-3

```
M239C   CP $D9              Return with the carry flag
        RET C               if the code is not in the range of
        CP $DF              $D9 - $DE (INK to OVER).
        CCF                 Complement carry flag.
        RET C
        PUSH AF             Save the color item code while
        RST $20             getting the next character.
        POP AF
```

CO-TEMP-4 (COLITM)

```
M23A6   SUB $C9             Reduce the token range from $D9-$DE
                            to control characters ($10-$10).
        PUSH AF             Save the control character to the stack.
        CALL EXPT-1NUM      Move the parameter to the calculator stack.
        POP AF              Get control character back.
        AND A
        CALL UNSTACK-Z      If syntax is being checked, return.
```

```
        PUSH AF             Save control character; move
        CALL FIND-INT1      parameter to D.
        LD D,A
        POP AF              Get control character back.
        RST $10             Print control character.
        LD A,D              Get the parameter
        RST $10             and print it, too.
        RET
```

TV_COL

System color variables (ATTR-T, MASK-T & P-FLAG) are updated as required. On entry the control character code is in A and the parameter is in D. All changes are to the 'temporary' system variables.

CO-TEMP-5 (CO-TEMPS)

```
M23BB   SUB $11             Reduce the range and
        ADC A,$00           jump forward for INK and PAPER.
        JR Z, CO-TEMP-7
        SUB $02             Reduce again and jump forward
        ADC A,$00           for FLASH and BRIGHT
        JR Z, CO-TEMP-C
```

Color control code is now $01 (INVERSE) or $02 (OVER) and PFLAG will be updated accordingly.

```
        CP $01              Prepare to jump with OVER.
        LD A,D              Get the parameter.
        LD B, $01           Set the mask for OVER.
        JR NZ,CO-TEMP-6     Jump.
        RLCA                Bit 2 of A needs to be reset for
        RLCA                INVERSE 0 and set for INVERSE 1.
        LD B,$04            Mask to set the bit.
```

CO-TEMP-6

```
M23D2   LD C,A              Save A register.
        LD A,D              Correct range for INVERSE and OVER
        CP $02              is only 0-1.
        JR NC,REPORT-K      Not in range, report error.
        LD A,C              Restore A.
        LD HL,PFLAG         Get PFLAG.
        JR CO-CHANGE        Exit via CO-CHANGE and use B as mask
                            for PFLAG.
```

CO-TEMP-7 (COLOR)

PAPER and INK are handled here. Carry flag is set on entry for INK.

```
M23DE   LD A,D              Get the parameter.
        LD B,$07            Mask for INK.
        JR C,CO-TEMP-8      Jump forward to handle INK.
        RLCA                Multiply the parameter for
```

```
            RLCA                    PAPER by eight.
            RLCA
            LD B,$38                Mask for PAPER.
```

CO-TEMP-8

```
M23E8       LD C,A                  Save the parameter in C.
            LD A,D                  Get the parameter.
            CP $0A                  Only allow for PAPER/INK (range 0-9).
            JR C,REPORT-K
```

REPORT-K

```
M23EE       RST $08                 Error: Invalid color.
            DEFB $13
```

CO-TEMP-9

```
M23F0       LD HL,ATTRT             Prepare to alter ATTRT, MASKT and PFLAG.
            CP $08                  Jump forward for PAPER/INK 0-5, 7.
            JR C,CO-TEMP-B
            LD A,(HL)               Get the current value of ATTRT and use
            JR Z,CO-TEMP-A          with PAPER/INK 8.
            OR B                    For PAPER/INK 9, the colors have to be
            CPL                     black and white.
            AND $24
            JR Z,CO-TEMP-A          Jump for black PAPER/INK.
            LD A,B                  Continue for white PAPER/INK.
```

CO-TEMP-A

```
M2401       LD C,A                  Move value to C.
```

CO-TEMP-B

Mask (B) and value (C) are used to change ATTRT.

```
M2402       LD A,C                  Move the value to A.
            CALL CO-CHANGE          Change ATTRT.
            LD A,$07                Bits of MASKT are set only when using
            CP D                    PAPER/IN 8 or 9.
            SBC A,A
            CALL CO-CHANGE          Change MASKT.
            RLCA                    Create the appropriate mask in B
            RLCA                    in order to change bits 4 and 6
            AND $50                 as necessary.
            LD B,A
            LD A,$08                Bits of PFLAG are only set when using
            CP D                    PAPER/INK 9.
            SBC A,A                 Continue into CO-CHANGE to update PFLAG.
```

CO-CHANGE Subroutine

This subroutine applies a masked version of A to a system variable. The B register holds a mask that shows which bits are to be 'copied over' from A to (HL).

CO-CHANGE

```
M2416   XOR (HL)          The bits, specified by the mask in B,
        AND B             are changed in the value and the
        XOR (HL)          result goes to the system variable.
        LD (HL),A
        INC HL            Move to the next system variable.
        LD A,B            Return with the mask in A.
        RET
```

CO-TEMP-C (HIFLSH)

This routine handles BRIGHT and FLASH.

```
M241D   SBC A,A           Zero flag is set for BRIGHT.
        LD A,D            Get the parameter.
        RRCA              Rotate it to the right.
        LD B,$80          Mask for FLASH.
        JR NZ, CO-TEMP-D  Jump forward for FLASH.
        RRCA              Rotate again to prepare
        LD B,$40          mask for BRIGHT.
```

CO-TEMP-D

```
M2427   LD C,A            Save the value to C.
        LD A,D            Get the parameter and test its range;
        CP $08            only 0, 1 and 8 are allowed.
        JR Z, CO-TEMP-E
        CP $02
        JR NC,REPORT-K    Not an allowed value.
```

CO-TEMP-E

```
M2431   LD A,C            Get the value.
        LD HL,ATTRT       Put address of ATTRT in HL then
        CALL CO-CHANGE    change the system variable.
        LD A,C            Get the value again, this time for MASK-T.
        RRCA              Bit set for FLASH/BRIGHT 8 (bit 3) is
        RRCA              moved to bit 7 (for FLASH) or bit 6
        RRCA              (for BRIGHT).
        JR CO-CHANGE      Exit via CO-CHANGE.
```

BORDER Command Routine

BORDER is used with OUT to alter the color of the border. Parameter is saved in the system variable BORDCR.

BORDER

```
M243E   CALL FIND-INT1          Get the parameter and
        CP $08                  test its range.
        JR NC,REPORT-K
        OUT ($FE),A             OUT instruction is used to set border color.
        RLCA                    Multiply the parameter by 8.
        RLCA
        RLCA
        BIT 5,A                 If the border color is a light color, then
        JR NZ,BORDER-1          set the INK color in the editing area to black.
        XOR $07                 Change the INK color.
```

BORDER-1

```
M2450   LD (BORDCR),A           Set the system variable and
        RET                     return.
```

RESET Command Routine

In theory, this would reset the computer. In practice, it's essentially commented out. RESET * pulls some characters and then peters out. RESET # checks for a valid channel number but does not actually reset it. And a plain RESET does nothing.

Wes Brzozowski states this section "would have CLOSEd all 16 streams & rebuilt the SYSCON table after the execution of the BASIC command `RESET *`. The rebuilt table would have been a 'cold reset', using the EXROM routine BLDSCT."

RESET (RSET)

```
M2454   RST $18                 Get current character
        CP $2A                  Is it *?
        JR NZ, CHK-#            No, check to see if it's a channel reset.
        CALL NEXT-CHAR
        CALL CHECK-END
        RET
```

RESET-STREAMS

Would have closed all 16 streams and rebuilt the SYSCON table after execution "RESET *" in BASIC. The rebuilt table would have been a "cold reset," using the EXROM routine BLDSCT.

```
M2460   LD A,$10                Value for 16 streams.
        LD HL,CH_0              Start at channel 0.
```

DO-STREAMS

```
M2465   CALL RESTORE-STREAM     Reset the stream
        INC HL                  Increment the channel pointer.
        INC HL
        DEC A                   Decrement the channel counter.
```

```
        JR NZ, DO-STREAMS       If not zero, loop back around.
        LD HL, $09F4
        PUSH HL
        LD B, $FE
```

RESET-EXBNK

```
M2473   LD C,$88                Set up to reset the expansion
        PUSH BC                 bus banks.
        LD BC,$0000
        PUSH BC
        PUSH BC
        CALL CALL_BANK
        RET
```

CHK-#

```
M247F   CP $23                  Is it a #?
        JR Z, GET-PARAM         Yes, go look for the parameters.
        CALL CHECK-END          No, see if we're at the end of the line.
        RET
```

RESET-MODE

Would have performed a "warm reset" of the SYSCON table after the BASIC command RESET. This would have used the EXROM routine RESET-SYSCONF.

```
M2487   LD HL,$0C4C             Set up parameters to perform
        PUSH HL                 a warm reset.
        LD BC,$FEFE
        PUSH BC
        LD BC,$0000
        PUSH BC
        PUSH BC
        CALL CALL_BANK
        RET
```

Reset Parameters Subroutine

GET-PARAM

```
M2498   RST $20                 Get next character
        CALL EXPT-1NUM          Make sure it's a number.
        CALL CHECK-END          Are we at the end of the line?
        CALL FIND-INT1
        CP $11
        JR NC, REPORT-O
        AND A
        JP M, REPORT-O
        ADD A,A
        ADD A,$16
        LD L,A
        LD H,$5C
        LD E,(HL)
```

```
        INC HL
        LD D,(HL)
        LD A,D
        OR E
        JR NZ, SKIP-ERR
```

REPORT-O

```
M24B7   RST $08                 Error: Invalid stream.
        DEFB $17
```

SKIP-ERR

```
M24B9   LD A,D
        CP $80
        RET C
        JP REPORT-J
```

EXPBNK-RESET

Would have run expansion bank code upon execution of the BASIC command "RESET # stream".

```
M24C0   SUB $80
        LD D,A
        LD DE,(SYSCON)
        ADD HL,DE
        INC HL
        LD B,(HL)
        LD D,$00
        LD E,$12
        ADD HL,DE
        PUSH HL
        JR RESET-EXBNK
```

New Device Routines

These routines were intended to support devices attached to the expansion bank, like disk drives. Each device would have had its own device code. The code checks to see if it's an extended save, load, verify or merge command, all of which would have an asterisk (*) between the command and the string with the device code, options and file name.

The *Third Party Software Guide* listed a number of potential devices, including a modem, stringy floppy, floppy disk, hard disk, RS232 interface, Centronics (parallel) printer interface, 80 column printer, local area network, and RAM, in addition to the printer, keyboard and screen.

These routines call XPASSING, in EXROM, which would have passed the information along to the device.

NEWDEV
Vector into save, load, verify and merge routine in EXROM.

```
M24D2   RST $18          Get current character.
        CP '*'           Is it an asterisk? The asterisk is used to leverage
                         disk commands.
        JP NZ, DOSAVE    No, save the program to tape.
        RST $20          Get next character.
        CALL EXPT-EXP    Check if the character was a string.
        CP ','           Is it a comma?
        JP NZ, REPORT-C  No, report BAD BASIC.
        CALL SYNTAX-Z    Check if interpreting.
        JR NZ, CALL-REP-J Yes, jump forward (invalid IO dev).
        CALL SKIPIT      No, skip the rest of the line.
        CALL CHECK-END
```

CALL-REP-J
```
M24EC   JR REPORT-J      This is probably placeholder; in a fully-
                         debugged and complete ROM, execution
                         would continue to the routine below.
```

DEVICE-COMMAND
This code is never called. If the call to REPORT-J above this were replaced with a NOP, execution would continue here, picking up string parameters and eventually invoking the NEW-DEV-SAVE and NEW-DEV-LOAD routines.

Would have passed the information in an extended LOAD or SAVE command (ie, LOAD * "D",list of information) onto the stack, and then CALLed the NEW-DEV-SAVE or NEW-DEV-LOAD routine, perhaps for a disk or microdrive, in an expansion bank. This would have CALLed the routine PASSEM, which is also never used. This is fortunate, because it tries to CALL an EXROM routine with a mislinked address, and also has a RET command missing, following a CALL at M25DE.

```
M24EE   CALL STK-FETCH
        DEC BC
        LD A,B
        OR C
        JR NZ,REPORT-J
        LD A,(DE)
        AND $DF
        LD C,A
        CALL SRCHSC
        JP NC,REPORT-J
        PUSH HL
        LD DE,$0014
        ADD HL,DE
        LD A,(HL)
        BIT 1,A
        JP Z, REPORT-J
        POP HL
        EX DE,HL
```

```
        CALL PASSEM
        EX DE,HL
        LD A,(TADDR)
        AND A
        CP $00
        JR C,NEW-DEV-SAVE
        JR Z,NEW-DEV-LOAD
        ADD A,$D4
        LD C,A
```

EXPBNK-DEV
Routine for handling devices attached via expansion bank.

```
M251E   PUSH BC
        LD D,(HL)
        LD E,$88
        LD BC,$000C
        ADD HL,BC
        LD C,(HL)
        INC HL
        LD B,(HL)
        PUSH BC
        PUSH DE
        LD HL,(STKEND)
        DEC HL
        LD C,(HL)
        INC C
        LD (STKEND),HL
        LD B,$00
        PUSH BC
        LD BC,$0000
        PUSH BC
        CALL CALL_BANK          call bank
        RET
```

NEW-DEV-SAVE
New device save. Routine to support SAVE for devices not defined in ROM.

```
M253F   LD C,$F8                Code for SAVE.
        JR EXPBNK-DEV           Jump to routine for handling expansion bank
                                devices.
```

NEW-DEV-LOAD
New device load. Routine to support LOAD from devices not defined in ROM.

```
M2543   LD C,$EF                Code for LOAD.
        JR EXPBNK-DEV           Jump to routine for handling expansion bank
                                devices.
```

DOSAVE
Puts the address of SAVE-LOAD-VERIFY-MERGE (in EXROM) in to BC, checks whether display file 2 is active, and invokes the routine via the function dispatcher.

```
M2547   POP AF
        LD BC, $01AB            SLVM address in EXROM
        PUSH BC
        LD BC,$FEFE             dock bank, chunk 0 active
        PUSH BC
        LD BC,$0000             no in or out params
        PUSH BC
        PUSH BC
        LD A,(VIDMOD)           Determine where the function
        AND A                   dispatcher is.
        JR NZ,DOSAVE-DF2        Has been moved to high memory, so
                                call there.
        CALL CALL_BANK          CALL_BANK
```

CK-END
```
M255E   CALL CHECK-END          Is this the end of the line?
        RET
```

DOSAVE-DF2
Call SLVM via the function dispatcher in high memory.

```
M2562   CALL CALL_BANK-HI       CALL_BANK when DF2 active
        JR CK-END
```

REPORT-J
```
M2567   RST $08                 Error: Invalid IO device.
        DEFB $12
```

SKIPIT

Skip the rest of a line. Get characters until a newline ($0D), a colon (:) or quote mark is found.

SKIPIT
```
M2569   LD A,(ARSFLG)           Reset AROS string quote flag.
        RES AROSQTE,A
        LD (ARSFLG),A
        PUSH BC                 Briefly save BC.
        RST $18                 Get current character.
```

SKIPIT-L
Loop until the end of a sub-line, end of line or quote is found.

```
M2573   CP $22              jump if a quote
        JR Z, SKPT-NOT-EOL
        CP ':'              jump if colon (sub-line marker)
        JR Z, SKPT-NOT-EOL
        CP $0D              jump if newline
        JR Z, SKPT-NOT-EOL
        RST $20             Get next character
        JR SKIPIT-L         and loop back around
```

SKPT-NOT-EOL

```
M2582   CP ':'              Jump if not end of a sub-line.
        JR NZ, SKPT-FP-CHK
        LD A,(ARSFLG)       Jump if still in a string literal
        BIT AROSQTE,A       delimited by quote marks.
        JR NZ, SKPT-NEXTCHAR
```

SKPT-FP-CHK
Look back as many as five characters to determine whether or not the quote, :, or newline are actually components of a floating point number

```
M258D   PUSH HL             Save current character pointer.
        LD B,$05            Set up to look for "number slug"
```

SFP-LOOP

```
M2590   DEC HL              Point back.
        LD A,(HL)
        CP $0E              Look back 5 characters to determine
        JR Z, SKPT-RESTHL   if the character was part of a
        DJNZ SFP-LOOP       floating point number.
        POP HL              Restore the character pointer.
        RST $18             Get current character.
        CP $22              Jump if not a quote.
        JR NZ, SKPT-RETURN
```

SKPT-QUOTE
at this point we have found a quote in a literal string. we check the quote flag to determine whether this is the first or second quote found. if the first, set the quote flag, else reset it

```
M259E   LD A,(ARSFLG)       Jump to reset the
        BIT AROSQTE,A       AROS quote flag.
        JR NZ, RES-AROSQTE
        SET AROSQTE,A       No previous quote, so
        LD (ARSFLG),A       set the flag and move on
        JR SKPT-NEXTCHAR    to the next character.
```

RES-AROSQTE

```
M25AC   RES AROSQTE,A       Reset the AROS quote flag
        LD (ARSFLG),A       land move on.
        JR SKPT-NEXTCHAR
```

SKPT-RETURN

```
M25B3   POP BC              Restore BC.
        RET                 Return to caller. HL and CH_ADD point to the
                            next character to be interpreted.
```

SKPT-RESTHL

```
M25B5   POP HL              restore the character pointer
```

SKPT-NEXTCHAR

```
M25B6   RST $20             Get next character.
        JR SKIPIT-L         Back around for more characters.
```

PASSEM

Pass data to an expansion bank, via PASSING in EXROM.

```
M25B9   LD BC,$FEFE         Enable the EXROM for chunk 0.
        CALL BANK_ENABLE
        CALL XPASSING       Mislinked address ($0F09). This is in the middle of
                            an instruction. Landing at this address would
                            result in LD (BC),A, followed by a jump to OUT-
                            CODE, which would (eventually) send what in A to
                            the selected expansion bank. This address should
                            be $0F43.
        LD BC,$FF00         Enable the home ROM in all chunks.
        CALL BANK_ENABLE
```

CAT Command

CAT

```
M25C8   LD B,$CF            Put “CAT” token in B.
        JR CAT-ETC          Jump ahead to complete.
```

FORMAT Command

FORMAT

```
M25CC   LD B,$D0            Put "FORMAT" token in B.
        JR CAT-ETC          Jump ahead to complete.
```

MOVE Command

MOVE

```
M25D0   LD B,$D1            Put "MOVE" token in B.
        JR CAT-ETC          Jump ahead to complete.
```

ERASE Command

ERASE

```
M25D4   LD B,$D2            Put "ERASE" token in B then fall through to CAT-
                            ETC.
```

CAT-ETC

```
M25D6   CALL SYNTAX-Z       Check if interpreting.
        JR NZ, REPORT-J     Goes to error routine but would have gone
                            to the routine that passed parameters.
        CALL SKIPIT         Skip the rest of the line.
        CALL CHECK-END      Returns here if interpreting.
```

REPORT-J

```
M25E1   JP REPORT-J         Error - INVALID IO DEVICE. Removing this
                            would allow the following routine to work and
                            complete these extended commands.
```

Pass Disk Parameters

This is part of the code that would have passed parameters from the BAS1C commands CAT, FORMAT, MOVE, and ERASE.

The blocking JP at M25E1, is where Timex deleted code to make room in the HOME ROM.[1]

```
M25E4   LD BC,$000C         offset into jump table
```

CALL_BANK with HL pointing to a jump table, BC containing an offset into it and DE containing the bank and horizontal select respectively. the top of the FP stack at (STKEND) contains a byte with the number of parameters out expected

```
        ADD HL,BC
        LD C,(HL)           |get jump address
        INC HL              |
        LD B,(HL)
        PUSH BC             push address onto stack
        PUSH DE             push horiz sel and bank
        LD HL,(STKEND)
        DEC HL              |get the number of params out
        LD C,(HL)           |from CALL_BANK from the
        INC C               |free space
        LD (STKEND),HL
        LD B,$00            number of parameters out
        PUSH BC
        LD BC,$0000         no parameters in
```

[1] "The Mystery of the Missing 253: Conclusion."

```
        PUSH BC
        CALL CALL_BANK          CALL_BANK
        RET
```

REPORT-J

```
        RST $08                 Error: Invalid IO device.
        DEFB $12
```

Pixel Address Subroutine

Screen address calculator. Called by POINT subroutine and PLOT command routine. Entered with the coordinates of a pixel in BC and returns with address of the display file byte which contains that pixel in HL holding the and position of the pixel within the byte in A (zero for left-hand = MS end, one for right-hand = LS end).

PIXEL-ADD (SCRMBL)

```
M2603   LD A,$AF            Test that the y coordinate (in B) is not
        SUB B               greater than 175.
        JP C, REPORT-B
        LD B,A              B now contains 175 minus y.
        AND A               A holds B.
        RRA                 Shift A to the right, dropping bit 0.
        SCF
        RRA                 Shift again, A is 10b7b6b5b4b3b2.
        AND A
        RRA                 A is 010b7b6b5b4b3.
        XOR B
        AND $F8             Finally 010b7b6b2b1b0, so that H
        XOR B               becomes 64+8*INT(B/64) + B (mod 8), the
        LD H,A              high byte of the pixel address.
        LD A,C              C contains X.
        RLCA                A starts as c7c6c5c4c3c2c1c0.
        RLCA
        RLCA                And is now c2c1c0c7c6c5c4c3.
        XOR B
        AND $C7
        XOR B               Now c2c1b5b4b3c5c4c3.
        RLCA
        RLCA                Finally b5b4b3c7c6c5c4c3, so that L
        LD L,A              becomes 32*INT(B(mod64)/8)+INT(x/8),
        LD A,C              the low byte.
        AND $07             A holds x(mod8). The pixel is bit 7 within
        RET                 the byte.
```

Point Subroutine

Called by the POINT function in SCANNING. Entered with the coordinates of a pixel on the calculator stack and returns a last value of 1 if that pixel is ink color or 0 if it is paper color.

POINT-SUB (F_PNT)

```
M2624   CALL STK-TO-BC      Y coordinate to B, X to C.
        CALL PIXEL-ADD      Pixel address in HL.
        LD B,A              B will count A+1 loops to get the desired
        INC B               bit of HL to location 0.
        LD A,(HL)
```

POINT-LP

```
M262D   RLCA
        DJNZ POINT-LP
        AND $01             Bit 1 for INK, 0 for PAPER.
        JP STACK-A          Put A on calculator stack.
```

PLOT Command Routine

This routine consists of a main subroutine plus one line to call it and one line to exit from it. The main routine is used twice by CIRCLE and the subroutine is called by DRAW. The routine is entered with the coordinates of a pixel on the calculator stack. It finds the address of that pixel and plots it, taking account of the status of INVERSE and OVER held in the P-FLAG.

PLOT

```
M2635   CALL STK-TO-BC      Y coordinate to B, X to C.
        CALL PLOT-SUB       Call the subroutine.
        JP TEMPS            Exit, setting temporary colors.
```

PLOT-SUB (PLOTBC)

```
M263E   LD (XCOORD),BC      Set the system variable.
        CALL PIXEL-ADD      Pixel address to HL.
        LD B,A              B will count A+1 loops to get a zero
        INC B               in the correct place in A.
        LD A,$FE            Enter the zero.
```

PLOT-LOOP

```
M2649   RRCA                Line up the pixel bit position in the byte.
        DJNZ PLOT-LOOP
        LD B,A              Copy to B.
        LD A,(HL)           Put pixel-byte in A.
        LD C,(IY+OPFLAG)    Get PFLAG and
        BIT XOR_CH,C        test for OVER.
        JR NZ, PL-TST-IN    Jump if OVER 1.
        AND B               OVER 0; make the pixel zero.
```

PL-TST-IN

```
M2656   BIT INV_CH,C        Test for INVERSE.
        JR NZ, PLOT-END     INVERSE 1 leaves the pixel as it was
                            (OVER 1) or zero (OVER 0).
        XOR B               INVERSE 0 set the pixel to its complement
        CPL                 (OVER 1) or 1 (OVER 0).
```

PLOT-END

```
M265C   LD (HL),A               Byte is saved; other bits are unchanged.
        JP PO-ATTR
```

STK-TO-BC Subroutine

Loads two floating point numbers into the BC register pair. It is used to pick up parameters in the range +00-+FF. It also obtains in DE the 'diagonal move' values (+/-1,+/-1) which are used in the line drawing subroutine of DRAW. Return with X and Y in D and E and the sign of the numbers in B and C ($01 signifies positive, $FF signifies negative)

STK-TO-BC (GET_XY)

```
M2660   CALL STK-TO-A           Get the first parameter
        LD B,A                  Save the number to B.
        PUSH BC                 Save number (B) and the sign info (C)
        CALL STK-TO-A           Get the second parameter.
        LD E,C                  Load the sign info into E.
        POP BC                  Get the first parameter back.
        LD D,C                  Put the first parameter sign in D.
        LD C,A                  Put the second parameter in C.
        RET
```

STK-TO-A Subroutine

Loads the A register with the floating point number held at the top of the calculator stack. The number must be in the range 00-FF.

STK-TO-A

```
M266D   CALL FP-TO-A            Put the modulus of rounded last value in A
        JP C, REPORT-B          if possible; else report error.
        LD C,$01                Put a 1 in C for positive last value.
        RET Z                   Return if value was positive.
        LD C,$FF                Otherwise, change C to $FF (minus 1).
        RET
```

CIRCLE Command Routine

This routine draws an approximation to the circle with centre co-ordinates X and Y and radius Z. These numbers are rounded to the nearest integer before use. Thus Z must be less than 87.5, even when (X,Y) is in the centre of the screen. The method used is to draw a series of arcs approximated by straight lines. It is illustrated in the BASIC program in the appendix. The notation of that program is followed here. CIRCLE has four parts: I. Tests the radius. If its modulus is less than 1, just plot X,Y; II. Calls CD-PRMS-1 at 2470-24B6, which is used to set the initial parameters for both CIRCLE and DRAW; III. Sets up the remaining parameters for CIRCLE, including the initial displacement for the first 'arc' (a straight line in fact); IV. Jumps into DRAW to use the arc-drawing loop at 2420-24FA. Parts i. to iii. will now be explained in turn. i. 2320-23AA. The radius, say Z', is obtained from the calculator stack. Its modulus Z is formed and used from now on. If Z is less than 1, it is deleted from the stack and the point X,Y is plotted by a jump to PLOT.

CIRCLE

```
M2679   RST $18                 Get current character,
```

```
         CP $2C                  test for comma.
         JP NZ,REPORT-C          Error - BAD BASIC
         RST $20                 Get next character (radius).
         CALL EXPT-1NUM          Put radius on calculator stack.
         CALL CHECK-END          Move to next statement if checking syntax.
         RST $28                 Call calculator.
         DEFB $2A                ABS
         DEFB $3D                FLOAT
         DEFB $38                QUIT
         LD A,(HL)               Get exponent of radius.
         CP $81                  Is it less than 1?
         JR NC, C-R-GRE-1        If not, jump ahead.
         RST $28                 Call calculator and delete from stack.
         DEFB $02                DROP
         DEFB $38                QUIT
         JR PLOT                 Just plot point X, Y.
```

C-R-GRE-1

```
M2694    RST $28                 Call calculator.
         DEFB $A3                CONST 3 (PI/2)
         DEFB $38                Increase exponent to 83, changing
         LD (HL),$83             PI/2 into 2*PI.
         RST $28
         DEFB $C5                T -> MEM 5
         DEFB $02                DROP
         DEFB $38                QUIT
         CALL CD-PRMS1
         PUSH BC
         RST $28                 Call calculator.
         DEFB $31                DUP
         DEFB $E1                MEM 1 -> T
         DEFB $04                TIMES
         DEFB $38                QUIT
         LD A,(HL)
         CP $80
         JR NC, C-ARC-GE1
         RST $28                 Call calculator.
         DEFB $02                DROP
         DEFB $02                DROP
         DEFB $38                QUIT
         POP BC
         JP PLOT
```

C-ARC-GE1

```
M26B3    RST $28                 Call calculator.
         DEFB $C2                T -> MEM 2
         DEFB $01                SWAP
         DEFB $C0                T -> MEM 0
         DEFB $02                DROP
```

```
DEFB $03        SUB
DEFB $01        SWAP
DEFB $E0        MEM 0 -> T
DEFB $0F        ADD
DEFB $C0        T -> MEM 0
DEFB $01        SWAP
DEFB $31        DUP
DEFB $E0        MEM 0 -> T
DEFB $01        SWAP
DEFB $31        DUP
DEFB $E0        MEM 0 -> T
DEFB $A0        CONST 0 (0)
DEFB $C1        T -> MEM 1
DEFB $02        DROP
DEFB $38        QUIT
INC (IY+(MEM2-Y))
CALL FIND-INT1
LD L,A
PUSH HL
CALL FIND-INT1
POP HL
LD H,A
LD (XCOORD),HL
POP BC
JP DRW-STEPS
```

(The stack now holds X+Z, Y - Z*SIN (PI/A), Y - Z*SIN (PI/A), X+Z).

DRAW Command Routine

Routine is entered with the coordinates of a point in COORDS. If only two parameters X, Y are given with the DRAW command, it draws an approximation to a straight line from the point X0, Y0 to X0+X, Y0+Y. If a third parameter G is given, it draws an approximation to a circular arc from X0, Y0 to X0+X, Y0+Y turning anti-clockwise through an angle G radians.

The routine has four parts:
1. Just draws a line if only 2 parameters are given or if the diameter of the implied circle is less than 1;
2. Calls CD-PRMS1 at 247D-24B6 to set the first parameters;
3. Sets up the remaining parameters, including the initial displacements for the first arc;
4. Enters the arc-drawing loop and draws the arc as a series of smaller arcs approximated by straight lines, calling the line-drawing subroutine at 24B7-24FA as necessary.

Two subroutines, CD-PRMS1 and DRAW-LINE, follow the main routine. The above 4 parts of the main routine will now be treated in turn.

Part 1
If there are only 2 parameters, a jump is made to LINE-DRAW at 2477. A line is also drawn if the quantity Z=(ABS X + ABS Y)/ABS SIN(G/2) is less than 1. Z lies between 1 and 1.5 times the diameter of the implied circle. In this section mem-0 is set to SIN (G/2), mem-1 to Y, and mem-5 to G.

DRAW

```
M26DB   RST $18             Get current character.
        CP $2C              Is it a comma?
        JR Z, DR-3-PRMS     Yes, jump ahead.
        CALL CHECK-END      Move to next statement if checking syntax.
        JP LINE-DRAW        Just draw a line.
```

DR-3-PRMS

```
M26E6   RST $20             Get next character (the angle).
        CALL EXPT-1NUM      Move the angle to the calculator stack.
        CALL CHECK-END      Move to next statement if checking syntax.
        RST $28             Call calculator.
        DEFB $C5            T -> MEM 5
        DEFB $A2            CONST 1 (0.5)
        DEFB $04            TIMES
        DEFB $1F            SIN
        DEFB $31            DUP
        DEFB $30            NOT
        DEFB $30            NOT
        DEFB $00,$06        IFJUMP 26FC
        DEFB $02            DROP
        DEFB $38            QUIT
        JP LINE-DRAW        Line X0, Y0 to X0+X, Y0+Y
```

DR-SIN-NZ

```
M26FC   DEFB +C0,st-mem-0   (SIN (G/2) is copied to mem-0)
        DEFB +02,delete     X, Y are now on the stack.
        DEFB +C1,st-mem-1   (Y is copied to mem-1).
        DEFB +02,delete     X
        DEFB +31,duplicate  X, X
        DEFB +2A,abs        X, X' (X' = ABS X)
        DEFB +E1,get-mem-1  X, X', Y
        DEFB +01,exchange   X, Y, X'
        DEFB +E1,get-mem-1  X, Y, X', Y
        DEFB +2A,abs        X, Y, X', Y' (Y' = ABS Y)
        DEFB +0F,addition   X, Y, X'+Y'
        DEFB +E0,get-mem-0  X, Y, X'+Y', SIN (G/2)
        DEFB +05,division   X, Y, (X'+Y')/SIN (G/2)=Z', say
        DEFB +2A,abs        X, Y, Z (Z = ABS Z')
        DEFB +E0,get-mem-0  X, Y, Z, SIN (G/2)
        DEFB +01,exchange   X, Y, SIN (G/2), Z
        DEFB +3D,re-stack   (Z is re-stacked to make sure
        DEFB +38,end-calc   that its exponent is available).
        LD A,(HL)           Get exponent of Z.
        CP $81
        JR NC, DR-PRMS
        RST $28             Call calculator.
        DEFB $02            DROP
        DEFB $02            DROP
```

```
DEFB $38                QUIT
JP LINE-DRAW
```

Part 2
Call CD-PRMS1. This subroutine saves in the B register the number of shorter arcs required for the complete arc, viz. A=4*INT (G'*SQR Z/8)+4, where G' = mod G, or 252 if this expression exceeds 252 (as can happen with a large chord and a small angle). So A is 4, 8, 12, ... , up to 252. The subroutine also stores in mem-0 to mem-4 the quantities G/A, SIN (G/2*A), 0, COS (G/A), SIN (G/A).

DR-PRMS
```
M271A   CALL CD-PRMS1
```

Part 3
Set up the rest of the parameters as follow. The stack will hold these 4 items, reading up to the top: X0+X and Y0+Y as end of last arc; then X0 and Y0 as beginning of first arc. Mem-0 will hold X0 and mem-5 Y0. Mem-1 and mem-2 will hold the initial displacements for the first arc, U and V; and mem-3 and mem-4 will hold COS (G/A) and SIN (G/A) for use in the arc-drawing loop. The formulae for U and V can be explained as follows. Instead of stepping along the final chord, of length L, say, with displacements X and Y, we want to step along an initial chord (which may be longer) of length L*W, where W=SIN (G/2*A)/SIN (G/2), with displacements X*W and Y*W, but turned through an angle - (G/2 - G/2*A), hence with true displacements: U = Y*W*SIN (G/2 - G/2*A) + X*W*COS (G/2 - G/2*A) Y = Y*W*COS (G/2 - G/2*A) - X*W*SIN (G/2 - G/2*A) These formulae can be checked from a diagram, using the normal expansion of COS (P - Q) and SIN (P - Q), where Q = G/2 - G/2*A.

```
PUSH BC
RST $28                 Call calculator.
DEFB $02                DROP
DEFB $E1                MEM 1 -> T
DEFB $01                SWAP
DEFB $05                DIV
DEFB $C1                T -> MEM 1
DEFB $02                DROP
DEFB $01                SWAP
DEFB $31                DUP
DEFB $E1                MEM 1 -> T
DEFB $04                TIMES
DEFB $C2                T -> MEM 2
DEFB $02                DROP
DEFB $01                SWAP
DEFB $31                DUP
DEFB $E1                MEM 1 -> T
DEFB $04                TIMES
DEFB $E2                MEM 2 -> T
DEFB $E5                MEM 5 -> T
DEFB $E0                MEM 0 -> T
DEFB $03                SUB
DEFB $A2                CONST 1 (0.5)
DEFB $04                TIMES
```

```
DEFB $31        DUP
DEFB $1F        SIN
DEFB $C5        T -> MEM 5
DEFB $02        DROP
DEFB $20        COS
DEFB $C0        T -> MEM 0
DEFB $02        DROP
DEFB $C2        T -> MEM 2
DEFB $02        DROP
DEFB $C1        T -> MEM 1
DEFB $E5        MEM 5 -> T
DEFB $04        TIMES
DEFB $E0        MEM 0 -> T
DEFB $E2        MEM 2 -> T
DEFB $04        TIMES
DEFB $0F        ADD
DEFB $E1        MEM 1 -> T
DEFB $01        SWAP
DEFB $C1        T -> MEM 1
DEFB $02        DROP
DEFB $E0        MEM 0 -> T
DEFB $04        TIMES
DEFB $E2        MEM 2 -> T
DEFB $E5        MEM 5 -> T
DEFB $04        TIMES
DEFB $03        SUB
DEFB $C2        T -> MEM 2
DEFB $2A        ABS
DEFB $E1        MEM 1 -> T
DEFB $2A        ABS
DEFB $0F        ADD
DEFB $02        DROP
DEFB $38        QUIT
LD A,(DE)
CP $81
POP BC
JP C, LINE-DRAW
PUSH BC
RST $28         Call calculator.
DEFB $01        SWAP
DEFB $38        QUIT
LD A,(XCOORD)
CALL STACK-A    Put A on calculator stack.
RST $28         Call calculator.
DEFB $C0        T -> MEM 0
DEFB $0F        ADD
DEFB $01        SWAP
DEFB $38        QUIT
LD A,(YCOORD)
```

```
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $C5            T -> MEM 5
        DEFB $0F            ADD
        DEFB $E0            MEM 0 -> T
        DEFB $E5            MEM 5 -> T
        DEFB $38            QUIT
        POP BC
```

Part 4

This is the arc-drawing loop. This is entered at 2439 with the co-ordinates of the starting point on top of the stack, and the initial displacements for the first arc in mem-1 and mem-2. It uses simple trigonometry to ensure that all subsequent arcs will be drawn to points that lie on the same circle as the first two, subtending the same angle at the centre. It can be shown that if 2 points X1, Y1 and X2, Y2 lie on a circle and subtend an angle N at the centre, which is also the origin of co-ordinates, then X2 = X1*COS N - Y1*SIN N, and Y2 = X1*SIN N + Y1*COS N. But because the origin is here at the increments, say Un = Xn+1 - Xn and Vn = Yn+1 - Yn, thus achieving the desired result. The stack is shown below on the (n+1)th pass through the loop, as Xn and Yn are incremented by Un and Vn, after these are obtained from Un-1 and Vn-1. The 4 values on the top of the stack at 2425 are, in DRAW, reading upwards, X0+X, Y0+Y, Xn and Yn but to save space these are not shown until 2439. For the initial values in CIRCLE, see the end of CIRCLE, above. In CIRCLE too, the angle G must be taken to be 2*PI.

DRW-STEPS

```
M2779   DEC B               B counts passes through the loop.
        JR Z, ARC-END       Jump when B has reached zero.
        JR ARC-START        Jump to loop start.
```

ARC-LOOP

```
M277E   RST $28             Call calculator.
        DEFB $E1            MEM 1 -> T
        DEFB $31            DUP
        DEFB $E3            MEM 3 -> T
        DEFB $04            TIMES
        DEFB $E2            MEM 2 -> T
        DEFB $E4            MEM 4 -> T
        DEFB $04            TIMES
        DEFB $03            SUB
        DEFB $C1            T -> MEM 1
        DEFB $02            DROP
        DEFB $E4            MEM 4 -> T
        DEFB $04            TIMES
        DEFB $E2            MEM 2 -> T
        DEFB $E3            MEM 3 -> T
        DEFB $04            TIMES
        DEFB $0F            ADD
        DEFB $C2            T -> MEM 2
        DEFB $02            DROP
        DEFB $38            QUIT
```

ARC-START

```
M2792   PUSH BC             Save arc counter.
        RST $28             Call calculator.
        DEFB $C0            T -> MEM 0
        DEFB $02            DROP
        DEFB $E1            MEM 1 -> T
        DEFB $0F            ADD
        DEFB $31            DUP
        DEFB $38            QUIT
        LD A,(XCOORD)
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $03            SUB
        DEFB $E0            MEM 0 -> T
        DEFB $E2            MEM 2 -> T
        DEFB $0F            ADD
        DEFB $C0            T -> MEM 0
        DEFB $01            SWAP
        DEFB $E0            MEM 0 -> T
        DEFB $38            QUIT
        LD A,(YCOORD)
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $03            SUB
        DEFB $38            QUIT
        CALL DRAW-LINE      Draw next 'arc'.
        POP BC              Restore loop counter.
        DJNZ ARC-LOOP       Keep drawing more arcs.
```

ARC-END

```
M27B8   RST $28             Call calculator.
        DEFB $02            DROP
        DEFB $02            DROP
        DEFB $01            SWAP
        DEFB $38            QUIT
        LD A,(XCOORD)
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $03            SUB
        DEFB $01            SWAP
        DEFB $38            QUIT
        LD A,(YCOORD)
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $03            SUB
        DEFB $38            QUIT
```

```
LINE-DRAW
M27D0   CALL DRAW-LINE
        JP TEMPS
```

Initial Parameters Subroutine

Called by both CIRCLE and DRAW to set their initial parameters. It is called by CIRCLE with X, Y and the radius Z on the top of the stack, reading upwards. It is called by DRAW with its own X, Y, SIN (G/2) and Z, as defined in DRAW i. above, on the top of the stack. In what follows the stack is only shown from Z upwards. The subroutine returns in B the arc-count A as explained in both CIRCLE and DRAW above, and in mem-0 to mem-5 the quantities G/A, SIN (G/2*A), 0, COS (G/A), SIN (G/A) and G. For a circle, G must be taken to be equal to 2*PI.

```
CD-PRMS1
M27D6   RST $28             Call calculator.
        DEFB $31            DUP
        DEFB $28            ROOT
        DEFB $34,$32,$00    LITERAL 8200000000
        DEFB $01            SWAP
        DEFB $05            DIV
        DEFB $E5            MEM 5 -> T
        DEFB $01            SWAP
        DEFB $05            DIV
        DEFB $2A            ABS
        DEFB $38            QUIT
        CALL FP-TO-A        A1 to A from the stack, if possible.
        JR C, USE-252       If A1rounds to 256 or more, use 252.
        AND $FC             4*INT(A1 / 4) to A.
        ADD A,$04           Add 4, giving arc-count A.
        JR NC, DRAW-SAVE    Jump if still under 256.

USE-252
M27EE   LD A,$FC            Just use 252.

DRAW-SAVE
M27F0   PUSH AF             Save the arc-count.
        CALL STACK-A        Put A on calculator stack.
        RST $28             Call calculator.
        DEFB $E5            MEM 5 -> T
        DEFB $01            SWAP
        DEFB $05            DIV
        DEFB $31            DUP
        DEFB $1F            SIN
        DEFB $C4            T -> MEM 4
        DEFB $02            DROP
        DEFB $31            DUP
        DEFB $A2            CONST 1 (0.5)
        DEFB $04            TIMES
        DEFB $1F            SIN
        DEFB $C1            T -> MEM 1
```

```
        DEFB $01        SWAP
        DEFB $C0        T -> MEM 0
        DEFB $02        DROP
        DEFB $31        DUP
        DEFB $04        TIMES
        DEFB $31        DUP
        DEFB $0F        ADD
        DEFB $A1        CONST 1 (256)
        DEFB $03        SUB
        DEFB $1B        NEGATE
        DEFB $C3        T -> MEM 3
        DEFB $02        DROP
        DEFB $38        QUIT
        POP BC          Restore arc-count to BC.
        RET
```

Line-Drawing Subroutine

Called by DRAW to draw an approximation to a straight line from the point X0, Y0 held in COORDS to the point X0+X, Y0+Y, where the increments X and Y are on the top of the calculator stack. Intersperses as many horizontal or vertical steps as are needed among a basic set of diagonal steps, using an algorithm that spaces the horizontal or vertical steps as evenly as possible.

DRAW-LINE (DRAW-L)

```
M2810   CALL STK-TO-BC      ABS Y to B, ABS X to C,
                            SGN Y to D, SGN X to E.
```

DRAWLN

```
M2813   LD A,C              Jump if ABS X is greater than or equal
        CP B                to ABS Y, so that the smaller goes to L,
        JR NC, DL-X-GE-Y    and the larger to H.
        LD L,C
        PUSH DE             Save diag step to DE.
        XOR A               Insert a vertical step in to DE.
        LD E,A
        JR DL-LARGER        Jump to set H.
```

DL-X-GE-Y

```
M281D   OR C                Return if ABS X and ABS Y are both zero.
        RET Z
        LD L,B              Smaller (ABS Y) goes to L.
        LD B,C              ABS X to B, for H.
        PUSH DE             Save the diagonal step here, too.
        LD D,$00            Horizontal step for DE.
```

DL-LARGER

```
M2824   LD H,B              Larger of ABS X, ABS Y to H.
```

The algorithm starts here. The larger of ABS X and ABS Y, say H, is put into A and reduced to INT (H/2). The H - L horizontal or vertical steps and L diagonal steps are taken (where L is the smaller of ABS X and ABS Y) in this way: L is added to A; if A now equals or exceeds H, it is reduced by H and a diagonal step is taken; otherwise a horizontal or vertical step is taken. This is repeated H times (B also holds H). Note that meanwhile the exchange registers H' and L' are used to hold COORDS.

```
        LD A,B              B to A as well as H.
        RRA                 A starts at INT (H/2).
```

D-L-LOOP

```
M2827   ADD A,L             L is added to A.
        JR C, D-L-DIAG      If 256 or more, jump for diagonal step.
        CP H                If A is less than H, jump for horizontal
        JR C, D-L-HR-VT     or vertical step.
```

D-L-DIAG

```
M282D   SUB H               Reduce A by H.
        LD C,A              Restore it to C.
        EXX                 Exchange registers.
        POP BC              Diag step to B'C'.
        PUSH BC             Save it.
        JR D-L-STEP         Jump to take the step.
```

D-L-HR-VT

```
M2834   LD C,A              Save A (unreduced) to C.
        PUSH DE             Step to stack.
        EXX                 Eexchange registers.
        POP BC              Step to B'C'.
```

D-L-STEP

```
M2838   LD HL,(XCOORD)      Take first step: put XCOORDS to H'L'.
        LD A,B              Y-step from B' to A.
        ADD A,H             Add in H'.
        LD B,A              Result to B'.
        LD A,C              Test that X step is in range.
        INC A
        ADD A,L             Add L' to C' in A, jump on
        JR C, D-L-RANGE     carry for further test.
        JR Z,REPORT-B       Zero after carry means X is out of range.
```

D-L-PLOT

```
M2845   DEC A               Restore true value to A.
        LD C,A              Value to C' for plotting.
        CALL PLOT-SUB       Plot the step.
        EXX                 Restore the main registers.
        LD A,C              C back to A to continue..
```

```
            DJNZ D-L-LOOP       Loop back for 8 steps.
            POP DE              Clear machine stack.
            RET

D-L-RANGE
M2850       JR Z, D-L-PLOT      Zero after carry means X is in range.

REPORT-B
M2852       RST $08             Error: Integer out of range.
            DEFB $0A
```

Expression Evaluation

The ZX Spectrum and TS 2068 have a comprehensive expression evaluator that accommodates a wide range of variable types, functions and operations.

This part of ROM is fairly slow as all potentials alternatives are considered as the expression is scanned and evaluated.

Simple strings are managed dynamically and old copies are reclaimed once they are redundant. As a result, there's no need for garbage collection, unlike other BASICs.

Scanning Subroutine

Produces an evaluation result of the 'next expression'. Result is returned as the 'last value' on the calculator stack.

For a numerical result, the last value will be the actual floating point number.

For a string result the last value will consist of a set of parameters:
- The first of the five bytes is unspecified
- the second and third bytes hold the address of the start of the string
- the fourth and fifth bytes hold the length of the string.

Bit 6 of FLAGS is set for a numeric result and reset for a string result.

When a next expression consists of only a single operand [A, RND, A$ (4, 3 TO 7)], then the last value is simply the value that is obtained from evaluating the operand.

When the next expression contains a function and an operand [CHR$ A, NOT A, SIN 1], the operation code of the function is stored on the machine stack until the last value of the operand has been calculated. This last value is then subjected to the appropriate operation to give a new last value.

In the case of there being an arithmetic or logical operation to be performed [A+B, A*B, A=B], then both the last value of the first argument and the operation code have to be kept until the last value of the second argument has been found. The calculation of the

last value of the second argument may also involve the storing of last values and operation codes whilst the calculation is being performed.

When a complex expression is evaluated [CHR$ (T+A - 26*INT ((T+A)/26)+65)], a hierarchy of operations to be performed is built up until the point is reached from which it must be dismantled to produce the final last value.

Each operation code has associated with it an appropriate priority code and operations of higher priority are always performed before those of lower priority.

The subroutine begins with the A register being set to hold the first character of the expression and a starting priority marker - zero - being put on the machine stack.

SCANNING (EXPRN)

```
M2854   RST $18              Get current character into A.
        LD B,$00             Starting priority marker is stacked.
        PUSH BC
```

Search the scanning function table (M294C) for the code in A. If the operator or token is found, jump to the handler for it. Four subroutines follow; they are called by routines from the scanning function table.

S-LOOP-1

```
M2858   LD C,A
        LD HL, $294C         Index into the scanning function
        CALL INDEXER         table with the code in C.
        LD A,C               Restore the code to A.
        JP NC, S-ALPHNUM     Jump if item not found.
        LD B,$00             Get offset value to make
        LD C,(HL)            required address and
        ADD HL,BC            jump to it.
        JP (HL)
```

S-QUOTE-S

The scanning quotes subroutine checks that every string quote is matched by another one.

```
M2868   CALL NEXTCH          Get the next character.
        INC BC               Increase length count.
        CP $0D               Is it a carriage return?
        JP Z, REPORT-C       Report error - BAD BASIC.
        CP $22               Loop back if not quote.
        JR NZ, S-QUOTE-S
        CALL NEXTCH          Get the next character.
        CP $22               Flag if it is another quote.
        RET
```

S-2-COORD
Called by S-SCREEN$, S-ATTR, S-POINT and STICK to make sure the required two coordinates are given in their proper form.

```
M287B   RST $20             Get next character with filtering
        CP '('              Is it an open parenthesis?
        JR NZ,S-RPORT-C     No, report the error.
        CALL NEXT-2NUM      Put the coordinates on the calculator
        RST $18             stack and get current character.
        CP ')'              Is it a close parenthesis? If not report error.
```

S-RPORT-C
```
M2886   JP NZ,REPORT-C      Error: BAD BASIC.
```

SYNTAX-Z (INTPTQ)
Checks the interpret flag. Returns Z flag=1 if syntax checking, Z flag=0 if interpreting statement.

```
M2889   BIT INTPT,(IY+OFLAGS)   Test if interpreting or checking syntax.
        RET
```

S-SCRN$-S (F_SCRN)
Used by S-SCREENS$ to find the character that appears at line x, column y of the screen. It only searches the character set 'pointed to' to CHARS.
Note: This is normally the characters +20 (space) to +7F (©) although the user can alter CHARS to match for other characters, including user-defined graphics.

```
M288E   CALL STK-TO-BC      Get the x and y screen coords.
        LD HL,(CHARS)       Get the pointer to the character bit map table.
        LD DE,$0100         Point to the bit map.
        ADD HL,DE           Point to the position in the character set.
        LD A,C
        RRCA                Do some math to get
        RRCA                the low byte of the
        RRCA                required screen address.
        AND $E0
        XOR B
        LD E,A
        LD A,C
        AND $18
        XOR $40
        LD D,A              DE now holds the screen address.
        LD B,$60            Count 96 characters.
```

S-SCRN-LP
```
M28A8   PUSH BC             Save the count.
        PUSH DE             And the screen pointer.
        PUSH HL             And the character set pointer.
        LD A,(DE)           Get the first row of screen character.
        XOR (HL)            Match with row from character set.
        JR Z, S-SC-MTCH     Jump if direct match found.
```

```
        INC A               Test for match with inverse character.
        JR NZ, S-SC-NXT     Jump if direct match found.
        DEC A               Restore A.
```

S-SC-MTCH

```
M28B3   LD C,A              Inverse status to C.
        LD B,$07            B counts through the other 7 rows.
```

S-SC-ROWS

```
M28B6   INC D               Move DE to next row.
        INC HL              Move HL to next row.
        LD A,(DE)           Get the screen row.
        XOR (HL)            Match with row from ROM.
        XOR C               Include inverse status.
        JR NZ, S-SCR-NXT    Jump if row fails to match.
        DJNZ S-SC-ROWS      Jump back until all rows are done.
        POP BC              Discard character set pointer.
        POP BC              Discard screen pointer.
        POP BC              Final count to BC.
        LD A,$80            Last character code is set plus one.
        SUB B               A now hold required code.
        LD BC,$0001         One space is needed in workspace.
        RST $30             Make the space.
        LD (DE),A           Put the character in it.
        JR S-SCR-STO        Jump to stack the character.
```

S-SCR-NEXT

```
M28CC   POP HL              Restore character set pointer.
        LD DE,$0008         Move it 8 bytes, to next character.
        ADD HL,DE
        POP DE              Restore the screen pointer.
        POP BC              And the counter.
        DJNZ S-SCRN-LP      Loop back for the 96 characters.
        LD C,B              Stack the empty string.
```

S-SCR-STO

```
M28D6   RET                 RET corrects an error in the Spectrum ROM here.
```

S-SATTR-S (F_ATTR)

Scanning attributes subroutine, called by S-ATTR to return the value of ATTR (x,y) which codes the attributes of line x, column y on the television screen.

```
M28D7   CALL STK-TO-BC      x to C, y to B.
        LD A,C              x is copied to A and the number
        RRCA                is manipulated to to get the
        RRCA                location in memory.
        RRCA
        LD C,A
        AND $E0
        XOR B
```

```
             LD L,A          Low byte of attribute address in L.
             LD A,C          More math to make the high byte.
             AND $03
             XOR $58
             LD H,A          Which goes in H.
             LD A,(HL)       Copy the attribute byte to A.
             JP STACK-A      Put A on calculator stack.
```

PI Routine

```
PI
M28ED        CALL SYNTAX-Z   jump if interpreting
             JR Z, PI-1      Exit routine if syntax checking.
             RST $28         Call calculator.
             DEFB $A3        CONST 3 (PI/2)
             DEFB $38        QUIT

PI-1
M28F5        JP S-NUMERIC
```

STICK Command Routine

Checks to see if the values passed are in range (1 or 2), reads the appropriate joystick through the AY-3-8912 registers and returns the value, if any.

```
STICK
M28F8        CALL S-2-COORD      Make sure stick value is a number.
             CALL NZ, READ-STICK
             RST $20             Get next character.
             JP S-NUMERIC        Mark as a numeric result.

READ-STICK
M2902        CALL STK-TO-BC      Put floating point value in BC.
             LD A,B              Copy to A.
             CALL TEST-STICK-ARG Is the stick device correct (1 or 2)?
             LD A,C
             CALL TEST-STICK-ARG Is the stick number correct (1 or 2)?
             LD D,C
             LD A,$0E
             OUT ($F5),A         Set up to read the joystick
             LD C,$F6            through the AY-3-8912 IO port.
             IN A,(C)
             CPL
             LD B,D
             DJNZ STICK-BITS
             AND $0F
             CP $0F
             JR C, STICK-R-1
             AND $00
```

```
STICK-R-1
M2922   CALL STACK-A          Put stick value on calculator stack.
        RET

STICK-BITS
M2926   RLCA                  Set the bit values to return.
        AND $01
        JR STICK-R-1

TEST-STICK-ARG
M292B   SUB $02               Test that the stick value in both
        ADC A,$00             positions is 2 or less.
        JR NZ, REPORT-A
        RET

REPORT-A
M2932   RST $08               Error: Invalid argument.
        DEFB $09
```

FREE Function Command

```
FREE
M2934   CALL SYNTAX-Z         Return if syntax checking.
        JR Z, FREE-1
        LD HL,(RAMTOP)        Get top of RAM.
        LD DE,(STKEND)        And bottom of the stack.
        AND A                 Clear A.
        SBC HL,DE             Subtract stack from RAMTOP.
        LD C,L                Put the value in BC.
        LD B,H
        CALL STACK-BC         Convert to floating point.

FREE-1
M2948   RST $20               Get next character.
        JP S-NUMERIC          Mark as numeric result.
```

Scanning Function Table

This table contains 8 functions and 4 operators. It thus incorporates 5 new Spectrum functions and provides a neat way of accessing some functions and operators which already existed on the ZX81.

```
M294C   DEFB $22,$24          S-QUOTE     M2971
        DEFB $28,$57          S-BRACKET   M29A6
        DEFB $2E,$FA          S-DECIMAL   M2A4B
        DEFB $2B,$1A          S-U-PLUS    M296D
        DEFB $7C,$16          S-STICK     M296B
        DEFB $7E,$12          S-FREE      M2969
        DEFB $A8,$5A          S-FN        M29B3
        DEFB $A5,$5B          S-RND       M29B6
```

```
        DEFB $A7,$88                    S-PI        M29E5
        DEFB $A6,$93                    S-INKEY$    M29F2
        DEFB $C4,$EA                    S-BIN       M2A4B
        DEFB $AA,$C3                    S-SCREEN$   M2A26
        DEFB $AB,$CB                    S-ATTR      M2A30
        DEFB $A9,$D2                    S-POINT     M2A39
        DEFB $00      end of table
```

Scanning Function Routines

S-FREE

```
M2969   JR FREE
```

S-STICK

```
M296B   JR STICK
```

S-U-PLUS

```
M296D   RST $20             For unary addition, get the next character
        JP S-LOOP-1         and jump back to main re-entry of SCANNING.
```

S-QUOTE

This routine deals with string quotes, whether simple like "name" or more complex like "a ""white"" lie" or the seemingly redundant VAL$ """a""".

```
M2971   RST $18             Get current character.
        INC HL              Point to the start of the string.
        PUSH HL             Save the start address.
        LD BC,$0000         Set length to zero.
        CALL S-QUOTE-S      Call the quote matching routine.
        JR NZ, S-Q-PRMS     Jump if zero: no more quotes.
```

S-Q-AGAIN

```
M297C   CALL S-QUOTE-S      Call again for 3rd quote.
        JR Z, S-Q-AGAIN     And again for fifth, seventh, etc.
        CALL SYNTAX-Z       If syntax checking, jump to reset
        JR Z, S-Q-PRMS      bit 6 of FLAGS and continue scanning.
        RST $30             Make space for the string and terminating
                            quote.
        POP HL              Get pointer to the start.
        PUSH DE             Save pointer to the first space.
```

S-Q-COPY

```
M2989   LD A,(HL)           Get a character from the string.
        INC HL              Point to the next one.
        LD (DE),A           Copy last one from work space.
        INC DE              Point to last space.
        CP $22              Is last character a question mark?
        JR NZ, S-Q-COPY     If not, jump to copy next one; if
        LD A,(HL)           next one is a quote, jump to copy
        INC HL              the one after it.
```

```
         CP $22
         JR Z, S-Q-COPY          Otherwise, finish copying.
```

S-Q-PRMS

```
M2997    DEC BC                  Get true length of BC.
```

Note that the first quote was not counted into the length; the final quote was, and is discarded now. Inside the string, the first, third, fifth, etc., quotes were counted in but the second, fourth, etc., were not.

```
         POP DE                  Restore start of copied string.
```

S-STRING

```
M2999    LD HL,FLAGS             This entry point is used when bit 6 is to be
         RES NUM,(HL)            reset and a string stacked if executing a line.
         BIT INTPT,(HL)          Interpreting?
         CALL NZ,STK-ST-$        Stack the string.
         JP S-CONT-2             Jump to continue scanning the line.
```

Note that in copying the string to the work space, every two pairs of string quotes inside the string ("") have been reduced to one pair of string quotes(").

S-BRACKET

```
M29A6    RST $20                 Get next character
         CALL SCANNING
         CP $29
         JP NZ,REPORT-C          Error - BAD BASIC
         RST $20                 Get next character
         JP S-CONT-2
```

S-FN

```
M29B3    JP S-FN-SBRN
```

S-RND (RND)

```
M29B6    CALL SYNTAX-Z           Jump if syntax checking
         JR Z,S-RND-END
         LD BC,(SEED)            Get the current seed value.
         CALL STACK-BC           Put it on the calculator stack.
         RST $28                 Call calculator.
         DEFB $A1                CONST 1 (256)
         DEFB $0F                ADD
         DEFB $34,$37,$16        LITERAL decimal 75
         DEFB $04                TIMES
         DEFB $34,$80,$41,$00,$00,$80    LITERAL 65537
         DEFB $32                INTDIV
         DEFB $02                DROP
         DEFB $A1                CONST 1 (256)
         DEFB $03                SUB
         DEFB $31                DUP
         DEFB $38                QUIT
         CALL FP-TO-BC           Use this last value to
```

```
              LD (SEED),BC          be new value for SEED.
              LD A,(HL)             Fetch exponent of last value.
              AND A                 Jump forward if zero
              JR Z, S-RND-END
              SUB $10               Divide  by 65536.
              LD (HL),A
```

S-RND-END

```
M29E3   JR S-PI-END           Jump past PI routine.
```

S-PI (F_PI)

Scanning-PI routine: unless syntax is being checked the value of 'PI' is calculated and forms the 'last value' on the calculator stack.

```
M29E5   CALL SYNTAX-Z         Jump if syntax checking.
        JR Z,S-PI-END
        RST $28               Call calculator.
        DEFB $A3              CONST 3 (PI/2)
        DEFB $38              QUIT
        INC (HL)
```

S-PI-END

```
M29EE   RST $20               Get next character
        JP S-NUMERIC
```

S-INKEY$ (F_INKY)

```
M29F2   LD BC,$105A           Priority $10, operation code $5A
        RST $20               for read-in subroutine.
        CP $23                If next character is #, jump.
        JP Z, S-PUSH-PO       There will be a numerical argument.
        LD HL,FLAGS
        RES NUM,(HL)          Reset bit 6 for string result.
        BIT INTPT,(HL)        Test for syntax checking.
        JR Z, S-INK$-EN       Jump if required.
        CALL KEY-SCAN         Fetch a key-value in DE.
        LD C,$00              Prepare empty string; stack it if too
        JR NZ,S-IK$-STK       many keys pressed.
        CALL K-TEST           Test the key value; stack empty string
        JR NC,S-IK$-STK       if unsatisfactory.
        DEC D                 $FF to D for L mode (bit 3 set).
        LD E,A                Key-value to E for decoding.
        CALL K-DECODE         Decode key value.
        PUSH AF               Save ASCII value.
        LD BC,$0001           One space needed in workspace.
        RST $30               Make the space.
        POP AF                Recall ASCII value.
        LD (DE),A             Prepare to stack it as string.
        LD C,$01              Length is 1.
```

S-IK$-STK

```
M2A1E   LD B,$00          Complete length.
        CALL STK-ST-$     Stack the string.
```

S-INK$-EN

```
M2A23   JP S-CONT-2       Jump forward.
```

S-SCREEN$

```
M2A26   CALL S-2-COORD    Check that 2 coordinates are provided.
        CALL NZ,S-SCRN$-S Call subroutine unless checking syntax; then
        RST $20           get next character and jump back.
        JP S-STRING
```

S-ATTR

```
M2A30   CALL S-2-COORD    Check that 2 coordinates are provided.
        CALL NZ, S-ATTR-S Call subroutine unless checking syntax; then
        RST $20           get next character and jump forward.
        JR S-NUMERIC
```

S-POINT

```
M2A39   CALL S-2-COORD    Check that 2 coordinates are provided.
        CALL NZ,POINT-SUB Call sub unless checking syntax; then
        RST $20           get next character and jump forward.
        JR S-NUMERIC
```

S-ALPHNUM

```
M2A42   CALL ALPHANUM     Is character alphanumeric?
        JR NC,S-NEGATE    Jump if not a letter or digit.
        CP 'A'            Jump if a letter; otherwise continue
        JR NC,S-LETTER    to S-DECIMAL.
```

S-DECIMAL

Deals with a decimal point or a number that starts with a digit. Takes care of the expression 'BIN', which is handled in the decimal to floating-point subroutine.

```
M2A4B   CALL SYNTAX-Z     Jump forward if line is being
        JR NZ,S-STK-DEC   executed.
```

If syntax is being checked, then the floating-point form has to be calculated and copied into the BASIC line. When a line is being executed, the floating-point form will always be available so it is copied to the calculator stack to form a 'last value'. During syntax checking:

```
        CALL DEC-TO-FP    Convert to floating point.
        RST $18           Set HL to point one past the last digit.
        LD BC,$0006       Open space for a floating point
        CALL MAKE-ROOM    number.
        INC HL            Point to the first free space.
        LD (HL),$0E       Enter the number maker code.
        INC HL            Point to the second location.
        EX DE,HL          Put pointer in DE.
```

```
              LD HL,(STKEND)        Get the old STKEND value.
              LD C,$05              5 bytes to move.
              AND A                 Clear carry.
              SBC HL,BC             New STKEND = old STKEND - 5.
              LD (STKEND),HL        Move number from calculator
              LDIR                  stack to the line.
              EX DE,HL              Put line pointer in HL.
              DEC HL                Point back to the last byte added.
              CALL TEMP-PTR1        Set CH_ADD.
              JR S-NUMERIC          Jump to indicate a numeric result.
```

During execution: step over the ASCII digits of a number until the slug character is reached.

S-STK-DEC

```
M2A73  RST $18                      Get current character.
```

S-SD-SKIP

```
M2A74  INC HL                       Move on to the next character
       LD A,(HL)                    until the number marker code
       CP $0E                       is found.
       JR NZ,S-SD-SKIP
       INC HL                       Point to the fp number.
       CALL STACK-NUM               Stack the fp value at (HL).
       LD (CH_ADD),HL               Update CH_ADD.
```

S-NUMERIC

A numeric result has now been identified, coming from RND, PI, ATTR, POINT or a decimal number, therefore bit 6 of FLAGS must be set.

```
M2A81  SET NUM,(IY+OFLAGS)          Set the numeric marker flag.
       JR S-CONT-1
```

Scanning Variable Routine

When a variable name has been identified a call is made to LOOK-VARS to look for extant variables in the variables area (or in the program area at DEF FN statements for a user-defined function FN).

If an appropriate numeric value is found then it is copied to the calculator stack using STACK-NUM.

String or string array entries have the appropriate parameters passed to the calculator stack by the STK-VAR subroutine (or in the case of a user-defined function, by the STK-F-ARG subroutine as called from LOOK-VARS).

S-LETTER

```
M2A87  CALL LOOK-VARS               Check in the variables area for matching entry.
       JP C, REPORT-2               Jump to error if variable was not found.
       CALL Z ,STK-VAR              Stack parameters of the string entry/return
                                    numeric element base address.
       LD A,(FLAGS)
```

```
        CP $C0              Test bits 6 and 7.
        JR C, S-CONT-1      One or both bits are reset.
        INC HL              A numeric value is to be stacked.
        CALL STACK-NUM
```

S-CONT-1

```
M2A9B   JR S-CONT-2
```

The character is tested against the code for '-', thus identifying the 'unary minus' operation.

Before the actual test, B is set to hold the priority +09 and C the operation code +D8 that are required for this operation.

S-NEGATE

```
M2A9D   LD BC,$09DB         Priority $09, operation code $D8.
        CP $2D              Is it a minus?
        JR Z, S-PUSH-PO     Jump forward if unary minus.
```

Next test the character against the code for 'VAL$', with priority 16 decimal and operation code 18 hex.

```
        LD BC,$1018         Priority $10, operation code $18.
        CP $AE              Is it VAL$?
        JR Z, S-PUSH-PO     Jump forward if it is VAL$.
```

Character must now represent one of the functions CODE to NOT, with codes +AF to +C3.

```
        SUB $AF             Range of the functions is changed from
                            $AF to $C3 to $0 to $14.
        JP C ,REPORT-C      Report an error if out of range.
```

The function 'NOT' is identified and dealt with separately from the others.

```
        LD BC,$04F0         Priority $04, operation code $F0.
        CP $14              Is it NOT?
        JR Z, S-PUSH-PO     Jump if so.
        JP NC,REPORT-C      Check the range again.
```

The remaining functions have priority 16 decimal. The operation codes for these functions are now calculated. Functions that operate on strings need bit 6 reset and functions that give string results need bit 7 reset in their operation codes.

```
        LD B,$10            Priority $10.
        ADD A,$DC           Function range now $DC to $EF.
        LD C,A              Transfer operation code.
        CP $DF              Separate CODE, VAL and LEN,
        JR NC, S-NO-TO-S    which operate on strings to give
        RES 6,C             numerical results.
```

S-NO-TO-S

```
M2AC5   CP $EE              Separate STR$ and CHR$, which
        JR C, S-PUSH-PO     operate on numbers to give string results.
        RES 7,C             Mark the operation codes.
```

The priority code and the operation code for the function being considered are now pushed on to the machine stack. A hierarchy of operations is thereby built up.

S-PUSH-PO

```
M2ACB   PUSH BC             Stack priority and operation code before
        RST $20             moving onto consider the next part of the
        JP S-LOOP-1         expression.
```

Scanning the line continues. Current argument may be followed by a '(', a binary operator or, if the end of the expression has been reached, then e.g. a carriage return character or a colon, a separator or a 'THEN'.

S-CONT-2

```
M2AD0   RST $18             Get current character.
```

S-CONT-3

```
M2AD1   CP $28              Jump forward if it is left parenthesis.
        JR NZ, S-OPERTR
```

If the 'last value' is numeric then the parenthesized expression is a true sub-expression and must be evaluated by itself. However if the 'last value' is a string then the parenthesized expression represents an element of an array or a slice of a string. A call to SLICING modifies the parameters of the string as required.

```
        BIT NUM,(IY+OFLAGS)     Jump forward if dealing with a number,
        JR NZ,S-LOOP            parenthesized expression.
        CALL SLICING            Modify the parameters of the last value.
        RST $20                 Get next character.
        JR S-CONT-3
```

If the present character is indeed a binary operator it will be given an operation code in the range +C3 - +CF hex, and the appropriate priority code.

S-OPERTR

```
M2AE1   LD B,$00            Original code to BC to index
        LD C,A              into table of operators.
        LD HL,OPTBL
        CALL INDEXER        Index into the table.
        JR NC, S-LOOP       Jump forward if no operation found.
        LD C,(HL)           Get required code from the table.
        LD HL,OPPRI-$C3     Pointer to the priority table.
        ADD HL,BC           Index into the table.
        LD B,(HL)           Find the appropriate priority.
```

At this stage there are:
- A 'last value' on the calculator stack.
- The starting priority market on the machine stack below a hierarchy, of unknown size, of function and binary operation codes. This hierarchy may be null.
- The BC register pair holding the 'present' operation and priority, which if the end of an expression has been reached will be priority zero.

Initially the 'last' operation and priority are taken off the machine stack and compared against the 'present' operation and priority.

If the 'present' priority is higher than the 'last' priority, then an exit is made from the loop as the 'present' priority is considered to bind tighter than the 'last' priority.

However, if the present priority is less binding then the operation specified as the 'last' operation is performed. The 'present' operation and priority go back on the machine stack to be carried round the loop again. In this manner the hierarchy of functions and binary operations that have been queued are dealt with in the correct order.

S-LOOP

```
M2AF2   POP DE              Get the last operation and priority.
        LD A,D              Priority goes to A.
        CP B                Compare last against present.
        JR C, S-TIGHTER     Exit to wait for argument.
        AND A               Are both priorities zero?
        JP Z,  GET-CHAR     Exit via GET-CHAR, making last value the result.
```

Before the 'last' operation is performed, the 'USR' function is separated into 'USR number' and 'USR string' according as bit 6 of FLAGS was set or reset when the argument of the function was stacked as the 'last value'.

```
        PUSH BC             Stack the present values.
        LD HL,FLAGS
        LD A,E              Last operation is compared against code for
        CP $ED              USR, which will give USR # unless modified.
        JR NZ,S-STK-LST     Jum if not USR.
        BIT NUM,(HL)        Is this a number?
        JR NZ, S-STK-LST    Jump if yes (USR #).
        LD E,$99            Modify last operation code offset.
```

S-STK-LST

```
M2B0A   PUSH DE             Stack last values briefly.
        CALL SYNTAX-Z       Do not perform operation if syntax checking.
        JR Z, S-SYNTEST
        LD A,E              Last operation code.
        AND $3F             Strip off bits 6 and 7 to convert operation code
        LD B,A              to calculator offset.
        RST $28             Call calculator.
        DEFB $3B            XEQTB
        DEFB $38            QUIT
        JR S-RUNTEST
```

An important part of syntax checking involves the testing of the operation to ensure that the nature of the 'last value' is of the correct type for the operation under consideration.

S-SYNTEST

```
M2B19   LD A,E              Get the last operation code.
        XOR (IY+OFLAGS)     Test against last operation code. Correct
        AND $40             if same syntax.
```

S-RPORT-C

```
M2B1F   JP NZ,REPORT-C      Jump if syntax fails.
```

Before jumping back to go round the loop again the nature of the 'last value' must be recorded in FLAGS.

S-RUNTEST

```
M2B22   POP DE              Get the last operation code.
        LD HL,FLAGS
        SET NUM,(HL)        Assume result to be numeric.
        BIT INTPT,E         Jump forward if interpreting.
        JR NZ, S-LOOPEND
        RES NUM,(HL)        Otherwise, it is a string.
```

S-LOOPEND

```
M2B2E   POP BC              Get the present value in to BC.
        JR S-LOOP
```

When the 'present' operation binds tighter, the 'last' and the 'present' values go back on the machine stack. However if the 'present' operation requires a string as its operand then the operation code is modified to indicate this requirement.

S-TIGHTER

```
M2B31   PUSH DE             Last values go to the stack.
        LD A,C              Get present operation code.
        BIT NUM,(IY+OFLAGS) Do not modify the operation code if dealing
        JR NZ, S-NEXT       with numeric operand.
        AND $3F             Clear bits 6 and 7.
        ADD A,$08           Increase the code by $08.
        LD C,A              Return the code to C.
        CP $10              Is the operation AND?
        JR NZ, S-NOT-AND    Jump if it is not.
        SET 6,C             AND requires a numeric operand.
        JR S-NEXT
```

S-NOT-AND

```
M2B46   JR C, S-RPORT-C     The operation -, *, /, ^ and OR are not
                            possible between strings.
        CP $17              Is the operation +?
        JR Z, S-NEXT        Jump if yes.
        SET 7,C             Other operations have a numeric result.
```

```
S-NEXT
M2B4E   PUSH BC          Present values go to the stack.
        RST $20          Get next character.
        JP S-LOOP-1      Go around loop again.
```

TABLE OF OPERATORS

```
OPTBL
M2B53   DEFB $2B,$CF          '+' ADD      M2B7A
        DEFB $2D,$C3          '-' SUB      M2B6E
        DEFB $2A,$C4          '*' MULT     M2B6F
        DEFB $2F,$C5          '/' DIV      M2B70
        DEFB $5E,$C6          '^' POW      M2B71
        DEFB $3D,$CE          '=' EQUALS   M2B79
        DEFB $3E,$CC          '>' GT       M2B77
        DEFB $3C,$CD          '<' LT       M2B78
        DEFB $C7,$C9          '<=' LTE     M2B74
        DEFB $C8, $CA         '>=' GTE     M2B75
        DEFB $C9, $CB         '<>' NE      M2B76
        DEFB $C5, $C7         OR           M2B72
        DEFB $C6, $C8         AND          M2B73
        DEFB $00
```

TABLE OF PRIORITIES (precedence table)

```
OPPRI
M2B6E   DEFB $06      "-"
        DEFB $08      "*"
        DEFB $08      "/"
        DEFB $0A      "^"
        DEFB $02      OR
        DEFB $03      AND
        DEFB $05      "<="
        DEFB $05      ">="
        DEFB $05      "<>
        DEFB $05      ">"
        DEFB $05      "<"
        DEFB $05      "="
        DEFB $06      "+"
```

Scanning Function Subroutine

Called by the 'scanning FN routine' to evaluate a user defined function which occurs in a BASIC line.

The subroutine can be considered in four stages:
1. The syntax of the FN statement is checked during syntax checking.
2. During line execution, a search is made of the program area for a DEF FN statement, and the names of the functions are compared, until a match is found - or an error is reported.
3. The arguments of the FN are evaluated by calls to SCANNING.
4. The function itself is evaluated by calling SCANNING, which in turn calls LOOK-VARS and so the 'STACK FUNCTION ARGUMENT' subroutine.

S-FN-SBRN

M2B7B	CALL SYNTAX-Z	Jump if not syntax checking.
	JR NZ, SF-RUN	
	RST $20	Get next character.
	CALL ALPHA	If it is not alphabetic, then
	JP NC,REPORT-C	report the error.
	RST $20	Get next character.
	CP $24	Is it $?
	PUSH AF	Save the zero flag to the stack.
	JR NZ, SF-BRKT-1	Jump if it was not $.
	RST $20	But get next character if it was.

SF-BRKT-1

M2B8E	CP $28	If the character is not '(', then
	JR NZ,SF-RPRT-C	report the error.
	RST $20	Get next character.
	CP $29	Is it ')'?
	JR Z,SF-FLAG-6	Jump if it was; there are no arguments.

SF-ARGMTS

M2B97	CALL SCANNING	Within this loop call SCANNING to check syntax of each argument and insert floating point numbers.
	RST $18	Get current character.
	CP $2C	If it is not a comma then jump, there
	JR NZ,SF-BRKT-2	are no more arguments.
	RST $20	Get next character in the argument.
	JR SF-ARGMNTs	Jump back to process it.

SF-BRKT-2

M2BA2	CP $29	Is the current character a ')'?

SF-RPRT-C

M2BA4	JP NZ,REPORT-C	No, report the error.

SF-FLAG-6

M2BA7	RST $20	Get next character.

```
        LD HL,FLAGS
        RES NUM,(HL)        Assume string; reset NUM.
        POP AF              Restore zero flag, jump if FN is
        JR Z,SF-SYN-EN      a string value.
        SET NUM,(HL)        Otherwise, set NUM of FLAGS.
```

SF-SYN-EN

```
M2BB2   JP S-CONT-2         Jump back to continue scanning the line.
```

During line execution, a search must first be made for a DEF FN statement.

SF-RUN

```
M2BB5   RST $20             Get the first character of the name.
        AND $DF             Reset bit 5 for upper case.
        LD B,A              Copy name to B.
        RST $20             Get next character.
        SUB $24             Subtract code for $.
        LD C,A              Copy to C (zero for string; non-zero for numeric).
        JR NZ, SF-ARGMT1    Jump if non-zero; numeric.
        RST $20             Get next character, the '('.
```

SF-ARGMT1

```
M2BC0   RST $20             Get first character of first argument.
        PUSH HL             Save pointer to it to the stack.
        LD HL,(PROG)        Point to the start of the program.
        DEC HL              Go back one location.
```

SF-FND-DF

```
M2BC6   LD DE,$00CE         Search for DEF FN.
        PUSH BC             Save the name and string status.
        CALL LOOK-PROG      Search the program.
        POP BC              Restore name and status.
        JR NC, SF-CP-DEF    Jump if found.
```

REPORT-P

```
M2BD0   RST $08             Error: FN without DEF FN.
        DEFB $18
```

When a DEF FN statement is found, the name and status of the two functions are compared: if they do not match, the search is resumed.

SF-CP-DEF

```
M2BD2   PUSH HL          Save pointer to the DEF FN character
                         in case search has to be resumed.
        CALL FN-SKPOVER  Get the name of the DEF FN function.
        AND $DF          Reset bit 5 for upper case.
        CP B             Does it match the FN name?
        JR NZ, SF-NOT-FD Jump if it does not match.
        CALL FN-SKPOVER  Get the next character in the DEF FN.
        SUB $24          Subtract code for $.
        CP C             Compare status with that of FN.
        JR Z, SF-VALUES  Jump if match found.
```

SF-NOT-FD

```
M2BE3   POP HL           Restore pointer to the DEF FN.
        DEC HL           Step back one.
        LD DE,$0200      Use search routine to find end of
        PUSH BC          DEF FN, preparing for the next
        CALL EACH-STMT   search; save the name and status.
        POP BC
        JR SF-FND-DF
```

The correct DEF FN statement has now been found. The arguments of the FN statement will be evaluated by repeated calls of SCANNING, and their 5 byte values (or parameters, for strings) will be inserted into the DEF FN statement in the spaces made there at syntax checking. HL will be used to point along the DEF FN statement (calling FN-SKPOVR as needed) while CH_ADD points along the FN statement (calling RST 0020, NEXT-CHAR, as needed).

SF-VALUES

```
M2BEF   AND A            IF HL is pointing to a $, move on to '('.
        CALL Z ,FN-SKPOVR
        POP DE           Discard pointer to DEF FN.
        POP DE           Get pointer to first argument of FN and
        LD (CH_ADD),DE   copy it to CH_ADD.
        CALL FN-SKPOVR   Move past the '('.
        PUSH HL          Save this pointer to the stack.
        CP $29           Is it pointing to a ')'?
        JR Z, SF-R-BR-2  If so jump: FN has no arguments.
```

SF-ARG-LP

```
M2C01   INC HL           Point to the next code and
        LD A,(HL)        put it in A.
        CP $0E           Is it the number marker code?
        LD D,$40         Set bit 6 of D for numerical argument.
        JR Z, SF-ARG-VL  Jump on zero: numerical argument.
        DEC HL           Ensure HL points to the $.
        CALL FN-SKPOVR
        INC HL           HL now points to the number marker.
        LD D,$00         Reset bit 6 of D: string argument.
```

SF-ARG-VL

```
M2C10   INC HL              Point to the 1st of 5 bytes of DEF FN.
        PUSH HL             Save pointer to the stack.
        PUSH DE             Save string status of the argument.
        CALL SCANNING       Evaluate the argument.
        POP AF              Get the number/string flag to A.
        XOR (IY+OFLAGS)     Test NUM against result of SCANNING.
        AND $40
        JR NZ,REPORT-Q      Give report Q if they don't match.
        POP HL              Put pointer to the 1st of 5 spaces DE.
        EX DE,HL
        LD HL,(STKEND)      Point HL to STKEND.
        LD BC,$0005         5 spaces to move.
        SBC HL,BC           Decrease STKEND by 5.
        LD (STKEND),HL      deleting the last value from the stack.
        LDIR                Copy the 5 bytes into the spaces in DEF FN.
        EX DE,HL            Point HL at the next code.
        DEC HL              Ensure that HL points to the character after
        CALL FN-SKPOVR      the 5 bytes.
        CP $29              Is it a ')'?
        JR Z, SF-R-BR-2     Jump if it is: no more arguments here.
        PUSH HL             It is a comma; save the pointer to it.
        RST $18             Get current character after last arg evaluated
                            from FN.
        CP $2C              If it is not a comma, jump: mismatched
                            arguments in FN and DEF FN.
        JR NZ, REPORT-Q
        RST $20             Advance CH_ADD to the next argument of FN.
        POP HL              Point HL to the comma in DEF FN.
        CALL FN-SKPOVR      Move HL to the next arg in DEF FN.
        JR SF-ARG-LP        Jump back to process the argument.
```

SF-R-BR-2

```
M2C43   PUSH HL             Save the pointer to ')' in DEF FN.
        RST $18             Get current character after last arg in FN.
        CP $29              Is it ')'?
        JR Z, SF-VALUE      If so, jump to evaluate. If not, report error.
```

REPORT-Q

```
M2C49   RST $08             Error: Parameter error.
        DEFB $19
```

Finally, the function itself is evaluated by calling SCANNING, after first setting DEFADD to hold the address of the arguments as they occur in the DEF FN statement. This ensures that LOOK-VARS, when called by SCANNING, will first search these arguments for the required values, before making a search of the variables area.

SF-VALUE

```
M2C4B   POP DE              Restore pointer to ')' in DEF FN.
        EX DE,HL            Put the pointer in HL.
```

```
LD (CH_ADD),HL          Put it in CH_ADD.
LD HL,(DEFADD)          Get old value of DEFADD.
EX (SP),HL              Stack it and get the start address and arguments
LD (DEFADD),HL          of DEF FN in to DEFADD.
PUSH DE                 Save address of ')' in FN.
RST $20                 Move CH_ADD past ')' and
RST $20                 '=' to start of expression for function in DEF FN.
CALL SCANNING           Evaluate the function.
POP HL                  Restore address of ')' in FN.
LD (CH_ADD),HL          Store it in CH_ADD.
POP HL                  Restore original value of DEFADD
LD (DEFADD),HL          and put it back into DEFADD.
RST $20                 Move to next char in the BASIC line.
JP S-CONT-2             Jump back to continue scanning.
```

Function Skipover Subroutine

Used by FN and STK-F-ARG to move HL along the DEF FN statement while leaving CH_ADD undisturbed, as it points along the FN statement.

FN-SKPOVR (NXT_HL)

```
M2C69   INC HL          Point to next code in statement.
        LD A,(HL)       Copy code to A.
        CP $21          Jump back or skip over if it is control
        JR C, FN-SKPOVR code or a space.
        RET
```

LOOK-VARS Subroutine

Called to search the variables area or the arguments of a DEF FN statement.

Entered with the system variable CH_ADD pointing to the first letter of the name of the variable whose location is being sought. The name will be in the program area or the work space.

Builds up a discriminator byte, in C, that is based on the first letter of the variable name. Bits 5 & 6 of this byte indicate the type of the variable. B is used as a bit register to hold flags.

If no matching name was found:
1. the carry flag is set
2. the zero flag is set when the search was for an array variable
3. HL points to the first letter of the variable name

If the search found a match:
1. the carry flag is reset
2. the zero flag is set for both simple string variables and all array variables.
3. HL points to the letter of a single letter variable name o the last character of a multi variable name as it appear in the variable list

Bit 6 of C is reset when dealing with an array of numbers and set when dealing with an array of strings.

Bit 7 of C is reset during line execution and set during syntax checking.

LOOK-VARS (FIND_N)

```
M2C70   SET NUM,(IY+OFLAGS)     Start by assuming numeric value.
        RST $18                 Get current character.
        CALL ALPHA              Jump if it is not a letter.
        JP NC,REPORT-C          Error - BAD BASIC.
        PUSH HL                 Save the pointer to the first letter.
        AND $1F                 Transfer bits 0 to 4 to C.
        LD C,A
        RST $20                 Get next character into A.
        PUSH HL                 Save the pointer to the second character.
        CP '('                  Jump if an array.
        JR Z, V-RUN/SYN         Separate arrays of numbers.
        SET 6,C                 Set bit 6.
        CP '$'                  Jump if a string
        JR Z, V-STR-VAR         Separate all the strings.
        SET 5,C                 Set bit 5.
        CALL ALPHANUM           If the variable name has only one character
        JR NC, V-TEST-FN        then jump forward.
```

Now find the end character of a name that has more than one character.

V-CHAR

```
M2C92   CALL ALPHANUM           Is the character alphanumeric?
        JR NC, V-RUN/SYN        Jump out of the loop when the end
                                end of the name is found.
        RES 6,C                 Mark the discriminator byte.
        RST $20                 Get next character.
        JR V-CHAR               And test it.
```

Simple strings and arrays of strings require that bit 6 of FLAGS is reset.

V-STR-VAR

```
M2C9C   RST $20                 Step CH_ADD past the $.
        RES NUM,(IY+OFLAGS)     Reset NUM to indicate a string result.
```

If DEFADD-hi is non-zero (indicating that a function [FN] is being evaluated) and if in run-time, search the arguments in the DEF FN statement.

V-TEST-FN

```
M2CA1   LD A,(DEFADD+1)         Is DEFADD-hi zero?
        AND A
        JR Z, V-RUN/SYN         If so, jump forward.
        CALL SYNTAX-Z           Is this run-time?
        JP NZ, STK-F-ARG        If yes, jump forward to search DEF FN.
```

Otherwise (or if the variable was not found in the DEF FN statement), search variables area, unless syntax is being checked.

V-RUN/SYN

```
M2CAD   LD B,C          Put discriminator byte in B.
        CALL SYNTAX-Z   Jump forward if in run-time.
        JR NZ, V-RUN
        LD A,C          Move discriminator to A.
        AND $E0         Drop the character code.
        SET 7,A         Indicate syntax by setting bit 7.
        LD C,A          Put it back into C.
        JR V-SYNTAX     Jump forward to continue.
```

A BASIC line is being executed so make a search of the variables area.

V-RUN

```
M2CBB   LD HL,(VARS)    Point to the variable region.
```

Enter a loop to process the names of the existing variables.

V-EACH

```
M2CBE   LD A,(HL)       Get the first letter of the variable name.
        AND $7F         Match on bits 0 to 6.
        JR Z, V-80-BYTE Jump when the $80 byte is found.
        CP C            Jump if this is not the name
        JR NZ,V-NEXT    we are seeking.
        RLA             Rotate and double to test
        ADD A,A         bits 5 and 6.
        JP P ,V-FOUND-2 Strings and array variables.
        JR C, V-FOUND-2 Simple numeric and FOR-NEXT vars.
```

Long names are matched fully.

```
        POP DE          Get the pointer to the 2nd character.
        PUSH DE         Save it again.
        PUSH HL         Save the 1st letter pointer.
```

V-MATCHES

```
M2CD0   INC HL          Look at the next character.
```

V-SPACES

```
M2CD1   LD A,(DE)       Get each character in turn.
        INC DE          Point to next character.
        CP ' '          Is it a space?
        JR Z, V-SPACES  Ignore the spaces
        OR $20          Set bit 5 to match upper and lower case.
        CP (HL)         Compare.
        JR Z,V-MATCHES  Matched: go back around for next character.
        OR $80          Check for match against bit 7.
        CP (HL)
        JR NZ,V-GET-PTR Jump forward if the last characters do not match.
        LD A,(DE)       Check that the end of the name
        CALL ALPHANUM   has been reached before
```

```
        JR NC, V-FOUND-1        jumping forward.
```

In all cases where the names fail to match the HL register pair has to be made to point to the next variable in the variables area.

V-GET-PTR

```
M2CE7   POP HL                  Fetch the variable pointer.
```

V-NEXT

```
M2CE8   PUSH BC                 Briefly save BC.
        CALL NEXT-ONE           Make DE point to next variable.
        EX DE,HL                Switch the two pointers.
        POP BC                  Restore BC and go
        JR V-EACH               back around for another.
```

Come here if no entry was found with the correct name.

V-80-BYTE

```
M2CF0   SET 7,B                 Signal that no matching variable was found.
```

Come here if checking syntax.

V-SYNTAX

```
M2CF2   POP DE                  Drop the pointer to the 2nd character.
        RST $18                 Get current character.
        CP '('                  Is it '('?
        JR Z, V-PASS            Jump forward.
        SET 5,B                 Indicate this is not an array and
        JR V-END                jump forward.
```

Come here when an entry with the correct name was found.

V-FOUND-1

```
M2CFC   POP DE                  Drop the saved variable pointer.
```

V-FOUND-2

```
M2CFD   POP DE                  Drop the second char pointer.
        POP DE                  Drop the first letter pointer.
        PUSH HL                 Save the last letter pointer.
        RST $18                 Get current character.
```

Move past the characters in a multi-character name. If the matching variable name has more than a single letter, then the other characters must be bypassed.

V-PASS

```
M2D01   CALL ALPHANUM           Jump when we have reached the
        JR NC,V-END             last character (bit 7 set).
        RST $20                 Get next character,
        JR V-PASS               go back to test it.
```

V-END

```
M2D09   POP HL                  HL holds pointer to the letter of a short name
```

```
                        or the last character of a long name.
        RL B            Rotate the register.
        BIT 6,B         Set bit 6.
        RET
```

Stack Function Argument Subroutine

Called by LOOK-VARS when DEFADD-hi in non-zero, to search the arguments area of a DEF FN statement, before searching in the variables area.

If the variable is found in the DEF FN statement, then the parameters of a string variable are stacked and a signal is given that there is no need to call STK/VAR. But it is left to SCANNING to stack the value of a numerical variable in the usual way.

STK-F-ARG

```
M2D0F   LD HL,(DEFADD)      Point to 1st character in arguments
        LD A,(HL)           and put it in to A.
        CP $29              Is it ')'?
        JP Z, V-RUN/SYN     Jump to search the variables.
```

SFA-LOOP

```
M2D18   LD A,(HL)           Get the next argument in the loop.
        OR $60              Set bits 5 & 6, assuming a simple
        LD B,A              numeric variable; copy to B.
        INC HL              Point to the next code.
        LD A,(HL)           Put in in A.
        CP $0E              Is the the number marker code?
        JR Z,SFA-CP-VR      Yes, jump.
        DEC HL              Ensure HL points to the character,
        CALL FN-SKPOVR      not a space of control code.
        INC HL              HL now points to the number marker.
        RES 5,B             Reset bit 5: string variable.
```

SFA-CP-VR

```
M2D29   LD A,B              Get the variable name into a.
        CP C                Is it the one we are seeking?
        JR Z, SFA-MATCH     Yes, jump.
        INC HL              Pass over the 5 bytes of a
        INC HL              floating-point number or
        INC HL              string parameters to get to next
        INC HL              argument.
        INC HL
        CALL FN-SKPOVR      Skip to next character.
        CP $29              Is it ')'?
        JP Z, V-RUN/SYN     Yes, jump to search variables area.
        CALL FN-SKPOVR      Point to net argument.
        JR SFA-LOOP         Jump back to process it.
```

A match has been found. The parameters of a string variable are stacked, avoiding the need to call the STK-VAR subroutine.

SFA-MATCH

```
M2D3F   BIT 5,C             Test for numeric variable.
        JR NZ, SFA-END      Jump if numeric; SCANNING will stack it.
        INC HL              Point to the first of 5 bytes to stack.
        LD DE,(STKEND)      Point DE to STKEND.
        CALL MOVE-FP        Stack the 5 bytes.
        EX DE,HL            Point HL to new position of STKEND.
        LD (STKEND),HL      Reset STKEND.
```

SFA-END

```
M2D4F   POP DE              Discard the LOOK-VARS pointers.
        POP DE
        XOR A               Return from search with both carry and
        INC A               zero flags reset, indicating that call to
        RET                 STK-VAR is not required.
```

STK-VAR Subroutine

Usually used either to find the parameters that define an existing string entry in the variables area or to return in HL the base address of a particular element or an array of numbers. When called from DIM the subroutine only checks the syntax of the BASIC statement.

The parameters that define a string may be altered by calling SLICING if this should be specified.

Initially, A and B are cleared and bit 7 of C is tested to determine whether syntax is being checked.

STK-VAR (GET_EL)

```
M2D54   XOR A               Clear the array flag.
        LD B,A              Clear B for later.
        BIT 7,C             Jump forward if syntax is being checked.
        JR NZ,SV-COUNT
```

Next, simple strings are separated from array variables.

```
        BIT 7,(HL)          Jump forward if dealing with an array variable.
        JR NZ, SV-ARRAYS
        INC A               Indicate this is a string variable.
```

SV-SIMPLE$

```
M2D5F   INC HL              Move along the entry.
        LD C,(HL)           Pick up the low length counter.
        INC HL              Advance the pointer.
        LD B,(HL)           Get the high length counter.
        INC HL              Advance the pointer.
        EX DE,HL            Transfer pointer to the actual string.
        CALL STK-STORE      Pass parameters to the calculator stack.
        RST $18             Get current character and jump forward
        JP SV-SLICE?        to see if a 'slice' is requested.
```

Base address of an element in an array is now found. Initially the 'number of dimensions' is collected.

SV-ARRAYS

```
M2D6C   INC HL              Step past the length bytes.
        INC HL
        INC HL
        LD B,(HL)           Get number of dimensions.
        BIT 6,C             Jump forward if handling array of numbers.
        JR Z, SV-PTR
```

If an array of strings has 1 dimension, then it can be handled as a simple string.

```
        DEC B               Decrease number of dimensions and jump
        JR Z, SV-SIMPLE$    if it is zero.
```

Check to ensure that the variable is followed by a subscript.

```
        EX DE,HL            Save pointer to DE.
        RST $18             Get current character.
        CP $28              Is it a '('?
        JR NZ, REPORT-3     No? Issue error message.
        EX DE,HL            Restore pointer.
```

For both numeric arrays and arrays of strings the variable pointer is transferred to the DE register pair before the subscript is evaluated.

SV-PTR

```
M2D7E   EX DE,HL            Pass pointer to DE.
        JR SV-COUNT
```

This loop finds the parameters of a specified element within an array. The loop is entered at the mid-point (SV-COUNT), where the element count is set to zero.

The loop is accessed 'B' times, this being, for a numeric array, equal to the number of dimensions that are being used, but for an array of strings 'B' is one less than the number of dimensions in use as the last subscript is used to specify a 'slice' of the string.

SV-COMMA

```
M2D81   PUSH HL          Save the counter.
        RST $18          Get current character.
        POP HL           Restore the counter.
        CP $2C           Is the current character a comma?
        JR Z, SV-LOOP    Jump forward to look for another subscript.
        BIT 7,C          If a line is being executed,
        JR Z, REPORT-3   then there is an error.
        BIT 6,C          Jump forward if handling array of strings.
        JR NZ, SV-CLOSE
        CP $29           Is current character a ')'?
        JR NZ, SV-RPT-C  Report an error if not.
        RST $20          Get next character.
        RET
```

For an array of strings the present subscript may represent a 'slice', or the subscript for a 'slice' may yet be present in the BASIC line.

SV-CLOSE

```
M2D96   CP $29           Is current character a ')'?
        JR Z, SV-DIM     Jump forward and check for another subscript.
        CP $CC           Is the character a 'TO'?
        JR NZ, SV-RPT-C  It must not be otherwise.
```

SV-CH_ADD

```
M2D9E   RST $18          Get current character.
        DEC HL           Point to the preceding character and
        LD (CH_ADD),HL   set CH_ADD.
        JR SV-SLICE      Evaluate the 'slice'.
```

Enter the loop here.

SV-COUNT

```
M2DA5   LD HL,$0000      Set counter to 0.
```

SV-LOOP

```
M2DA8   PUSH HL          Save the counter.
        RST $20          Advance CH_ADD.
        POP HL           Restore the counter.
        LD A,C           Get the discriminator byte.
        CP $C0           Jump unless checking the syntax
        JR NZ, SV-MULT   for an array of strings.
        RST $18          Get current character.
        CP $29           Is it a ')'?
        JR Z, SV-DIM     Jump forward; finished counting elements.
        CP $CC           Is it 'TO'?
        JR Z, SV-CH_ADD  Jump back if dealing with a 'slice'.
```

SV-MULT

```
M2DB9   PUSH BC             Save dimension number counter and
                            discriminator byte.
        PUSH HL             Save element counter.
        CALL DE,(DE+1)      Put dimension size in to DE.
        EX (SP),HL          Move counter to HL and stack variable pointer.
        EX DE,HL            Move counter to DE and dimension size to HL.
        CALL INT-EXP1       Evaluate the next subscript.
        JR C, REPORT-3      Give an error if out of range.
        DEC BC              Decrement counter.
        CALL GET-HL*DE      Multiply the counter by dimension size.
        ADD HL,BC           Add result of INT-EXP1 to counter.
        POP DE              Get variable pointer.
        POP BC              Get dimension number and discriminator byte.
        DJNZ SV-COMMA       Keep going around until B is 0.
```

The SYNTAX/RUN flag is checked before arrays of strings are separated from arrays of numbers.

```
        BIT 7,C             Report an error if checking syntax at this point.
```

SV-RPT-C

```
M2DD0   JR NZ, SL-RPT-C
        PUSH HL             Save the counter.
        BIT 6,C             Jump forward if handling an array of strings.
        JR NZ, SV-ELEM$
```

When dealing with an array of numbers the present character must be a ')'.

```
        LD B,D              Transfer variable pointer to
        LD C,E              BC.
        RST $18             Get current character.
        CP $29              It it ')'?
        JR Z, SV-NUMBER     Jump past error report if it's not needed.
```

REPORT-3

```
M2DDE   RST $08             Error: Subscript wrong.
        DEFB $02
```

The address of the location before the actual floating-point form can now be calculated.

SV-NUMBER

```
M2DE0   RST $20             Advance CH_ADD.
        POP HL              Get the counter.
        LD DE,$0005         5 bytes to each element in array of numbers.
        CALL GET-HL*DE      Computer total number of bytes required.
        ADD HL,BC           Point HL to location before required element.
        RET
```

When dealing with an array of strings the length of an element is given by the last 'dimension-size'. The appropriate parameters are calculated and then passed to the calculator stack.

SV-ELEM$

```
M2DEA   CALL DE,(DE+1)          Get the dimension size.
        EX (SP),HL              Variable pointer goes on stack and counter in HL.
        CALL GET-HL*DE          Multiply counter by dimension size.
        POP BC                  Get variable pointer.
        ADD HL,BC               Point HL to location before the string.
        INC HL                  Move to the actual start.
        LD B,D                  Transfer las dimension size to BC to form length.
        LD C,E
        EX DE,HL                Move start to DE.
        CALL STK-ST-0           Pass parameters to calculator stack.
```

There are three possible forms of the last subscript. The first is illustrated by - A$(2,4 TO 8) -, the second by - A$(2)(4 TO 8) - and the third by - A$(2) - which is the default form and indicates that the whole string is required.

```
        RST $18                 Get current character.
        CP $29                  Is it ')'?
        JR Z, SV-DIM            Jump if yes.
        CP $2C                  Is it a comma?
        JR NZ, REPORT-3         Report an error if not.
```

SV-SLICE

```
M2E03   CALL SLICING            Use SLICING to modify the set of parameters.
```

SV-DIM

```
M2E06   RST $20                 Get next character.
```

SV-SLICE?

```
M2E07   CP $28                  Is it a '('?
        JR Z, SV-SLICE          Jump back if there is another slide to consider.
```

When finished considering the last subscript a return can be made.

```
        RES NUM,(IY+OFLAGS)     Signal string result.
        RET
```

SLICING Subroutine

Strings are sliced using this subroutine. Entered with the parameters of the string on the top of the calculator stack and in the registers A, B, C, D & E. SYNTAX/RUN flag is tested and the parameters of the string are fetched only if a line is being executed.

SLICING (SLICER)

```
M2E10   CALL SYNTAX-Z           Check interpreting flag.
        CALL NZ,STK-FETCH       Take the parameters off the stack in run-time.
```

The possibility of the 'slice' being '()' has to be considered.

```
        RST $20             Get next character.
        CP $29              Is it ')'?
        JR Z, SL-STORE      Jump forward if yes.
```

Before proceeding the registers are manipulated as follows:

```
        PUSH DE             Start goes to the machine stack.
        XOR A               A register is cleared
        PUSH AF             and saved.
        PUSH BC             Save the length briefly.
        LD DE,$0001         Assume slice starts with first character.
        RST $18             Get current character.
        POP HL              Pass the length to HL.
```

The first parameter of the 'slice' is now evaluated.

```
        CP $CC              Is the character a 'TO'?
        JR Z, SL-SECOND     First parameter is 1, jump ahead.
        POP AF              Restore AF.
        CALL INT-EXP2       Put first parameter in BC. A is $FF
                            if there was an 'out of range' error.
        PUSH AF             Save the value.
        LD D,B              Transfer first parameter
        LD E,C              to DE.
        PUSH HL             Save length.
        RST $18             Get current character.
        POP HL              Restore length.
        CP $CC              Is current character 'TO'?
        JR Z, SL-SECOND     Jump to process second parameter.
        CP $29              It is a ')'?
```

SL-RPT-C

```
M2E38   JP NZ,REPORT-C      Error - BAD BASIC.
```

At this point a 'slice' of a single character has been identified. e.g. - A$(4).

```
        LD H,D              Last character of slice is also first
        LD L,E              character.
        JR SL-DEFINE        Jump forward.
```

The second parameter of a 'slice' is now evaluated.

SL-SECOND

```
M2E3F   PUSH HL             Save length.
        RST $20             Get next character.
        POP HL              Restore length.
        CP $29              Is character a ')'?
        JR Z, SL-DEFINE     Yes, no 2nd parameter, jump forward.
        POP AF              If first parameter was in range, A will
                            be 0; else $FF.
```

```
              CALL INT-EXP2       Put second parameter in BC.
              PUSH AF             Save the error register.
              RST $18             Get current character.
              LD H,B              Pass result from INT-EXP2
              LD L,C              to HL.
              CP $29              Is character a ')'?
              JR NZ, SL-RPT-C     No, report error.
```

The 'new' parameters are now defined.

SL-DEFINE

```
M2E52         POP AF              Fetch error register.
              EX (SP),HL          Second param to stack; start to HL.
              ADD HL,DE           First param is added to start.
              DEC HL              Back up one location.
              EX (SP),HL          New start goes to stack; second param to HL.
              AND A               Subtract first parameter from second to find
              SBC HL,DE           length of the slice.
              LD BC,$0000         Initialize new length.
              JR C, SL-OVER       A negative slide is a null string rather than error.
              INC HL              Allow for the inclusive byte.
              AND A               Test the error register.
              JP M, REPORT-3      Jump if either param was out of range for string.
              LD B,H              Transfer new length to BC.
              LD C,L
```

SL-OVER

```
M2E66         POP DE              Get new start value.
              RES NUM,(IY+OFLAGS) Confirm that string is still indicated.
```

SL-STORE

```
M2E6B         CALL SYNTAX-Z       Return if checking syntax; otherwise
              RET Z               continue to STK-STORE.
```

STK-STORE Subroutine

Passes the values held in the A, B, C, D & E to the calculator stack. The stack grows by 5 bytes with every call to this subroutine.

Normally used to transfer the parameters of strings but it is also used by STACK-BC and LOG (2^A) to transfer 'small integers' to the stack.

When storing the parameters of a string, the first value stored (coming from the A register) will be a zero if the string comes from an array of strings or is a 'slice' of a string. The value will be '1' for a complete simple string.

This 'flag' is used in the LET command routine when the '1' signals that the old copy of the string is to be 'reclaimed'.

STK-ST-0

```
M2E6F         XOR A               Signal: a string from an array of strings or a
                                  'sliced' string.
```

STK-ST-$ (PSHSTR)

```
M2E70   RES NUM,(IY+OFLAGS)     Reset to indicate a string result.
```

STK-STORE (PAEDCB)

```
M2E74   PUSH BC                 Save BC.
        CALL TEST-5-SP          Is there room for 5 bytes? Does not return if no.
        POP BC                  Restore BC
        LD HL,(STKEND)          Get address of first location above current stack.
        LD (HL),A               Transfer first byte.
        INC HL                  Step forward.
        LD (HL),E               Transfer second and third bytes.
        INC HL                  For a string, these are the 'start'.
        LD (HL),D
        INC HL                  Step forward.
        LD (HL),C               Transfer 4th and 5th bytes.
        INC HL                  For a string, these are 'length'.
        LD (HL),B
        INC HL
        LD (STKEND),HL          Save this address in STKEND
        RET                     and return.
```

INT-EXP Subroutine

Returns the result of evaluating the next expression as an integer value in BC. Also tests this result against a limit-value supplied in HL. Carry flag is set if there is an out of range error. The A register is used as an error register and holds +00 of there is no previous error and +FF if there has been one.

INT-EXP1

```
M2E8A   XOR A                   Clear error register.
```

INT-EXP2

```
M2E8B   PUSH DE                 Save DE and HL.
        PUSH HL
        PUSH AF                 Save AF.
        CALL EXPT-1NUM          Evaluate next expression to create a last value on
                                the calculator stack.
        POP AF                  Restore AF.
        CALL SYNTAX-Z           Jump forward if syntax checking.
        JR Z, I-RESTORE
        PUSH AF                 Save AF.
        CALL FIND-INT2          Last value is compressed to BC.
        POP DE                  Error register in DE.
        LD A,B                  Next expression that gives zero
        OR C                    is always in error, so jump forward if
        SCF                     it is so.
        JR Z, I-CARRY
        POP HL                  Get a copy of the limit value. This will be
        PUSH HL                 dimension size or string length.
        AND A                   Compare result of evaluating against the
```

```
         SBC HL,BC                 limit.
```

The carry flag and value held in D are manipulated to give the appropriate value for the 'error register'.

I-CARRY

```
M2EA6    LD A,D                    Fetch old error value.
         SBC A,$00                 Form new error value: $00 if no error, $FF if an
                                   out of range error.
```

Restore the registers before returning.

I-RESTORE

```
M2EA9    POP HL                    Restore HL and DE.
         POP DE
         RET
```

DE,(DE+1) Subroutine

Performs LD DE,(DE+1) and returns HL pointing to DE+2.

DE,(DE+1)

```
M2EAC    EX DE,HL                  Use HL.
         INC HL                    Point to DE+1.
         LD E,(HL)
         INC HL
         LD D,(HL)
         RET
```

GET-HL*DE Subroutine

Unless syntax is being checked this subroutine calls HL=HL*DE. Overflow of the 16 bits in HL gives the report 'out of memory'. While not exactly true, it implies that the memory is not large enough for the amount of memory requested.

GET-HL*DE

```
M2EB2    CALL SYNTAX-Z             Return if syntax checking.
         RET Z
         CALL HL=HL*DE             Perform the multiplication.
         JP C, REPORT-4            If carry is set, report out of memory.
         RET
```

LET Command Routine

Assignment routine for LET, READ and INPUT.

When the destination is a newly declared variable, DEST will point to the first letter of the variable's name as it occurs in the BASIC line. Bit 1 of FLAGX will be set.

If the destination variable exists already then bit 1 of FLAGX will be reset and DEST will point for a numeric variable to the location before the five bytes of the old number; and for a string variable to the first location of the old string.

The use of DEST applies to simple variables and to elements of arrays.

Bit 0 of FLAGX is set if the destination variable is a complete simple string variable, meaning the old copy should be deleted.

Initially the current value of DEST is collected and bit 1 of FLAGS tested.

```
LET
M2EBD   LD HL,(DEST)            Get the address of DEST.
        BIT UNFND,(IY+OFLAGX)   Jump if handling a variable that exists.
        JR Z, L-EXISTS
```

A newly declared variable is being used. Find the length of its name.

```
        LD BC,$0005             Assume it is numeric; set aside 5 bytes.
```

Enter a loop to deal with the characters of a long name. Spaces or color codes in the name are ignored.

```
L-EACH-CH
M2EC9   INC BC                  Add 1 to the counter for each character in name.

L-NO-SP
M2ECA   INC HL                  Move through variable name.
        LD A,(HL)               Get the code.
        CP $20                  Jump back if it's a space.
        JR Z,L-NO-SP
        JR NC,L-TEST-CH         Jump forward if code is $21 - $44.
        CP $10                  If less than $10, accept as end of name.
        JR C, L-SPACES
        CP $16                  Also accept $16 to $1F.
        JR NC, L-SPACES
        INC HL                  Step past control code after INK or OVER.
        JR L-NO-SP              Jump back; treat as spaces.
```

Separate 'numeric' and 'string' names.

L-TEST-CH

```
M2EDD   CALL ALPHANUM       Is the code alphanumeric?
        JR C, L-EACH-CH     If so, accept as character of a long name.
        CP $24              It is a '$'?
        JP Z, L-NEWS        Jump forward to handling as string.
```

The 'newly declared numeric variable' will require 'BC' spaces in the variables area for its name and its value. The room is made available and the name of the variable is copied over with the characters being 'marked' as required.

L-SPACES

```
M2EE7   LD A,C              Copy length to A.
        LD HL,(ELINE)       Make HL point to the $80 byte at end of
        DEC HL              the variables area.
        CALL MAKE-ROOM      Open up variables area.
        INC HL              Point to first new byte.
        INC HL              Point to second new byte.
        EX DE,HL            Make DE point to it.
        PUSH DE             Save pointer.
        LD HL,(DEST)        Get the pointer to the start of the name.
        DEC DE              Make DE point to the first new byte.
        SUB $06             Make B hold number of extra letters in name.
        LD B,A
        JR Z, L-SINGLE      Jump forward for short variable names.
```

The 'extra' codes of a long name are passed to the variables area.

L-CHAR

```
M2EFC   INC HL              Point to each extra code.
        LD A,(HL)           Get the code.
        CP $21              Accept codes from $21 to $FF.
        JR C, L-CHAR        Ignore codes $00 to $20.
        OR $20              Set bit 5, for lower case letters.
        INC DE              Transfer codes to the 2nd new
        LD (DE),A           byte onwards.
        DJNZ L-CHAR         Loop back for all extra codes.
```

The last code of a long variable name is ORed with $80.

```
        OR $80              Mark last code as required
        LD (DE),A           and overwrite.
```

The first letter of the name of the variable being handled is now considered.

```
        LD A,$C0            Mark the letter for a long name.
```

L-SINGLE

```
M2F0D   LD HL,(DEST)        Get the pointer to the letter. A is $00
        XOR (HL)            for short name and $C0 for long name.
        OR $20              Set bit 5 for lower case.
        POP HL              Drop pointer.
```

Call L-FIRST to enter the 'letter' into its appropriate location.

```
        CALL L-FIRST
```

The 'last value' is transferred to the variables area. At this point, HL always points to the location after the five locations allotted to the number. A 'RST 0028' instruction is used to call the CALCULATOR and the 'last value' is deleted. However this value is not overwritten.

L-NUMERIC

```
M2F17   PUSH HL             Save destination pointer.
        RST $28             Call calculator.
        DEFB $02            DROP
        DEFB $38            QUIT
        POP HL              Restore pointer.
        LD BC,$0005         Give number length of 5 bytes.
        AND A               Make HL point to first of five
        SBC HL,BC           and jump forward to make
        JR L-ENTER          transfer.
```

Come here if considering a variable that 'exists already'. First bit 6 of FLAGS is tested so as to separate numeric variables from string or array of string variables.

L-EXISTS

```
M2F24   BIT NUM,(IY+OFLAGS) Jump forward if handling a string
        JR Z, L-DELETE$     variable.
```

For numeric variables the 'new' number overwrites the 'old' number. So first HL has to be made to point to the location after the five bytes of the existing entry. At present HL points to the location before the five bytes.

```
        LD DE,$0006         Five bytes for a number plus 1.
        ADD HL,DE           HL points after.
        JR L-NUMERIC
```

The parameters of the string variable are fetched and complete simple strings separated from 'sliced' strings and array strings.

L-DELETE$

```
M2F30   LD HL,(DEST)        Fetch the start.
        LD BC,(STRLEN)      Get the length.
        BIT FLEX,(IY+OFLAGX) Jump if dealing with a simple string.
        JR NZ, L-ADD$
```

When dealing with a 'slice' of an existing simple string, a 'slice' of a string from an array of strings or a complete string from an array of strings there are two distinct stages involved.

The first is to build up the 'new' string in the work space, lengthening or shortening it as required. The second stage is then to copy the 'new' string to its allotted room in the variables area. However do nothing if the string has no 'length'.

```
LD A,B          Return if the string is
OR C            a null string.
RET Z
```

Then make the required number of spaces available in the work space.

```
PUSH HL         Save the start (DEST).
RST $30         Make the necessary room in workspace.
PUSH DE         Save pointer to first location.
PUSH BC         Save length.
LD D,H          Make DE point to the
LD E,L          last location.
INC HL          Make HL point one past new location.
LD (HL),$20     Enter a space character.
LDDR            Copy space into all new locations.
```

The parameters of the string being handled are now fetched from the calculator stack.

```
PUSH HL         Save the pointer.
CALL STK-FETCH  Get new parameters.
POP HL          Restore the pointer.
```

Note: At this point the required amount of room has been made available in the work space for the 'variable in assignment'. e.g. For statement - LET A$(4 to 8)="abcdefg" - five locations have been made. The parameters fetched above as a 'last value' represent the string that is to be copied into the new locations with Procrustean lengthening or shortening as required. The length of the 'new' string is compared to the length of the room made available for it.

```
EX (SP),HL      Length of new area in HL, pointer to stack.
AND A           Compare two lengths and jump
SBC HL,BC       forward if the new string will fit in the space.
ADD HL,BC
JR NC, L-LENGTH
LD B,H          Modify new length if it
LD C,L          is too long.
```

L-LENGTH

```
M2F59   EX (SP),HL          Length of new area to stack, pointer to HL.
```

As long as the new string is not a 'null string' it is copied into the work space. Procrustean lengthening is achieved automatically if the 'new' string is shorter than the room available for it.

```
        EX DE,HL            Start of new string to HL, pointer to DE.
        LD A,B              Jump forward if the new string is null.
        OR C
        JR Z, L-IN-W/S
        LDIR                Otherwise move new string to workspace.
```

The values that have been saved on the machine stack are restored.

L-IN-W/S

```
M2F61   POP BC              Length of new area.
        POP DE              Pointer to new area.
        POP HL              Start: pointer to variable.
```

L-ENTER Subroutine

This short subroutine is used to pass either a numeric value, from the calculator stack, or a string, from the work space, to its appropriate position in the variables area. The subroutine is therefore used for all except 'newly declared' simple strings and 'complete & existing' simple strings.

L-ENTER

```
M2F64   EX DE,HL            Change the pointers.
        LD A,B              Check again that the length is not zero.
        OR C
        RET Z
        PUSH DE             Save destination pointer.
        LDIR                Move numeric value or string.
        POP HL              Return with HL pointing to first byte
        RET                 or numeric value or string.
```

LET Subroutine Continues

When handling a 'complete & existing' simple string the new string is entered as if it were a 'newly declared' simple string before the existing version is 'reclaimed'.

L-ADD$

```
M2F6D   DEC HL              Make HL point to the letter of the variable's
        DEC HL              name.
        DEC HL
        LD A,(HL)           Pick up the letter.
        PUSH HL             Save the pointer to the existing version.
        PUSH BC             Save the length of the existing string.
        CALL L-STRING       Add the new string to the variable area.
        POP BC              Restore the length.
        POP HL              Restore the pointer.
```

```
              INC BC                  All one byte for the letter and
              INC BC                  two bytes for the length.
              INC BC
              JP RECLAIM-2            Exit to reclaim space.
```

'Newly declared' simple strings are handled as follows:

L-NEW$

```
M2F7E         LD A,$DF                Prepare the marking of the variable letter.
              LD HL,(DEST)            Get pointer to letter.
              AND (HL)                Mark as required.
```

L-STRING Subroutine

The parameters of the 'new' string are fetched, sufficient room is made available for it and the string is then transferred.

L-STRING

```
M2F84         PUSH AF                 Save the variable's letter.
              CALL STK-FETCH          Get the start and length of the new string.
              EX DE,HL                Move start to HL.
              ADD HL,BC               Make HL point one past string.
              PUSH BC                 Save length.
              DEC HL                  Make HL point to start of string.
              LD (DEST),HL            Save pointer.
              INC BC                  Allow one byte for the letter and two
              INC BC                  bytes for the length.
              INC BC
              LD HL,(ELINE)           Make HL point to the $80 byte.
              DEC HL
              CALL MAKE-ROOM          Open up the variables area.
              LD HL,(DEST)            Restore pointer to the end of new string.
              POP BC                  Make a copy of length
              PUSH BC                 of the new string.
              INC BC                  Add one to length in case new string is null.
              LDDR                    Make copy of new string plus one byte.
              EX DE,HL                Make HL point to the byte
              INC HL                  that is to hold the high-length.
              POP BC                  Get length.
              LD (HL),B               Enter the high-length.
              DEC HL                  Back one.
              LD (HL),C               Enter low-length.
              POP AF                  Get the variable's letter.
```

L-FIRST Subroutine

Entered with the letter of the variable, suitably marked, in the A register. The letter overwrites the 'old 80-byte' in the variables area. Returns with HL pointing to the 'new 80-byte'.

L-FIRST

```
M2FA8   DEC HL              Make HL point to the old $80 byte.
        LD (HL),A           Overwrite with letter of variable.
        LD HL,(ELINE)       Make HL point to new $80 byte.
        DEC HL              Finish with all the newly declared
        RET                 variables.
```

STK-FETCH Subroutine

Collects the last value from the calculator stack. The five bytes can be either a floating-point number, in short or long form, or set of parameters that define a string.

STK-FETCH (PGPSTR)

```
M2FAF   LD HL,(STKEND)      Get end of stack.
        DEC HL              Back up one value.
        LD B,(HL)           Get the fifth value.
        DEC HL              Back up.
        LD C,(HL)           Get fourth value.
        DEC HL              Back up.
        LD D,(HL)           Get third value.
        DEC HL              Back up.
        LD E,(HL)           Get second value.
        DEC HL              Back up.
        LD A,(HL)           Get first value.
        LD (STKEND),HL      Save the new STKEND.
        RET
```

DIM Command Routine

Creates new arrays in the variables area. Starts by searching existing variables for an existing array with the same name. If an array is found, it is reclaimed before the new array is created. New numeric arrays have all elements set to zero. New string arrays are set to spaces.

DIM

```
M2FC0   CALL LOOK-VARS      Search for the variable.
```

D-RPORT-C

```
M2FC3   JP NZ,REPORT-C      If there is an error, REPORT C.
        CALL SYNTAX-Z       Jump forward if interpreting.
        JR NZ, D-RUN
        RES 6,C             Test syntax for string arrays as if
        CALL STK-VAR        they were numeric.
        CALL CHECK-END      Move on to next statement if ok.
```

An existing array is reclaimed.

D-RUN

```
M2FD3   JR C, D-LETTER      Jump forward if there is no prior array.
        PUSH BC             Save discriminator byte.
        CALL NEXT-ONE       Find start of the next variable.
        CALL RECLAIM-2      Reclaim the prior array.
        POP BC              Restore the discriminator byte.
```

The initial parameters of the new array are found.

D-LETTER

```
M2FDD   SET 7,C             Set bit 7 of discriminator byte.
        LD B,$00            Make dimension counter zero.
        PUSH BC             Save both.
        LD HL,$0001         Put size of elements in HL; 1 for
        BIT 6,C             string array, 5 for numeric.
        JR NZ, D-SIZE
        LD L,$05
```

D-SIZE

```
M2FEB
        EX DE,HL            Put element size in DE.
```

This loop is accessed for each dimension specified in the parenthesized expression of the DIM statement. The total number of bytes required for the elements of the array is built up in the DE register pair.

D-NO-LOOP

```
M2FEC   RST $20             Advance CH_ADD through each pass.
        LD H,$FF            Set the limit value.
        CALL INT-EXP1       Evaluate a parameter.
        JP C, REPORT-3      Error if out of range.
        POP HL              Get dimension counter and discriminator byte.
        PUSH BC             Save parameter on each pass through loop.
        INC H               Increase dimension counter on each pass.
        PUSH HL             Restock the dimension counter and discriminator
                            byte.
        LD H,B              Move parameter to HL pair.
        LD L,C
        CALL GET-HL*DE      Byte total is built in HL and moved to DE.
        EX DE,HL
        RST $18             Get current character.
        CP $2C              Go around the loop again if there is
        JR Z,D-NO-LOOP      another dimension.
```

At this point DE holds the number of bytes required for the elements of the new array and the size of each dimension is on the machine stack. Check that there is indeed a closing bracket to the parenthesized expression.

```
        CP $29              Is current character a ')'?
        JR NZ, D-RPORT-C    Jump back if not.
        RST $20             Move CH_ADD past character.
```

Now make room for the dimension-sizes.

```
POP BC              Get dimension counter and discriminator byte.
LD A,C              Put discriminator byte in A.
LD L,B              Move counter to L.
LD H,$00            Clear H.
INC HL              Increase dimension counter by
INC HL              two and double the result to get
ADD HL,HL           correct overall length for the variable
ADD HL,DE           by adding the element byte total.
JP C, REPORT-4      Give "out of memory" if necessary.
PUSH DE             Save element byte total.
PUSH BC             Save dimension counter and discriminator byte.
PUSH HL             Save overall length.
LD B,H              Move overall length to BC.
LD C,L
```

Make room for the new array at the end of the variables area.

```
LD HL,(ELINE)       Make HL point to the $80 byte.
DEC HL
CALL MAKE-ROOM      Create the space.
INC HL              Point HL to the first new space.
```

Save the parameters in variables area.

```
LD (HL),A           Store the letter.
POP BC              Get overall length and subtract 3.
DEC BC
DEC BC
DEC BC
INC HL              Advance HL.
LD (HL),C           Store the low length.
INC HL              Advance HL.
LD (HL),B           Store high length.
POP BC              Get dimension counter.
LD A,B              Move it to A.
INC HL              Advance A.
LD (HL),A           Store dimension counter.
```

Clear elements of the new array.

```
LD H,D              Point HL to last location of the array
LD L,E              and DE to location before it.
DEC DE
LD (HL),$00         Put a zero in the location; overwrite
BIT 6,C             with space if it's a string array.
JR Z, DIM-CLEAR
LD (HL),$20
```

DIM-CLEAR

```
M303A   POP BC          Fetch element byte total.
        LDDR            Clear array and one extra location.
```

Save the dimension size.

DIM-SIZES

```
M303D
        POP BC          Get the dimension size.
        LD (HL),B       Save the high byte.
        DEC HL          Go back one.
        LD (HL),C       Save the low byte.
        DEC HL          Back again.
        DEC A           Decrement the dimension counter.
        JR NZ, DIM-SIZES Repeat until all dimensions have
        RET             been stored.
```

ALPHANUM Subroutine

Returns with the carry flag set if the value of the A register is a valid digit or letter.

ALPHANUM (ALNUMQ)

```
M3046   CALL NUMERIC    Test if this is a digit; carry will be reset if so.
        CCF             Complement the flag.
        RET C           Return if set.
```

ALPHA Subroutine

Return with carry set if the character in A is an alpha character.

ALPHA (ALPHAQ)

```
M304B   CP 'A'          Test against A ($41).
        CCF             Complement carry flag.
        RET NC          Return if not a valid character code.
        CP '['          Test against "[" ($5B), one more than "Z".
        RET C           Uppercase letter (A..Z), return with carry set,
        CP 'a'          Test against a ($61).
        CCF             Complement the carry flag.
        RET NC          Return if not a valid character code.
        CP '{'          Test against "{" ($7B), one more than "z".
        RET             Lowercase letter (a..z) so return with carry set.
```

Decimal To Floating Point Subroutine (STKUSN)

As part of syntax checking, decimal numbers that occur in a BASIC line are converted to floating-point forms. This subroutine reads the decimal number digit by digit and gives its result as a last value on the calculator stack. But first it deals with the alternative notation BIN, which introduces a sequence of 0's and 1's giving the binary representation of the required number.

DEC-TO-FP (STKUSN)

```
M3059   CP $C4              Jump if not BINary number.
        JR NZ,NOT-BIN
        LD DE,$0000         Initialize DE.
```

BIN-DIGIT

```
M3060   RST $20             Get next character.
        SUB '1'             Subtract code for "1".
        ADC A,$00           A will be 0; bit 0 will have carry set. Bit 1
                            will have carry reset.
        JR NZ,BIN-END       Any other character causes jump.
        EX DE,HL            Save result so far to HL.
        CCF                 Carry will be 1 for 1 and 0 for 0.
        ADC HL,HL           Shift the result into DE.
        JP C, REPORT-6      Number > 65535 is an error.
        EX DE,HL            Put result back into DE.
        JR BIN-DIGIT        Go back for the next digit.
```

BIN-END

```
        LD B,D              Copy result to BC
        LD C,E              then stack BC.
        JP STACK-BC
```

For other numbers, any integer part is converted first. If the next character is a decimal, then the decimal fraction is considered.

NOT-BIN (STKNUM)

```
M3076   CP '.'              Jump if first character is a decimal point.
        JR Z,DECIMAL
        CALL INT-TO-FP      Stack the integer portion of the number.
        CP '.'              Jump if not decimal point and check
        JR NZ,E-FORMAT      for an exponent.
        RST $20             Get next character
        CALL NUMERIC        Not a digit, so see if
        JR C,E-FORMAT       it is an "E" or "e" for an exponent.
        JR DEC-STO-1        Build the fractional portion of the number.
```

DECIMAL

```
M3089   RST $20             Get next character.
        CALL NUMERIC        Jump if it is not a decimal digit.
```

DEC-RPT-C

```
M308D   JP C,REPORT-C       Report an error if it is not.
        RST $28             Call calculator.
        DEFB $A0            Use the calculator to stack the integer
        DEFB $38            part.
```

DEC-STO-1

```
M3093   RST $28             Call calculator.
        DEFB $A1            Find the floating point version of 1
```

```
         DEFB $C0          and save it to the memory area.
         DEFB $02
         DEFB $38
```

NXT-DGT-1

```
M3098    RST $18           Get current character.
         CALL STK-DIGIT    Jump if the character was not
         JR C,E-FORMAT     a decimal digit.
         RST $28           Call calculator.
         DEFB $E0          For each iteration of the loop,
         DEFB $A4          the number saved in the memory is fetched,
         DEFB $05          divided by 10 and restored.
         DEFB $C0
         DEFB $04          Multiplied by the saved number and added
         DEFB $0F          to the last value.
         DEFB $38
         RST $20           Get next character.
         JR NXT-DGT-1      Loop until done.
```

Process E notation, i.e. xEm or xem where m is a positive or negative integer.

E-FORMAT

```
M30A9    CP 'E'            Jump if an exponent is present.
         JR Z,SIGN-FLAG
         CP 'e'            Is it lowercase e?
         RET NZ            Return if no exponent
```

SIGN-FLAG

```
M30B0    LD B,$FF          Use B as sign flag. $FF for +.
         RST $20           Get next character.
         CP $2B            It is it a plus?
         JR Z,SIGN-DONE    If yes, jump forward.
         CP $2D            Is it a minus?
         JR NZ,ST-E-PART   Neither, jump forward.
         INC B             Change sign of flag.
```

SIGN-DONE

```
M30BC    RST $20           Get next character.
```

ST-E-PART

```
M30BD    CALL NUMERIC      Jump if A is not a decimal digit.
         JR C,DEC-RPT-C    Report an error if not.
         PUSH BC           Save the B flag.
         CALL INT-TO-FP    Stack ABS m (the exponent).
         CALL FP-TO-A      Transfer ABS m to A.
         POP BC            Restore B flag.
         JP C,REPORT-6     Report overflow if ABS m is >255
         AND A
         JP M, REPORT-6    or >127.
         INC B             Test the sign flag in B. + will set to $0.
```

```
                JR Z, E-FP-JUMP         Jump if sign is +.
                NEG                     Negate m if sign is -.
```

E-FP-JUMP

```
M30D6   JP E-TO-FP              Jump to assign to the last value the result
                                of x*10^m.
```

Numeric Subroutine

Checks to see if a character is between '0' and '9'. Will return with carry set if not a digit.

NUMERIC (DIGITQ)

```
M30D9   CP $30                  Return if less than '0'.
        RET C
        CP $3A                  Is it 9 or less?
        CCF                     Make carry correct for return.
        RET
```

STK DIGIT Subroutine

Returns if the current value in A does not represent a digit. If it does, put the floating-point form of the digit on the calculator stack. Makes an ASCII character for a decimal digit into its binary representation. CF = 1 if not a digit

STK-DIGIT (ASC2BIN?)

```
M30E0   CALL NUMERIC            Return if A is not a digit.
        RET C
        SUB $30                 Make A the binary value of the digit.
```

STACK-A Subroutine

Put A onto the calculator stack. Munges BC.

STACK-A (STK_A)

```
M30E6   LD C,A                  Save A to BC.
        LD B,$00                Clear B.
```

STACK-BC Subroutine

Gives the floating-point form of the absolute binary value in BC. The form used in this and hence in the two previous subroutines as well is the one reserved in the computer for small integers n, where -65535 <= n <= 65535.

The first and fifth bytes are zero; the third and fourth bytes are the less significant and more significant bytes of the 16 bit integer n in two's complement form (if n is negative, these two bytes hold 65536+n); and the second byte is a sign byte, 00 for '+' and FF for '-'.

STACK-BC (STK_BC)

```
M30E9   LD IY,ERR_NR            Re-initialize IY to ERR_NR.
        XOR A                   Clear A.
        LD E,A                  And E, to indicate positive value.
```

```
LD D,C          Copy least significant byte to D.
LD C,B          Copy most significant byte to C.
LD B,A          Clear B.
CALL STK-STORE  Stack the number.
RST $28         Make HL point to STKEND-5.
DEFB $38
AND A           Clear carry flag.
RET
```

Integer To Floating-Point Subroutine

Returns a last value on the calculator stack that is the result of converting an integer in a BASIC line, i.e. the integer part of the decimal number or the line number, to its floating-point form.

Repeated calls to CH_ADD+1 fetch each digit of the integer in turn. Exits when a code that does not represent a digit has been fetched.

Carry flag set if character is not a decimal digit.

INT-TO-FP (ININT)

```
M30F9  PUSH AF         Save the first digit in A.
       RST $28         Call calculator.
       DEFB $A0        Set last value to zero.
       DEFB $38
       POP AF          Restore the first digit.
```

As long as the code represents a digit, the digit is stacked under the last value. The last value is then multiplied by decimal 10 and added to the digit to form a new last value which is carried back to the start of the loop.

NXT-DGT-2

```
M30FE  CALL STK-DIGIT  If the code represents a digit, stack it.
       RET C           Return if the character was not a digit.
       RST $28         Call calculator.
       DEFB $01        Digit goes under last value.
       DEFB $A4        Define decimal 10.
       DEFB $04        Multiply last value by 10.
       DEFB $0F        Add digit to last value.
       DEFB $38
       CALL CH_ADD+1   Get the next character.
       JR NXT-DGT-2    Loop back and process.
```

Arithmetic Routines

The TS 2068 stores numbers in two ways:

- Integer values in the range -65535 to +65535 are in an integral (or short) form.
- All other numbers are stored in a five byte floating point form.

E-Format To Floating-Point Subroutine

Puts number on the top of the calculator stack that is the result of converting the number given in the form xEm, where m is a positive or negative integer.

Entered with x at the top of the calculator stack and m in A. Finds the absolute value of m and multiplies or divides x by 10^m according to whether m is positive or negative.

```
E-TO-FP (XEY)
M310D   RLCA                    Test the sign of m by rotating bit 7 to carry.
        RRCA
        JR NC, E-SAVE           Jump if m is positive.
        CPL                     Negate m in A without disturbing carry.
        INC A

E-SAVE
M3113   PUSH AF                 Save m.
        LD HL,MEMBOT            Sign flag is stored in first byte.
        CALL FP-0/1
        RST $28                 Call calculator.
        DEFB $A4                x 10
        DEFB $38
        POP AF                  Recall in A.

E-LOOP
M311E   SRL A                   Short out the next bit of m.
        JR NC,E-TST-END         Jump if carry is set.
        PUSH AF                 Save the rest of m.
        RST $28                 Call calculator.
        DEFB $C1                10^(2^n)
        DEFB $E0                Jump true.
        DEFB $00,$04            IFJUMP E-DIVSN
        DEFB $04                x 10
        DEFB $33,$02            JUMP E-FETCH

E-DIVSN
M312B   DEFB $05                DIV

E-FETCH
M312C   DEFB $E1                MEM 1 -> T
        DEFB $38                QUIT
        POP AF                  Restore rest of m in A.
```

E-TST-END

```
M312F   JR Z, E-END     If m has been reduced to zero, jump.
        PUSH AF         Save m.
        RST $28         Call calculator.
        DEFB $31        DUP
        DEFB $04        TIMES
        DEFB $38        QUIT
        POP AF          Recall m.
        JR E-LOOP       Loop until all bits of m processed.
```

E-END

```
M3139   RST $28         Call calculator.
        DEFB $02        Deleted the final power of 10, leaving
        DEFB $38        the last value of x*10^m on the stack.
        RET
```

INT-FETCH Subroutine

Stores small integer n (-65535<=n<=65535) from location HL in DE. Does not delete the number from the stack or from memory. HL points to the sign byte of the integer value. C has the sign byte.

INT-FETCH (INT2COMPL)

```
M313D   INC HL          Point to the sign byte.
        LD C,(HL)       Move sign byte to C.
```

The following will two's complement the number if negative (C is FF) but leave it alone if positive (C is 00).

```
        INC HL          Get least significant byte of integer.
        LD A,(HL)       Move to A.
        XOR C           Generate the two's complement.
        SUB C           of the least significant byte and store to
        LD E,A          E.
        INC HL          Advance HL.
        LD A,(HL)       Generate the two's complement
        ADC A,C         of the most significant byte and store to
        XOR C           E.
        LD D,A          Move most significant byte to D.
        RET
```

INT-STORE Subroutine

Store the number in DE as an unsigned integer. Returns HL pointing to the first byte of n on the stack.

P-INT-STO (STDE_U)

```
M314A   LD C,$00        Set up to store number known to be positive.
```

INT-STORE (STDE_S)
Store the integer number in CDE to (HL). Take the two's complement of the number prior to storage.

```
M314C   PUSH HL          Pointer to first location saved.
        LD (HL),$00      First byte set to zero.
        INC HL
        LD (HL),C        Save second byte.
```

Same mechanism as used in INT-FETCH to two's complement negative numbers. Required before and after multiplication of small integers. Addition is performed without further two's complementing.

```
        INC HL           Point to 3rd location.
        LD A,E           Get least significant byte.
        XOR C            Two's complement if negative.
        SUB C
        LD (HL),A        Store the byte.
        INC HL           Point to 4th location.
        LD A,D           Get most significant byte.
        ADC A,C          Two's complement if negative.
        XOR C
        LD (HL),A        Save the byte.
        INC HL           Point to 5th location.
        LD (HL),$00      Set it to zero.
        POP HL           Restore the number pointer 1st byte of n
        RET              on the stack.
```

Floating-Point To BC Subroutine

Compresses floating-point last value into BC. Returns with the carry flag set if the result is too large (>65536 decimal). Zero flag is reset if the last value is negative. The low byte of the result is also copied to A.

FP-TO-BC (FP2BC)

```
M3160   RST $28          Call calculator.
        DEFB $38         Loads HL with the low byte of the value.
        LD A,(HL)        Put exponent byte in A.
        AND A            Jump if it is already an integer.
        JR Z,FP-DELETE
        RST $28          Call calculator. Round the value on
        DEFB $A2         the calc stack.
        DEFB $0F
        DEFB $27
        DEFB $38
```

FP-DELETE

```
M316B   RST $28          Call calculator. Drop the top of the calc stack
        DEFB $02         since it will now be put in the
        DEFB $38         BC register.
        PUSH HL          Save the registers
```

```
PUSH DE          DE contains the address of the dropped value.
EX DE,HL         HL points to the dropped value.
LD B,(HL)        Save the sign byte
CALL INT-FETCH   Copy bytes 2, 3, and 4 to C, E and D.
XOR A            Clear A.
SUB B            Set the carry unless B is zero.
BIT 7,C          Set the zero flag if number is positive.
LD B,D           Copy value to BC.
LD C,E
LD A,E           Copy E to A.
POP DE           Restore registers.
POP HL
RET
```

LOG (2^A) Subroutine

Calculates the approximate number of digits before the decimal in x, the number to be printed, or, if there are no digits before the decimal, then the approximate number of leading zeros after the decimal. Entered with A containing e', the true exponent of x, or e'-2, and calculates z=log to the base 10 of (2^A). Sets A equal to ABS INT (Z + 0.5), as required, using FP-TO-A for this purpose.

LOG(2^A)

```
M317F   LD D,A           Stack A as 00 00 A 00 00 (for positive A) or
        RLA              00 FF A FF 00 (for negative A).
        SBC A,A
        LD E,A           Bytes are loaded into A, E, D, C, B and then
        LD C,A           STK-STORE is called
        XOR A            to put the number
        LD B,A           of the calculator stack.
        CALL STK-STORE
        RST $28          Call calculator.
        DEFB $34,
        DEFB $EF,$7F
        DEFB $1A,$20,$9A,$85
        DEFB $04         TIMES
        DEFB $27         INT
        DEFB $38         QUIT
```

Floating-Point To A Subroutine

Uses FP-TO-BC to get the last value into A where possible.

Tests whether the modulus of the number rounds to more than 255 and if it does, returns with the carry flag set. Otherwise, returns with the modulus of the number, rounded to the nearest integer in A and the zero flag set to indicate that the number was positive, or reset for negative.

FP-TO-A (FP2A)

```
M3193   CALL FP-TO-BC       Convert the top of calculator stack to BC.
        RET C               Return if number was greater than 255.
        PUSH AF             Save A.
        DEC B               Test if out of range.
        INC B               Jump if the sign byte is 0.
        JR Z,FP-A-END
        POP AF              Restore A.
        SCF                 Set carry if value was negative.
        RET
```

FP-A-END

```
M319F   POP AF              Restore A.
        RET
```

Print a Floating-Point Number Subroutine (OUTPUT)

Converts number to string for printing. Prints x, the last value on the calculator stack. Print format never occupies more than 14 spaces.

The 8 most significant digits of x, correctly rounded, are stored in an ad hoc print buffer in mem-3 and mem-4. Small numbers, numerically less than 1, and large numbers, numerically greater than 2^27, are dealt with separately.

The former are multiplied by 10^n, where n is the approximate number of leading zeros after the decimal, while the latter are divided by 10^(n-7), where n is the approximate number of digits before the decimal.

Printing is done using E-format if there are more than 8 digits before the decimal or for small numbers with more than 4 leading zeros after the decimal.

PRINT-FP (OUTPUT)

```
M31A1   RST $28             Call calculator.
        DEFB $31            DUP
        DEFB $36            MINUSQ
        DEFB $00,$0B        IFJUMP PF-NEGATVE
        DEFB $31            DUP
        DEFB $37            PLUSQ
        DEFB $00,$0D        IFJUMP PF-POSTVE
        DEFB $02            DROP
        DEFB $38            QUIT
        LD A,$30            Character code for 0.
        RST $10             Print 0.
        RET
```

PF-NEGTVE

```
M31B0   DEFB $2A            .abs
        DEFB $38            .end
        LD A, 2DH           "-"
        RST $10             Print the minus.
        RST $28             Call calculator.
```

PF-POSTVE

```
M31B6   DEFB $A0        CONST 0 (0)
        DEFB $C3        T -> MEM 3
        DEFB $C4        T -> MEM 4
        DEFB $C5        T -> MEM 5
        DEFB $02        DROP
        DEFB $38        QUIT
        EXX             H'L', used to cold calculator offsets (for STR$)
        PUSH HL         is saved to the stack.
        EXX
```

Start of a loop which deals with large numbers. Every number x is split into its integer part i and the fractional part f. If i is a small integer (-65535 <= i <= 65535), it is stored in D'E' for insertion into the print buffer.

PF-LOOP

```
M31BF   RST $28         Call calculator.
        DEFB $31        DUP
        DEFB $27        INT
        DEFB $C2        T -> MEM 2
        DEFB $03        SUB
        DEFB $E2        MEM 2 -> T
        DEFB $01        SWAP
        DEFB $C2        T -> MEM 2
        DEFB $02        DROP
        DEFB $38        QUIT
        LD A,(HL)       Is i a small integer?
        AND A
        JR NZ, PF-LARGE Jump if not.
        CALL INT-FETCH  i copied to DE.
        LD B,$10        B set to count 16 bits.
        LD A,D          D copied to A for testing.
        AND A           Is it zero?
        JR NZ, PF-SAVE  Jump if not zero.
        OR E            Test E.
        JR Z, PF-SMALL  Jump if DE zero: x is a pure fraction.
        LD D,E          Move E to D and set B for 8 bits.
        LD B,$08        D was zero and E was not.
```

PF-SAVE

```
M31DC   PUSH DE         Transfer DE to D'E'.
        EXX
        POP DE
        EXX
        JR PF-BITS
```

Pure fractions are multiplied by 10^n, where n is the approximate number of leading zeros after the decimal; and -n is added to the second byte of mem-5, which holds the number of digits needed before the decimals; a negative number here indicates leading zeros after the decimal.

PF-SMALL

```
M31E2   RST $28          Call calculator.
        DEFB $02         DROP
        DEFB $E2         MEM 2 -> T
        DEFB $38         QUIT
        LD A,(HL)        Exponent byte e of f is copied to A.
        SUB $7E          A becomes e-126, e'+2, where e' is true
                         exponent of f.
        CALL LOG(2^A)
        LD D,A           n copied from A to D.
        LD A,(MEM5+1)    Current count collected from second byte of
        SUB D            mem-5 and n is subtracted from it.
        LD (MEM5+1),A
        LD A,D           n copied from D to A.
        CALL E-TO-FP     y=f*10^n performed and stacked.
        RST $28          Call calculator.
        DEFB $31         DUP
        DEFB $27         INT
        DEFB $C1         T -> MEM 1
        DEFB $03         SUB
        DEFB $E1         MEM 1 -> T
        DEFB $38         QUIT
        CALL FP-TO-A     i2 is transferred from stack to A.
        PUSH HL          Pointer to f2 is saved.
        LD (MEM3),A      i2 stored in first byte of mem3.
        DEC A            i2 will not count as digit for printing if it's 0.
        RLA
        SBC A,A          A manipulated so that non-zero result will
        INC A            produce 1.
        LD HL,MEM5       Zero or one is inserted into first byte of mem5.
        LD (HL),A        And added to second byte of mem5.
        INC HL
        ADD A,(HL)
        LD (HL),A
        POP HL           Restore pointer to f2.
        JP PF-FRACTN     Jump to store f3 in buffer.
```

Numbers greater than 2^27 are multiplied by 2^(-n+7), reducing the number of digits before the decimal to 8, and the loop is re-entered at PF-LOOP.

PF-LARGE

```
M3215   SUB $80          e-$80 = e', the true exponent of i.
        CP $1C           Is it less than 28 decimal?
        JR C, PF-MEDIUM  Jump if less.
        CALL LOG(2^A)    n is built in A.
        SUB $07          And reduced to n-7.
        LD B,A           Copy to B.
        LD HL,MEM5+1     Added to second byte of mem5,
        ADD A,(HL)       the number of digits required before
```

```
              LD (HL),A          the decimal.
              LD A,B             i is multiplied by 10^(-n+7).
              NEG                To bring into medium range for printing.
              CALL E-TO-FP
              JR PF-LOOP         Loop again to handle medium-sized number.
```

Integer part of x is stored in the print buffer in mem-3 and mem-4.

PF-MEDIUM

```
M322E EX DE,HL           DE points to i, HL to f.
      CALL FETCH-TWO     Mantissa of i is in D',E',D,E.
      EXX                Exchange registers.
      SET 7,D            True numerical bit 7 to D'.
      LD A,L             Exponent byte e of i to A.
      EXX                Back to main registers.
      SUB $80            True exponent e' to A.
      LD B,A             B holds required bit count.
```

When i is a small integer (less than 65536), re-enter here.

PF-BITS

```
M323A SLA E              Mantissa of i is rotated left and
      RL D               all bits of i are shifted into mem4
      EXX                and each byte of mem4 is decimal
      RL E               adjusted after each shift.
      RL D               All four bytes of i.
      EXX                Back to the main registers.
      LD HL,MEM4+4       Address of 5th byte of mem4 to HL.
      LD C,$05           Count of 5 bytes in C.
```

PF-BYTES

```
M3249 LD A,(HL)          Get the byte of mem4 to HL.
      ADC A,A            Shift left, taking in each new bit.
      DAA                Decimal adjust the byte.
      LD (HL),A          Restore it to mem4.
      DEC HL             Point to next byte of mem4.
      DEC C              Decrease byte count by one.
      JR NZ,PF-BYTES     Loop for each byte of mem4.
      DJNZ PF-BITS       Loop for each bit of INT (x).
```

Decimal adjusting each byte of mem-4 gave 2 decimal digits per byte, at most 9 digits. Digits will now be repacked, one to a byte, in mem-3 and mem-4, using the instruction RLD.

```
              XOR A           Clear A to receive digits.
              LD HL,MEM4      Source: first byte of mem4.
              LD DE,MEM3      Destination: first byte of mem3.
              LD B,$09        At most 9 digits.
              RLD             Discard left nibble of mem4.
              LD C,$FF        $FF in C will signal leading 0, $00 is non-
                              leading 0.
```

PF-DIGITS

```
M3260         RLD             Left nibble of (HL) to A, right nibble of
                              (HL) to left.
              JR NZ, PF-INSERT Jump if digit in A is not zero.
              DEC C           Test for leading zero:
              INC C           it will now give zero reset.
              JR NZ, PF-TEST-2 Jump if it was a leading zero.
```

PF-INSERT

```
M3268         LD (DE),A       Insert the digit.
              INC DE          Point to next destination.
              INC (IY+(MEM5-Y)) One more digit for printing, and one more
              INC (IY+(MEM5-Y+1)) before the decimal.
              LD C,$00        Change flag from leading zero to other zero.
```

PF-TEST-2

```
M3272         BIT 0,B         Source pointer needs to be incremented
              JR Z,PF-ALL-9   every second time through the loop when
              INC HL          B is odd.
```

PF-ALL-9

```
M3277         DJNZ PF-DIGITS  Jump back for all 9 digits.
              LD A,(MEM5)     Get counter: were there 9 digits, excluding
              SUB $09         leading zeros?
              JR C, PF-MORE   If not, jump to get more digits.
              DEC (IY+(MEM5-Y)) Reduce count to 8.
              LD A,$04        Compare 9th digit, byte 4 of mem4 with 4
              CP (IY+(MEM4-Y+3)) to set carry for rounding up.
              JR PF-ROUND
```

PF-MORE

```
M328A         RST $28         Call calculator.
              DEFB $02        DROP
              DEFB $E2        MEM 2 -> T
              DEFB $38        QUIT
```

The fractional part of x is now stored in the print buffer.

PF-FRACTN

```
M328E         EX DE,HL        Point DE to f.
              CALL FETCH-TWO  Mantissa of f in D',E',D,E.
              EXX             Exchange the registers.
```

```
        LD A,$80            Reduce exponent of f to zero
        SUB L               by shifting the bits of f $80-e
        LD L,$00            places right, where L' contained e.
        SET 7,D             True numerical bit to bit 7 of D'.
        EXX                 Restore main registers.
        CALL SHIFT-FP       Make the shift.
```

PF-FRN-LP

```
M329E   LD A,(IY+(MEM5-Y))  Get the digit count.
        CP $08              Are there already 8 digits?
        JR C, PR-FR-DGT     If not, jump forward.
        EXX                 If 8 digits, use f to round i up,
        RL D                rotating D' left to set the carry.
        EXX                 Restore main registers.
        JR PF-ROUND
```

PF-FR-DGT

```
M32AB   LD BC,$0200         Initialize C to zero, B to 2.
```

PF-FR-EXX

```
M32AE   LD A,E              D'E'DE is multiplied by 2 in stages,
        CALL CA=10*A+C      first DE, then D'E', each byte by byte
        LD E,A              in two steps and the integer part of the
        LD A,D              result is put in C to pass to print buffer.
        CALL CA=10*A+C
        LD D,A
        PUSH BC             Count and result alternate between BC and B'C'.
        EXX
        POP BC
        DJNZ PF-FR-EXX      Look back once through the exchange registers.
        LD HL,MEM3          Get base address of number.
        LD A,C              Result to A temporarily.
        LD C,(IY+(MEM5-Y))  Count of digits so far in number to C.
        ADD HL,BC           Get address of first empty byte.
        LD (HL),A           Store the next digit.
        INC (IY+(MEM5-Y))   Step up count of digits.
        JR PF-FRN-LP        Loop back until all 8 digits are processed.
```

The digits stored in the print buffer are rounded to a maximum of 8 digits for printing.

PF-ROUND

```
M32CB   PUSH AF             Save carry flag for rounding.
        LD HL,MEM3          Start with base address of number.
        LD C,(IY+(MEM5-Y))  Copy number of digits to BC.
        LD B,$00
        ADD HL,BC           Get address of last byte in number.
        LD B,C              Copy C to B as counter.
        POP AF              Restore carry flag.
```

PF-RND-LP

```
M32D7   DEC HL              Move back to last byte of number.
        LD A,(HL)           Put the byte in A.
        ADC A,$00           Add in the carry to round up.
        LD (HL),A           Put the byte back.
        AND A               If byte is 0 or 10, B will be decremented
        JR Z,PF-R-BACK      and the final zero will not be counted
        CP $0A              for printing.
        CCF                 Reset carry for valid digit.
        JR NC, PF-COUNT     Jump if carry reset.
```

PF-R-BACK

```
M32E4   DJNZ PF-RND-LP      Jump back for more rounding or final zeros.
        LD (HL),$01         There is an overflow to the left; an extra 1
        INC B               is needed here.
        INC (IY+(MEM5-Y+1)) It is also an extra digit before the decimal.
```

PF-COUNT

```
M32EC   LD (IY+(MEM5-Y)),B  B now sets count of the digits to be printed.
        RST $28             Call calculator to delete f.
        DEFB $02            DROP
        DEFB $38            QUIT
        EXX                 Calculator offset saved on the stack is
        POP HL              restored to H'L'.
        EXX
```

The number can now be printed. C will be set to hold the number of digits to be printed, not counting final zeros. B will hold the number of digits required before the decimal.

```
        LD BC,(MEM5)        Set the counters.
        LD HL,MEM3          Start of the digits.
        LD A,B              If more than 9 or fewer than -4 digits are
        CP $09              required before the decimal, then E-format
        JR C, PF-NOT-E      will be needed.
        CP $FC              Fewer than 4 means more than 4 leading zeros
        JR C, PF-E-FRMT     after the decimal.
```

PF-NOT-E

```
M3305   AND A               Are there no digits before the decimal?
        CALL Z, OUT-CODE    If so, print an initial zero.
```

The next entry point is also used to print the digits needed for E-format printing.

PF-E-SBRN

```
M3309   XOR A               Start by setting A to zero.
        SUB B               Subtract B. Minus will mean there are digits
        JP M, PF-OUT-LP     before the decimal, jump forward to print them.
        LD B,A              A is now used as a counter.
        JR PF-DC-OUT
```

```
PF-OUT-LP
M3311   LD A,C          Copy number of digits to be printed to A. If
        AND A           A is zero, there are still final zeros to print
        JR Z, PF-OUT-DT B is non-zero), so jump.
        LD A,(HL)       Get a digit from print buffer.
        INC HL          Point to next digit.
        DEC C           Decrease county by one.

PF-OUT-DT
M3318   CALL OUT-CODE   Print the digit.
        DJNZ PF-OUT-LP  Loop back until B is zero.

PF-DC-OUT
M331D   LD A,C          Time to print the decimal unless C is 0.
        AND A           In that case, return.
        RET Z
        INC B           Add 1 to B: include the decimal.
        LD A,$2E        Code for period in A.

PF-DEC-0S
M3323   RST $10         Print the period.
        LD A,$30        Character code for 0.
        DJNZ PF-DEC-0S  Loop back to print all necessary zeros.
        LD B,C          Set count for all remaining digits.
        JR PF-OUT-LP    Jump back to print the digits.

PF-E-FRMT
M332B   LD D,B          Count of digits is copied to D.
        DEC D           Decrement to give the exponent.
        LD B,$01        One digits is required before decimal.
        CALL PF-E-SBRN  Print all parts of the number before E.
        LD A,$45        Character code for E.
        RST $10         Print the E.
        LD C,D          Exponent to C for printing.
        LD A,C          And to A for testing.
        AND A           Test the sign.
        JP P, PF-E-POS  Jump if positive.
        NEG             Otherwise, negate to A.
        LD C,A          Copy back to C for printing.
        LD A,$2D        Character code for minus.
        JR PF-E-SIGN    Jump to print the sign.

PF-E-POS
M3342   LD A,$2B        Character code for plus.

PF-E-SIGN
M3344   RST $10         Print the sign.
        LD B,$00        BC holds exponent for printing.
        JP OUT-NUM      Jump to print and finish.
```

CA=10*A+C Subroutine

Multiplies each byte of D'E'DE by 10 and returns the integer part of the result in C. On entry, A contains the byte to be multiplied by 10 and C contains the carry over from the previous byte. On return, A contains the resulting byte and C the carry forward to the next byte.

C=10*A+C

```
M334A   PUSH DE         Save the DE pair in use.
        LD L,A          Copy multiplicand from A to HL.
        LD H,$00
        LD E,L          Copy it to DE.
        LD D,H
        ADD HL,HL       Double HL.
        ADD HL,HL       And again.
        ADD HL,DE       Add DE to give HL=5*A.
        ADD HL,HL       Again, so HL=10*A.
        LD E,C          Copy C to DE for addition.
        ADD HL,DE       HL-10*A+C.
        LD C,H          Copy H to C.
        LD A,L          Copy L to A.
        POP DE          Restore DE.
        RET
```

Prepare To Add Subroutine

First of four subroutines that are used by the main arithmetic operation routines: SUBTRACTION, ADDITION, MULTIPLICATION and DIVISION. This subroutine prepares a floating-point number for addition by replacing the sign bit with a true numerical bit 1, and negating the number (two's complement) if it is negative. The exponent is returned in the A register and the first byte is set to $00 for a positive number and $FF for a negative number.

PREP-ADD

```
M335A   LD A,(HL)       Transfer the exponent to A.
        LD (HL),$00     Presume a positive number.
        AND A           If the number is zero then the
        RET Z           preparation is already finished.
        INC HL          Point to the sign byte.
        BIT 7,(HL)      Set the zero flag for positive number.
        SET 7,(HL)      Restore the true numeric bit.
        DEC HL          Point to the first byte again.
        RET Z           Positive numbers have been prepared,
                        but negative numbers need to be
                        two's complemented.
        PUSH BC         Save any earlier exponent.
        LD BC,$0005     There are 5 bytes to be handled.
        ADD HL,BC       Point one-past the last byte.
        LD B,C          Transfer the '5' to B.
        LD C,A          Save the exponent in C.
        SCF             Set carry flag for negation.
```

NEG-BYTE

```
M336E   DEC HL          Point to each byte in turn.
        LD A,(HL)       Get each byte.
        CPL             One's complement the byte.
        ADC A,$00       Add in carry for negation.
        LD (HL),A       Restore the byte.
        DJNZ NEG-BYTE   Loop 5 times.
        LD A,C          Restore exponent to A.
        POP BC          Restore any earlier exponent.
        RET
```

Fetch Two Numbers Subroutine

This routine is called by the addition, multiplication, and division routines to fetch two floating point numbers to the main and alternate register sets. When called from the multiplication and division routines, the sign of the result is stored in the second byte of the first number (M2).
M1 - M5 are in H', B', C', C, B.
N1 - N5 are in: L', D', E', D, E.
HL points to the first byte of the first number.

FETCH-TWO (SUMSLD)

```
M3379   PUSH HL         Save HL and AF
        PUSH AF
        LD C,(HL)       M1 to C
        INC HL          point to M2
        LD B,(HL)       M2 to B
        LD (HL),A       save the sign of the result
        INC HL          point to M3
        LD A,C          M1 to A
        LD C,(HL)       M3 to C
        PUSH BC         save M2 & M3
        INC HL          point to M4
        LD C,(HL)       M4 to C
        INC HL          point to M5
        LD B,(HL)       M5 to B
        EX DE,HL        HL now points to N1
        LD D,A          M1 to D
        LD E,(HL)       N1 to E
        PUSH DE         Save M1 & N1
        INC HL          point to N2
        LD D,(HL)       N2 to D
        INC HL          point to N3
        LD E,(HL)       N3 to E
        PUSH DE         save N2 & N3
        EXX             to alternate register set
        POP DE          D' = N2, E' = N3
        POP HL          H' = M1, L' = N1
        POP BC          B' = M2, C' = M3
        EXX             back to normal reg set
```

```
        INC HL          point to N4
        LD D,(HL)       D = N4
        INC HL          point to N5
        LD E,(HL)       E = N5
        POP AF          Restore AF
        POP HL          Restore the pointer to the first number.
        RET
```

Shift ADDEND Subroutine

This subroutine will shift a number up to 32 places to the right to properly align it for addition. Prior to this routine, the number with the small exponent has been put in the addend position. Any overflow to the right, into the carry, is ripped right back to the beginning of the number then the number is set to zero so that addition will not alter the other number (augend).

SHIFT-FP (SHIFT)

```
M339C   AND A           If shift argument is zero, no
        RET Z           shift is needed.
        CP $21          If we shift more than 32 bits,
        JR NC, ADDED-0  jump forward.
        PUSH BC         Save BC briefly.
        LD B,A          Transfer exponent difference to B to count shifts
                        right.
```

ONE-SHIFT

```
M33A4   EXX             Arithmetic shift right for L',
        SRA L           preserving the sign marker bits.
        RR D            Rotate right with carry D', E', D & E.
        RR E
        EXX
        RR D
        RR E
        DJNZ ONE-SHIFT
        POP BC          Restore the BC.
        RET NC          Done if no carry to retrieve.
        CALL ADD-BACK   Retrieve carry.
        RET NZ          Return unless carry rippled right back. If so, zero
                        the result.
```

ADDEND-0

```
M33B8   EXX             Fetch L', D' & E'
        XOR A           Clear A.
```

ZEROS-4/5

```
M33BA   LD L,$00        Set addend to zero in D', E', D & E,
        LD D,A          along with sign indicator L', which
        LD E,L          was $00 for positive and $FF for negative.
        EXX
        LD DE,$0000
        RET
```

ADD-BACK Subroutine

This subroutine adds back into the number any carry which has overflowed to the right. In the extreme case, the carry ripples right back to the left of the number. When this subroutine is called during addition, this ripple means that a mantissa of 0.5 was shifted a full 32 places right, and the addend will now be set to zero; when called from MULTIPLICATION, it means that the exponent must be incremented, and this may result in overflow.

```
ADD-BACK
M33C3   INC E               Add carry to rightmost byte.
        RET NZ              Return if no overflow to left.
        INC D               Continued to the next byte.
        RET NZ              Return if no overflow to left.
        EXX                 Get next byte.
        INC E               Increment.
        JR NZ, ALL-ADDED    Jump if no overflow.
        INC D               Increment last byte.

ALL-ADDED
M33CC   EXX                 Restore original registers.
        RET
```

Subtraction Operation (SUB)

Changes the sign of the subtrahend and carried on into ADDITION. HL points to the minuend and DE points to the subtrahend. (See ADDITION for more details.)

```
SUBTRACT
M33CE   EX DE,HL            Swap the number pointers.
        CALL NEGATE         Negate the subtrahend.
        EX DE,HL            Restore the number pointers.
```

Addition Operation (ADD)

Carries out floating-point addition of two numbers, each with a 4-byte mantissa and a 1-byte exponent. In The two numbers at the top of the calculator stack are added/multiplied/divided to give one number at the top of the calculator stack, a 'last value'.

HL points to the second number from the top, the augend/multiplier/dividend. DE points to the number at the top of the calculator stack, the addend/multiplicand/divisor. Afterwards HL points to the resultant 'last value' whose address can also be considered to be STKEND - 5.

The addition subroutine first tests whether the 2 numbers to be added are 'small integers'. If they are, it adds them quite simply in HL and BC, and puts the result directly on the stack. No twos complementing is needed before or after the addition, since such numbers are held on the stack in twos complement form, ready for addition.

```
ADDITION (FP_ADD)
M33D3   LD A,(DE)           If both numbers are not
        OR (HL)             internal integer format,
        JR NZ, FULL-ADDN    jump to full addition.
```

```
        PUSH DE             Save the second number pointer.
        INC HL              Point to the sign byte of the
        PUSH HL             first number and save it.
        INC HL              Point to the least sig byte of M.
        LD E,(HL)           Put it in E.
        INC HL              point to most sig byte of M.
        LD D,(HL)           Put it in D.
        INC HL              Points to unused byte of M.
        INC HL              Point to exponent byte of N.
        INC HL              Point to sign of N.
        LD A,(HL)           Save N sign.
        INC HL              Point to N least sig byte.
        LD C,(HL)           Put it in C.
        INC HL              Point to N most sig byte.
        LD B,(HL)           Put it in B.
        POP HL              Point to sign of N1.
        EX DE,HL            HL = N, DE points to N sign.
        ADD HL,BC           HL = M + N.
        EX DE,HL            DE = M + N, HL points to N1 sign.
        ADC A,(HL)          If the signs were the same and if
        RRCA                the result generated a carry,
        ADC A,$00           we have overflowed 32 bits so
        JR NZ,ADDN-OFLW     jump to handle overflow. At this point CF=1 if the
                            result is negative, CF=0 if positive.
        SBC A,A             Generate $FF for negative, $00 for positive.
M33F1   LD (HL),A           Store sign byte.
        INC HL              Point to the next location.
        LD (HL),E           Store least sig byte.
        INC HL              Point to the next location.
        LD (HL),D           Store most sig byte.
        DEC HL              Move pointer back to address of
        DEC HL              first byte of the result.
        DEC HL
        POP DE              Restore STKEND to DE.
        RET
```

The number -65536 decimal can arise here in the form 00 FF 00 00 00 as the result of the addition of two smaller negative integers, e.g. -65000 and -536. It is simply stacked in this form. This is a mistake: this math system cannot handle this number. Most functions treat it as zero, and it is printed as -1E-38, obtained by treating is as 'minus zero' in an illegitimate format. One possible remedy would be to test for this number at about byte 33F1 and, if it is present, to make the second byte 80 hex and the first byte 91 hex, so producing the full five byte floating-point form of the number, i.e. 91 80 00 00 00, which causes no problems.

ADDN-OFLW

```
M33FB   DEC HL              Restore number pointers.
        POP DE
```

FULL-ADDN

M33FD	CALL RE-ST-TWO	Re-stack both numbers in full five byte floating-point form.

The full ADDITION subroutine calls PREP-ADD for each number, then gets the two numbers from the calculator stack and puts the one with the smaller exponent into the addend position. It then calls SHIFT-FP to shift the addend up to 32 decimal places right to line it up for addition. The actual addition is done in a few bytes, a single shift is made for carry (overflow to the left) if needed, the result is twos complemented if negative, and any arithmetic overflow is reported; otherwise the subroutine jumps to TEST-NORM to normalize the result and return it to the stack with the correct sign bit inserted into the second byte.

	EXX	Exchange the registers.
	PUSH HL	Save the next literal address.
	EXX	Exchange the registers.
	PUSH DE	Save addend pointer.
	PUSH HL	Save augend pointer.
	CALL PREP-ADD	Prepare the augend.
	LD B,A	Save its exponent.
	EX DE,HL	Prepare the addend.
	CALL PREP-ADD	Prepare the addend.
	LD C,A	Save addend exponent.
	CP B	If the first exponent is smaller,
	JR NC,SHIFT-LEN	keep the first number in the added position;
	LD A,B	otherwise, swap the
	LD B,C	numbers.
	EX DE,HL	

SHIFT-LEN

M3414	PUSH AF	Save the larger exponent.
	SUB B	Get the difference in exponents.
	CALL FETCH-TWO	Get the numbers into the machine registers from the calc stack.
	CALL SHIFT-FP	Shift the addend right.
	POP AF	Restore the larger exponent.
	POP HL	HL points to the result.
	LD (HL),A	Save the exponent of the result.
	PUSH HL	Save the result address.
	LD L,B	M4 to H and M5 to L.
	LD H,C	
	ADD HL,DE	Add the two right bytes.
	EXX	
	EX DE,HL	Add the two left bytes with
	ADC HL,BC	carry.
	EX DE,HL	
	LD A,H	Add H'L' and the carry; the
	ADC A,L	result will ensure that a
	LD L,A	single shift right is called
	RRA	if two positive numbers overflowed

```
                XOR L           or two negs have not overflowed left.
                EXX
                EX DE,HL        The result is now in D'E'DE.
                POP HL          Get the result pointer,
                RRA             then test for shift. (H'L' were
                                $00 for positive numbers and $FF).

                JR NC,TEST-NEG
                LD A,$01        Do a single right shift,
                CALL SHIFT-FP   perform the shift.
                INC (HL)        Add one to the exponent.
                JR Z,ADD-REP-6
```

TEST-NEG

```
M343B           EXX             Test for negative result: put sign bit
                LD A,L          of L' in A. A correctly indicates sign
                AND $80         of result.
                EXX
                INC HL          Store the sign in 2nd byte position of
                LD (HL),A       calculator stack.
                DEC HL          Point back to the exponent.
                JR Z,GO-NC-MLT  If it is zero, then do not two's complement.
                LD A,E          Get first byte,
                NEG             negate it.
                CCF             Complement the carry for continued
                LD E,A          negation and store the byte.
                LD A,D          Get next byte.
                CPL             One's complement it.
                ADC A,$00       Add carry for negation.
                LD D,A          Store the byte.
                EXX             Put next byte
                LD A,E          in A.
                CPL             One's complement it.
                ADC A,$00       Add carry for negation.
                LD E,A          Store the byte.
                LD A,D          Get last byte.
                CPL             One's complement it.
                ADC A,$00       Add carry for negation.
                JR NC,END-COMPL Done if no carry.
                RRA             Get .5 into the mantissa and
                EXX             increment the exponent. This
                INC (HL)        will be necessary when two
                                negative numbers add to a power of 2.
```

ADD-REP-6

```
M345E           JP Z, REPORT-6  Report overflow.
                EXX
```

```
END-COMPL
M3462   LD D,A          Store the last byte.
        EXX

GO-NC-MLT
M3464   XOR A           Clear the carry.
        JP TEST-NORM
```

HL=HL*DE Subroutine (MULT)

Called by GET-HL*DE and by MULTIPLICATION to perform 16-bit multiplication. Any overflow of the 16 bits available is dealt with on return from the subroutine.

```
HL=HL*DE
M3468   PUSH BC         Save the result sign byte.
        LD B,$10        16 bit multiplication.
        LD A,H          High byte to A.
        LD C,L          Low byte to C.
        LD HL,$0000     Zero result registers.

HL-LOOP
M3470   ADD HL,HL       Double the result.
        JR C, HL-END    Jump if overflow.
        RL C            Rotate bit 7 into carry.
        RLA             Rotate carry bit to bit 0 and bit 7 in to carry.
        JR NC, HL-AGAIN Jump if carry flag is reset.
        ADD HL,DE       Otherwise, add DE once.
        JR C,HL-END     Jump if overflow.

HL-AGAIN
M347B   DJNZ HL-LOOP    Loop for 16 passes.

HL-END
M347D   POP BC          Restore BC.
        RET
```

Prepare To Multiply Or Divide Subroutine

Prepares a floating-point number for multiplication or division, returning with carry set if the number is zero, putting the sign of the result into the A register and replacing the sign bit in the number by the true numeric bit,

```
PREP-M/D
M347F   CALL TEST-ZERO  If number is zero, return with carry set.
        RET C
        INC HL          Point to the sign byte.
        XOR (HL)        Get sign for result in A.
        SET 7,(HL)      Set true numeric bit.
        DEC HL          Point to the exponent.
        RET
```

Multiplication Operation (TIMES)

First tests whether the two numbers to be multiplied are small integers. If they are, it uses INT-FETCH to get them from the stack, HL=HL*DE to multiply them and INT-STORE to return the result to the stack. Any overflow of this short multiplication (i.e. if the result is not itself a small integer) causes a jump to multiplication in full five byte floating-point form.

MULTIPLY (FP_TIMES)

```
M3489   LD A,(DE)          Test whether first bytes of both number
        OR (HL)            are zero.
        JR NZ,MULT-LONG    If not, jump for long multiplication.
        PUSH DE            Save pointer to 2nd number.
        PUSH HL            Save pointer to 1st number
        PUSH DE            Working copy of 2nd number pointer.
        CALL INT-FETCH     Fetch sign in C, number in DE.
        EX DE,HL           Number to HL.
        EX (SP),HL         Number to stack, second pointer to HL.
        LD B,C             Save first sign in B.
        CALL INT-FETCH     Fetch second sign in C, number in DE.
        LD A,B             Get sign of result in A. Like signs produce $00,
        XOR C              unlike $FF.
        LD C,A             Save the sign in C.
        POP HL             Get 1st number.
        CALL HL-HL*DE      Perform a 32 bit multiply.
        EX DE,HL           transfer product to DE
        POP HL             Restore pointer to 1st number.
        JR C,MULT-OFLW     Jump if overflow to full mult.
        LD A,D             Ensure that 00 FF 00 00 00 is replaced
        OR E               by zero.
        JR NZ,MULT-RSLT
        LD C,A
```

MULT-RSLT

```
M34A9   CALL INT-STORE     Store the result on the stack.
        POP DE             Restore STKEND to DE.
        RET
```

MULT-OFLW

```
M34AE   POP DE             Restore point to 2nd number.
```

MULT-LONG

```
M34AF   CALL RE-ST-TWO     Restack both numbers in full five byte form.
```

The full MULTIPLICATION subroutine prepares the first number for multiplication by calling PREP-M/D, returning if it is zero; otherwise the second number is prepared by again calling PREP-M/D, and if it is zero the subroutine goes to set the result to zero. Next it fetches the two numbers from the calculator stack and multiplies their mantissas in the usual way, rotating the first number (treated as the multiplier) right and adding in the second number (the multiplicand) to the result whenever the multiplier bit is set. The exponents are then added together and checks are made for overflow and for underflow

(giving the result zero). Finally, the result is normalized and returned to the calculator stack with the correct sign bit in the second byte.

```
                XOR A                   Set A to zero for sign of first number.
                CALL PREP-M/D           Prepare the first number  and return if
                RET C                   zero.
                EXX                     Exchange registers.
                PUSH HL                 Save next literal address
                EXX                     Restore register.
                PUSH DE                 Save pointer to multiplicand.
                EX DE,HL                Exchange pointers.
                CALL PREP-M/D           Prepare 2nd number for multiplication.
                EX DE,HL                Exchange pointers.
                JR C, ZERO-RSLT         Jump forward if 2nd number is zero.
                PUSH HL                 Save pointer to result.
                CALL FETCH-TWO          Get the numbers into the machine
                                        registers from the calc stack.
                LD A,B                  M5 to A.
                AND A                   Reset carry flag.
                SBC HL,HL               Initialize accumulator.
                EXX
                PUSH HL                 Save M1 and N1
                SBC HL,HL               Initialize H'L'.
                EXX
                LD B,$21                For 33 bits.
                JR STRT-MLT

MLT-LOOP
M34D3           JR NC, NO-ADD           Jump forward if multiplier bit (carry) was reset.
                ADD HL,DE               Add multiplicand
                EXX                     in D'E'DE to
                ADC HL,DE               accumulator in H'L'HL.
                EXX

NO-ADD
M34DA           EXX                     Whether multiplicand was added or not, shift
                RR H                    result to right in H'L'HL so that any bit that
                RR L                    drops into carry is picked up by the next
                EXX                     byte and shift continued into B'C'CA.
                RR H
                RR L

STRT-MLT
M34E4           EXX                     Shift right the multiplier in B'C'CA.
                RR B
                RR C                    A final bit dropping into the carry will
                EXX                     trigger another add of the multiplicand
                RR C                    as a result.
                RRA
                DJNZ MLT-LOOP           Loop 33 times for all bits.
```

```
              EX DE,HL
              EXX                   Move H'L'HL to D'E'DE.
              EX DE,HL
              EXX
```

Add the exponents together.

```
              POP BC                Restore the exponents.
              POP HL                Restore the pointer to the result.
              LD A,B                Add the exponents.
              ADD A,C
              JR NZ,MAKE-EXPT       If sum >0 then clear carry,
              AND A                 else leave unchanged.
```

MAKE-EXPT

```
M34FA         DEC A                 Prepare to increase exponent
              CCF                   by $80.
```

The rest of the subroutine is common to both MULTIPLICATION and DIVISION.

DIVN-EXPT

```
M34FC         RLA                   These few bytes very cleverly
              CCF                   make the correct exponent byte.
              RRA                   Rotating left then right gets the exponent
                                    byte into A.
              JP P, OFLW1-CLR       If the sign flag is reset, no report of
                                    arithmetic overflow is required.
              JR NC, REPORT-6       Report overflow if carry reset.
              AND A                 Clear carry.
```

OFLW1-CLR

```
M3505         INC A                 The exponent is complete but if A is zero a
              JR NZ, OFLW2-CLR      a further check for overflow is required.
              JR C, OFLW2-CLR
              EXX                   If there is no carry set and the
              BIT 7,D               result is already in normal form
              EXX                   (bit 7 of D' set) then there is overflow
              JR NZ, REPORT-6       to report. But if bit 7 of D' is reset,
                                    the result is just in range, i.e. just under 2^127.
```

OFLW2-CLR

```
M3510         LD (HL),A             Store the exponent.
              EXX                   Pass the fifth result byte to A for
              LD A,B                the normalization sequence, i.e.
              EXX                   the overflow from L to B'.
```

The remainder of the subroutine deals with normalization and is common to all the arithmetic routines.

TEST-NORM (TESTNORML)

```
M3514   JR NC,NORMALISE     If no carry then normalize now.
        LD A,(HL)           Otherwise, deal with underflow (zero)
        AND A               or near underflow
```

NEAR-ZERO

```
M3518   LD A,$80            (result 2^-128).
        JR Z, SKIP-ZERO
```

ZERO-RSLT

```
M351C   XOR A               Test if A is zero (case 2^-128) and if so
```

SKIP-ZERO

```
M351D   EXX                 produce 2^-128 if number is normal;
        AND D               otherwise rocude zero.
        CALL ZEROS-4/5      the exponent must then be set to
        RLCA                zero (for zero) or 1 (for 2^-128).
        LD (HL),A           Restore the exponent byte
        JR C,OFLOW-CLR      Jump if case 2^-128,
        INC HL              otherwise, put zero into second
        LD (HL),A           byte of result on calculator
        DEC HL              stack.
        JR M3554
```

The actual normalization operation.

NORMALISE (NORML)

```
M352B   LD B,$20            Normalize the result by up to 32
```

SHIFT-ONE

```
M352D   EXX                 decimal shifts left of
        BIT 7,D             D'E'DE (with A adjoined), until bit
        EXX                 7 of D' is set. A hold zero after
        JR NZ, NORML-NOW    addition so no precision is
        RLCA                gained or lost. A hold the fifth
        RL E                byte from B' after multiplication or
        RL D                division; but as only about 32
        EXX                 bits can be correct, no precision
        RL E                lost. Note that A is rotated
        RL D                circularly with branch at carry ...
        EXX                 ...eventually a random process
        DEC (HL)            The exponent is decremented on each shift
        JR Z, NEAR-ZERO     If the exponent becomes zero, then numbers from
                            2^-129 are rounded up to 2^-128.
        DJNZ SHIFT-ONE      Loop back, up to 32 times.
        JR ZERO-RSLT        If bit 7 never became 1 then the whole result is
                            zero.
```

Finish the normalization by considering the carry.

NORML-NOW

```
M3545   RLA                 After normalization, add back any
        JR NC, OFLW-CLR     final carry that went into A.
        CALL ADD-BACK       Jump forward if the carry does not
        JR NZ,OFLW-CLR      ripple right back.
        EXX                 If it should ripple back, then
        LD D,$80            set the mantissa to 0.5 and increment
        EXX                 the exponent.
        INC (HL)            This may lead to arithmetic
        JR Z, REPORT-6      overflow.
```

The final part of the subroutine involves passing the result to the bytes reserved for it on the calculator stack and resetting the pointers.

OFLOW-CLR

```
M3554   PUSH HL             Save the result pointer.
        INC HL              Point to the sign byte of the result.
        EXX                 The result is moved from
        PUSH DE             D'E'DE to BCDE
        EXX                 then to ACDE.
        POP BC
        LD A,B              Retrieve the sign bit
        RLA                 and put into bit 7 of the
        RL (HL)             mantissa.
        RRA
        LD (HL),A           Store the first byte.
        INC HL              Next.
        LD (HL),C           Store the second byte.
        INC HL              Next.
        LD (HL),D           Store the third byte.
        INC HL              Next.
        LD (HL),E           Store the fourth byte.
        POP HL              Restore the result pointer.
        POP DE              Restore the pointer to 2nd number.
        EXX
        POP HL              Restore the next literal address.
        EXX
        RET
```

REPORT-6 (ERR6)

```
M356C   RST $08             error: Number too big.
        DEFB $05
```

Division Operation

Prepared the divisor by calling PREP-M/D, reporting arithmetic overflow if it is zero; then it prepares the dividend again calling PREP-M/D, returning if it is zero. Fetches the two numbers from the calculator stack and divides their mantissa by means of the usual restoring division, trial subtracting the divisor from the dividend and restoring if there is carry, otherwise adding 1 to the quotient. The maximum precision is obtained for a 4-byte division, and after subtracting the exponents the subroutine exits by joining the later part of MULTIPLICATION.

DIVISION (FP_DIVIDE)

```
M356E   CALL RE-ST-TWO      Convert operand to FP form.
        EX DE,HL            Exchange N and M pointers.
        XOR A               Clear the exponent register
        CALL PREP-M/D       Prepare the divisor and give the
        JR C, REPORT-6      report for arithmetic overflow if it is zero.
        EX DE,HL            Restore the pointers.
        CALL PREP-M/D       Prepare the dividend
        RET C               Return if 0, already done,
        EXX
        PUSH HL             Save next literal pointer.
        EXX
        PUSH DE             Save number pointers.
        PUSH HL
        CALL FETCH-TWO      Fetch the numbers to the machine registers.
        EXX
        PUSH HL             Save M1 and N1 on the stack.
        LD H,B
        LD L,C              Copy the dividend from
        EXX                 B'C'CB to H'L'HL.
        LD H,C
        LD L,B
        XOR A               Clear the carry.
        LD B,$DF            Count from -33 to -1.
        JR DIV-START
```

Now enter the division loop.

DIV-LOOP

```
M3591   RLA                 Shift the result left into B'C'CA,
        RL C                shifting out the bits already
        EXX                 there, picking up 1 from the
        RL C                carry whenever it is set, and
        RL B                rotating left each byte with
        EXX                 carry to achieve the 32 bit shift.
```

```
DIV-34TH
M359A   ADD HL,HL       Move what remains of the dividend left in H'L'HL
        EXX             before the next trial subtraction; if a
        ADC HL,HL       bit drops into the carry, force no restore and a bit
        EXX             for the quotient, thus retrieving the lost
        JR C,SUBN-ONLY  bit and allowing a full 32-bit divisor.

DIV-START
M35A1   SBC HL,DE       Trial subtract divisor in D'E'DE
        EXX             from rest of dividend in H'L'HL;
        SBC HL,DE       there is no initial carry .
        EXX
        JR NC, NO-RSTORE  Jump forward if there is no carry.
        ADD HL,DE       Otherwise restore, i.e. add back
        EXX             the divisor. Then clear the carry
        ADC HL,DE       so that there will be no bit for
        EXX             the quotient.
        AND A
        JR COUNT-ONE    Jump forward to the counter.

SUBN-ONLY
M35B1   AND A           Just subtract with no restore
        SBC HL,DE       and go on to set the carry flag
        EXX             because the lost bit of the dividend
        SBC HL,DE       is to be retrieved and used for the
        EXX             quotient.

NO-RSTORE
M35B8   SCF             Force a bit into the quotient.

COUNT-ONE
M35B9   INC B           Step the loop count.
        JP M,DIV-LOOP   Loop 32 times.
        PUSH AF         Save any 33rd bit.
        JR Z,DIV-34TH   Trial subtract yet again for any 34th bit;
        PUSH AF         above saves this bit, too.
        LD E,A          Now move the four bytes that
        LD D,C          form the mantissa bytes of the
        EXX             result from B'C'CA to D'E'DE.
        LD E,C
        LD D,B
        POP AF          Then put the 34th and 33rd bits
        RR B            into 'B' to be picked up on
        POP AF          normalization.
        RR B
        EXX
        POP BC          Restore the exponent bytes, M1 & N1.
        POP HL          Restore result pointer/
        LD A,B          Compute exponent difference.
        SUB C
```

```
        JP DIVN-EXPT        Exit via the exponent routine
                            in the multiplication routine.
```

Integer Truncation Towards Zero Subroutine

Returns the result of integer truncation of x, the last value, towards zero. If I(x) is a 'short integer' the subroutine returns it in that form. It returns x if the exponent byte is A0 hex or greater (x has no significant non-integral part). Otherwise the correct number of bytes of x are set to zero and, if needed, one more byte is split with a mask.

TRUNCATE (FP_TRUNC)

```
M35D3   LD A,(HL)           Put the exponent byte into A.
        AND A               If A is zero, return since x is already a
        RET Z               small integer.
        CP $81              Compare exponent to $81.
        JR NC, T-GR-ZERO    Jump if e is greater than $80.
        LD (HL),$00         Number is <1, so prepare
        LD A,$20            to zero out all 32 bits
        JR NIL-BYTES
```

T-GR-ZERO

```
M35E0   CP $91              If the number is not $91,
        JR NZ,T-SMALL       jump ahead.
```

The next 26 bytes seem designed to test whether x is in fact -65536 decimal (91 80 00 00 00), and if it is, to set it to 00 FF 00 00 00. This is a mistake. As already stated at byte 33F1 above, the system cannot handle this number. The result here is simply to make INT (-65536) return the value -1.

```
M35E4   INC HL              Point HL to the 4th byte of x, where the
        INC HL              17 bits of the integer part end.
        INC HL
        LD A,$80            Bit mask into A.
        AND (HL)            Test the bits for zero.
        DEC HL
        OR (HL)
        DEC HL              Point to 2nd byte.
        JR NZ, T-FIRST      If already non-zero, end the test.
        LD A,$80            Otherwise test for -65536 is complete.
        XOR (HL)
```

T-FIRST

```
M35F2   DEC HL              Point to the 1st byte.
        JR NZ,T-EXPNENT
        LD (HL),A           Put zero in 1st byte.
        INC HL              Point to 2nd byte.
        LD (HL),$FF         Set to $FF.
        DEC HL              Point back to1st byte.
        LD A,$18            Set last 24 bits to zero.
        JR NIL-BYTES
```

T-SMALL

```
M35FE   JR NC,X-LARGE       Jump with exponent byte 92 or more.
        PUSH DE             Save STKEND in DE.
        CPL                 Effectively subtracts
        ADD A,$91           A-1 from $91, range becomes 0 - $15.
        INC HL              Point to 2nd byte.
        LD D,(HL)           Store in D.
        INC HL              Point to 3rd byte.
        LD E,(HL)           Store in E.
        DEC HL              Point back to 1st byte.
        DEC HL
        LD C,$00            Assume positive number.
        BIT 7,D             Test for negative.
        JR Z, T-NUMERIC     Jump if positive.
        DEC C               Make negative to positive.
```

T-NUMERIC

```
M3611   SET 7,D             Insert true numeric bit in D.
        LD B,$08            Test whether A >= 8.
        SUB B               If >8, two bytes are needed.
        ADD A,B             Leave A unchanged.
        JR C, T-TEST        Jump if two bytes needed.
        LD E,D              Put the one byte in E.
        LD D,$00            And set D to zero.
        SUB B               Count the shifts needed.
```

T-TEST

```
M361D   JR Z, T-STORE       Jump if no shifts needed.
        LD B,A              Set up the shift counter.
```

T-SHIFT

```
M3620   SRL D               Shift DE right B times to produce the
        RR E                correct number.
        DJNZ T-SHIFT        Loop until B is zero.
```

T-STORE

```
M3626   CALL INT-STORE      Store the result on the stack.
        POP DE              Restore STKEND to DE.
        RET
```

Large values of x remains to be considered.

T-EXPNENT

```
M362B   LD A,(HL)           Get the exponent of x in A.
```

X-LARGE

```
M362C   SUB $A0         Subtract $A0 from e.
        RET P           Return if positive.
        NEG             Negate remainder to get number of bits to
                        become zero.
```

Now the bits of the mantissa can be cleared.

NIL-BYTES

```
M3631   PUSH DE         Save STKEND.
        EX DE,HL        Make HL point one past the 5th byte.
        DEC HL          Point to the 5th byte of X.
        LD B,A          Get the number of bits to be set.
        SRL B           Divide by 8 to determine
        SRL B           the number of whole bytes
        SRL B           to change.
        JR Z, BITS-ZERO Jump forward if result is 0.
```

BYTE-ZERO

```
M363D   LD (HL),$00     Set the bytes to zero.
        DEC HL          Reset the whole bytes
        DJNZ BYTE-ZERO
```

BITS-ZERO

```
M3642   AND $07         Determine the number of bits,
        JR Z, IX-END    Jump to end if nothing more do do.
        LD B,A          Initialize the bit counter.
        LD A,$FF        Set up the mask.
```

LESS-MASK

```
M3649   SLA A           Shift 0's into the mask.
        DJNZ LESS-MASK
        AND (HL)        Zero the unwanted bits
        LD (HL),A       and store.
```

IX-END

```
M364F   EX DE,HL        Restore HL.
        POP DE          Restore STKEND.
        RET
```

Re-Stack Two Subroutine

Re-stacks two small integers in full five byte floating-point form for the binary operations of addition, multiplication and division.

RE-ST-TWO

```
M3652   CALL RSTK-SUB   Call subroutine then continue for 2nd call.
```

RESTK-SUB

```
M3655   EX DE,HL        Exchange pointers at each call.
```

Re-Stack Subroutine

Re-stack one number in full five byte floating-point form.

```
RE-STACK (FLOAT)
M3656   LD A,(HL)          If the first byte is not zero,
        AND A              return. Is not a small integer.
        RET NZ
        PUSH DE            Save the pointer to number N.
        CALL INT-FETCH     Get sign in C and number in DE.
        XOR A              Zero Z.
        INC HL             Point to M5.
        LD (HL),A          Zero it out.
        DEC HL             Point to M4.
        LD (HL),A          Zero it.
        LD B,$91           Set exponent for 16 bit number.
        LD A,D             Test whether D is zero.
        AND A              If so, only 8 bits are needed.
        JR NZ, RS-NRMLSE   Jump if more than 8 bits necessary.
        OR E               Test E.
        LD B,D             Save zero to B.
        JR Z, RS-STORE     Jump if E is zero.
        LD D,E             Move E to D.
        LD E,B             Zero E.
        LD B,$89           Exponent for 8 bit number.

RS-NRMLSE
M3670   EX DE,HL           HL now has number, DE has pointer.

RSTK-LOOP
M3671   DEC B              For every left shift,
        ADD HL,HL          decrement the exponent once.
        JR NC, RSTK-LOOP   Loop until carry is set.
        RRC C              Sign bit to carry flag.
        RR H               Insert it in place as the number
        RR L               is shifted back one place.
        EX DE,HL           Pointer to byte 4 back to HL.

RS-STORE
M367C   DEC HL             Pointer to 3rd location.
        LD (HL),E          Store the 3rd byte.
        DEC HL             Point to 2nd location.
        LD (HL),D          Store the second byte.
        DEC HL             Point to 1st location.
        LD (HL),B          Store the exponent byte.
        POP DE             Restore the other pointer to DE.
        RET
```

Floating-Point Calculator

The calculator handles operations on numbers and strings. Its operations are specified by literals: commands unique to the calculator.

The calculator also contains routines for all mathematical functions. The approximations to SIN X, EXP X, LN X & ATN X are obtained by developing Chebyshev polynomials.

Table Of Constants

This table holds five useful and frequently needed numbers. The numbers are stored condensed form which is expanded by the STACK LITERALS subroutine to give the required floating-point form.

FPCONST

```
M3684   stk-zero    DEFB $00,$B0,$00            zero
        stk-one     DEFB $40,$B0,$00,$01        one
        stk-half    DEFB $30,$00                one half
        stk-pi/2    DEFB $F1,$49,$0F,$DA,$A2    PI/2
        stk-ten     DEFB $40,$B0,$00,$0A        ten
```

Table Of Addresses

This table is a look-up table of the addresses of the sixty-six operational subroutines of the calculator. The offsets used to index into the table are derived either from the operation codes used in SCANNING or from the literals that follow a RST 0028 instruction.

FPJMPTBL

```
M3696   DEFW $3AAA .jrnz        Jump if true
        DEFW $37FB  .swap       Swap/exchange
        DEFW $3760  .drop       Drop (delete)
        DEFW $33CE  .-          Subtract
        DEFW $3489  .*          Times
        DEFW $356E  ./          Divide
        DEFW $3C6C .^           To The
        DEFW $3936  .or         OR
        DEFW $393F  .and        AND
        DEFW $3956  .<=         Less than/equal
        DEFW $3956  .>=         Greater than/equal
        DEFW $3956  .<>         Not equal
        DEFW $3956  .>          Greater than
        DEFW $3956  .<          Less than
        DEFW $3956  .=          Equal
        DEFW $33D3  .add        Add
        DEFW $3948  .and$       String and
        DEFW $3956  .<=$        String less than/equal
        DEFW $3956  .>=$        String greater than/equal
        DEFW $3956  .<>$        String not equal
```

DEFW $3956	.>$	String greater than
DEFW $3956	.<$	String less than
DEFW $3956	.=$	String equal
DEFW $39B7	.add$	Concatenate
DEFW $39F9	.val$	VAL$
DEFW $38D7	.usr$	USR$
DEFW $3A60	.read-in	Read in string
DEFW $382D	.neg	Negate
DEFW $3A84	.code	CODE
DEFW $39F9	.val	VAL
DEFW $3A8F	.len	LEN
DEFW $3BD0	.sin	SIN
DEFW $3BC5	.cos	COS
DEFW $3BF5	.tan	TAN
DEFW $3C4E	.asn	ASN
DEFW $3C5E	.acs	ACS
DEFW $3BFD	.atn	ATN
DEFW $3B2E	.ln	LN
DEFW $3ADF	.exp	EXP
DEFW $3ACA	.int	INT
DEFW $3C65	.root	ROOT
DEFW $3851	.sgn	SGN
DEFW $3829	.abs	ABS
DEFW $386B	.peek	PEEK
DEFW $3864	.in	IN
DEFW $3872	.usr	USR
DEFW $3A3A	.str$	STR$
DEFW $39E4	.chr$	CHR$
DEFW $391C	.not	NOT
DEFW $377F	.dup	Duplicate (Move FP)
DEFW $3ABB	.mod	INTDIV (N mod M)
DEFW $3AA1	.jr	Jump
DEFW $3785	.stk05	LITERAL (STK DATA)
DEFW $3A95	.djnz	LOOP
DEFW $3921	.<0	LESS-0
DEFW $3914	.>0	GREATER-0
DEFW $3AB6	.edu	QUIT
DEFW $3B9E	.trig	ANGLE
DEFW $35D3	.trung	TRUNC
DEFW $3761	.breg	XEQTB
DEFW $310D	.Etofp	XEY
DEFW $3656	.restk	CBSV
DEFW $3808	.stk#s	SERIES-06
DEFW $37DA	.stk#	STK-ZERO
DEFW $37EC	.putm_	ST-MEM-0
DEFW $37CE	.getm_	GET-MEM-0

The last four subroutines are multi-purpose subroutines and are entered with a parameter that is a copy of the right hand five bits of the original literal. The full set follows:

Offset	Series
3E	series-06, series-08, & series-0C; literals 86,88 & 8C.
3F	stk-zero, stk-one, stk-half, stk-pi/2 & stk-ten; literals A0 to A4.
40	st-mem-0, st-mem-1, st-mem-2, st-mem-3, st-mem-4 & st-mem-5; literals C0 to C5.
41	get-mem-0, get-mem-1, get-mem-2, get-mem-3, get-mem-4 & get-mem-5; literals E0 to E5.

Calculate Subroutine

Perform floating-point calculations. These can be considered to be of three types:
1. Binary operations, e.g. addition, where two numbers in floating-point form are added together to give one 'last value'.
2. Unary operations, e.g. sin, where the 'last value' is changed to give the appropriate function result as a new 'last value'.
3. Manipulatory operations, e.g. st-mem-0, where the 'last value' is copied to the first five bytes of the calculator's memory area

The operations to be performed are specified as a series of data-bytes, the literals, that follow an RST 0028 instruction that calls this subroutine. The last literal in the list is always 38 which leads to an end to the whole operation.

In the case of a single operation needing to be performed, the operation offset can be passed to the CALCULATOR in the B register, and operation 3B, the SINGLE CALCULATION operation, performed.

It is also possible to call this subroutine recursively (from within itself), and in such a case it is possible to use the system variable BREG as a counter that controls how many operations are performed before returning.

The first part of this subroutine performs the two tasks of setting the registers to hold their required values, and to produce an offset, and possibly a parameter, from the literal that is currently being considered.

The offset is used to index into the calculator's table of addresses, see above, to find the required subroutine address. The parameter is used when the multi-purpose subroutines are called.

Note: A floating-point number may in reality be a set of string parameters.

CALCULATE

```
M371A   CALL STK-PNTRS
```

Assume a unary operation and set HL to point to the start of the last value on the calculator stack and DE to one past this floating point number (STKEND).

GEN-ENT-1

```
M371D   LD A,B
        LD (BREG),A
```

Either transfer a single operation offset to BREG temporarily or, when using the subroutine recursively, pass the parameter to BREG to use as a counter.

GEN-ENT-2

```
M3721   EXX             Return address of the subroutine is stored
        EX (SP),HL      in H'L'. Save the pointer to the first literal.
        EXX             Enter the calculator at this point when BREG is
                        used as a counter and not to be disturbed.
```

RE-ENTRY

```
M3724   LD (STKEND),DE  Enter loop to handle each literal that follows the
                        calling instruction. Always set STKEND.
        EXX             Go to alternate register set and
        LD A,(HL)       set the literal for this loop.
        INC HL          Make H'L' point to the next literal.
```

SCAN-ENT

```
M372B   PUSH HL         Save the literal to the machine stack.
        AND A           Test the A register.
        JP P, FIRST-3D  Separate the simple literals from multi-purpose
                        literals. Jump with literals $00-$3D.
        LD D,A          Save the literal in D.
        AND $60         Continue with only bits 5 & 6.
        RRCA            Four shift rights to make them bits 1 & 2.
        RRCA
        RRCA
        RRCA
        ADD A,$7C       Offsets required are $3E-$41.
        LD L,A          L will hold double the required offset.
        LD A,D          Produce parameter in A.
        AND $1F
        JR ENT-TABLE    Jump forward to find address of subroutine.
```

FIRST-3D

```
M373F   CP $18          Jump forward if performing unary operation.
        JR NC, DOUBLE-A
        EXX             All of the subroutines that perform binary
        LD BC,$FFFB     operations require that HL point to the first
        LD D,H          operand and DE to the second, as they
        LD E,L          appear on the calculator stack.
        ADD HL,BC
        EXX
```

DOUBLE-A

```
M374B   RLCA            Each entry in the table of address takes
        LD L,A          two bytes: double the offset.
```

ENT-TABLE

```
M374D   LD DE,$3696     Base address of the table.
        LD H,$00        Address of the required table entry
        ADD HL,DE       will be put in HL and the subroutine
        LD E,(HL)       address loaded in to DE.
        INC HL
```

```
        LD D,(HL)
        LD HL,$3724         Re-entry address is put on machine stack
        EX (SP),HL          beneath the subroutine address.
        PUSH DE
        EXX                 Return main set of registers.
        LD BC,(STKEND+1)    Current value of BREG transferred to B,
                            giving the single operation offset.
```

Delete Subroutine

Contains only the single RET instruction. The literal '02' results in this subroutine being considered as a binary operation that is to be entered with a first number addressed by the HL register pair and a second number addressed by the DE register pair, and the result produced again addressed by the HL register pair. The single RET instruction leads to the first number being considered as the resulting 'last value' and the second number considered as being deleted.

DELETE (FP_DROP)

```
M3760   RET
```

Single Operation Subroutine

Called from SCANNING. Used to perform a single arithmetic operation. The effect of calling this subroutine is essentially to make a jump to the appropriate subroutine for the single operation.

FP-CALC-2 (FP_XEQTB)

```
M3761   POP AF              Discard re-entry address.
        LD A,(BREG)         Transfer offset to A.
        EXX                 Enable alternate register set.
        JR SCAN-ENT         Jump back to find the required address,
                            stack the re-entry address and jump to
                            subroutine.
```

Test 5-Spaces Subroutine (ROOMQ)

Tests whether there is sufficient room in memory for another 5-byte floating-point number to be added to the calculator stack.

TEST-5-SP

```
M3768   PUSH DE             Save registers.
        PUSH HL
        LD BC,$0005         One floating point number size.
        CALL TEST-ROOM      Check for room.
```

If we returned, there was room.

```
        POP HL              Restore the registers.
        POP DE
        RET
```

Stack Number Subroutine

Called by BEEP and SCANNING twice to copy STKEND to DE, move a floating-point number to the calculator stack, and reset STKEND from DE.

STACK-NUM (STK_M)

```
M3773   LD DE,(STKEND)      Copy STKEND to DE as destination address.
        CALL MOVE-FP        Move the number.
        LD (STKEND),DE      Reset STKEND from DE.
        RET
```

Move A Floating-Point Number Subroutine

Moves a floating-point number to the top of the calculator stack or from the top of the stack to the calculator's memory area. It is also called through the calculator when it simply duplicates the number at the top of the calculator stack, the last value, thereby extending the stack by five bytes.

MOVE-FP (FP_DUP)

```
M377F   CALL TEST-5-SP      Find out if there is room.
        LDIR                Move the 5 bytes.
        RET
```

Stack Literals Subroutine

Places on the calculator stack, as a last value, the floating-point number supplied to it as 2, 3, 4 or 5 literals. When called by using offset 34 the literals follow the 34 in the list of literals; when called by the SERIES GENERATOR, see below, the literals are supplied by the sub-routine that called for a series to be generated; and when called by SKIP CONSTANTS & STACK A CONSTANT the literals are obtained from the calculator's table of constants.

In each case, the first literal supplied is divided by $40, and the integer quotient plus 1 determines whether 1, 2, 3 or 4 further literals will be taken from the source to form the mantissa of the number. Any unfilled bytes of the five bytes that go to form a 5-byte floating-point number are set to zero. The first literal is also used to determine the exponent, after reducing mod $40, unless the remainder is zero, in which case the second literal is used, as it stands, without reducing mod $40. In either case, $50 is added to the literal, giving the augmented exponent byte, e (the true exponent e' plus $80). The rest of the 5 bytes are stacked, including any zeros needed, and the subroutine returns.

STK-DATA (FP_LIT)

```
M3785   LD H,D              Set HL to point one past the present
        LD L,E              last value.
```

STK-CONST

```
M3787   CALL TEST-5-SP      Determine if there is room.
        EXX                 Go to alternate register set and
        PUSH HL             stack the pointer to next literal.
        EXX                 Switch over to result point and
        EX (SP),HL          the next literal pointer.
        PUSH BC             Save BC.
        LD A,(HL)           Put first literal into A
```

```
                AND $C0             and divide by $40 to give integer
                RLCA                values of 0, 1, 2 or 3.
                RLCA
                LD C,A              Transfer to C and increment, giving
                INC C               range of 1, 3, 4 or 4 for number of
                                    literals needed.
                LD A,(HL)           Assign literal again.
                AND $3F             Reduce by mod $40 and
                JR NZ,FORM-EXP      discarded as inappropriate if remainder
                INC HL              is zero. If so, fetch the next literal
                LD A,(HL)           and use undivided.

FORM-EXP
M379D   ADD A,$50           Exponent (e) is formed by addition
        LD (DE),A           of $50 and passed to calc stack.
        LD A,$05            Number of literals specified in C
        SUB C               are taken from source and entered
        INC HL              into the bytes of the result.
        INC DE
        LD B,$00
        LDIR
        POP BC              Restore BC.
        EX (SP),HL          Return result pointer to HL
        EXX                 and the next literal pointer to
        POP HL              its usual position in H'L'.
        EXX
        LD B,A              Number of zero bytes at this stage is defined
        XOR A               by 5-C-1.

STK-ZEROS
M37B0   DEC B               Add this number of zeroes to make
        RET Z               up the required 5 bytes.
        LD (DE),A
        INC DE
        JR STK-ZEROES
```

Skip Constants Subroutine

Entered with HL holding the base address of the calculator's table of constants and A holding a parameter that shows which of the five constants is being requested. Performs the null operations of loading the five bytes of each unwanted constant into the locations 0000, 0001, 0002, 0003 and 0004 at the beginning of the ROM until the requested constant is reached. The subroutine returns with the HL register pair holding the base address of the requested constant within the table of constants.

```
SKIP-CONS
M37B6   AND A               Returns if the parameter is zero

SKIP-NEXT
M37B7   RET Z               or when the constant has been reached.
        PUSH AF             Save parameter.
```

```
        PUSH DE             Save result pointer.
        LD DE,$0000         Dummy address.
        CALL STK-CONST      Perform fake stacking of an expanded const.
        POP DE              Restore pointer.
        POP AF              Restore parameter.
        DEC A               Count the loops.
        JR SKIP-NEXT
```

Memory Location Subroutine

Finds the base address for each five byte portion of the calculator's memory area to or from which a floating-point number is to be moved from or to the calculator stack.

Note that when a FOR-NEXT variable is being handled then the pointers are changed so that the variable is treated as if it were the calculator's memory area.

LOC-MEM (ARRAY)

```
M37C5   LD C,A              Copy parameter to C.
        RLCA                Double parameter.
        RLCA                Double result.
        ADD A,C             Add to original value to get 5x.
        LD C,A              Move to BC
        LD B,$00
        ADD HL,BC           And produce the new base address.
        RET
```

Get From Memory Area Subroutine

Called using the literals E0 to E5; parameter derived from these literals is held A. Calls MEMORY LOCATION to put the required source address into the HL register pair and MOVE A FLOATING-POINT NUMBER to copy the five bytes involved from the calculator's memory area to the top of the calculator stack to form a new last value.

GET-MEM-0-5

```
M37CE   PUSH DE             Save result pointer.
        LD HL,(MEM)         Fetch pointer to current memory area.
        CALL LOC-MEM        Find the base address.
        CALL MOVE-FP        Move the five bytes.
        POP HL              Set result pointer.
        RET
```

Stack A Constant Subroutine

Uses SKIP CONSTANTS to find the base address of the requested constants from the calculator's table of constants and then calls STACK LITERALS, entering at STK-CONST, to make the expanded form of the constant the last value on the calculator stack.

STK-ZERO (FP_TO_MEM)

```
M37DA   LD H,D              Set HL to hold result pointer.
        LD L,E
        EXX                 Go to alternate register set.
        PUSH HL             Save next literal pointer.
        LD HL,$3684         Base address of calculator's table of consts.
        EXX                 Back to main set of registers.
        CALL SKIP-CONS      Find the required const.
        CALL STK-CONST      Expand the constant.
        EXX
        POP HL              Restore next literal pointer.
        EXX
        RET
```

Store In Memory Area Subroutine

Called using the literals C0 to C5; parameter derived from these literals is held A. This subroutine is very similar to the GET FROM MEMORY subroutine but the source and destination pointers are exchanged.

ST-MEM-0-5 (FP_FROM_MEM)

```
M37EC   PUSH HL             Save the result pointer.
        EX DE,HL            Source to DE.
        LD HL,(MEM)         Get point to current memory area.
        CALL LOC-MEM        Find the base address.
        EX DE,HL            Exchange source and destination pointers.
        CALL MOVE-FP        Move the five bytes.
        EX DE,HL            Last value +5 (STKEND) to DE.
        POP HL              Result pointer to HL.
        RET
```

Note that the pointers HL and DE remain as they were, pointing to STKEND-5 and STKEND respectively, so that the last value remains on the calculator stack. If required it can be removed by using delete.

Exchange Subroutine

Exchanges the first number with the second number, i.e. the topmost two numbers on the calculator stack are exchanged.

EXCHANGE (FP_SWAP)

```
M37FB   LD B,$05          Five bytes are involved.
```

SWAP-BYTE

```
M37FD   LD A,(DE)         Each byte of the 2nd number.
        LD C,(HL)         Each byte of the 1st number.
        EX DE,HL          Switch source and destination.
        LD (DE),A         Now to the 1st number.
        LD (HL),C         Now to the 2nd number.
        INC HL            Move to operate on next pair
        INC DE            of bytes.
        DJNZ SWAP-BYTE    Exchange all 5 bytes.
        EX DE,HL          Get the pointers correct as 5 is an odd number.
        RET
```

Series Generator Subroutine

Generates the series of Chebyshev polynomials which are used to approximate to SIN, ATN, LN and EXP and derive the other arithmetic functions which depend on these (COS, TAN, ASN, ACS, ** and SQR). The SERIES GENERATOR returns to the calling routine a last value that bears a simple relationship to the requested function, for instance, SIN X.

The setting of the loop counter: The calling subroutine passes its parameters in the A register for use as a counter. The calculator is entered at GEN-ENT-1 so that the counter can be set.

SERIES-06-ETC (FP_FLOAT)

```
M3808   LD B,A            Move the parameter to B.
        CALL GEN-ENT-1    Effectively RST 28 but set the counter.
        DEFB $31          duplicate Z,Z
        DEFB $0F          addition 2*Z
        DEFB $C0          st-mem-0 2*Z mem-0 holds 2*Z
        DEFB $02          delete -
        DEFB $A0          stk-zero 0
        DEFB $C2          st-mem-2 0 mem-2 holds 0
```

G-LOOP

```
M3812   DEFB $31          duplicate B(R),B(R)
        DEFB $E0          get-mem-0 B(R),B(R),2*Z
        DEFB $04          multiply B(R),2*B(R)*Z
        DEFB $E2          get-mem-2 B(R),2*B(R)*Z,B(R-1)
        DEFB $C1          st-mem-1 mem-1 holds B(R-1)
        DEFB $38          end-calc
        DEFB $03          subtract B(R),2*B(R)*Z-B(R-1)
```

The next constant is placed on the calculator stack.

```
        CALL STK-DATA           B(R),2*B(R)*Z-B(R-1),A(R+1)
```

The calculator is re-entered without disturbing BREG.

```
        CALL GEN-ENT-2
        DEFB $0F                addition B(R),2*B(R)*Z-B(R-1)+A(R+1)
        DEFB $01                exchange 2*B(R)*Z-B(R-1)+A(R+1),B(R)
        DEFB $C2                st-mem-2 mem-2 holds B(R)
        DEFB $02                delete 2*B(R)*Z-B(R-1)+A(R!1) = B(R!1)
        DEFB $35                dec-jr-nz B(R+1)
        DEFB $EE,               to G-LOOP
```

The subtraction of B(N-2): The loop above leaves B(N) on the stack and the required result is given by B(N) - B(N-2).

```
        DEFB $E1                get-mem-1 B(N),B(N-2)
        DEFB $03                subtract B(N)-B(N-2)
        DEFB $38                end-calc
        RET
```

Absolute Magnitude Function

Performs its unary operation by ensuring that the sign bit of a floating-point number is reset. Small integers have to be treated separately. Most of the work is shared with the unary minus operation.

ABS (FP_ABS)

```
M3829   LD B,$FF                Set B to $FF.
        JR NEG-TEST
```

Unary Minus Operation

Performs its unary operation by changing the sign of the last value on the calculator stack. Zero is returned unchanged.

Full five byte floating-point numbers have their sign bit manipulated so that it ends up reset (for abs) or changed (for negate). Small integers have their sign byte set to zero (for abs) or changed (for negate).

NEGATE (FP_NEGATE)

```
M382D   CALL TEST-ZERO          Return if number is zero.
        RET C
        LD B,$00                Prep B for negate.
```

NEG-TEST

```
M3833   LD A,(HL)               If the first byte is zero, jump to handle
        AND A                   a small integer.
        JR Z,INT-CASE
        INC HL                  Point to the 2nd byte.
        LD A,B                  Get the sign mask.
        AND $80                 $80 for abs, $00 for negate.
        OR (HL)                 Set bit 7 for abs, no change for negate.
```

```
        RLA             Bit 7 is changed, leading to bit 7
        CCF             of byte 2 reset for abs and simply
        RRA             changed for negate.
        LD (HL),A       Store new 2nd byte.
        DEC HL          Point back to 1st byte.
        RET
```

INT-CASE
Integer case does a similar operation with the sign byte.

```
M3842   PUSH DE         Save the working registers.
        PUSH HL
        CALL INT-FETCH  Get sign in C, number in DE.
        POP HL          Restore pointer to the number in HL.
        LD A,B          Get $FF for abs, $00 for negate.
        OR C            $FF for abs, no change for negate.
        CPL             $00 for abs, changed byte for negate,
        LD C,A          store in C.
        CALL INT-STORE  Store result on the stack.
        POP DE          Restore STKEND to DE.
        RET
```

SIGNUM Function

Handles the function SGN X and returns a last value of 1 if X is positive, zero if X is zero and -1 if X is negative.

SGN

```
M3851   CALL TEST-ZERO  Return if X is zero.
        RET C
```

FP_SGN

```
M3855   PUSH DE         Save pointer to STKEND.
        LD DE,$0001
        INC HL          Point to the sign byte.
        RL (HL)         Rotate bit 7 to carry flag.
        DEC HL          Point back to start of number.
        SBC A,A         C = $00 for positive,
        LD C,A          $FF for negative.
        CALL INT-STORE  Stack 1 or -1 as required.
        POP DE          Restore pointer to STKEND.
        RET
```

IN Function

Handles the function IN X. It inputs at processor level from port X, loading BC with X and performing the instruction IN A,(C).

IN (FP_IN)

```
M3864   CALL FIND-INT2      X is compressed to BC.
        IN A,(C)            Get the data from the port.
        JR IN-PK-STK        Jump to stack result.
```

PEEK Function

Handles the function PEEK X. The last value is unstacked by calling FIND-INT2 and replaced by the value of the contents of the required location.

PEEK (FP_PEEK)

```
M3868   CALL FIND-INT2      Get last value to BC.
        LD A,(BC)           Get the memory value to A.
```

IN-PK-STK

```
M386F   JP STACK-A          Put A on calculator stack.
```

USR Function

This subroutine handles the function USR #. The number is obtained in BC, a return address is stacked and the machine code is executed from the location.

USR-NO (FP_USR)

```
M3872   CALL FIND-INT2      Get the number from the stack, rounded to the
                            nearest integer, test that it is in range and return
                            in BC.
        CALL USR-IN-AROS    See if the USR code is in AROS.
        LD HL,$3882         Set first return address to USRRET.
        PUSH HL
        LD HL,$30E9         Push address of STACK-BC.
        PUSH HL
        PUSH BC             Push the address of the function.
        RET
```

USRRET

Return from USR function.

```
M3882   POP AF              Get USR bank.
        INC A
        RET Z               Return with Z set.
```

The code below, which is unused, would have supported user routines in other banks beyond the DOCK bank.

```
M3885   PUSH BC                 Save BC.
        LD BC,$FF00             Set HOME bank, all chunks.
        CALL BANK_ENABLE
        POP BC                  Restore BC.
        RET
```

USR-IN-AROS
Check to see if this USR function is in an AROS.

```
M388E   LD HL,(SYSCON)          Address of the SYSCON table.
        INC HL                  Skip to the type byte.
        LD A,(HL)               Get the type.
        CP $02                  Is it an AROS?
        JR NZ, BANK-255         No, jump ahead.
        INC HL                  Skip LSB of starting address.
        INC HL                  Skip MSB of starting address.
        INC HL                  HL points to memory chunk specification.
        LD A,B                  MSB of USR address to A.
        BIT 7,A                 Is the USR function in chunk 7?
        JR Z, BANK-255          Yes,  enable the HOME bank.
        AND $06                 Reduce to chunks 2 and 3. If it set Z,
        JR Z, CK-A-4            test the memory chunk spec for chunk 4.
        SUB $04                 Remove 4 from what's left.
        JP M, CK-A-5            Negative result? Look for chunk 5.
        JR Z, CK-A-6            Zero? Look for chunk 6.
        LD A,(HL)               Put the memory chunk spec in A.
        JP M, BANK-255          Is it negative? Set the HOME bank.
        JR USR-BANK             Otherwise, it's in the DOCK bank.
```

CK-A-6
USR test for chunk 6.

```
M38B0   LD A,(HL)               Get the chunk spec.
        BIT 6,A                 Is chunk 6 set?
        JR Z, USR-BANK          Yep, enable that bank/chunk.
        JR BANK-255             Nope, HOME bank.
```

CK-A-5
USR test for chunk 5.

```
M38B7   LD A,(HL)               Same as above but for chunk 5.
        BIT 5,A
        JR Z, USR-BANK
        JR M38C5
```

CK-A-4
USR test for chunk 4.

```
M38BE   LD A,(HL)               Same as above for chunk 4.
        BIT 4,A
        JR Z, USR-BANK
```

```
        JR BANK-255
```

BANK-255
USR code is in the HOME bank.

```
M38C5   POP HL
        LD A,$FF            Home bank.
        PUSH AF             Push to stack.
        PUSH HL
        RET
```

USR-BANK
USR switch to bank (AROS).

```
M38CB   POP HL
        PUSH AF
        PUSH HL
        PUSH BC             Briefly save BC.
        LD C,A              Chunk map to C.
        LD B,$00            Dock bank.
        CALL BANK_ENABLE
        POP BC              Restore BC.
        RET
```

USR-STRING Function

This subroutine handles the function USR X$. Returns in BC the address of the bit pattern for the user-defined graphic corresponding to X$. It reports error A if X$ is not a single letter between a and u nor a user-defined graphic.

USR-$ (FP_USRS)

```
M38D7   CALL STK-FETCH      Get the parameters of the string.
        DEC BC              Decrease length by 1 to test it.
        LD A,B              If the length was not 1, then jump
        OR C                to give error report A.
        JR NZ, REPORT-A
        LD A,(DE)           Get the single code of the string.
        CALL ALPHA          Is it a letter?
        JR C, USR-RANGE     If so, jump to get its address.
        SUB $90             Reduce range for user-defined graphics
                            to 0 to 20 decimal.
        JR C, REPORT-A      Report A if out of range.
        CP $15              Test the range again.
        JR NC, REPORT-A     Report A if out of range.
        INC A               Make range 1 to 21 decimal, a to u.
```

USR-RANGE

```
M38EE   DEC A               Name the range 0 to 20 decimal.
        ADD A,A             Multiply by 8 to get offset for the
        ADD A,A             address.
        ADD A,A
        CP $A8              Test the range of the offset.
        JR NC, REPORT-A     Error if out of range.
```

```
              LD BC,(UDG)          Put address of user-defined graphic in BC.
              ADD A,C              Add C to the offset.
              LD C,A               Move the result to C.
              JR NC, USR-STACK     Jump if no carry.
              INC B                Increment B to complete the address.

USR-STACK
M38FF         JP STACK-BC          Jump to stack the address.

REPORT-A
M3902         RST $08              Error: Invalid argument.
              DEFB $09
```

TEST-ZERO Subroutine

Tests whether a floating-point number is zero. Requires that the first four bytes of the number are zero. Returns with the carry flag set if the number was in fact zero.

```
TEST-ZERO (TEST0)
M3904         PUSH HL              Save HL, BC, and A.
              PUSH BC
              LD B,A
              LD A,(HL)            OR all of the components of the
              INC HL               floating point value together. this will
              OR (HL)              produce zero if the number is zero.
              INC HL
              OR (HL)
              INC HL
              OR (HL)
              LD A,B               Restore HL, BC, and A.
              POP BC
              POP HL
              RET NZ               Return if not zero.
              SCF                  Set carry and return if zero.
              RET
```

Greater Than Zero Operation

Returns a last value of one if the current last value is greater than zero and zero otherwise. Used by other subroutines to jump on plus.

```
GREATER-0 (FP_PLUSQ)
M3914         CALL TEST-ZERO
              RET C                Return if (HL) is zero.
              LD A,$FF             Jump forward to less than zero
              JR SIGN-TO-C         but signal the opposite if action is necessary.
```

NOT Function

Returns a one on the stack if the current last value is zero and zero otherwise. Used by other subroutines to jump on zero.

NOT (FP_NOT)

```
M391C   CALL TEST-ZERO      Test (HL) for zero.
        JR FP-0/1           Jump forward.
```

Less Than Zero Operation

Returns a one on the stack if the current last value is less than zero and zero otherwise. Used by other subroutines to jump on minus.

LESS-0 (FP_MINUSQ)

```
M3921   XOR A               Clear A.
```

SIGN-TO-C

```
M3922   INC HL              Point to the sign byte of the integer to be tested.
        XOR (HL)            Carry is reset for positive, set for negative.
        DEC HL
        RLCA
```

Zero Or One Subroutine

Sets the last value to zero if the carry flag is reset and to one if it is set. When called from E-TO-FP, it creates the zero or one not on the stack but in mem-0.

FP-0/1 (STBOOL)

```
M3926   PUSH HL             Save the result pointer.
        LD A,$00            Clear A without disturbing carry.
        LD (HL),A           Store 0 into the first byte.
        INC HL              Point to 2nd byte.
        LD (HL),A           Set it to 0 as well.
        INC HL              Point to 3rd byte.
        RLA                 Rotate carry into A, making A $1 if
                            carry is set or $0 if not.
        LD (HL),A           Set 3rd byte to $1 or $0.
        RRA                 Restore A to $0.
        INC HL              Point to 4th byte.
        LD (HL),A           Set 4th byte to $0.
        INC HL              Point to 5th byte.
        LD (HL),A           Set 5th byte to $0.
        POP HL              Restore result pointer and return.
        RET
```

OR Operation

Performs the binary operation 'X OR Y' and returns X if Y is zero and the value 1 otherwise.

OR (FP_OR)

```
M3936   EX DE,HL          Point HL at Y, the second number.
        CALL TEST-ZERO    Test whether Y is zero.
        EX DE,HL          Restore the pointers.

        RET C             Return if Y was zero; X is now last value.
        SCF               Set the carry flag and jump back to
        JR FP-0/1         set the last value to 1.
```

Number And Number Operation

Performs the binary operation 'X AND Y' and returns X if Y is non-zero and zero otherwise.

NO-&-NO (FP_AND)

```
M393F   EX DE,HL          Point HL to Y, DE to X.
        CALL TEST-ZERO    Test whether Y is zero.
        EX DE,HL          Swap registers back.
        RET NC            Return with X as last value if Y was non-zero.
        AND A             Reset the carry flag and jump back to
        JR FP-0/1         set the last value to zero.
```

String And Number Operation

This subroutine performs the binary operation 'X$ AND Y' and returns X$ if Y is non-zero and a null string otherwise.

STR-&NO (FP_STGAND)

```
M3948   EX DE,HL          Point HL to Y, DE to X$.
        CALL TEST-ZERO    Test whether Y is zero.
        EX DE,HL          Swap pointers back.
        RET NC            Return with X$ as last value if Y was not zero.
        PUSH DE           Save the pointer to the number.
        DEC DE            Point to 5th byte of string parameters
                          (length-high).
        XOR A             Clear A.
        LD (DE),A         Set length-high to 0.
        DEC DE            Point to length-low.
        LD (DE),A         Set length-low to 0.
        POP DE            Restore the pointer.
        RET
```

Comparison Operations

Perform the twelve possible comparison operations (no-l-eql, no-gr-eq, nos-neql, no-grtr, no-less, nos-eql, str-l-eql, str-gr-eq, strs-neql, str-grtr, str-less & strs-eql). The single operation offset is in B at the start of the subroutine.

NO-L-EQL (FP_LE, FP_GE, FP_NE, FP_GT, FP_LT, FP_EQU, FP_STLE, FP_STGE, FP_STNE, FP_STGT, FP_STLT, FP_STEQU)

```
M3956   LD A,B              Copy offset to A.
        SUB $08             Make range $01-$06 & $09-$0E.
        BIT 2,A             Range changed to $00-$02, $04-$06,
        JR NZ, EX-OR-NOT    $08-$0A, $0C-$0E.
        DEC A
```

EX-OR-NOT

```
M395E   RRCA                Reduced again to $00-$07 with carry set for GTE
                            or LTE. The operations with
        JR NC, NU-OR-STR    carry set are treated as their
        PUSH AF             complementary operation once the values
        PUSH HL             have been exchanged.
        CALL EXCHANGE
        POP DE
        EX DE,HL
        POP AF
```

NU-OR-STR

```
M3969   BIT 2,A             Separate the numerical comparisons from
        JR NZ, STRINGS      strings by testing bit 2.
        RRCA                Numerical operations are $00-$01 with
                            carry set for equal or not equal.
        PUSH AF             Save the offset.
        CALL SUBTRACT       Subtract numbers for final tests.
        JR END-TESTS
```

STRINGS

```
M3974   RRCA                String comparisons are $02-$03 with
                            carry set for equal and not equal.
        PUSH AF             Save the offset.
        CALL STK-FETCH      Lengths and starting address of the strings
        PUSH DE             are fetched from calculator stack.
        PUSH BC
        CALL STK-FETCH
        POP HL              Length of the 2nd string.
```

```
BYTE-COMP
M397F   LD A,H
        OR L
        EX (SP),HL
        LD A,B
        JR NZ, SEC-PLUS     Jump unless 2nd string is null.
        OR C

SECND-LOW
M3986   POP BC              Second string is either null or less than first.
        JR Z, BOTH-NULL
        POP AF
        CCF                 Complement carry to get correct test
        JR STR-TEST         result.

BOTH-NULL
M398D   POP AF              Carry is used as it stands.
        JR STR-TEST

SEC-PLUS
M3990   OR C
        JR Z,FRST-LESS      First string is null, second is not.
        LD A,(DE)           Neither string is null, compare
        SUB (HL)            next bytes.
        JR C, FRST-LESS     First byte is less.
        JR NZ, SECND-LOW    Second byte is less.
        DEC BC              Bytes are equal, decrement length
        INC DE              and jump to BYTE-COMP
        INC HL              to compare next bytes of the
        EX (SP),HL          reduced strings.
        DEC HL
        JR BYTE-COMP

FRST-LESS
M39A0   POP BC
        POP AF
        AND A               Carry cleared for correct result.

STR-TEST
M39A3   PUSH AF             For string tests, put a zero on calc stack.
        RST $28             Call calculator.
        DEFB $A0            stk-zero
        DEFB $38

END-TESTS
M39A7   POP AF              These three test, give the correct results
        PUSH AF             for all twelve comparisons.
        CALL C, NOT         The initial carry is set for not equal and
        POP AF              equal, and the final carry is set for GT, LT
        PUSH AF             and equal.
```

```
        CALL NC,GREATER-0
        POP AF
        RRCA
        CALL NC, NOT
        RET
```

String Concatenation Operation

Performs the binary operation A$+B$. Parameters for these strings are fetched and the total length found. Sufficient room to hold both the strings is made available in the work space and the strings are copied over. Subroutine produces a temporary variable A$+B$ that resides in the work space.

STRS-ADD (FP_CONCAT)

```
M39B7   CALL STK-FETCH          Get parameters of 2nd string and save.
        PUSH DE
        PUSH BC
        CALL STK-FETCH          Get parameters of 1st string and save.
        POP HL
        PUSH HL                 Lengths are no in HL and BC.
        PUSH DE                 Save parameters of first string.
        PUSH BC
        ADD HL,BC               Calculate total length of two and pass
        LD B,H                  to BC.
        LD C,L
        RST $30                 Make room for the string.
        CALL STK-ST-$           Pass parameters of string to calculator stack.
        POP BC                  Retrieve parameters of first string
        POP HL                  and copy string to work space
        LD A,B                  as long as the string is not null.
        OR C
        JR Z, OTHER-STR
        LDIR
```

OTHER-STR

```
M39D2   POP BC                  Exactly the same procedure is followed
        POP HL                  for the second string to produce
        LD A,B                  A$+B$.
        OR C
        JR Z,STK-PNTRS
        LDIR
```

STK-PNTRS Subroutine

Resets HL to point to the first byte of the last value (STKEND-5) and DE to point one-past the 'last value' (STKEND).

STK-PNTRS

```
M39DA   LD HL,(STKEND)          Get current value of STKEND.
        LD DE,$FFFB             Set DE to -5 (two's complement).
        PUSH HL                 Stack value for STKEND.
```

```
        ADD HL,DE       Calculate STKEND - 5.
        POP DE          Put STKEND in DE.
        RET
```

CHR$ Function

Handles the function CHR$ X and creates a single character string in the work space.

CHRS (FP_CHR)

```
M39E4   CALL FP-TO-A    Last value is compressed into A.
        JR C, REPORT-B  Error if X was > 255 or
        JR NZ, REPORT-B X was negative.
        PUSH AF         Save compressed value of X.
        LD BC,$0001     Make one space available.
        RST $30
        POP AF          Fetch the value.
        LD (DE),A       Copy value to work space.
        CALL STK-ST-$   Pass parameters of new string to calc stack.
        EX DE,HL        Reset pointers.
        RET
```

REPORT-B

```
M39F7   RST $08         Error: Integer out of range.
        DEFB $0A
```

VAL And VAL$ Function

Handles the functions VAL X$ and VAL$ X$. When handling VAL X$, it return a last value that is the result of evaluating the string (without its bounding quotes) as a numerical expression. When handling VAL$ X$, it evaluates X$ (without its bounding quotes) as a string expression and returns the parameters of that string expression as a last value on the calculator stack.

VAL, VAL$ (FP_VALS)

```
M39F9   LD HL,(CH_ADD)  Save current value of CH_ADD to machine stack.
        PUSH HL
        LD A,B          Offset for VAL vs VAL$ must be in B, copy to A.
        ADD A,$E3       Produce $00 and carry set for VAL, $FB carry
                        reset for VAL$.
        SBC A,A         And $FF (bit 6 set) for VAL, $00 (bit 6 reset)
                        for VAL$.
        PUSH AF         Save this flag to the machine stack.
        CALL STK-FETCH  Parameters of the string are fetched;  the
        PUSH DE         starting address is saved; one byte is added
        INC BC          to the length and room for the string is
        RST $30         created in work space.
        POP HL          Starting address of string goes to HL as source.
        LD (CH_ADD),DE  Pointer for first new space to CH_ADD and
        PUSH DE         machine stack.
        LDIR            Copy string to work space.
        EX DE,HL        Switch pointers.
```

```
              DEC HL                  Extra byte is replaced with
              LD (HL),$0D             carriage return.
              RES INTPT,(IY+OFLAGS)   Reset syntax flag and
              CALL SCANNING           scan string for correct syntax.
              RST $18                 Get current character.
              CP $0D                  Check if this is the end of the expression.
              JR NZ, V-RPORT-C        If not, report an error.
              POP HL                  Get starting address of string.
              POP AF                  Fetch flag for VAL/VAL$ and compare
              XOR (IY+OFLAGS)         with NUM of result of syntax scan.
              AND $40
```

V-RPORT-C

```
M3A27         JP NZ,REPORT-C          Report an error if they do not match.
              LD (CH_ADD),HL          Start address to CH_ADD again.
              SET INTPT,(IY+OFLAGS)   Set flag for line execution.
              CALL SCANNING           String is treated as next expression and
                                      a last value is produced.
              POP HL                  Restore original value of CH_ADD.
              LD (CH_ADD),HL
              JR STK-PNTRS            Exit via this routine which resets pointers.
```

STR$ Function

Handles the function STR$ X and returns a last value, which is a set of parameters that define a string containing what would appear on the screen if X were displayed by a PRINT command.

STR$ (FP_STR)

```
M3A3A         LD BC,$0001             Make 1 space in work space and
              RST $30                 copy address to KCUR, the address
              LD (KCUR),HL            of the cursor.
              PUSH HL                 Save that address to the stack.
              LD HL,(CURCHL)          Save current channel address to machine stack.
              PUSH HL
              LD A,$FF                Open channel "R" to print the string.
              CALL CHAN-OPEN
              CALL PRINT-FP           Last value (X) is printed out in the work space and
                                      work space is expanded with each character.
              POP HL                  Restore CURCHL to HL and
              CALL CHAN-FLAG          restore the flags appropriate to it.
              POP DE                  Restore start address of string.
              LD HL,(KCUR)            Cursor address is one past end of string
              AND A                   and the difference is the length.
              SBC HL,DE
              LD B,H                  Transfer length to BC.
              LD C,L
              CALL STK-ST-$           Pass parameters of new string to calc stack.
              EX DE,HL                Reset pointers.
              RET
```

READ-IN Subroutine

Called via the calculator offset through the first line of the S-INKEY$ routine in SCANNING. Provides for the reading in of data through different streams from those available on the standard 2068. Like INKEY$, it returns a string.

READ-IN (FP_INKEY)

```
M3A60   CALL FIND-INT1      Convert numeric parameter to A.
        CP $10              Is it smaller than 16 decimal?
        JP NC,REPORT-B      If not, report the error.
        LD HL,(CURCHL)      Get the current channel pointer
        PUSH HL             and save it on machine stack.
        CALL CHAN-OPEN      Select the requested stream.
        CALL INPUT-AD       Accept a signal (key value).
        LD BC,$0000         Default length of string is zero.
        JR NC,R-I-STORE     Jump if no signal.
        INC C               Make length 1.
        RST $30             Make a space in work space.
        LD (DE),A           Put the string in it.
```

R-I-STORE

```
M3A7A   CALL STK-ST-$       Pass parameters of string to calc stack.
        POP HL              Restore CURCHL and appropriate flags.
        CALL CHAN-FLAG
        JP STK-PNTRS
```

CODE Function

Handles the function CODE A$ and returns the code of the first character in A$ or zero if A$ is null.

CODE (FP_CODE)

```
M3A84   CALL STK-FETCH      Get the parameters of the string.
        LD A,B              Test the length of the string.
        OR C
        JR Z, STK-CODE      Jump forward if null.
        LD A,(DE)           Put code of first character in A.
```

STK-CODE

```
M3A8C   JP STACK-A          Put A on calculator stack.
```

LEN Function

Handles the function LEN A$ and returns a last value that is equal to the length of the string.

LEN (FP_LEN)

```
        CALL STK-FETCH      Get the parameters of the string.
        JP STACK-BC         Exit via STACK-BC, which puts BC (length)
                            on the stack.
```

Decrease The Counter Subroutine

Called by the SERIES GENERATOR subroutine and in effect is a DJNZ operation but the counter is the system variable, BREG, rather than B.

DEC-JR-NZ (FP_LOOP)

```
        EXX             Switch to alternate register set
        PUSH HL         and save the next literal pointer on machine stack.
        LD HL,BREG      Make HL point to BREG.
        DEC (HL)        Decrement BREG.
        POP HL          Restore the next literal pointer.
        JR NZ, JUMP-2   Jump on non-zero.
        INC HL          Skip over next literal.
        EXX             Return to main register set.
        RET
```

Jump Subroutine

Executes an unconditional jump when called by the literal 33. It is also used by the subroutines DECREASE THE COUNTER and JUMP ON TRUE.

JUMP (FP_JUMP)

```
M3AA1   EXX             Switch to alternate register set.
```

JUMP-2

```
M3AA2   LD E,(HL)       Put next literal (jump length) in E'.
        LD A,E          Make A $00 or $FF based on
        RLA             whether E' is positive or negative
        SBC A,A         and copy to D'.
        LD D,A
        ADD HL,DE       Add to H'L' for next literal pointer.
        EXX             Return to main register set.
        RET
```

Jump On True Subroutine

Executes a conditional jump if the last value on the calculator stack (the number addressed currently by DE) is true.

JUMP-TRUE (FP_IFJUMP)

```
M3AAA   INC DE          Point to the 3rd byte, which is 0 or 1.
        INC DE
        LD A,(DE)       Copy byte to A.
        DEC DE          Point back to 1st byte.
        DEC DE
        AND A           Is A zero?
        JR NZ, JUMP     Jump if byte is non-zero.
        EXX             Go to alternate register set.
        INC HL          Skip jump length.
        EXX             Back to main registers.
        RET
```

END-CALC Subroutine

Ends a calculator (RST 0028) operation.

END-CALC (FP_QUIT)

```
M3AB6   POP AF          Discard return address to calculator.
        EXX             Get address from H'L', put it on
        EX (SP),HL      machine stack and make an indirect
                        jump to it.
        EXX             H'L' holds any earlier address in the
        RET             calculator chain of addresses.
```

MODULUS Subroutine

Calculates M (mod M), where M is a positive integer held at the top of the calculator stack and N is the integer held on the stack beneath M. Returns the integer quotient INT (N/M) at the top of the calculator stack and the remainder N-INT (N/M) in the second place on the stack. Called during the calculation of a random number to reduce N mod 65537 decimal.

N-MOD-M (FP_INTDIV)

```
M3ABB   RST $28         Call calculator.
        DEFB $C0        st-mem-0
        DEFB $02        delete
        DEFB $31        duplicate
        DEFB $E0        get-mem-0
        DEFB $05        division
        DEFB $27        int
        DEFB $E0        get-mem-0
        DEFB $01        exchange
        DEFB $C0        st-mem-0
        DEFB $04        multiply
        DEFB $03        subtract
        DEFB $E0        get-mem-0
        DEFB $38        end-calc
        RET
```

INT Function

Handles the function INT X and returns a last value that is the integer part of the value supplied. When X is a negative integer (X) is returned, otherwise (X)-1 is returned.

INT (FP_INT)

```
M3ACA   RST $28         Call calculator.
        DEFB $31        duplicate
        DEFB $36        less-0
        DEFB $00,$04    jump-true X-NEG
        DEFB $3A        truncate
        DEFB $38        end-calc
        RET
```

X-NEG

```
M3AD2   DEFB $31            duplicate
        DEFB $3A            truncate
        DEFB $C0            st-mem-0
        DEFB $03            subtract
        DEFB $E0            get-mem-0
        DEFB $01            exchange
        DEFB $30            not
        DEFB $00,$03        jump-true EXIT
        DEFB $A1            stk-one
        DEFB $03            subtract
```

EXIT

```
M3ADD   DEFB $38            end-calc
        RET
```

Exponential Function

Handles the function EXP X and is the first of four routines that use SERIES GENERATOR to produce Chebyshev polynomials.

EXP

```
M3ADF   RST $28             Call calculator.
        DEFB $3D            re-stack X (in full floating-point form)
        DEFB $34            stk-data
        DEFB $F1            exponent
        DEFB $38,$AA,$3B,$29
        DEFB $04            multiply
        DEFB $31            duplicate
        DEFB $27            int
        DEFB $C3            st-mem-3
        DEFB $03            subtract
        DEFB $31            duplicate
        DEFB $0F            addition
        DEFB $A1            stk-one
        DEFB $03            subtract
        DEFB $88            series-08
        DEFB $13,$36        EXP 63,$36,(00,00,00)
        DEFB $58,$65,$66    EXP 68 $65,$66,(00,00)
        DEFB $9D,$78,$65,$40 EXP 6D $78,$65,$40,(00)
        DEFB $A2,$60,$32,$C9 EXP 72 $60,$32,$C9,(00)
        DEFB $E7            EXP 77 $21,$F7,$AF,$24
        DEFB $21,$F7,$AF,$24
        DEFB $EB            EXP 7B $2F,$B0,$B0,$14
        DEFB $2F,$B0,$B0,$14
        DEFB $EE            EXP 7E $7E,$BB,$94,$58
        DEFB $7E,$BB,$94,$58
        DEFB $F1            EXP 81 $3A,$7E,$F8,$CF
        DEFB $3A,$7E,$F8,$CF
        DEFB $E3            get-mem-3
```

```
              DEFB $38                end-calc
              CALL FP-TO-A            Put abs value of N mod 256 in A.
              JR NZ,N-NEGTV           Jump forward if N was negative.
              JR C, REPORT-6          Error if ABS N > 256.
              ADD A,(HL)              Add ABS N to exponent.
              JR NC, RESULT-OK

REPORT-6
M3B1E         RST $08                 Error: Number too big.
              DEFB $05

N-NEGTV
M3B20         JR C, RSLT-ZERO         If N < -255, result is 0.
              SUB (HL)                Subtract ABS N from exponent if negative.
              JR NC, RSLT-ZERO        Zero result if e less than zero.
              NEG                     Minus e changed to e.

RESULT-OK
M3B27         LD (HL),A               Save exponent to A.
              RET

RSLT-ZERO
M3B29         RST $28                 Call calculator.
              DEFB $02                delete
              DEFB $A0                stk-zero
              DEFB $38                end-calc
              RET
```

Natural Logarithm Function

Handles the function LN X. Use SERIES GENERATOR to produce Chebyshev polynomials.

```
LN (FP_LN)
M3B2E         RST $28                 Call calculator.
              DEFB $3D                re-stack
              DEFB $31                duplicate
              DEFB $37                greater-0
              DEFB $00,$04            jump-true VALID
              DEFB $38                end-calc

REPORT-A
M3B35         RST $08                 Error: Invalid argument.
              DEFB $09

VALID
M3B37         DEFB $A0                stk-zero
              DEFB $02                delete
              DEFB $38                end-calc
              LD A,(HL)               Exponent (e) goes into A.
              LD (HL),$80             X is reduced to X'
              CALL STACK-A            Put A on calculator stack.
```

```
                RST $28                 Call calculator.
                DEFB $34 stk-data
                DEFB $38,$00            EXP 88 (00,00,00,00)
                DEFB $03                subtract
                DEFB $01                exchange
                DEFB $31                duplicate
                DEFB $34                stk-data
                DEFB $F0                EXP 80 $4C,$CC,$CC,$CD
                DEFB $4C,$CC,$CC,$CD
                DEFB $03                subtract
                DEFB $37                greater-0
                DEFB $00,$08            jump-true GRE.8
                DEFB $01                exchange
                DEFB $A1                stk-one
                DEFB $03                subtract
                DEFB $01                exchange
                DEFB $38                end-calc
                INC (HL)                Double X' to give 2*X'.
                RST $28                 Call calculator.

GRE.8
M3B58           DEFB $01                exchange
                DEFB $34                stk-data LN2
                DEFB $F0,$31,$72,$17,$F8 EXP 80 $31,$72,$17,$F8
                DEFB $04                multiply
                DEFB $01                exchange
                DEFB $A2                stk-half
                DEFB $03                subtract
                DEFB $A2                stk-half
                DEFB $03                subtract
                DEFB $31                duplicate
                DEFB $34                stk-data
                DEFB $32,$20            EXP 82 $20,(00,00,00,00)
                DEFB $04                multiply
                DEFB $A2                stk-half
                DEFB $03                subtract
                DEFB $8C                series12
                DEFB $11,$AC            EXP 61 AC,(00,00,00)
                DEFB $14,$09            EXP 64, 09,(00,00,00)
                DEFB $56,$DA,$A5        EXP 66 $DA,$A5,(00,00)
                DEFB $59,$30,$C5        EXP 69 $30,$C5,(00,00)
                DEFB $5C,$90,$AA        EXP 6C $90,$AA,(00,00)
                DEFB $9E,$70,$6F,$61    EXP 6E $70,$6F,$6,(00)
                DEFB $A1,$CB,$DA,$96    EXP 71 $CB,$DA,$96,(00)
                DEFB $A4,$31,$9F,$B4    EXP 74 $31,$9F,$B4,(00)
                DEFB $E7                EXP 77 $A0,$FE,$5C,$FC
                DEFB $A0,$FE,$5C,$FC
                DEFB $EA                EXP 7A $1B,$43,$CA,$36
                DEFB $1B,$43,$CA,$36
```

```
              DEFB $ED                EXP 7D $A7,$9C,$7E,$5E
              DEFB $A7,$9C,$7E,$5E
              DEFB $F0                EXP 80 $6E,$23,$80,$93
              DEFB $6E,$23,$80,$93
              DEFB $04                multiply
              DEFB $0F                add
              DEFB $38                end-calc
              RET
```

Reduce Argument Subroutine

Transforms the argument X of SIN X or COS X into a value V.

GET-ARGT (FP_ANGLE)

```
M3B9E RST $28                 Call calculator.
      DEFB $3D                re-stack
      DEFB $34                stk-data
      DEFB $EE                EXP 7E $22,$F9,$83,$6E
      DEFB $22,$F9,$83,$6E
      DEFB $04                multiply
      DEFB $31                duplicate
      DEFB $A2                stk-half
      DEFB $0F                addition
      DEFB $27                int
      DEFB $03                subtract
      DEFB $31                duplicate
      DEFB $0F                addition
      DEFB $31                duplicate
      DEFB $0F                addition
      DEFB $31                duplicate
      DEFB $2A                abs
      DEFB $A1                stk-one
      DEFB $03                subtract
      DEFB $31                duplicate
      DEFB $37                greater-0
      DEFB $C0                st-mem-0
      DEFB $00,$04            jump-true ZPLUS
      DEFB $02                delete
      DEFB $38                end-calc
      RET
```

ZPLUS

```
M3BBC DEFB $A1                stk-one
      DEFB $03                subtract
      DEFB $01                exchange
      DEFB $36                less-0
      DEFB $00,$02            jump-true YNEG
      DEFB $1B                negate
```

YNEG

```
M3BC3   DEFB $38                end-calc
        RET
```

COSINE Function

Handles the function COS X and returns a last value that is an approximation to COS X.

COS (FP_COS)

```
M3BC5   RST $28                 Call calculator.
        DEFB $39                get-argt
        DEFB $2A                abs
        DEFB $A1                stk-one
        DEFB $03                sub
        DEFB $E0                get-mem-0
        DEFB $00,$06            jump-true C-ENT
        DEFB $1B                negate
        DEFB $33,$03            jump C-ENT
```

SINE Function

Handles the function SIN X. Uses SERIES GENERATOR to produce Chebyshev polynomials.

SIN (FP_SIN)

```
M3BD0   RST $28                 Call calculator.
        DEFB $39                get-argt
```

C-ENT

```
M3BD2   DEFB $31                duplicate
        DEFB $31                duplicate
        DEFB $04                multiply
        DEFB $31                duplicate
        DEFB $0F                addition
        DEFB $A1                stk-one
        DEFB $03                subtract
        DEFB $86                series-06
        DEFB $14,$E6            EXP 64 $E6,(00,00,00)
        DEFB $5C,$1F,$0B        EXP 6C $1F,$0B,(00,00)
        DEFB $A3,$8F,$38,$EE    EXP 73 $8F,$38,$EE,(00)
        DEFB $E9                EXP 79 $15,$63,$BB,$23
        DEFB $15,$63,$BB,$23
        DEFB $EE                EXP 7E $92,$0D,$CD,$ED
        DEFB $92,$0D,$CD,$ED
        DEFB $F1
        DEFB $23,$5D,$1B,$EA    EXP 81 $23,$5D,$1B,$EA
        DEFB $04                multiply
        DEFB $38                end-calc
        RET
```

TAN FUNCTION

Handles the function TAN X. Returns SIN X/COS X, with overflow if COS X = 0.

TAN (FP_TAN)

```
M3BF5   RST $28         Call calculator.
        DEFB $31        duplicate
        DEFB $1F        sin
        DEFB $01        exchange
        DEFB $20        cos
        DEFB $05        divide
        DEFB $38        end-calc
        RET
```

ARCTAN Function

Handles the function ATN X. Uses SERIES GENERATOR to produce Chebyshev polynomials. It returns a real number between -PI/2 and PI/2, which is equal to the value in radians of the angle whose tan is X.

ATN (FP_ATN)

```
M3BFD   CALL RE-STACK   Use floating point form of X.
        LD A,(HL)       Get exponent of X.
        CP $81
        JR C, SMALL     Jump forward where i: Y=X.
        RST $28         Call calculator.
        DEFB $A1        stack-one
        DEFB $1B        negate
        DEFB $01        exchange
        DEFB $05        divide
        DEFB $31        duplicate
        DEFB $36        less-0
        DEFB $A3        stk-pi/2
        DEFB $01        exchange
        DEFB $00,$06    jump-true CASES
        DEFB $1B        negate
        DEFB $33,$03    jump CASES
```

SMALL

```
M3C13   RST $28
        DEFB $A0        stk-zero
```

CASES

```
M3C15   DEFB $01        exchange
        DEFB $31        duplicate
        DEFB $31        duplicate
        DEFB $04        multiply
        DEFB $31        duplicate
        DEFB $0F        addition
        DEFB $A1        stk-on
        DEFB $03        subtract
```

```
        DEFB $8C                series-0C
        DEFB $10,$B2            EXP 60 $B2,(00,00,00)
        DEFB $13,$0E            EXP 63 $0E,(00,00,00)
        DEFB $55,$E4,$8D        EXP 65 $E4,$8D,(00,00)
        DEFB $58,$39,$BC        EXP 68 $39,$BC,(00,00)
        DEFB $5B,$98,$FD        EXP 6B $98,$FD,(00,00)
        DEFB $9E,$00,$36,$75    EXP 6E $00,$36,$75,(00)
        DEFB $A0,$DB,$E8,$B4    EXP 70 $DB,$E8,$B4,(00)
        DEFB $63,$42,$C4        EXP 73 $42,$C4,(00,00)
        DEFB $E6                EXP 76 $B5,$09,$36,$BE
        DEFB $B5,$09,$36,$BE
        DEFB $E9                EXP 79 $36,$73,$1B,$5D
        DEFB $36,$73,$1B,$5D
        DEFB $EC                EXP 7C $D8,$DE,$63,$BE
        DEFB $D8,$DE,$63,$BE
        DEFB $F0                EXP 80 $61,$A1,$B3,$0C
        DEFB $61,$A1,$B3,$0C
        DEFB $04                multiply
        DEFB $0F                addition
        DEFB $38                end-calc
        RET
```

ARCSIN Function

Handles the function ASN X and returns a real number from -PI/2 to PI/2 (inclusive) which is equal to the value in radians of the angle whose sine is X. Thereby if Y = ASN X then X = SIN Y.

ASN (FP_ASN)

```
M3C4E   RST $28             Call calculator.
        DEFB $31            duplicate
        DEFB $31            duplicate
        DEFB $04            multiply
        DEFB $A1            stk-one
        DEFB $03            subtract
        DEFB $1B            negate
        DEFB $28            sqr
        DEFB $A1            stk-one
        DEFB $0F            addition
        DEFB $05            division
        DEFB $24            atn
        DEFB $31            duplicate
        DEFB $0F            addition
        DEFB $38            end-calc
        RET
```

ARCCOS Function

Handles the function ACS X and returns a real number from zero to PI (inclusive) which is equal to the value in radians of the angle whose cosine is X.

ACS (FP_ACS)

```
M3C5E   RST $28         Call calculator.
        DEFB $22        asn
        DEFB $A3        stk-PI/2
        DEFB $03        subtract
        DEFB $1B        negate
        DEFB $38        end-calc
        RET
```

SQUARE ROOT Function

Handles the function SQR X and returns the positive square root of the real number X if X is positive, and zero if X is zero.

SQR (FP_ROOT)

```
M3C65   RST $28         Call calculator.
        DEFB $31        duplicate
        DEFB $30        not
        DEFB $00,$1E    jump-true LAST
        DEFB $A2        stk-half
        DEFB $38        end-calc
```

EXPONENTIATION Operation

Performs the binary operation of raising the first number, X, to the power of the second number, Y.

TO-POWER (FP_TO_THE)

```
M3C6C   RST $28         Call calculator.
        DEFB $01        exchange
        DEFB $31        duplicate
        DEFB $30        not
        DEFB $00,$07    jump-true XIS0
        DEFB $25        ln
        DEFB $04        multiply
        DEFB $38        ex-calc
        JP EXP
```

XIS0

```
M3C78   DEFB $02        delete
        DEFB $31        dup
        DEFB $30        not
        DEFB $00,$09    jump-true ONE
        DEFB $A0        stk-zero
        DEFB $01        exchange
        DEFB $37        greater-0
```

```
              DEFB $00,$06          jump-true LAST
              DEFB $A1              stk-one
              DEFB $01              exchange
              DEFB $05              division

ONE
M3C85         DEFB $02              delete
              DEFB $A1              stk-one

LAST
M3C87         DEFB $38              end-calc
              RET
```

Tape Messages

```
SEPRMT
M3C89         DEFM $80&"Start tape, then press any key"&('.'+$80)
              DEFM $0D&"Program:"&(' '+$80)
              DEFM $0D&"Number array:"&(' '+$80)
              DEFM $0D&"Character array:"&(' '+$80)
              DEFM $0D&"Bytes:"&(' '+$80)
              DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
              DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
              DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
              DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
              DEFB $FF,$FF,$FF,$FF
```

Character Set

```
CHRSET
M3D00   DEFB $00,$00,$00,$00,$00,$00,$00,$00
        DEFB $00,$10,$10,$10,$10,$00,$10,$00   !
        DEFB $00,$24,$24,$00,$00,$00,$00,$00   "
        DEFB $00,$24,$7E,$24,$24,$7E,$24,$00   #
        DEFB $00,$08,$3E,$28,$3E,$0A,$3E,$08   $
        DEFB $00,$62,$64,$08,$10,$26,$46,$00   %
        DEFB $00,$10,$28,$10,$2A,$44,$3A,$00   &
        DEFB $00,$08,$10,$00,$00,$00,$00,$00   '
        DEFB $00,$04,$08,$08,$08,$08,$04,$00   (
        DEFB $00,$20,$10,$10,$10,$10,$20,$00   )
        DEFB $00,$00,$14,$08,$3E,$08,$14,$00   *
        DEFB $00,$00,$08,$08,$3E,$08,$08,$00   +
        DEFB $00,$00,$00,$00,$00,$08,$08,$10   ,
        DEFB $00,$00,$00,$00,$3E,$00,$00,$00   -
        DEFB $00,$00,$00,$00,$00,$18,$18,$00   .
        DEFB $00,$00,$02,$04,$08,$10,$20,$00   /
        DEFB $00,$3C,$46,$4A,$52,$62,$3C,$00   0
        DEFB $00,$18,$28,$08,$08,$08,$3E,$00   1
        DEFB $00,$3C,$42,$02,$3C,$40,$7E,$00   2
        DEFB $00,$3C,$42,$0C,$02,$42,$3C,$00   3
        DEFB $00,$08,$18,$28,$48,$7E,$08,$00   4
        DEFB $00,$7E,$40,$7C,$02,$42,$3C,$00   5
        DEFB $00,$3C,$40,$7C,$42,$42,$3C,$00   6
        DEFB $00,$7E,$02,$04,$08,$10,$10,$00   7
        DEFB $00,$3C,$42,$3C,$42,$42,$3C,$00   8
        DEFB $00,$3C,$42,$42,$3E,$02,$3C,$00   9
        DEFB $00,$00,$00,$10,$00,$00,$10,$00   :
        DEFB $00,$00,$10,$00,$00,$10,$10,$20   ;
        DEFB $00,$00,$04,$08,$10,$08,$04,$00   <
        DEFB $00,$00,$00,$3E,$00,$3E,$00,$00   =
        DEFB $00,$00,$10,$08,$04,$08,$10,$00   >
        DEFB $00,$3C,$42,$04,$08,$00,$08,$00   ?
        DEFB $00,$3C,$4A,$56,$5E,$40,$3C,$00   @
        DEFB $00,$3C,$42,$42,$7E,$42,$42,$00   A
        DEFB $00,$7C,$42,$7C,$42,$42,$7C,$00   B
        DEFB $00,$3C,$42,$40,$40,$42,$3C,$00   C
        DEFB $00,$78,$44,$42,$42,$44,$78,$00   D
        DEFB $00,$7E,$40,$7C,$40,$40,$7E,$00   E
        DEFB $00,$7E,$40,$7C,$40,$40,$40,$00   F
        DEFB $00,$3C,$42,$40,$4E,$42,$3C,$00   G
```

```
DEFB $00,$42,$42,$7E,$42,$42,$42,$00   H
DEFB $00,$3E,$08,$08,$08,$08,$3E,$00   I
DEFB $00,$02,$02,$02,$42,$42,$3C,$00   J
DEFB $00,$44,$48,$70,$48,$44,$42,$00   K
DEFB $00,$40,$40,$40,$40,$40,$7E,$00   L
DEFB $00,$42,$66,$5A,$42,$42,$42,$00   M
DEFB $00,$42,$62,$52,$4A,$46,$42,$00   N
DEFB $00,$3C,$42,$42,$42,$42,$3C,$00   O
DEFB $00,$7C,$42,$42,$7C,$40,$40,$00   P
DEFB $00,$3C,$42,$42,$52,$4A,$3C,$00   Q
DEFB $00,$7C,$42,$42,$7C,$44,$42,$00   R
DEFB $00,$3C,$40,$3C,$02,$42,$3C,$00   S
DEFB $00,$FE,$10,$10,$10,$10,$10,$00   T
DEFB $00,$42,$42,$42,$42,$42,$3C,$00   U
DEFB $00,$42,$42,$42,$42,$24,$18,$00   V
DEFB $00,$42,$42,$42,$42,$5A,$24,$00   W
DEFB $00,$42,$24,$18,$18,$24,$42,$00   X
DEFB $00,$82,$44,$28,$10,$10,$10,$00   Y
DEFB $00,$7E,$04,$08,$10,$20,$7E,$00   Z
DEFB $00,$0E,$08,$08,$08,$08,$0E,$00   [
DEFB $00,$00,$40,$20,$10,$08,$04,$00   \
DEFB $00,$70,$10,$10,$10,$10,$70,$00   ]
DEFB $00,$10,$38,$54,$10,$10,$10,$00   (up arrow)
DEFB $00,$00,$00,$00,$00,$00,$00,$FF   _
DEFB $00,$1C,$22,$78,$20,$20,$7E,$00   (pound)
DEFB $00,$00,$38,$04,$3C,$44,$3C,$00   a
DEFB $00,$20,$20,$3C,$22,$22,$3C,$00   b
DEFB $00,$00,$1C,$20,$20,$20,$1C,$00   c
DEFB $00,$04,$04,$3C,$44,$44,$3C,$00   d
DEFB $00,$00,$38,$44,$78,$40,$3C,$00   e
DEFB $00,$0C,$10,$18,$10,$10,$10,$00   f
DEFB $00,$00,$3C,$44,$44,$3C,$04,$38   g
DEFB $00,$40,$40,$78,$44,$44,$44,$00   h
DEFB $00,$10,$00,$30,$10,$10,$38,$00   i
DEFB $00,$04,$00,$04,$04,$04,$24,$18   j
DEFB $00,$20,$28,$30,$30,$28,$24,$00   k
DEFB $00,$10,$10,$10,$10,$10,$0C,$00   l
DEFB $00,$00,$6C,$92,$92,$92,$92,$00   m
DEFB $00,$00,$78,$44,$44,$44,$44,$00   n
DEFB $00,$00,$38,$44,$44,$44,$38,$00   o
DEFB $00,$00,$78,$44,$44,$78,$40,$40   p
DEFB $00,$00,$3C,$44,$44,$3C,$04,$06   q
DEFB $00,$00,$1C,$20,$20,$20,$20,$00   r
DEFB $00,$00,$38,$40,$38,$04,$78,$00   s
DEFB $00,$10,$38,$10,$10,$10,$0C,$00   t
DEFB $00,$00,$44,$44,$44,$44,$38,$00   u
DEFB $00,$00,$44,$44,$28,$28,$10,$00   v
DEFB $00,$00,$92,$92,$92,$92,$6C,$00   w
DEFB $00,$00,$44,$28,$10,$28,$44,$00   x
```

```
DEFB $00,$00,$44,$44,$44,$3C,$04,$38    y
DEFB $00,$00,$7C,$08,$10,$20,$7C,$00    z
DEFB $00,$0E,$08,$30,$08,$08,$0E,$00    {
DEFB $00,$08,$08,$08,$08,$08,$08,$00
DEFB $00,$70,$10,$0C,$10,$10,$70,$00    }
DEFB $00,$14,$28,$00,$00,$00,$00,$00
DEFB $3C,$42,$99,$A1,$A1,$99,$42,$3C   (copyright)
```

Extension ROM

In order to add new commands and extend the Timex/Sinclair 2068, Timex engineers and programmers moved some core routines to a second, 8K ROM.

The Extension ROM (EXROM) is paged in/out using the same SCLD registers that make DOCK cartridges possible: the *Display Enhancement Register* ($FF) and the *Horizontal Select Register* ($F4).

Timex added a dispatcher, a body of code that was copied to RAM at startup that provided a common programming interface to many commonly used routines. Timex provided the dispatcher to allow for future ROM upgrades. If all programmers used the dispatcher, their code would theoretically be future-proof.

This ROM also contains routines to support the extra video modes Timex added to the TS 2068, routines for bank switching and other service routines.

EXROM

XBASIC

General routines used when the EXROM is switched in.

XRST0
The very first bit of code handles the case where the computer started with the EXROM, instead of HOME ROM, enabled. We shouldn't end up here, but in case we do, handle the exception.

```
M0000   DI                      Disable interrupts.
        JR EXROM-STARTUP        Jump to bank transfer code.
```

XRST8
This version of RST 8 handles any RST 8 that happens when EXROM is enabled, sending the event back to the HOME ROM through the dispatcher.

```
M0008   LD HL,(CH_ADD)          HL gets the address of the character in dispute.
        LD (X_PTR),HL           put it into the "syntax error" character pointer.
        POP HL                  The address on the stack points to the error
        LD L,(HL)               code; get error code and put it into
        LD (IY+OERRNR),L        ERR_NR.
        LD SP,(ERRSP)           Get the error stack pointer.
        LD HL, $1354            Address of SET-STK.
        PUSH HL
M001B   LD H,$FF                Address for home bank.
        LD L,$00
        PUSH HL
M0020   PUSH AF                 Save AF.
        LD A,(VIDMOD)
        AND A                   Check the video mode.
        NOP
        JR Z, XRST8-NORMVID     Jump if normal video.
M0028   POP AF                  Restore AF.
        CALL GOTO_BANK-HI       Jump to RAM service when in extended
                                video mode.
```

XRST8-NORMVID

```
M002C   POP AF                  Restore AF.
        CALL GOTO_BANK          Jump to RAM service when in normal video
                                mode.
```

XRST38
Interrupt routine. Called when the keyboard interrupt is received from the SCLD.

```
M0038   PUSH AF              Save AF and
        DI                   disable interrupts.
        LD A,(VIDMOD)        Check the video mode.
        AND A
        NOP
        JR Z, KEYB-NORMVID   Jump if only one display file active.
```

KEYB-EXTVID
```
M0041   POP AF               Restore AF.
        JP MFA6E             Jump to RAM INT service when in extended
                             video mode (two display files).
```

KEYB-NORMVID
```
M0045   POP AF               Restore AF. Jump to RAM INT service
        JP X38INT            when in normal video mode.
```

EXROM-STARTUP
Code executed on power-up if the SCLD has reset to in such a state so as to point to EXROM.

```
M0049   LD A,$01             Set HSR to point lower 8K to DOCK/
        OUT (HSR),A          EXROM.
        JR REBOOT
```

MOVE-TO-$6000
This section of code is transferred to HOME RAM by the REBOOT code fragment during a reset that starts with EXROM active. The short routine ensures that all chunks of the HOME ROM and RAM are active, sets DE to $FFFF and then jumps to the START/NEW routine in the HOME ROM.

```
M004F   XOR A                Clear A.
        OUT (HSR),A          Set HSR to selects all chunks to HOME/EXROM.
        OUT (DECR),A         Bank select now transfers us to HOME.
        LD DE,$FFFF          Bank $FFFF.
        JP START/NEW         Routine in HOME ROM that initializes the
                             computer.
```

REBOOT
```
M005A   LD HL,$004F          Starting address of bytes to transfer.
        LD DE,$6000          Where to transfer it.
        LD BC,$000B          How many bytes to transfer.
        LDIR                 Copy the bytes.
        JP M6000             Jump to the routine and continue with
                             initialization.
```

Cassette Handling Routines

The vast majority of the cassette code in the ZX Spectrum was moved to the EXROM to make room for the cartridge routines and new BASIC commands.

SA-BYTES Subroutine

This subroutine is called to SAVE the header and later the program/data block.

Writes the number of bytes specified in DE from memory to tape. The IX register points to the first byte to save. This byte should be preceded by the byte in A which is used to identify the type of block: $00 for data, $FF for header.

This routine aborts with Report code D if the BREAK key is pressed.

SA-BYTES (W_TAPE)

```
M0068   LD HL,$00E5         Address for SA/LD-RET is pushed
        PUSH HL             as common exit route.
        LD HL,$1F80         Number of half-cycles for the header,
                            approximately 5 seconds.
        BIT 7,A             Test bit 7,
        JR Z, SA-FLAG       skip to SA-FLAG if a header is being saved.
        LD HL,$0C98         Set for the two-second gap after header.
```

SA-FLAG

```
M0076   EX AF,AF'           Save the the flags.
        INC DE              Increase length by one.
        DEC IX              Decrease start.
        DI                  Don't interrupt the tape output.
        LD A,$02            Start sound with '0' and set border red
        LD B,A              (alternates with cyan).
```

SA-LEADER

Write the tape header.

```
M007E   DJNZ SA-LEADER      Delay loop for tone output.
        OUT ($FE),A         Output byte $02/$0D to tape port.
        XOR $0F             Flip the border color and tape output signal.
        LD B,$A4            Hold count. Half-cycle time for 806.5Hz.
        DEC L               jump if header not done.
        JR NZ, SA-LEADER
        DEC B               Adjust B for processing time.
        DEC H               Jump back if header not done.
        JP P, SA-LEADER
        LD B,$2F            First half-cycle of the 2400Hz transition.
```

SA-SYNC-1

```
M0090   DJNZ SA-SYNC-1      Self loop.
        OUT ($FE),A         Switch to mic on and red.
        LD A,$0D            Prep for mic off and cyan.
        LD B,$37            A short delay.
```

SA-SYNC-2

```
M0098   DJNZ SA-SYNC-2      Self loop.
        OUT ($FE),A         Output mic off, cyan border.
        LD BC,$3B0E         B=half-cycle frequency for 2400Hz. Tape output
                            high, border = YELLOW.
        EX AF,AF'           Restore the saved flag, which is 1st byte to be
                            saved.
        LD L,A              Transfer to L.
        JP SA-START         Jump forward to SA-START.
```

SA-LOOP

During the save loop a parity byte is maintained in H. The save loop begins by testing if reduced length is zero and if so, the final parity byte is saved reducing count to $FFFF.

```
M00A4   LD A,D              Get the high byte.
        OR E                Test against low byte.
        JR Z, SA-PARITY     Forward to SA-PARITY if zero.
        LD L,(IX+$00)       Load currently addressed byte in L.
```

SA-LOOP-P

```
M00AB   LD A,H              Get parity byte.
        XOR L               XOR with next byte.
```

SA-START

```
M00AD   LD H,A              Put parity byte back in H.
        LD A,$01            Tape output low, border = blue
        SCF                 Set carry flag ready to rotate in.
        JP SA-8-BITS        Start sending the byte.
```

SA-PARITY

```
M00B4   LD L,H              Transfer running parity byte to L
        JR SA-LOOP-P        and back to SA-LOOP-P.
```

SA-BIT-2

```
M00B7   LD A,C              Tape output high, border = YELLOW
        BIT 7,B             Set zero flag. This will indicate that the 2nd
                            half-cycle has been sent.
```

SA-BIT-1

```
M00BA   DJNZ SA-BIT-1       Wait a half-cycle.
        JR NC, SA-OUT       Jump forward if data bit is '0'
        LD B,$42            Wait a bit longer for a '1'.
```

SA-SET

```
M00C0   DJNZ SA-SET         Self loop, roughly 66*13 clock cycles.
```

SA-OUT

```
M00C2   OUT ($FE),A       Flip the bit.
        LD B,$3E          Set up delay.
        JR NZ,SA-BIT-2    Jump if next half-cycle is still to be sent.
        DEC B             B = $3D (make up for processing time).
        XOR A             Zero A, clear carry flag and
        INC A             reset zero flag.
```

SA-8-BITS

```
M00CB   RL L              Rotate left through carry.
        JP NZ,SA-BIT-1    Jump back until all 8 bits are done.
        DEC DE            Decrement the byte count,
        INC IX            point to the next byte to send.
        LD B,$31          Half-cycle value for a '0'.
        LD A,$7F          Test for space key and
        IN A,($FE)        return to common exit (to restore border)
        RRA               return if space key press detected.
        RET NC            Return to SA/LD-RET.
        LD A,D            Continue if more bytes are waiting
        INC A             to be written.
        JP NZ,SA-LOOP     Jump to SA-LOOP if more bytes.
        LD B,$3B          Wait for one last half-cycle.
```

SA-DELAY

```
M00E2   DJNZ SA-DELAY     Self loop.
        RET               And done.
```

SA/LD-RET Subroutine

The address of this routine is pushed on the stack prior to any load/save operation and it handles normal completion with the restoration of the border and also abnormal termination when the break key (space key) is pressed during a tape operation.

SA/LD-RET (W_BORD)

```
M00E5   PUSH AF           Save AF.
        LD A,(BORDCR)     Get border color.
        AND $38           Mask off PAPER bits.
        RRCA              Rotate to
        RRCA              get number in 0-7.
        RRCA
        OUT ($FE),A       Set the border color.
        LD A,$7F          Read from port address $7FFE, the row
        IN A,($FE)        with space key at outside.
        RRA               Test for space key.
        EI                Enable interrupts.
        JR C, SA/LD-END   Jump if space key was not pressed.
```

REPORT-D

```
        RST $08           Error: BREAK.
        DEFB $0C
```

SA/LD-END

```
M00FA   POP AF          Restore AF.
        RET
```

LD-BYTES Subroutine

This routine is used to load bytes. On entry A is set to $00 for a header or to $FF for data. IX points to the start of receiving location and DE holds the length of bytes to be loaded. If carry flag is set on entry, data is loaded. If carry flag is reset, data is verified.

Routine will abort with Report code D if the BREAK key is pressed.

LD-BYTES (R_TAPE)

```
M00FC   INC D           Reset the zero flag without disturbing carry.
        EX AF,AF'       Preserve the entry flags.
        DEC D           Restore high byte of length.
        DI              Disable interrupts.
        LD A,$0F        Make the border white and mic off.
        OUT ($FE),A     Output to port.
        LD HL,$00E5     Address of SA/LD-RET
        PUSH HL         is saved on stack as terminating routine.
        IN A,($FE)      Read the ear state (bit 6)
        RRA             and rotate to bit 5.
        AND $20         Mask for the tape bit.
        OR $02          Combine with red color.
        LD C,A          Store initial state long-term in C.
        CP A            Set the zero flag.
```

LD-BREAK

```
M0111   RET NZ          Return if at any time space is pressed.
```

LD-START

```
M0112   CALL LD-EDGE-1  Check for transition on the tape input
        JR NC, LD-BREAK Back to LD-BREAK with time out and
                        no edge present on tape.
```

Transition detected, try to read the tape.

```
        LD HL,$0415     Set up 16-bit outer loop counter for approx 1
                        second delay.
```

LD-WAIT

```
M011A   DJNZ LD-WAIT    Self-loop.
        DEC HL          Decrease outer loop counter.
        LD A,H          Jump if HL is not zero.
        OR L
        JR NZ, LD-WAIT  Back to LD-WAIT if not zero, with B=0.
        CALL LD-EDGE-2  Get a bit of data.
        JR NC,LD-BREAK  Back to LD-BREAK if no edges at all.
```

LD-LEADER

```
M0126   LD B,$9C          Set timing value.
        CALL LD-EDGE-2    Get a bit of data.
        JR NC,LD-BREAK    Back to LD-BREAK with time-out.

        LD A,$C6          Two edges must be spaced apart.
        CP B
        JR NC,LD-START    Back to LD-START if too close together for
                          lead-in.
        INC H             Proceed to test 256 edged sample.
        JR NZ,LD-LEADER   Back if not enough cycles detected.
```

LD-SYNC

```
M0135   LD B,$C9          Initial timing value to B.
        CALL LD-EDGE-1    Check again for the edge of a bit.
        JR NC,LD-BREAK    Back to LD-BREAK with time-out.
        LD A,B            Get augmented timing value from B.
        CP $D4            Compare.
        JR NC,LD-SYNC     Back to LD-SYNC if gap is too big.
        CALL LD-EDGE-1    Check for the edge of a bit.
        RET NC            Return with time-out.
        LD A,C            Get long-term mask from C
        XOR $03           and make it blue/yellow.
        LD C,A            Store new long-term mask.
        LD H,$00          Set up parity byte to zero.
        LD B,$B0          Set timing value.
        JR LD-MARKER      Jump forward.
```

LD-LOOP

```
M014F   EX AF,AF'         Restore entry flags.
        JR NZ,LD-FLAG     Forward if awaiting initial flag, which is to be
                          discarded.
        JR NC,LD-VERIFY   Forward if not loading.
        LD (IX+$00),L     Save received byte at (IX).
        JR LD-NEXT
```

LD-FLAG

```
M0159   RL C              Preserve carry (verify) flag in long-term state byte.
                          Ok to lose bit 7.
        XOR L             Compare type in A with first byte in L.
        RET NZ            Return if no match (ie CODE vs DATA).
        LD A,C            Get byte with stored carry.
        RRA               Rotate to carry flag again.
        LD C,A            Restore long-term port state.
        INC DE            Increment counter to compensate for its
        JR LD-DEC         decrease after the jump.
```

LD-VERIFY

```
M0163   LD A,(IX+$00)       Get the last byte received.
        XOR L               Compare with that on tape.
        RET NZ              Return if not a match.
```

LD-NEXT

```
M0168   INC IX              Increment the byte pointer.
```

LD-DEC

```
M016A   DEC DE              Decrement the block length.
        EX AF,AF'           Store the flags.
        LD B,$B2            Set timing value.
```

LD-MARKER

```
M016E   LD L,$01            Initialize to %00000001.
```

LD-8-BITS

```
M0170   CALL LD-EDGE-2      Read a bit.
        RET NC              Return with time-out.
        LD A,$CB            Comparison value.
        CP B                Compare to incremented value in B. If B is higher,
                            then bit on tape was set. If lower, bit on tape was
                            reset.
        RL L                Rotate the carry bit in to L.
        LD B,$B0            Reset B timer byte.
        JP NC,LD-8-BITS     Jump back until carry is set.
        LD A,H              Get the running parity byte.
        XOR L               Include the new byte.
        LD H,A              Store back in parity register.
        LD A,D              Check length of
        OR E                expected bytes.
        JR NZ,LD-LOOP       Back while there are more bytes.
        LD A,H              Get parity byte.
        CP $01              Set carry if zero.
        RET
```

LD-EDGE-2 and LD-EDGE-1 Subroutines

An edge is a transition from one mic state to another, which is registered as a change in bit 6 of value input from port $FE.

The first entry point (LD-EDGE-2) looks for two adjacent edges (the length of a complete pulse). The second entry point (LD-EDGE-1) is used to find the time before the next edge.

B holds a count, up to 256, within which the edge (or edges) must be found. The gap between two edges will be more for a '1' than a ‘0’, so the value of B denotes the state of the bit (two edges) read from tape. C contains the previous border color and 'edge-type'.

The subroutines return with the carry flag set if the required number of 'edges' has been found in the time allowed; and the change to the value in the B register shows just how long it took to find the 'edge(s)'. The carry flag will be reset if there is an error.

The zero flag then signals 'BREAK pressed' by being reset, or 'time-up' by being set.

LD-EDGE-2 (RD_BIT)

```
M0189   CALL LD-EDGE-1      Find the tape tone.
        RET NC              Return if space pressed to time-out.
```

LD-EDGE-1 (R_EDGE)

```
M018D   LD A,$16            Delay value.
```

LD-DELAY

```
M018F   DEC A               Decrement counter.
        JR NZ, LD-DELAY     Loop back 22 times.
        AND A               Reset carry flag.
```

LD-SAMPLE

```
M0193   INC B               Increment the time-out counter.
        RET Z               Return with failure when $FF passed.
        LD A,$7F            Prepare to read keyboard and ear port.
        IN A,($FE)          Read the keyboard and the tape input bit
        RRA                 (tape bit 6; space is bit 0), move bit 6 to 5.
        RET NC              Return if space/BREAK was pressed.
        XOR C               Compare with initial long-term state.
        AND $20             Isolate bit 5.
        JR Z, LD-SAMPLE     Back to LD-SAMPLE if no edge.
        LD A,C              Get comparison value.
        CPL                 Switch the bits.
        LD C,A              Save it back.
        AND $07             Mask for the BORDER color.
        OR $08              Set the tape out bit to 0.
        OUT ($FE),A         Output the new border color and tape bit
        SCF                 Set carry flag to signal edge found within
                            time allowed.
        RET
```

SAVE, LOAD, VERIFY & MERGE Command Routines

Entry point for all four tape commands. The low TADDR byte distinguishes between the four commands. The first part of the following routine constructs the header information in the work space.

Because these tape routines were originally in the HOME ROM, a method of replacing calls to other HOME ROM routines had to be substituted. The common code block sets up CALL-BANK, which invokes the routine in the HOME ROM. Here's the code block:

```
        PUSH IX             Save the return address.
        EXX                 Switch to the alternate register set.
        LD HL, {ROUTINE}    Push the address of {ROUTINE} to HL.
        PUSH HL             Push it to the stack.
```

```
              LD L, $00               Horizontal select value for HOME ROM.
              LD H, HOMEROM           HOME ROM bank ($FF).
              PUSH HL                 Push that to the stack.
              LD HL,$0000             And push
              PUSH HL                 PRM_IN and
              PUSH HL                 PRM_OUT.
              EXX                     Switch back to regular register set.
              CALL CALL-BANK          Call the routine in the HOME bank.
              POP IX                  Restore the return address.
```

SAVE-ETC (SLVM)

```
M01AB         LD A, (TADDR)           Get low byte of the syntax table pointer.
              LD BC, P-SAVE+1         Address of P-SAVE in the syntax table,
              SUB C                   generate a value based on the instruction.
              LD (TADDR),A            Put back in TADDR.
                                      $00 - SAVE
                                      $01 - LOAD
                                      $02 - VERIFY
                                      $03 - MERGE
```

Get the file name. The string parameters will be on the top of the floating point stack. Evaluate the next expression and return here if it is a string.

```
              PUSH IX                 Save the return address.
              EXX                     Switch to the alternate register set.
              LD HL, EXPT-EXP         Push the address of EXPT-EXP to HL.
              PUSH HL                 Push it to the stack.
              LD L, $00               Horizontal select value for HOME ROM.
              LD H, HOMEROM           HOME ROM bank ($FF).
              PUSH HL                 Push that to the stack.
              LD HL,$0000             And push
              PUSH HL                 PRM_IN and
              PUSH HL                 PRM_OUT.
              EXX                     Switch back to regular register set.
              CALL CALL-BANK          Call the routine in the HOME bank.
              POP IX                  Restore the return address.

              BIT INTPT,(IY+OFLAGS)   Jump if syntax checking.
              JR Z, SD-DATA
```

Prepare to allocate a 17 byte header buffer if the value at TADDR is zero (SAVE).

```
              LD BC,$0011             17 bytes.
              LD A,(TADDR)            Jump if SAVE (only one buffer needed).
              AND A                   And to itself, which results in $00 if SAVE.
              JR Z, SA-SPACE          Jump ahead if SAVE.
```

Otherwise, allocate two header buffers worth of memory. They are needed for LOAD, VERIFY, and MERGE. One buffer is for data coming from tape, the other for the

arguments coming from the command line. DE points to the start of the newly allocated memory.

```
        LD C,$22
```

SA-SPACE
Make the space for the buffer(s).

```
M01DD   PUSH IX                 See the annotated example under SAVE-ETC.
        EXX
        LD HL,BC-SPACES         Push address for BC-SPACES to stack.
        PUSH HL
        LD L,$00
        LD H, HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
```

Return here if the buffer(s) could be allocated.

```
        PUSH DE                 Copy the location of the header buffer
        POP IX                  to IX.
```

Ten spaces are required for the default filename but it is simpler to overwrite the first file-type indicator byte as well.

```
        LD B,$0B                Set counter to 11.
        LD A,' '                And prepare a space.
```

SA-BLANK

```
M01FB   LD (DE),A               Set workspace location to space.
        INC DE                  Increment to next location.
        DJNZ SA-BLANK           Loop back until all eleven are cleared.
        LD (IX+$01),$FF         Set first byte of ten character filename to
                                $FF to signal a null string.
```

Check the name provided to the command, get the string descriptor from the calculator stack.

```
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,STK-FETCH
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
```

```
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        LD HL, $FFF6        Test the length of the program name
        DEC BC              (a length of zero will cause an overflow
        ADD HL,BC           hence an error).
        INC BC              Restore true length.
        JR NC, SA-NAME      Jump forward if the name was less than
                            11 characters.
```

File name is more than 10 characters or null string.

```
        LD A,(TADDR)        Get the command from TADDR.
        AND A               Test if it is SAVE ($00).
        JR NZ, SA-NULL      Jump forward if not SAVE.
```

REPORT-F
No more than 10 characters are allowed for SAVE.

```
        RST $08             Error: Invalid file name.
        DEFB $0E
```

SA-NULL
Continue with LOAD, MERGE, VERIFY and SAVE within 10 character limit.

```
M022A   LD A,B              Test if length of filename
        OR C                is zero.
        JR Z, SA-DATA       Jump forward if zero or null.
        LD BC,$000A         Otherwise, trim length to 10.
```

SA-NAME

```
M0231   PUSH IX             Save the header header buffer pointer
        POP HL              and pop in to HL.
        INC HL              HL now points to first byte of filename.
        EX DE,HL            Transfer destination address to DE, start
                            of string in command to HL.
        LDIR                Copy up to 10 bytes, less than 10 if spaces
                            follow.
```

SA-DATA
Call GETCURCH, skipping spaces and control characters.

```
M0238   PUSH IX             See annotated example under SAVE-ETC.
        EXX
        LD HL, GETCURCH
        PUSH HL
        LD L,$00
        LD H,HOMEROM
```

```
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        CP $E4              Is the character after filename the DATA token?
        JP NZ, SA-SCR$      No, jump forward to check for SCREEN$.
        LD A,(TADDR)        Get command from TADDR.
        CP $03              Is it MERGE?
        JP Z, BADBAS        Declare a syntax error: can't MERGE DATA.
```

Get the next character and then attempt to find it in the variables area.

```
        PUSH IX             See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-CHAR
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        EXX                 Back-to-back HOME ROM call.
        LD HL,FIND_N        Call FIND_N to look for the variable.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        SET 7,C             Force an array name.
        JR NC, SA-V-OLD     Jump if we found the variable name.
        LD HL,$0000         Signal 'using a new array.'
        LD A,(TADDR)        Check value of TADDR and give
        DEC A               error if trying to SAVE or VERIFY new array
        JR Z,SA-V-NEW       Otherwise, jump forward.
```

REPORT-2

```
        RST $08             Error: Variable not found.
        DEFB $01
```

SA-V-OLD (SLVBADBAS)

```
M0295   JP NZ,BADBAS            Variable found was not an array variable or a
                                string.
        BIT INTPT,(IY+OFLAGS)   Jump forward if syntax checking.
        JR Z,SA-DATA-1
        INC HL                  Point to the 'low length' of the variable.
        LD A,(HL)               Low byte goes into
        LD (IX+BLKLEN),A        workspace, followed by
        INC HL                  the high byte.
        LD A,(HL)
        LD (IX+BLKLEN+1),A
        INC HL                  Step past length bytes.
```

SA-V-NEW

```
M02A9   LD (IX+ARG1+1),C        Copy the array's name.
        LD A,$01                Assume an array of numbers.
        BIT 6,C                 Jump if true.
        JR Z,SA-V-TYPE
        INC A                   Nope, array of characters.
```

SA-V-TYPE

```
M02B3   LD (IX+DATTYPE),A       Save the type in the first location of header area.
```

SA-DATA-1

```
M02B6   EX DE,HL                Save the pointer to DE.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL, NEXT-CHAR        Get the next character.
        PUSH HL
        LD L, $00
        LD H, HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        CP ')'                  Is it a close parenthesis?
        JR NZ, SA-V-OLD         Report C if not.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-CHAR         Get the next character.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
```

```
         PUSH HL
         EXX
         CALL CALL-BANK
         POP IX
         BIT INTPT,(IY+OFLAGS)     Return if syntax checking.
         RET Z
         EX DE,HL                  Save pointer to DE.
         JP SA-ALL
```

SA-SCR$

```
M02F2    CP $AA                    Is the current code the SCREEN$ token?
         JR NZ,SA-CODE             Jump forward if not.
         LD A,(TADDR)              Is user trying to MERGE with SCREEN$?
         CP $03                    Can't MERGE SCREEN$ data.
         JP Z, BADBAS              Report a syntax error if ttrue.
         PUSH IX                   See annotated example under SAVE-ETC.
         EXX
         LD HL,NEXT-CHAR           Get the next character.
         PUSH HL
         LD L,$00
         LD H,HOMEROM
         PUSH HL
         LD HL,$0000
         PUSH HL
         PUSH HL
         EXX
         CALL CALL-BANK
         POP IX
         BIT INTPT,(IY+OFLAGS)     Return if syntax checking.
         RET Z
         LD (IX+BLKLEN),00         Load screen length into buffer
         LD (IX+BLKLEN+1),$1B
         LD HL,$4000               Screen address.
         LD (IX+ARG1),L
         LD (IX+ARG1+1),H
         JP SA-TYPE-3
```

SA-CODE

```
M032E    CP $AF                    Check if the current code is CODE token.
         JP NZ, SA-LINE            Nope, jump ahead.
         LD A,(TADDR)              Make sure user is not trying to
         CP $03                    MERGE with CODE.
         JP Z, BADBAS              Report syntax error if true.
         PUSH IX                   See annotated example under SAVE-ETC.
         EXX
         LD HL,NEXT-CHAR           Get next character.
         PUSH HL
         LD L,$00
         LD H,HOMEROM
```

```
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        EXX                     Back-to-back calls to HOME ROM.
        LD HL, PR-ST-END        Check for end of line (newline)
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        JR NZ, SA-CODE-1        Jump if not the end of a statement.
        LD A,(TADDR)            Make sure user is not trying to SAVE name
        AND A                   CODE by itself (missing location).
        JP Z, BADBAS            Report syntax error if user made this mistake.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL, USE-ZERO         Put a zero on the calculator stack for the
        PUSH HL                 start location.
        LD L,$00
        LD H, HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        JR SA-CODE2             Continue forward.
```

SA-CODE-1
Look for a starting address.

```
M0387   PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,EXPT-1NUM         Get the first number.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
```

```
              PUSH HL
              EXX
              CALL CALL-BANK
              EXX                   Back-to-back calls to HOME ROM.
              LD HL, GETCURCH       Get the next character after the number.
              PUSH HL
              LD L,$00
              LD H, HOMEROM
              PUSH HL
              LD HL,$0000
              PUSH HL
              PUSH HL
              EXX
              CALL CALL-BANK
              POP IX
              CP ','                Is the current character a comma?
              JR Z, SA-CODE-3       Yes, continue forward.
              LD A,(TADDR)          Check if user is trying to SAVE
              AND A                 with only address; no length.
              JP Z, BADBAS          Report syntax error if there's no length.
```

SA-CODE-2

```
M03BC         PUSH IX               See annotated example under SAVE-ETC.
              EXX
              LD HL, USE-ZERO       Put a zero on the calculator stack for
              PUSH HL               length.
              LD L,$00
              LD H, HOMEROM
              PUSH HL
              LD HL,$0000
              PUSH HL
              PUSH HL
              EXX
              CALL CALL-BANK
              POP IX
              JR SA-CODE-4          Jump forward.
```

SA-CODE-3

```
M03D5         PUSH IX               See annotated example under SAVE-ETC.
              EXX
              LD HL,NEXT-CHAR       Move CH_ADD forward.
              PUSH HL
              LD L,$00
              LD H,HOMEROM
              PUSH HL
              LD HL,$0000
              PUSH HL
              PUSH HL
              EXX
```

```
            CALL CALL-BANK          CALL BANK
            EXX                     Back-to-back calls to HOME ROM.
            LD HL,EXPT-1NUM         And get the value for length.
            PUSH HL
            LD L,$00
            LD H,HOMEROM
            PUSH HL
            LD HL,$0000
            PUSH HL
            PUSH HL
            EXX
            CALL CALL-BANK
            POP IX
```

SA-CODE-4

```
M03FF       BIT INTPT,(IY+OFLAGS)   Return if syntax checking.
            RET Z
            PUSH IX                 See annotated example under SAVE-ETC.
            EXX
            LD HL,FIND-INT2         Compress length into the BC register.
            PUSH HL
            LD L,$00
            LD H,HOMEROM
            PUSH HL
            LD HL,$0000
            PUSH HL
            PUSH HL
            EXX
            CALL CALL-BANK
            POP IX
            LD (IX+BLKLEN),C        Put the length of the code segment
            LD (IX+BLKLEN+1),B      into the buffer.
            PUSH IX                 See annotated example under SAVE-ETC.
            EXX
            LD HL,FIND-INT2         Compress starting address into BC.
            PUSH HL
            LD L,$00
            LD H,HOMEROM
            PUSH HL
            LD HL,$0000
            PUSH HL
            PUSH HL
            EXX
            CALL CALL-BANK
            POP IX
            LD (IX+ARG1),C          Put the starting address of the code
            LD (IX+ARG1+1),B        segment into the buffer.
            LD H,B                  Put the code address into HL.
            LD L,C
```

SA-TYPE-3
SCREEN$ and CODE are both type 3.

```
M0440   LD (IX+DATTYPE),03      Put the file type (BINARY) into the buffer.
        JP SA-ALL               Jump forward.
```

SA-LINE
Get the line number that follows LINE.

```
M0447   CP $CA                  Is the current code the LINE token?
        JR Z,SA-LINE-1          Yes, jump forward.
        BIT INTPT,(IY+OFLAGS)   Return if syntax checking.
        RET Z
        LD (IX+ARG1+1),$80      No further parameters, put in end code.
        JR SA-TYPE-0
```

SA-LINE-1

```
M0456   LD A,(TADDR)            Make sure the user has the syntax
        AND A                   correct: SAVE name LINE number.
        JP NZ,BADBAS            No? Issue a syntax error.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-CHAR         Advance CH_ADD.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        EXX
        LD HL,EXPT-1NUM         Get the number.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        BIT INTPT,(IY+OFLAGS)   Return if syntax checking.
        RET Z
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,FIND-INT2         Compress the line number to BC.
        PUSH HL
```

```
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        LD (IX+ARG1),C          Put the line number into the buffer.
        LD (IX+ARG1+1),B
```

SA-TYPE-0
Find and store parameters that describe the program and its variables in the header area of the work space.

```
M04A9   LD (IX+$00),00          Put file type into the buffer.
        LD HL,(ELINE)           Pointer to the end of the variables area.
        LD DE,(PROG)            Compute the length of the program
        SCF                     and variables to the $80 byte
        SBC HL,DE               just after the variables.
        LD (IX+BLKLEN),L        Store block length in the
        LD (IX+BLKLEN+1),H      buffer.
        LD HL,(VARS)            Compute the variables area length
        SBC HL,DE
        LD (IX+ARG2),L          Store length of the variables area
        LD (IX+ARG2+1),H        to the buffer.
        EX DE,HL                Transfer point to HL.
```

In all cases, the header information is prepared:
- The location 'IX+00' holds the type number.
- Locations 'IX+01 to IX+0A' holds the name (+FF in 'IX+01' if null).
- Locations 'IX+0B & IX+0C' hold the number of bytes that are to be found in the 'data block'.
- Locations 'IX+0D to IX+10' hold a variety of parameters whose exact interpretation depends on the 'type'.

The routine continues, separating SAVE from LOAD, VERIFY and MERGE.

SA-ALL

```
M04C9   LD A,(TADDR)            Jump forward
        AND A                   if this is a SAVE.
        JP Z,SA-CONTRL
```

For LOAD, VERIFY or MERGE commands, the first seventeen bytes of the header area hold the prepared information, as detailed above; and it is now time to fetch a 'header' from the tape.

```
        PUSH HL                 Save the destination pointer.
        LD BC,$0011             Point to the buffer used to store
        ADD IX,BC               the header coming in from the tape.
```

LD-LOOK-H

```
M04D6   PUSH IX              Make a copy of the base address.
        LD DE,$0011          Load 17 bytes.
        XOR A                Signal 'header'.
        SCF                  Signal 'LOAD'.
        CALL LD-BYTES        Look for a header.
        POP IX               Restore the base address.
        JR NC,LD-LOOK-H      Loop until successful.
        LD A,$FE             Ensure channel 'S' is open.
        PUSH IX              See annotated example under SAVE-ETC.
        EXX
        LD HL,CHAN-OPEN      Open the channel.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        LD (IY+$52),03       Set the scroll counter to 3.
        LD C,$80             Signal 'names do not match'.
        LD A,(IX+DATTYPE)    Compare the new type
        CP (IX+(-$11))       against the old type.
        JR NZ,LD-TYPE        Jump if they do not match.
        LD C,$F6             If they do, signal that ten characters need
                             to match.
```

LD-TYPE

```
M050D   CP $04               If the header type =>4 it's not a
        JR NC,LD-LOOK-H      TS file type, so loop back.
        LD DE,$3CA8          Base address of message block.
        PUSH BC              Briefly save C.
        PUSH IX              See annotated example under SAVE-ETC.
        EXX
        LD HL,PO-MSG         Print the appropriate message. Message
        PUSH HL              number is in A.
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        POP BC               Restore BC.
```

```
                PUSH IX         Make DE
                POP DE          ppoint to new type and
                LD HL,-16       HL point to old name.
                ADD HL,DE
                LD B,$0A        Name length counter (10 chars).
                LD A,(HL)       Get the first letter of the file name.
                INC A           Jump if a name was given.
                JR NZ,LD-NAME
                LD A,C          Load a with the length of the name.
                ADD A,B         Add to the max length of a name.
                LD C,A          Save the length of the name.

LD-NAME
M053D           INC DE          Get a character from the buffer.
                LD A,(DE)
                CP (HL)         Compare it to the received name.
                INC HL
                JR NZ,LD-CH-PR  Jump if it does not match.
                INC C           If it does match, increment the match counter.

LD-CH-PR
M0544           PUSH IX         See annotated example under SAVE-ETC.
                EXX
                LD HL,WRCH      Print the new character.
                PUSH HL
                LD L,$00
                LD H,HOMEROM
                PUSH HL
                LD HL,$0000
                PUSH HL
                PUSH HL
                EXX
                CALL CALL-BANK
                POP IX
                DJNZ LD-NAME    Loop back for 10 characters.
                BIT 7,C         If the names matched, C will have rolled
                JP NZ,LD-LOOK-H over.
                LD A,$0D        Print a newline.
                PUSH IX         See annotated example under SAVE-ETC.
                EXX
                LD HL,WRCH      Print the newline.
                PUSH HL
                LD L,$00
                LD H,HOMEROM
                PUSH HL
                LD HL,$0000
                PUSH HL
                PUSH HL
                EXX
```

```
        CALL CALL-BANK
        POP IX
        POP HL                  Get the pointer.
        LD A,(IX+DATTYPE)       SCREEN$ and CODE are handled by
        CP $03                  VERIFY.
        JR Z,VR-CONTRL
        LD A,(TADDR)            Jump forward if the command is
        DEC A                   LOAD.
        JP Z, LD-CONTRL
        CP $02                  Jump if using MERGE, otherwise
        JP Z, ME-CONTRL         continue in VERIFY.
```

VERIFY Control Routine

Verification LOADs a block of data, a byte at a time, and checks but does not store the bytes. This routine also LOADs blocks of data that have been described with SCREEN$ and CODE.

VR-CONTRL

```
M058F   PUSH HL                 Save the pointer.
        LD L,(IX+(-$06))        Get the number of bytes as described
        LD H,(IX+(-$05))        in the old header.
        LD E,(IX+BLKLEN)        Get the number number from the new
        LD D,(IX+BLKLEN+1)      header.
        LD A,H                  Jump forward if length is not specified.
        OR L
        JR Z,VR-CONT-1
        SBC HL,DE               Report R if attempting to load a block
        JR C,REPORT-R           larger than requested.
        JR Z, VR-CONT-1         Blocks are of equal length..
        LD A,(IX+DATTYPE)       Report R if trying to verify blocks of
        CP $03                  unequal size (old length > new length).
        JR NZ,REPORT-R
```

VR-CONT-1

```
M05AD   POP HL                  Get the pointer (start).
        LD A,H                  This pointer will be used unless it is zero.
        OR L
        JR NZ,VR-CONT-2
        LD L,(IX+ARG1)          Use new header instead.
        LD H,(IX+ARG1+1)
```

VR-CONT-2

```
M05B8   PUSH HL                 Move the pointer to IX.
        POP IX
        LD A,(TADDR)            Jump forward unless using
        CP $02                  VERIFY with carry flag set
        SCF                     to signal LOAD.
        JR NZ,VR-CONT-3
        AND A                   Signal VERIFY.
```

VR-CONT-3

```
M05C4   LD A,$FF        Signal 'accept data block only' before loading
                        block.
```

LOAD A DATA BLOCK Subroutine

This subroutine is common to all the LOAD routines. In the case of LOAD and VERIFY, it acts as a full return from the cassette handling routines but in the case of MERGE the data block has yet to be 'MERGEd'.

LD-BLOCK (R_TAPE1)

```
M05C6   CALL LD-BYTES   LOAD/VERIFY a data block.
        RET C           Return unless there's an error.
```

REPORT-R

```
M05CA   RST $08         Error: Tape loading error.
        DEFB $1A
```

LOAD Control Routine

This routine controls the LOADing of a BASIC program and its variables or an array.

LD-CONTRL (LOAD)

```
M05CC   LD E,(IX+BLKLEN)        Get the number of bytes
        LD D,(IX+BLKLEN+1)      as given in the new header.
        PUSH HL                 Save the destination pointer.
        LD A,H                  Jump forward unless tying to
        OR L                    LOAD a previously undeclared array.
        JR NZ,LD-CONT-1
        INC DE                  Add three bytes to the length: for the
        INC DE                  name, low length and
        INC DE                  high length.
        EX DE,HL
        JR LD-CONT-2
```

LD-CONT-1

Check if there is enough room for the new data block.

```
M05DD   LD L,(IX+(-$06))        Get the size of the existing
        LD H,(IX+(-$05))        program+variables or array.
        EX DE,HL
        SCF                     Jump if the existing data block
        SBC HL,DE               is at least long enough to hold
        JR C, LD-DATA           the new data.
```

LD-CONT-2

Test for room for data block.

```
M05E9   LD DE,$0005             Add 5 bytes to the block length.
        ADD HL,DE               BC = number of additional bytes needed
        LD B,H                  for this data block.
        LD C,L
```

```
        PUSH IX             See annotated example under SAVE-ETC.
        EXX
        LD HL,TEST-ROOM     Check to see if BC spaces are available.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
```

LD-DATA
Handle LOADing arrays.

```
M0606   POP HL              Restore the existing data block length.
        LD A,(IX+DATTYPE)   Jump forward
        AND A               if this is a program.
        JR Z, LD-PROG
        LD A,H              Jump if block length is zero
        OR L                to create the variable.
        JR Z, LD-DATA-1
        DEC HL              Get the size of exiting array
        LD B,(HL)           from the length bytes in variables area.
        DEC HL              Put the array size into BC.
        LD C,(HL)
        DEC HL              Point to its old name.
        INC BC              Add the length of the array name
        INC BC              and array length bytes.
        INC BC
        LD (X_PTR),IX       Save IX temporarily.
        PUSH IX             See annotated example under SAVE-ETC.
        EXX
        LD HL,RECLAIM-2     Delete the old array.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        LD IX,(X_PTR)       Restore IX.
```

LD-DATA-1

```
M0638   LD HL,(ELINE)          Find pointer to end marker of
        DEC HL                 variables area ($80).
        LD C,(IX+BLKLEN)       Get the length of new array.
        LD B,(IX+BLKLEN+1)
        PUSH BC                Save the length.
        INC BC                 Add three more bytes
        INC BC                 for the array name and
        INC BC                 length.
        LD A,(IX+(-$03))       Get the array ID for the incoming data.
        PUSH AF                Save it to make room at (HL) for the array,
        PUSH IX                See annotated example under SAVE-ETC.
        EXX
        LD HL,MAKE-ROOM        Create the space.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        INC HL
        POP AF                 Store the array name.
        LD (HL),A              Enter name.
        POP DE                 Get length and save its
        INC HL                 two bytes.
        LD (HL),E
        INC HL
        LD (HL),D
        INC HL                 Point to first location to fill with data from
        PUSH HL                tape. Move the location to IX.
        POP IX
        SCF                    "load..."
        LD A,$FF               "...a data block"
        JP LD-BLOCK            Read the data block and exit via SLVM.
```

LD-PROG

Load a BASIC program and its variables.

```
M0673   EX DE,HL               Save destination pointer.
        LD HL,(ELINE)          Point to the $80 byte at the end of the
        DEC HL                 variables area.
        LD (X_PTR),IX          Save IX temporarily.
        LD C,(IX+BLKLEN)       Get the data block length and briefly save
        LD B,(IX+BLKLEN+1)     it.
        PUSH BC
```

```
         PUSH IX          See annotated example under SAVE-ETC.
         EXX
         LD HL,RECLAIM-1  Reclaim area occupied by present program
         PUSH HL          and its variables.
         LD L,$00
         LD H, HOMEROM
         PUSH HL
         LD HL, $0000
         PUSH HL
         PUSH HL
         EXX
         CALL CALL-BANK
         POP IX
         POP BC           Pop the block length.
         PUSH HL          Push the data buffer address.
         PUSH BC          Push the block length.
         PUSH IX          See annotated example under SAVE-ETC.
         EXX
         LD HL,MAKE-ROOM  Make room for the new program.
         PUSH HL
         LD L,$00
         LD H, HOMEROM
         PUSH HL
         LD HL,$0000
         PUSH HL
         PUSH HL
         EXX
         CALL CALL-BANK
         POP IX
         LD IX,(X_PTR)    Get the buffer header pointer back.
         INC HL           Get the original address back.
         LD C,(IX+ARG2)   Get the variables address from the file.
         LD B,(IX+ARG2+1)
         ADD HL,BC        Point to the new variables area
         LD (VARS),HL     and save it in the system vars area.
         LD H,(IX+ARG1+1) Pet the putative line number.
         LD A,H
         AND $C0          Jump if no line number
         JR NZ, LD-PROG-1
         LD L,(IX+ARG1)   Get the lower byte of the line number
         LD (NEWPPC),HL   store the line number into the next line
         LD (IY+ONSPPC),00 number and set sub-line to 0.

LD-PROG-1
M06D5    POP DE           Restore the block length.
         POP IX           Restore the start.
         SCF              "load..."
         LD A,$FF         "...a data block"
         LD HL,(PROG)     Get the address of the program text
```

```
              DEC HL                  point to the $80 byte.
              LD (DATADD),HL          data terminator character (no data).
              JP LD-BLOCK             Read the tape.
```

MERGE Control Routine

There are three main parts to this routine.
1. LOAD the data block into the work space.
2. MERGE the lines of the new program into the old program.
3. MERGE the new variables into the old variables. Start therefore with the LOADing of the data block.

ME-CONTRL (MERGE)

```
M06E5   LD C,(IX+BLKLEN)        Get the data block length.
        LD B,(IX+BLKLEN+1)
        PUSH BC                 Save the block length
        INC BC                  with room for the $80 byte
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,BC-SPACES         Make length+1 spaces available in work
        PUSH HL                 space.
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        LD (HL),$80             $80 byte for the end of the edit buffer.
        EX DE,HL                Move start pointer to HL.
        POP DE                  Pop the original length.
        PUSH HL                 Save the pointer to the end of the
        PUSH HL                 reserved area ...and again
        POP IX                  Put pointer to the end of the reserved area
        SCF                     in IX."load..."
        LD A,$FF                "...a data block"
        CALL LD-BLOCK           Read the program lines in from tape.
        POP HL                  Point to the newly loaded data.
        LD DE,(PROG)            Set DE to start of old program.
```

ME-NEW-LP

```
M0717   LD A,(HL)               Get a line number and test it.
        AND $C0
        JR NZ,ME-VAR-LP         Jump when finished with all lines.
```

ME-OLD-LP

```
M071C   LD A,(DE)          Get the high line number byte
        INC DE             and
        CP (HL)            compare to the new BASIC.
        INC HL             Bump the new pointer.
        JR NZ,ME-OLD-L1    Old and new line numbers don't match, so
        LD A,(DE)          jump. Compare the low line number bytes.
        CP (HL)
```

ME-OLD-L1

```
M0724   DEC DE             Bump both BASIC pointers back
        DEC HL
        JR NC,ME-NEW-L2    Jump if the new line number is less than or equal
                           to the old line number to insert the new line into
                           the program.
        PUSH HL            Save the new pointer.
        EX DE,HL           HL=old pointer, DE=the new pointer
        PUSH IX            See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-ONE     Find the next variable or BASIC line.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        POP HL             Restore the new pointer.
        JR ME-OLD-LP       Back around to test the new line.
```

ME-NEW-L2

```
M0744   CALL ME-ENTER      Enter the new line and go
        JR ME-NEW-LP       back around the loop.
```

ME-VAR-LP

```
M0749   LD A,(HL)          Get each variable name in turn and test.
        LD C,A
        CP $80             Return when all variables have been
        RET Z              evaluated.
        PUSH HL            Save the current new pointer.
        LD HL,(VARS)       Get the pointer to the variables area
                           (old program).
```

ME-OLD-VP

```
M0752   LD A,(HL)          Get each variable name and test.
        CP $80
        JR Z,ME-VAR-L2     Jump if there are no more variables.
        CP C               Compare the 0 byte of each name.
        JR Z, ME-OLD-V2    Jump forward on match to compare the rest of
                           the names.
```

ME-OLD-V1

```
M075A   PUSH BC            Save new variable's name.
        PUSH IX            See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-ONE     Find the old variable.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        POP BC             Restore new variable's name.
        EX DE,HL           Restore pointer to DE and
        JR ME-OLD-VP       go around the loop again.
```

ME-OLD-V2

```
M0776   AND $E0            Compare bits 5, 6 and 7 only.
        CP $A0             Accept all variable types except
        JR NZ,ME-VAR-L1    long named variables.
        POP DE             Make DE pointer to the first character
        PUSH DE            of the new name.
        PUSH HL            Save the pointer to the old name.
```

ME-OLD-V3

```
M077F   INC HL             Update both old and
        INC DE             new pointers.
        LD A,(DE)          Compare the two letters.
        CP (HL)
        JR NZ,ME-OLD-V4    Jump if the variable names do not match.
        RLA                Loop back around if not the last character.
        JR NC, ME-OLD-V3
        POP HL             Restore the pointer to start of old name
        JR ME-VAR-L1       and jump forward bc successful match.
```

ME-OLD-V4

```
M078B   POP HL             Get pointer and jump back:
        JR ME-OLD-V1       unsuccessful match.
```

ME-VAR-L1

```
M078E   LD A,$FF                Signal 'replace' variable.
```

ME-VAR-L2

```
M0790   POP DE                  Get pointer to new name.
        EX DE,HL                Swap the pointers.
        INC A                   Set zero flag if replacement; reset for
                                addition.
        SCF                     Signal 'handling variables.'
        CALL ME-ENTER           Make the entry.
        JR ME-VAR-LP            Loop back to consider next new variable.
```

Merge A Line Or A Variable Subroutine

This subroutine is entered with the following parameters:
- Carry flag reset - MERGE a BASIC line; set - MERGE a variable.
- Zero reset - It will be an 'addition'; set - It is a 'replacement'.
- HL - Points to the start of the new entry.
- DE - Points to where it is to MERGE.

ME-ENTER

```
M0799   JR NZ,ME-ENT-1          Jump if handling an 'addition'.
        EX AF,AF'               Save the flags.
        LD (X_PTR),HL           Save new pointer wile old line
        EX DE,HL                or variable is reclaimed.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,NEXT-ONE          Find the next line or variable and delete it.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        EXX
        LD HL,RECLAIM-2
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        EX DE,HL
```

```
              LD HL,(X_PTR)
              EX AF,AF'          Restore the flags.
```

ME-ENT-1

```
M07CF         EX AF,AF'          Save the flags.
              PUSH DE            Save the pointer to destination.
              PUSH IX            See annotated example under SAVE-ETC.
              EXX
              LD HL,NEXT-ONE     Find the length of new next variable
              PUSH HL            or program line.
              LD L,$00
              LD H,HOMEROM
              PUSH HL
              LD HL,$0000
              PUSH HL
              PUSH HL
              EXX
              CALL CALL-BANK
              POP IX
              LD (X_PTR),HL      Save pointer to new variable/line.
              LD HL,(PROG)       Get the pointer to the BASIC program.
              EX (SP),HL         Save PROG on stack and get new pointer.
              PUSH BC            Save the length.
              EX AF,AF'          Retrieve the flags.
              JR C,ME-ENT-2      Jump forward if adding new variable.
              DEC HL             A new line is added before destination location.
              PUSH IX            See annotated example under SAVE-ETC.
              EXX
              LD HL,MAKE-ROOM    Make room for the new line.
              PUSH HL
              LD L,$00
              LD H,HOMEROM
              PUSH HL
              LD HL,$0000
              PUSH HL
              PUSH HL
              EXX
              CALL CALL-BANK
              POP IX
              INC HL
              JR ME-ENT-3        Jump forward.
```

ME-ENT-2

```
M080E         PUSH IX            See annotated example under SAVE-ETC.
              EXX
              LD HL,MAKE-ROOM    Make room for new variable.
              PUSH HL
              LD L,$00
              LD H,HOMEROM
```

```
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
```

ME-ENT-3

```
M0825   INC HL          Point to the 1st new location.
        POP BC          Retrieve the length.
        POP DE          Retrieve PROG and
        LD (PROG),DE    store it in its correct place.
        LD DE,(X_PTR)   Also fetch the new pointer.
        PUSH BC         Save the length and
        PUSH DE         the new pointer.
        EX DE,HL        Switch pointers and
        LDIR            copy the new variable/line into
                        the room made for it.
        POP HL          Get the new pointer.
        POP BC          Get the length.
        PUSH DE         Save the old pointer.
        PUSH IX         See annotated example under SAVE-ETC.
        EXX
        LD HL,RECLAIM-2 Remove the variable/line from the work
        PUSH HL         space.
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        POP DE          Return with the old pointer
        RET             in DE.
```

SAVE Control Routine

SAVing a program or a block of data is very straightforward.

SA-CONTRL (SAVE)

```
M0851   PUSH HL         Save the data buffer address.
        LD A,$FD        Ensure that channel '?K' is open.
        PUSH IX         See annotated example under SAVE-ETC.
        EXX
        LD HL,CHAN-OPEN Open the channel.
        PUSH HL
        LD L,$00
```

```
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        XOR A                   Signal 'first message'.
        LD DE,$3C89             Print the message 'Start Tape'.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,PO-MSG            Put message 0 to current channel.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        SET CLHS,(IY+OTVFLAG)   Signal 'screen must be cleared.'
        CALL AKEY               Wait user to press a key.
        PUSH IX                 Save the header buffer address.
        LD DE,$0011             Set tape header length.
        XOR A                   Signal 'header'.
        CALL SA-BYTES           Send the header.
        POP IX                  Restore the header buffer address.
        LD B,$32                Set delay for almost a second.

SA-1-SEC
M089A   HALT                    Wait for approximately 1000 ms.
        DJNZ SA-1-1SEC
        LD E,(IX+BLKLEN)        Get the length of the data block from the
        LD D,(IX+BLKLEN+1)      header.
        LD A,$FF                Signal 'data block.'
        POP IX                  Get start of block pointer and
        JP SA-BYTES             save the block.
```

AKEY Subroutine

Used just to wait for a key press in preparing to SAVE.

```
AKEY
M08AA   PUSH AF                 Save registers.
        PUSH BC
        PUSH DE
        LD BC,$9C40             Value to wait for keypress.
```

AKEY-PAUSE

```
M08B0   DEC BC                  Wait for approximately 238 ms.
        LD A,C
        OR B
        JR NZ, AKEY-PAUSE
```

AKEY-WAIT

```
M08B5   XOR A                   Zero A.
        IN A,($FE)              Poll the keyboard.
        AND $1F                 Use only the keyboard bits.
        CP $1F
        JR Z, AKEY-WAIT         Jump if no key is pressed.
        PUSH IX                 See annotated example under SAVE-ETC.
        EXX
        LD HL,CLS-LOWER         Clear the lower half of the screen.
        PUSH HL
        LD L,$00
        LD H,HOMEROM
        PUSH HL
        LD HL,$0000
        PUSH HL
        PUSH HL
        EXX
        CALL CALL-BANK
        POP IX
        POP DE                  Restore registers.
        POP BC
        POP AF
        RET                     Back to the "save" routine.
```

REPORT-8 (BADBAS)

```
M08D9   EXX
        LD HL,$1BED             Address for BAD BASIC COMMAND
        PUSH HL                 error routine.
        LD L,$00
        LD H,$FF
        PUSH HL
        EXX
        CALL GOTO-BANK
```

Extension Initialization Routine

EXTINIT Routine

Initialize the SYSCON table at $5EEA. Check if LROS or AROS is present, complete system variable initialization then jump to M18C6 in HOME ROM to continue AROS initialization.

EXINIT

```
M08E7   LD HL,$5EEA             Initialize the SYSCON
        LD (SYSCON),HL          system variable.
        CALL BLDSCT             Build the system config table
        LD HL,(SYSCON)          Point to SYSCON table.
        LD DE,$0008             LROS ID: check system config
        ADD HL,DE               table for an LROS.
        LD A,(HL)               Put the potential LROS value in A.
        CP $01                  Is an LROS present?
        JR NZ, CHK-SYSCN-AROS   No, jump ahead and check if AROS.
```

START-LROS

Finish setting up system variables then call the LROS.

```
M08FC   PUSH HL                 Save LROS pointer.
        CALL NORMSVAR           Continue initializing system variables.
        POP HL                  Get the LROS pointer back.
        INC HL                  Move forward one space in memory.
        LD E,(HL)               Put LROS entry point low byte in E.
        INC HL                  Next byte.
        LD D,(HL)               Put LROS entry point high byte in D.
        PUSH DE                 Save to the stack.
        LD B,$00                Specify dock bank.
        INC HL                  Next byte.
        LD C,(HL)               Put HSR byte in C.
        PUSH BC                 Save to the stack.
        EI                      Enable interrupts.
        CALL GOTO_BANK          Call GOTO_BANK to go to the LROS.
```

CHK-SYSCN-AROS

Check the system configuration table for an AROS.

```
M090F   LD HL,(SYSCON)          Put location of SYSCON table in HL.
        INC HL                  Next byte.
        LD A,(HL)               Get possible AROS ID.
        CP $02                  Is this an AROS?
        JR Z, WHICH-AROS        Jump if AROS cartridge.
        CALL NORMSVAR           Continue initializing system variables.
        JP NOT-AUTOSTART        Continue with this routine.
```

WHICH-AROS
AROS is present: determine whether it is machine language or BASIC.

```
M091E   DEC HL              Back up to get the
        LD A,(HL)           code type designator.
        CP $01              Is it a BASIC AROS?
        JR Z, AROS-INIT     Yes, jump ahead.
        CP $02              Is it machine code?
        JR NZ, REPORT-R     No, then there's an error.
        LD DE,$0006         Advance HL by 6 bytes
        ADD HL,DE           to get to the number
        LD C,(HL)           of bytes to reserve
        INC HL              for machine code variables and
        LD B,(HL)           put that value in BC.
        LD HL,$6840         Set HL to start of LROS vars,
        ADD HL,BC           add the amount of space needed,
        EX DE,HL            move that value to DE.
        LD HL,$6840         Reset HL.
        LDIR                Create space for the AROS variables.
        CALL INITSYSVAR     Finalize system variable initialization.
        LD HL,(SYSCON)      Reset HL to the SYSCON table.
        LD DE,$0005         Prepare to get the AROS type
        ADD HL,DE           specification.
        LD A,(HL)           Put the value inA.
        CP $00              Is it autostart? (0 is no)
        JR Z,NOT-AUTOSTART  Not autostart, continue with HOME ROM.
        DEC HL              Back up to put the chunk specification
        LD C,(HL)           in C.
        DEC HL              And again to put the starting address
        LD D,(HL)           in DE.
        DEC HL
        LD E,(HL)
        PUSH DE             Save the start address.
        LD B,$00            Save the chunk specification.
        PUSH BC
        EI                  Enable interrupts.
        CALL GOTO_BANK      Start the AROS.
```

AROS-INIT (CART-INIT)

```
M0956   CALL NORMSVAR       Continue initializing system variables.
        LD A,$80            Set flag to show an AROS is present.
        LD (ARSFLG),A
        LD HL,AROS          Jump to the AROS handler.
        PUSH HL             Save HL.
        LD B,$FF            Set HOME bank for
        LD C,$00            all of memory.
        PUSH BC             Push to the stack.
        CALL GOTO_BANK
```

REPORT-R (LROS_ERR)

```
M096A   RST $08                 Error: Missing LROS.
        DEFB $1B
```

NORMSVAR (NEW-INIT)

```
M096C   LD HL,$6840             Point past the RAM resident services and
        LD DE,$0015             reserve space for the CHANS table.
        ADD HL,DE
```

INITSYSVAR (NEW-INIT-2)

```
M0973   LD (DATADD),HL          Initialize the DATADD pointer.
        INC HL                  Point to the start of BASIC.
        LD (PROG),HL            Put that value in PROG.
        LD (VARS),HL            And in the variables pointer
        LD (HL),$80             Place end-of-table marker at the end of
                                the variables area.
        INC HL
        LD (ELINE),HL           Initialize the input line buffer.
        LD (HL),$0D             Put a carriage return in the buffer.
        INC HL
        LD (HL),$80             And an end marker.
        INC HL
        LD (WORKSP),HL          Initialize system variables.
        LD (STKBOT),HL
        LD (STKEND),HL
        XOR A
        LD (ARSFLG),A           Show no AROS present.
        LD (VIDMOD),A           Normal video mode.
        RET
```

NOT-AUTOSTART

AROS was not autostart, continue initialization.

```
M099A   LD D,$FF                Value for HOME bank.
        LD E,$80                HS select, top chunk in EXROM/DOCK.
        LD HL, MAIN-1           Jump address in HOME ROM.
        PUSH HL                 Stack the jump address.
        PUSH DE                 Stack bank/HS.
        LD HL,(SYSCON)          Point to the first expansion bank
        LD DE,$000C             entry.
        ADD HL,DE
        LD B,$00                Garbage code; not used.
```

NA-LOOP-1

```
M09AC   LD A,(HL)               Get the entry for an expansion bank,
        CP $80                  jump if at the end of the SYSCON area.
        JR Z, CART-END
        CP $00                  No entry for this bank, so move on
        JR Z, JUMP-END          to check the next one.
        INC HL                  Increment to the bank number.
```

```
            LD B,(HL)
            LD DE,$0014         Increment to entry 15 and get
            ADD HL,DE           the initialization flag.
            LD A,(HL)
            RRCA
            JR C,NA-CK-INIT     Initialize this bank?
            INC HL              Point to the next expansion bank entry in
            INC HL              the SYSCON area.
            INC HL
            JR NA-LOOP-1        Back around to check for more.

NA-CK-INIT
M09C4       INC HL              Point to entry 16 and get the "boot up"
            LD A,(HL)           priority.
            POP DE              Get current lowest priority bank.
            CP E                Compare it to the highest priority found.
            JR C, SET-EXP-INIT  If higher (lower value), set this to
            PUSH DE             Save current lowest bank/HS back.
            INC HL              Point to the next expansion bank
            INC HL              entry in the SYSCON.
            JR NA-LOOP-1        Loop to keep testing.

SET-EXP-INIT
M09CF       POP DE              Trash the call address.
            LD DE,$0005         Point to entry 10, the boot up address.
            SBC HL,DE           Make present priority and bank numbers
            PUSH HL             the new boot up values. The address of
            LD C,A              entry 10 becomes the boot up address.
            PUSH BC
            INC DE
            INC DE              Point to next entry in SYSCON.
            ADD HL,DE
            JR NA-LOOP-1        Back around to keep testing.
```

The code is incorrect. Contents of entry 10 should be the boot up address but instead contains bank number and priority.[2]

```
JUMP-END
M09DD       LD DE,$0018         Point to the next expansion bank entry
            ADD HL,DE           in the SYSCON area.
            JR NA-LOOP-1        Back around to check for more.

CART-END
M09E3       POP BC              Get the highest priority bank.
            LD A,B
            CP $FF              It it the HOME bank?
            JR Z, CALL-EXT-INIT Yes, jump ahead.
```

---

[2] "Mystery of the Missing 253" series.

```
        LD C, $58           Allow chunks 3, 4, and 6 to be in
                            expansion banks.
        JR GOTO-BANK
```

CALL-EXT-INIT

```
M09ED   LD C,$00            Allocate all chunks to HOME.
```

GOTO-BANK

```
M09EF   PUSH BC             Save bank/HS.
        EI                  Enable interrupts.
        CALL GOTO_BANK      Goto bank.
```

Build System Configuration (SYSCON) Table Subroutine

The system configuration table is a list and description of all the extra "memory" plugged into the TS2068. A detailed description of the system configuration table is available in the *Third Party Software Guide*.

BLDSCT

```
M09F4   LD HL,(SYSCON)      Point to SYSCON table.
        XOR A               Zero A.
        LD (MAXBNK),A       Zero out MAXBANK.
        LD (BS_MAX_BANK),A  Zero out dispatcher's copy of MAXBANK.
        LD DE,$0008         Value of LROS buffer.
        ADD HL,DE           Add to point to get to start of LROS buffer.
        LD E,$FF            Destination bank = $FF (home).
        LD D,$00            Source bank = $00 (dock).
        PUSH DE
        LD DE,$0001         Set source address.
        PUSH DE
        PUSH HL             Set destination address.
        LD DE,$0004         Set length.
        PUSH DE
        LD DE,$0001         Set direction (LDIR).
        PUSH DE
        CALL XFER_BYTES     Transfer the bytes from LROS to the SYSCON
                            table.
        LD A,(HL)           Check to see if
        CP $01              an LROS signature is present.
        JR Z,CHECK-CONF     Yes, jump forward.
        LD (HL),$00         Zero LROS signature to prevent system from
                            incorrectly seeing an LROS as present.
```

Transfer bytes from AROS to the SYSCON.

```
        LD E,$FF            Destination bank = $FF (home)
        LD D,$00            Source bank = $00 (dock)
        PUSH DE
        LD DE,$8000         Set source address.
        PUSH DE
```

```
            LD HL,(SYSCON)          Get SYSCON table address.
            PUSH HL                 And use as destination address.
            LD DE,$0008             Set length.
            PUSH DE
            LD DE,$0001             Set direction.
            PUSH DE
            CALL XFER_BYTES         Transfer bytes from AROS to SYSCON.
            INC HL                  Get AROS signature byte.
            LD A,(HL)
            CP $02                  Is it a valid AROS signature byte?
            JR Z,CHECK-CONF         Yes, jump forward.
            LD (HL),$00             Zero out AROS ID byte, as we do not have
                                    a valid AROS cartridge.
```

CHECK-CONF

```
M0A3E       LD HL,(SYSCON)          Get SYSCON table address.
            LD DE,$000D             Point to first expansion bank in SYSCON.
            ADD HL,DE
            LD D,$C0                Bank status register address.
            LD E,$00                $00 resets daisychain and starts setup mode.
            CALL WRITE_BS_REG       Write to bank status register.
```

CALL-RES-REG

```
M0A4C       CALL RESET-BS-REG       Find out how many valid expansion banks are
                                    available. Install a bank number into the bank
                                    selected by the daisychain.
            JP NC, SET-END-MARKER   If the bank does not exist, jump ahead.
            LD B,A                  Put a number into SYSCON $01. Its lowest
            SET 7,B                 7 bits are the bank number and the MSB
            LD (HL),B               is 1 to signify that the bank's not renumbered.
            RES 7,B
            INC HL                  Point to SYSCON $02.
            LD C,$FE                Destination bank: EXROM.
            PUSH BC
            LD DE,$08E7             Set source address.
            PUSH DE
            LD DE,$0000             Destination address.
            PUSH DE
            LD DE,$0001             Set length (1 byte).
            PUSH DE
            PUSH DE                 Set direction.
            CALL XFER_BYTES         Transfer the byte from new bank to $0000 of
                                    EXROM bank.
```

This is a bug: should be the other way around.[3]

```
            LD E,$FF                Destination bank = $FF (home).
```

[3] "Mystery of the Missing 253" series.

```
LD D,A                  Source bank provided by caller.
PUSH DE                 Set bank.
LD DE,$0000             Set source address.
PUSH DE
PUSH HL                 Set destination address.
LD DE,$0016             Set length.
PUSH DE
LD DE,$0001             Set direction.
PUSH DE
CALL XFER_BYTES         Transfer the bytes from $0000 of expansion bank
                        to SYSCON $02 and following. If present bank is
                        a RAM bank, then this is garbage. If a ROM bank,
                        these are overhead bytes.
LD D,(HL)               Get SYSCON $02 and compare it to ($08E7 in
                        EXROM). These should match if it's a RAM bank
                        and possibly match by coincidence if a ROM
                        bank.
LD A,($08E7)            No match definitely means a ROM bank.
CP D                    However, because the memory transfer
JP NZ, BLD-TBL-END      above is buggy, this test is meaningless. This
                        jump never happens.
LD C,$FE                Destination bank = $FE (external).
PUSH BC                 Set banks.
LD DE,$0A4C             Set source address.
PUSH DE
LD DE,$0000             Set destination address.
PUSH DE
LD DE,$0001             Length/direction.
PUSH DE                 Set length.
PUSH DE                 Set direction.
CALL XFER_BYTES         Move the byte from $0A4C to $0000 in EXROM,
                        which is a bug like the code above.
LD E,$FF                Destination bank = home.
LD D,A                  Source bank provided by caller.
PUSH DE                 Set bank.
LD DE,$0000             Set source address.
PUSH DE
PUSH HL                 Set destination address.
LD DE,$0016             Set block length.
PUSH DE
LD DE,$0001             Set direction.
PUSH DE
CALL XFER_BYTES         Transfer bytes to the bank.
LD D,(HL)               Compare SYSCON $02 to
LD A,($08E7)            $08E7.
CP D                    Do they match (RAM bank)?
JP NZ, BLD-TBL-END      Jump ahead if not.
DEC HL                  Point to SYSCON $00.
DEC HL
```

```
              CALL SET-RST-56          Copy RST $38 to destination bank.
              LD DE,$0015              Point to the next block of
              ADD HL,DE                the SYSCON table.
              JR CALL-WR-BS-REG
```

BLD-TBL-END
Here for a ROM bank.

```
M0AC2         LD A,D                   Reset SYSCON $02 bits.
              AND $DF                  This byte is normally a channel specification.
              LD (HL),A                Convert it to upper case.
              DEC HL                   Point to SYSCON $01
              CALL INIT-EXPROM-BANK    Initialize bank.
```

CALL-WR-BS-REG

```
M0ACA         LD D,$C0                 Send $01 to register $C0; steps
              LD E,$01                 daisychain.
              CALL WRITE_BS_REG        Load and do
              JP CALL-RES-REG          the next bank.
```

SET END MARKER Subroutine

SET-END-MARKER

```
M0AD4         DEC HL                   Put $80 at end of table.
              LD (HL),$80
              CALL CLEAR-SYSCONF       Renumber banks according to internal
              RET                      priorities and return.
```

Interruptable Restart Routine

Mark SYSCON $02 with chunks containing RAM and copy RST $38 routine to the destination bank.

Because of bugs in this routine, it does not actually do anything.[4]

SET-RST-56

```
M0ADB         LD (HL),$02              Put $02 at SYSCON $00 (mark a RAM bank).
              PUSH BC                  Push to stack.
              LD DE,$0038              Set source.
              PUSH DE
              PUSH DE                  Set destination.
              LD DE,$0010              Set block length.
              PUSH DE
              LD DE,$0001              Set direction.
              PUSH DE
              CALL XFER_BYTES          Transfer routine.
              INC HL                   Point SYSCON to $02: chunks available
              INC HL                   for RAM (high true).
```

[4] "Mystery of the Missing 253" series.

```
              LD A,(HL)
              SET 0,A              Set bit 0 of SYSCON $02. If we're here,
              LD (HL),A            there must be RAMi n chunk 0.
              LD DE,$0000          Initialize chunk address pointer.
              LD A,$01             Initialize chunk counter.

NEXT-CHUNK
M0AF9         EX AF,AF'            Save chunk counter.
              PUSH HL
              EX DE,HL
              LD DE,$2000          Point DE to next chunk.
              ADD HL,DE
              EX DE,HL
              POP HL
              LD B,$FE             Source bank = EXROM.
              LD A,(MAXBNK)        Get MAXBANK.
              LD C,A               Destination bank = MAXBANK.
              PUSH BC              Set banks.
              LD BC,$08E7          Set source.
              PUSH BC
              PUSH DE              Set destination
              LD BC,$0001
              PUSH BC              Set byte count and direction.
              PUSH BC
              CALL XFER_BYTES      Transfer byte from $08E7 to first byte in new RAM
                                   chunk.
              LD A,(MAXBNK)        Get MAXBANK.
              LD B,A               Source = MAXBANK.
              LD C,$00             Destination = DOCK.
              PUSH BC              Set banks.
              PUSH DE              Set source.
              INC HL               Increment to SYSCON $03.
              PUSH HL              Set destination.
              LD BC,$0001
              PUSH BC              Set block length and direction.
              PUSH BC
              CALL XFER_BYTES      Transfer byte that was written to RAM and copy to
                                   DOCK bank. Should have been copied to
                                   SYSCON $03. This is a bug.
              LD B,(HL)            Put contents of SYSCON $03 in B.
              DEC HL               Back HL up to SYSCON $02.
              LD A,($08E7)
              CP B                 Compare $08E7 in EXROM.
              JR NZ,RESET-FLAGS    No match? Jump ahead.
```

Move byte from $0A4C in EXROM to first byte of current chunk of RAM bank.

```
              LD B,$FE             Source bank = EXROM.
              LD A,(MAXBNK)        Get MAXBANK.
```

```
        LD C,A              Destination bank = MAXBANK.
        PUSH BC             Set banks.
        LD BC,$0A4C         Set source.
        PUSH BC
        PUSH DE             Set destination.
        LD BC,$0001
        PUSH BC             Set direction and byte count.
        PUSH BC
        CALL XFER_BYTES     Transfer byte.
        LD A,(MAXBNK)       Get MAXBANK.
        LD B,A              Source bank = MAXBANK.
        LD C,$00            Destination bank = DOCK (bug).
        PUSH BC             Set banks.
        PUSH DE             Set source.
        INC HL              Increment to SYSCON $03.
        PUSH HL             Set destination.
        LD BC,$0001
        PUSH BC             Set direction and byte count.
        PUSH BC
        CALL XFER_BYTES     Transfer bytes.
        LD B,(HL)           Get the value at (HL).
        DEC HL              Back up to SYSCON $02
        LD A,($0A4C)        Put the value in $0A4C in A.
        CP B                Compare it to value from SYSCON $03.
        JR NZ,RESET-FLAGS   No match? Jump ahead.
```

SET-FLAGS
Found a RAM bank. Set the appropriate bit in SYSCON $02.

```
M0B5E   EX AF,AF'
        LD B,(HL)           Get the byte at (HL).
        CP $01
        JR NZ,SET-CP-2      If 1, set bit 1 in B,
        SET 1,B             then move on.
        JR SAVE-SET-FLAG
```

SET-CP-2

```
M0B68   CP $02
        JR NZ,SET-CP-3      If 2, set bit 2 in B,
        SET 2,B             then move on.
        JR SAVE-SET-FLAG
```

SET-CP-3

```
M0B70   CP $03
        JR NZ,SET-CP-4      If 3, set bit 3 in B,
        SET 3,B             then move on.
        JR SAVE-SET-FLAG
```

SET-CP-4
```
M0B78   CP $04
        JR NZ,SET-CP-5          If 4, set bit 4 in B,
        SET 4,B                 then move on.
        JR SAVE-SET-FLAG
```

SET-CP-5
```
M0B80   CP $05
        JR NZ,SET-CP-6          If 5, set bit 5 in B,
        SET 5,B                 then move on.
        JR SAVE-SET-FLAG
```

SET-CP-6
```
M0B88   CP $06
        JR NZ, SET-7            If 6, set bit 6 in B,
        SET 6,B                 then move on.
        JR M0B92
```

SET-7
```
M0B90   SET 7,B                 Else set bit 7.
```

SAVE-SET-FLAG
```
M0B92   LD (HL),B               Save byte back.
        JR RESET-END
```

RESET-FLAGS

No RAM in this chunk. Reset appropriate bit in SYSCON $02.

```
M0B95   EX AF,AF'
        LD B,(HL)
        CP $01
        JR NZ,RESET-CP-3        If 1, reset bit 1 in B,
        RES 1,B                 then move on.
        JR RESET-END
```

RESET-CP-2
```
M0B9F   CP $02
        JR NZ,RESET-CP-3        If 2, reset bit 2 in B,
        RES 2,B                 then move on.
        JR SAVE-RESET-FLAGS
```

RESET-CP-3
```
M0BA7   CP $03
        JR NZ,RESET-CP-4        If 3, reset bit 3 in B,
        RES 3,B                 then move on.
        JR SAVE-RESET-FLAGS
```

RESET-CP-4
```
M0BAF   CP $04
        JR NZ,RESET-CP-5        If 4, reset bit 4 in B,
```

```
            RES 4,B                 then move on.
            JR SAVE-RESET-FLAGS

RESET-CP-5
M0BB7       CP $05
            JR NZ,RESET-CP-6        If 5, reset bit 5 in B,
            RES 5,B                 then move on.
            JR SAVE-RESET-FLAGS

RESET-CP-6
M0BBF       CP $06
            JR NZ,RESET-7           If 6, reset bit 6 in B,
            RES 6,B                 then move on.
            JR SAVE-RESET-FLAGS

RESET-7
M0BC7       RES 7,B                 Else reset bit 7.

SAVE-RESET-FLAG
M0BC9       LD (HL),B               Save byte back.

RESET-END
M0BCA       INC A                   Update chunk counter.
            CP $08                  Is it 8 yet?
            JP NZ, NEXT-CHUNK       No, keep going.
            RET
```

Reset BS Register Subroutine

Find true maximum bank number. Installs a bank number into the bank selected by the daisychain. Carry flag tells whether the bank is really there.

```
RESET-BS-REG
M0BD1       LD A,(MAXBNK)           Increment MAXBNK in
            INC A                   both places it's stored.
            LD (MAXBNK),A           Save new MAXBANK
            LD (BS_MAX_BANK),A      Save local copy of MAXBANK.
            LD D, ABN               Send new maximum bank number
            LD E,A                  Assigned Bank Number register.
            CALL WRITE_BS_REG       In setup mode, this installs the bank number into
                                    the bank selected by the daisy chain system.
            LD D, BNA               Send the new max bank number to
            LD E,A                  Bank Number Access register.
            CALL WRITE_BS_REG
            LD D, HS                Send $00 to HS.
            LD E,$00                Disables all chunks and resets
            CALL WRITE_BS_REG       the register.
            PUSH AF                 Save AF briefly (save maximum bank number).
            LD A,($A000)            Save value at $A000.
            EX AF,AF'               Exchange the register sets.
            LD A,$04                Send $04 (bank number) to $A0 register.
```

```
        LD ($A000),A
        LD D,$A0                Read the $C0/$A0 register pair.
        LD E,$C0
        CALL READ_BS_REG
        BIT 2,E                 If bit 2=0, then the bank exists.
        JR NZ,NO-BANK           Jump ahead if the bank does not exist.
        EX AF,AF'               Restore the value at $A000.
        LD ($A000),A
        POP AF                  Restore AF
        SCF                     Set carry flag: the bank exists.
        RET

NO-BANK
M0C0A   EX AF,AF'               Restore the value at $A000.
        LD ($A000),A
        POP AF                  Get back the new "max bank value"
        DEC A                   and go back to the old value.
        LD (MAXBNK),A           Save it back to MAXBANK,
        LD (BS_MAX_BANK),A      in both locations.
        LD D,$C0                Send $04 to register $C0 to
        LD E,$04                terminate setup mode.
        CALL WRITE_BS_REG
        AND A                   Reset carry flag: bank does not exist.
        RET
```

Initialize Expansion ROM Bank Subroutine

Checks to see if a ROM bank needs to be initialized. If so, it executes the bank's initialization routine through CALL_BANK.

This routine wipes out the SYSCON pointer and cannot work properly.

```
INIT-EXPROM-BANK
M0C1F   DEC HL                  Point to SYSCON $01.
        LD (HL),$01             Mark it as a ROM bank.
        LD DE,$0015
        ADD HL,DE
        LD A,(HL)               Get SYSCON $15, in A.
        RRA                     Move bit 0 to carry.
        JR C, EXTEND            If it's 0, then there's no initialization to do.
        LD DE,$0004             Point HL to SYSCON $01 for next
        ADD HL,DE               bank and return.
        RET

EXTEND
M0C2F   LD C,$08                Prepare to set a Horizontal Select.
        LD A,(MAXBNK)           Get the bank number.
        DEC HL                  Point to SYSCON $12 (address for
        DEC HL                  initialization code) and put the contents
        DEC HL                  of SYSCON $12 in to HL.
```

```
        LD D,(HL)           This is a bug: it wipes out the SYSCON
        DEC HL              pointer.
        LD E,(HL)
        LD H,D
        LD L,E
        LD B,A              Put bank number in B.
        PUSH HL             Push address of CALL routine.
        PUSH BC             Push bank number of call routine with HS
        LD BC,$0000         being every chunk, except 3, which
        PUSH BC             contains the CALL_BANK code.
        PUSH BC             There will be no output on input params.
        CALL CALL_BANK      Call to initialize.
        LD DE,$0008         If HL had not been wiped, this may have
        ADD HL,DE           updated the SYSCON pointer to SYSCON
        RET                 $01 for the next bank.
```

Reset System Configuration Subroutine

Resets the SYSCON table by fixing up any values that may have changed.

RESET-SYSCONF (RESSCT)

```
M0C4C   XOR A
        LD (MAXBNK),A
        LD (BS_MAX_BANK),A
        LD D,$C0
        LD E,$00
        CALL WRITE_BS_REG
        LD HL,(SYSCON)
        LD DE,$000C
```

RSC-1

```
M0C60   ADD HL,DE
        CALL RESET-BS-REG
        JP NC, SET-END-MARKER
        LD A,(HL)
        PUSH HL
        CP $80
        JR NZ, CALL-RES-BS-REG      (0C72)
        LD DE,$0018
        ADD HL,DE
        LD (HL),A
```

CALL-RES-BS-REG

```
M0C72   CALL RESET-BS-REG
        LD HL, BNADDR
        LD E,$FF
        LD D,A
        PUSH DE
        LD DE,$0000
        PUSH DE
```

```
        PUSH HL
        LD DE,$0016
        PUSH DE
        LD DE,$0001
        PUSH DE
        CALL XFER_BYTES
        EX AF,AF'
        LD A,(HL)
        CPL
        DEC HL
        LD (HL),A
        EX AF,AF'
        LD D,$FF
        LD E,A
        PUSH DE
        PUSH HL
        LD DE,$0002
        PUSH DE
        LD DE,$0001
        PUSH DE
        PUSH DE
        CALL XFER_BYTES
        LD E,$FF
        LD D,A
        PUSH DE
        LD DE,$0002
        PUSH DE
        INC HL
        PUSH HL
        LD DE,$0001
        PUSH DE
        PUSH DE
        CALL XFER_BYTES
        LD A,(HL)
        DEC HL
        LD B,(HL)
        CP B
        JR NZ, SKIP-WR-BS-REG     (0CCA)
        POP HL
        LD A,(HL)
        CP $02
        JR NZ, CALL-SET-RST-56    (0CC5)
        INC HL
        INC HL
        JR WRITE-BS-REG           (0CE8)
```

CALL-SET-RST-56

```
M0CC5   CALL SET-RST-56           Set RST 56
        JR WRITE-BS-REG           (0CE8)
```

```
SKIP-WR-BS-REG
M0CCA   LD C,(HL)
        POP HL
        INC HL
        INC HL
        LD A,(HL)
        CP C
        JR Z, WRITE-BS-REG          (0CE8)
        PUSH HL
        EX DE,HL
        LD HL,BNADDR
        LD BC,$0016
        LDIR
        POP HL
        DEC HL
        LD A,(MAXBNK)
        SET 7,A
        LD (HL),A
        CALL INIT-EXPROM-BANK
        INC HL

WRITE-BS-REG
M0CE8   LD D,$C0
        LD E,$01
        CALL WRITE_BS_REG
        LD DE,$0016
        JP RSC-1

SET-END-MARKER
M0CF5   LD (HL),$80
        CALL CLEAR-SYSCONF
        RET
```

Renumber Banks Based on Interrupt Priority Subroutine

```
CLEAR-SYSCONF
M0CFB   XOR A                       Load 0 into MAXBANK.
        LD (MAXBNK),A

GET-SYSCONF-TABLE
M0CFF   LD HL,(SYSCON)              Get SYSCON table address.
        LD DE,$000C                 Point to SYSCON $00 for first bank.
        ADD HL,DE

CHK-RENUM-BIT-LOOP
M0D06   LD A,(HL)                   Get SYSCON $00 and
        CP $80                      compare it to $80.
        JR Z,CLEAR-MAX-BANK         If equal, we've reached end of the table.
        INC HL                      Point to SYSCON $01: the bank number.
```

```
        LD A,(HL)                 Get the bank number and
        BIT 7,A                   check bit 7. This bit is only set if bank has
                                  not been renumbered yet.
        JR NZ,USR-SYS-1           Jump ahead if not renumbered.
        LD DE,$0017               Point to SYSCON $00 for next bank.
        ADD HL,DE
        JR CHK-RENUM-BIT-LOOP Loop back.

USR-SYS-1
M0D17   LD (BNADDR),HL            Save the bank number to a temp location.
        DEC HL                    Point to SYSCON $00.
        LD A,(HL)                 Get value and
        CP $02                    compare to $02 (ROM value).
        JR NZ,USR-SYS-2           Jump if a ROM bank.
        LD DE,$0017               Point to SYSCON $17, the interrupt
        ADD HL,DE                 priority.
        LD A,$FF                  Assign a priority of $FF (lowest priority).
        JR USR-SYS-3

USR-SYS-2
M0D28   LD DE,$0017               Point to SYSCON $17.
        ADD HL,DE
        LD A,(HL)                 Get the priority.

USR-SYS-3
M0D2D   LD (BANKP),A              Save the priority.

USR-SYS-LOOP
M0D30   INC HL                    Point to SYSCON $00 for next bank.
M0D31   LD A,(HL)                 Get SYSCON $00 and
        CP $80                    compare to $80, which marks end of table.
        JR Z,INCREASE-BANKS       Jump forward if end of table.
        INC HL                    Point to SYSCON $01.
        LD A,(HL)                 Get SYSCON $01
        BIT 7,A                   and check bit 7. Set if bank is not yet
                                  renumbered.
        JR NZ,USR-SYS-L-1         Jump ahead if it is not renumbered.
        LD DE,$0017               Point to SYSCON $17.
        ADD HL,DE
        JR USR-SYS-LOOP           Go back until done.

USR-SYS-L-1
M0D42   DEC HL                    Point to SYSCON $00.
        LD A,(HL)                 Get it and
        CP $02                    compare to $02 (ROM bank).
        JR NZ,USR-SYS-L-2         Jump forward if not ROM bank.
        LD DE,$0017               Point to SYSCON $17.
        ADD HL,DE
        JR USR-SYS-LOOP           Loop back until done.
```

```
USR-SYS-L-2
M0D4E   EX DE,HL            BUG: Should copy HL to DE.
        LD BC,$0017         Point HL to SYSCON $17.
        ADD HL,BC
        LD A,(BANKP)        Put previous priority in B.
        LD B,A
        LD A,(HL)           Get present priority and
        CP B                compare to B.
        JR NC,USR-SYS-LOOP  Go back if present priority is lower.
        LD (BANKP),A        Save new priority
        LD (BNADDR),DE      and bank number address.
        JR USR-SYS-LOOP     Loop back.
```

Increase Banks Subroutine

```
INCREASE-BANKS
M0D64   LD A,(MAXBNK)       Get MAXBNK and
        INC A               increment it then
        LD (MAXBNK),A       save it.
        LD HL,(BNADDR)      Get address of highest priority bank number,
        LD (HL),A           Save bank number to that address.
        LD E,$FF
        LD D,A
        PUSH DE             In theory, this is supposed to put the bank
        LD E,(HL)           number at the location  $0000 of the RAM
        LD D,$00            bank. Instead, due to a bug, it uses the
        PUSH DE             bank number as an address and copies
        LD E,$00            from that address in the home bank to
        PUSH DE             address $0000 of the expansion bank.
        LD E,$01
        PUSH DE
        PUSH DE
        CALL XFER_BYTES
        JP GET-SYSCONF-TABLE
```

Clear Max Banks Subroutine

```
CLEAR-MAX-BANK
M0D84   XOR A               Set MAXBNK to 0.
        LD (MAXBNK),A
        LD (BS_MAX_BANK),A
        LD D,$C0            Send $00 to register $C0.
        LD E,$00            Resets the daisychain.
        CALL WRITE_BS_REG
        LD HL,(SYSCON)      Get address of first bank number in the
        LD DE,$000D         SYSCON table.
        ADD HL,DE
        LD D,$A0
```

CALL-RESET-BS-REG

```
M0D9B   CALL RESET-BS-REG     Increments MAXBANK and checks
        RET NC                to see if that bank number exits. Returns
                              with carry flags set. Installs that number in the
                              bank — after they had been renumbered.
        LD E,(HL)             Get the correct (renumbered) bank
        CALL WRITE_BS_REG     number and install it in the bank. If we get here, D
                              ontains $A0, left over from the routine above.
        LD D,$C0              Send $01 to register $C0, which
        LD E,$01              advances the daisychain.
        CALL WRITE_BS_REG
        LD DE,$0018           Point to the next bank number in the
        ADD HL,DE             SYSCON table.
        JR CALL-RESET-BS-REG
```

Change Video Mode Routines

Open Display File Routine

Open the second display file. Move the UDG, then move the stack and RAM resident code to high memory. Update the video mode and the video port.

OPEN-DFILE (OPDFIL)

```
M0DB0   PUSH BC               Save the registers.
        PUSH DE
        PUSH HL
        PUSH AF
        LD HL,(PRAMT)         Get physical RAM top.
        LD DE,(UDG)           Get the pointer to the UDG.
        AND A                 Number of bytes occupied by
        SBC HL,DE             the UDG.
        LD B,H                Put into byte counter.
        LD C,L
        INC BC                Adjust for zero base.
        LD HL,(UDG)           Point to the present UDG.
        PUSH HL               Save briefly.
        LD DE,$0840           Number of bytes needed by RAM resident
                              routines and stack.
        AND A                 Compute the new address
        SBC HL,DE             for the UDG.
        EX DE,HL
        POP HL                Restore the present UDG address.
        LD (UDG),DE           Point to the new UDG location.
```

OPEN-XFER-UDG

```
M0DD1   LDIR                  Move the UDG.
        LD HL,$0000           HL will contain the new
```

```
                ADD HL,SP           SP used when the stack and
                LD BC,$97C0         RAM resident code are moved.
                ADD HL,BC
                DI                  Disable interrupts.
                LD SP,HL            Update the stack pointer.
                LD DE,$F7C0         New location for stack and RAM routines.
                LD HL,$6000         Old location.
                LD BC,$0840         Length of stuff to move.

OPEN-XFER-DISPATCHER
M0DE6           LDIR                Move it.
                LD HL,$1D00         Point to the fix table.
                LD BC,$97C0         Offset to be added to addresses.

OPEN-FIX-BL-LOOP
M0DEE           LD E,(HL)           Get fix offset.
                INC HL
                LD D,(HL)
                INC HL              Point to next entry.
                LD A,E              Jump if we have reached the end
                OR D                of the table.
                JR Z, OPEN-FIX-BL-DONE
                EX DE,HL            Point to moved code that
                ADD HL,BC           needs fixing.
                PUSH DE             Save the table pointer
                LD E,(HL)
                INC HL              Get the code needing fixing.
                LD D,(HL)
                EX DE,HL
                ADD HL,BC
                EX DE,HL            Fix the address and
                LD (HL),D           put it back.
                DEC HL
                LD (HL),E
                POP HL              Restore the table pointer.
                JR OPEN-FIX-BL-LOOP Back around for more entries.

OPEN-FIX-BL-DONE
M0E05           POP AF              Get caller's desired video mode
                LD (VIDMOD), A      and store it.
                PUSH AF             Save it again.
                EI                  Enable interrupts.

CLEAR-DFILE2
                LD HL,$6000         Point to the second display file.

CL-DF-LOOP
M0E0E           XOR A               Zero A.
                LD (HL),A           Clear the byte.
                INC HL              Step forward one.
```

```
LD A,H                  Loop until we reach $7AFF
CP $7B
JR NZ, CL-DF-LOOP       Keep clearing.
POP AF                  Get the requested video mode
PUSH AF                 save it again.
AND $7F                 Only use the lower seven bits (don't munge the
                        EXROM select).
LD B,A                  Save it in B.
IN A,(DECR)             Get the present value of the display enhancement
                        control register.
AND $80                 Save the EXROM select.
OR B                    OR in the request.
OUT (DECR),A            Update the display enhancement control register.
POP HL                  Restore registers.
POP DE
POP BC
POP AF
RET
```

Close Display File Routine

Close the second display file. Moves the stack and RAM resident code from high memory back to low memory. Updates SP and relocates the UDG.

This code does not work properly.

CLOSE-DFILE (CLDFIL)

```
M0E27   PUSH AF             Save registers.
        PUSH BC
        PUSH DE
        PUSH HL
        IN A,(DECR)         Get display mode value
        AND $80             and set for black and white,
        OUT (DECR),A        normal video, EXROM selected.
        LD HL,$0000
        ADD HL,SP           Set HL to the value the
        LD DE,$97C0         SP will need once the stack
        AND A               has been moved to low memory.
        SBC HL,DE
        DI                  Disable interrupts.
        LD SP,HL            Update the stack pointer.
        LD HL,$FFFF         Move the RAM resident code
        LD DE,$683F         and the stack back to low
        LD BC,$0840         memory (starting at $6000)
```

CLOSE-XFER-DISPACHER

```
M0E46   LDDR                Do the move.
        LD HL,$1D00         The location of the fix table.
        LD BC,$97C0         Offset value to convert code addresses from high
                            memory addresses to low memory addresses.
```

CLOSE-FIX-BL-LOOP

```
M0E4E   LD E,(HL)
        INC HL              Get a fix address.
        LD D,(HL)
        INC HL              Point to the next one.
        LD A,E
        OR D                Jump if the end of the table.
        JR Z, CLOSE-XFER-UDG
        PUSH HL             Save the table pointer.
        EX DE,HL
        LD E,(HL)           Get the JP/CALL address
        INC HL              in the RAM resident code.
        LD D,(HL)
        EX DE,HL            Convert the address from
        AND A               high memory to low memory.
        SBC HL,BC
        EX DE,HL
        LD (HL),D           Modify the code.
        DEC HL
        LD (HL),E
        POP HL              Restore the table pointer
        JR CLOSE-FIX-BL-LOOP back around for more entries.
```

CLOSE-XFER-UDG

```
M0E66   XOR A               Force normal video mode.
        LD (VIDMOD),A
        EI                  We can now allow interrupts again.
        LD HL,$F7BF         Address of UDG end when code is in high
                            memory.
        LD DE,$FFFF         New UDG location.
        PUSH HL             Save the old UDG address.
        LD BC,(UDG)         Get the current UDG pointer.
        AND A               Number of bytes to move.
        SBC HL,BC
        LD B,H              Transfer to byte counter.
        LD C,L
        INC BC              Bump by one.
        POP HL              Restore old address.
        LDDR                Move the UDG
        LD DE,$0840
        LD HL,(UDG)         Update the UDG pointer
        ADD HL,DE           in the system variables.
        LD (UDG),HL
        POP HL
        POP DE              Restore the caller's
        POP BC              registers.
        POP AF
        RET
```

Change Video Mode Routine

This function is used to open/close the second display file (and its corresponding attributes file). If the second display file is closed, it should not be used by the applications programmer because data structures used by the operating system overlay the memory area. To access the routine, pass the requested video mode in A.

System variable VIDMOD tracks the current display mode.

Using the second display file means that the function dispatcher and system stack are moved to the top of memory.

CHANGE-VIDEO (CHNG_V)

```
M0E8E   PUSH BC              Save the registers.
        PUSH DE
        PUSH HL
        PUSH AF
        LD B,A               Save the requester's mode.
        LD A,(VIDMOD)        Jump if present mode is NOT
        AND A                zero.
        JR NZ, CV-MODE-0
        OR B                 Jump if mode 0 is requested
        JP Z, CV-END         and current mode is '0'. Exit video mode change
                             with out doing anything.
```

Initiate a change from mode 0 to some other mode

```
        LD HL,$12C0          Number of bytes needed for RA services.
        LD B,H               Save for POINTERS later on, if needed.
        LD C,L
        LD DE,$0840          $2B00 total bytes needed.
        ADD HL,DE
        LD DE,(STKEND)       Add to free RAM space stored in
        ADD HL,DE            STKEND.
        LD DE,(RAMTOP)
        AND A                If the carry is set, RAMTOP is set
        SBC HL,DE            Too low to allow moving the RAM services,
        JP NC, CV-ABORT      error exit.
        LD HL,$683F          Fence value: all system variables that point above
                             this will be changed. (BC was set up earlier to
                             $12C0).
        LD DE,POINTERS       Address for POINTERS.
        PUSH DE
        LD DE,$FF00          Specify home bank, all chunks
        PUSH DE              in home.
        LD DE,$0000
        PUSH DE              No parameters in or out.
        PUSH DE
        CALL CALL_BANK
```

LD-UP-PROG

```
M0EC8   LD HL,(STKEND)
        EX DE,HL            Move the FP stack.
        LDDR
        POP AF              Restore caller's video mode.
        PUSH AF
        CALL OPEN-DFILE     Open the second display file.
        LD BC,$97C0
```

UPDATE-POINTERS

```
M0ED6   LD HL,(ERRSP)       Point the ERRSP to the newly moved
        ADD HL,BC           stack.
        LD (ERRSP),HL
        LD HL,(LISTSP)
        ADD HL,BC           Update list stack pointer.
        LD (LISTSP),HL
        LD HL,(MSTBOT)
        ADD HL,BC           Update machine stack pointer.
        LD (MSTBOT),HL
        JR CV-END           Jump to exit.
```

CV-MODE-0

```
M0EED   LD A,B              Jump if requested mode is
        AND A               '0' (move the stack and
        JR Z, CV-MODE-0-2   RAM resident code back down).
```

The requested mode was NOT zero and the present mode is not zero, so simply update the necessary items, then exit.

```
        AND $7F             Don't allow changes to the EXROM select.
        LD B,A              Briefly save.
        IN A,(DECR)         Get the present video control port value.
        AND $80             Save the EXROM select.
        OR B                OR in the requested mode
        OUT (DECR),A        and output to the port.
        LD A,B              Save the requested mode to
        LD (VIDMOD),A       VIDMOD.
        JR CV-END           Done.
```

CV-MODE-0-2
The present mode is not zero and the requested mode is zero, so prepare to move everything back around as needed.

```
M0F01   CALL CLOSE-DFILE        Close the display file.
        LD BC,$97C0
        LD HL,(ERRSP)           Update ERRSP to new
        AND A                   stack location.
        SBC HL,BC
        LD (ERRSP),HL
        LD HL,(LISTSP)
        AND A                   Do the same for the
        SBC HL,BC               LISTSP.
        LD (LISTSP),HL
        LD HL,(MSTBOT)
        AND A                   Yet again for MSTBOT.
        SBC HL,BC
        LD (MSTBOT),HL
        LD BC,$12C0             Number of bytes to delete.
        LD HL,$6840             Start address.
        LD DE,RECLAIM-2         Delete the bytes.
        PUSH DE
        LD DE,$FF00             HOME bank, all
        PUSH DE                 chunks.
        LD DE,$0000
        PUSH DE                 No params in or out.
        PUSH DE
        CALL CALL_BANK
        JR CV-END               Done.
```

CV-ABORT
```
M0F3A   SCF                     Set carry flag (indicates error)
        JR GET-REGISTERS        and exit.
```

CV-END
```
M0F3D   AND A                   Reset carry flag (all OK).
```

GET-REGISTERS
```
M0F3E   POP AF
        POP HL                  Restore caller's registers.
        POP DE
        POP BC
        RET
```

PASSING Routine

This routine was supposed to take information from PASSEM in the HOME ROM and pass it along to routines to handle expansion bus devices like a disk drive.

XPASSING (PASSIN)

```
M0F43   LD BC, (CH_ADD)        Point BC to CH_ADD.
        CALL M255D             Off by one byte to CK-END (M255E).
        LD HL,(CH_ADD)         Point HL to CH_ADD.
        AND A                  Clear flags.
        SBC HL,BC              Subtract
        DEC HL
        LD A,L
        LD HL,(STKEND)         Point HL to the end of the stack.
        LD (HL),A              Put A on the stack.
        INC HL
        POP BC
        LD (HL),B              Put B on the stack.
        INC HL
        LD (HL),C              Put C on the stack.
        INC HL
        LD (STKEND),HL
        LD HL,(CH_ADD)
        DEC HL
        BIT 0,A
        JR Z,PASS-2
```

PASS-LOOP

```
M0F67   DEC A
        LD B,(HL)
        DEC HL
        DEC A
        JP M, GET-GOSUB-ADDR
        LD C,(HL)
        DEC HL
        PUSH BC
        JR PASS-LOOP
```

PASS-2

```
M0F73   LD B,$20
        AND A
        RET Z
        LD C,(HL)
        DEC HL
        DEC A
        PUSH BC
        JR PASS-LOOP
```

GET-GOSUB-ADDR

```
M0F7D   LD HL,(STKEND)
        DEC HL
        LD A,(HL)
        DEC HL
        LD (STKEND),HL
        LD H,(HL)
        LD L,A
        PUSH HL
        RET
```

Bank Switching

These routines are shortcuts to finding and calling the appropriate routines, depending on whether dispatcher is in its normal location or has been relocated to accommodate a second display file.

Go To Bank Routine

GOTO-BANK (GOTO_B)

```
M0F8A   PUSH AF                 Determine where the function
        LD A,(VIDMOD)           dispatcher is located.
        AND A
        JR Z, GB-MODE-0         Jump to normal function dispatch.
        POP AF                  Restore AF.
        JP GOTO_BANK-HI         Jump to high function dispatch
```

GB-MODE-0

```
M0F95   POP AF                  Restore AF.
        JP GOTO_BANK
```

Call Bank Routine

CALL-BANK (CALL_B)

```
M0F99   PUSH AF                 Save AF.
        LD A,(VIDMOD)           Check VIDMOD.
        AND A                   Clear flags.
        JR Z, CB-MODE-0         Jump if non-expanded (single display file) video
                                mode.
        POP AF                  Restore AF.
        JP CALL_BANK-HI         Jump to expanded video mode (two display file)
                                function dispatcher.
```

CB-MODE-0

```
M0FA4   POP AF                  Restore AF
        JP CALL_BANK            Jump to normal function dispatcher.
```

Function Dispatcher

The function dispatcher provides a common interface to system routines via service code and jump flag parameter passed on the machine stack.

To call a function:
1. set up and memory and stack (machine and/or calculator) locations as if invoking the desired service directly
2. push the parameters for the dispatcher on the machine stack in the order outlined below
3. set up the registers as if invoking the desired service directly and call the dispatcher at its current location (Chunk 3 if VIDMOD=0 or Chunk 7 if VIDMOD<>0)

Parameters

PRM_OUT (16 bits)	Number of bytes of parameter data being passed on the stack to the specified service (number of stack "pushes" * 2). Zero if no parameters being passed. This parameter is passed to the dispatcher only if the Jump Flag (bit 15 of SVC_CODE) is *not* set.
PRM_IN (16 bits)	Number of bytes of parameter data to be passed back from the specified service (number of stack pushes * 2). Zero if no parameters are passed back. This parameter is passed to the dispatcher only if the Jump Flag (bit 15 of SVC_CODE) is *not* set.
SVC_CODE (16 bits)	Bits 0-14 identify the service to be invoked. Bit 15 (Jump Flag) is set if no return is desired (jump to service rather than call). Bit 15 is zero if return is desired.

The code for the function dispatcher is copied from $1000 in the Extension ROM to Chunk 3 ($6200) at system initialization. Since this is in the same memory area as the second display file, this code must be relocated, along with the machine stack, if the second display file is used. The CHANGE-VIDEO routine does the necessary relocation and modifications.

Because this code is not in a fixed location, access to these routines depends on the current video mode. Test the value of VIDMOD to determine the current video mode. A zero indicates that the second display file is not in use and that the OS RAM routines are in Chunk 3; a non-zero value indicates that the routines are in Chunk 7 ($F9C0).

Sample usage

```
LD DE, $0000
PUSH DE          Set input and output params
PUSH DE          to zero.
LD DE, function #  From table below.
PUSH DE          Put function # on stack.
CALL $6200 or $F9C0
```

Function Dispatcher Services

W_TAPE **$00**
Write a block to tape.

R_TAPE **$01**
Read a block from tape.

RD_BIT **$02**
Read a bit from tape.

R_EDGE **$03**
Read an edge from tape.

SLVM **$04**
General tape routine.

LOAD **$05**
Load.

MERGE **$06**
Merge.

SAVE **$07**
Save.

CHNG_VID **$08**
Change video mode.

W_BORD **$09**
Write border color.

RESERVED **$0A-$0D**

GET_STATUS **$0E**
Returns memory selection (low active) in C for bank number in B.

GET_NUMBER **$0F**

BANK_ENABLE **$10**

GOTO_BANK **$11**

CALL_BANK **$12**

XFER_BANK **$13**

RESERVED **$14-$18**

UPD_K **$19**

Process keyboard input.

PARP **$1A**

Generate DE+1 cycles of tone with a period of 8N+236 to *N+246 T-states. HL=N.

BEEP **$1B**

Beep command. Parameters on calculator stack. Exits via PARP.

K_DUMP **$1C**

COPY command. Dumps primary display file to printer.

SENDTV **$1D**

Character output to screen/printer. Character code in A.

SETAT **$1E**

Set print position to value in BC. Line number (0-23) in B; column number (0-31) in C.

STTBYT **$1F**

Set attribute byte for display file address in HL using ATTR_T, MASK_T and P_FLAG.

R_ATTS **$20**

Copy permanent attribute info to temporary attribute variables.

CLLHS **$21**

Clear lower half of the screen in the primary display file.

CLS **$22**

Clear entire screen in primary display file.

DUMPPR **$23**

Print/clear print buffer.

PRSCAN **$24**

Send scan (32 bytes) to printer. Pixel data address in HL. Number of scans remaining in B (1-8).

DESLUG **$25**

Remove number slugs from edit line buffer. Address in HL.

K_NEW **$26**

NEW command.

INIT **$27**

Initialize the system. Maximum RAM address in DE. A=0, power on state. A=-1, execute NEW.

INCH **$28**

Input a character to A from the currently selected channel. Returns NC if no input.

SELECT **$29**

Select channel (stream). Number in A.

INSERT **$2A**

Insert BC bytes before address in HL Copies up all from HL to STKEND and updates affected system variables. Returns BC=0, DE=address of last byte of inserted space, HL=address of byte before first.

RESET **$2B**

Reset calculator stack. Sets STKEND = STKBOT and MEM - MEMBOT ($5C92).

CLOSE **$2C**

CLOSE # command. Channel number on calculator stack.

CLCHAN **$2D**

Close channel. Value from STRMS (index into CHANS) in BC.

OPEN **$2E**

OPEN # command. Channel number and device specification on calculator stack.

OPCHAN **$2F**

Open channel. Device specification on calculator stack. Pointer into STRMS based on channel number in DE.

CAT **$30**

CAT command. Not implemented.

DELETE **$31**

DELETE command. Not implemented.

FORMAT **$32**

FORMAT command. Not implemented.

MOVE **$33**

MOVE command. Not implemented.

FLASHA **$34**

Flash character in A to screen. Calls SENDTV, assumes lower screen is selected. Used to flash the cursor.

FIND_L **$35**

Find BASIC program line with the number in HL. If line is found, returns Z and address of line in HL. Otherwise, returns NZ and HL either contains address of line with next larger line number of points to the variables area if there is no larger line number. Requested line number returned in BC and address of preceding line in DE (DE=HL if no preceding line).

SUBLIN **$36**

Finds either the D'th statement (D=statement number; E=0) or first statement whose keyword token matches E (D=0), in a line pointed to by HL. If the D'th statement is found, returns Z and HL and CH_ADD both point to 1 byte before

the statement. If the line contains exactly D-1 statements, then the next line counts as the D'th. If a match on E is found, it returns NZ, NC and both HL and CH_ADD point to the keyword. D is decremented by the number of statements examined (ie D=-2 if two statements). If no match is found for E then returns NZ, C with both HL and CH_ADD pointing to the end-of-line byte ($0D).

RECLEN **$37**

Returns in BC the length of the record pointed to by HL. The record can be a program line, string, numeric variable or array.

DELREC **$38**

Delete record point to by HL having length BC from program or variables memory. Updates affected system variables.

PUT_BC **$39**

Converts number in BC from binary to ASCII and outputs to currently selected channel. If BC is less than 0, outputs a 0.

SYNTAX **$3A**

Check syntax of command or program line in Edit Line buffer (E_LINE). ERR_NR=-1 if no errors, otherwise contains Error Number - 1.

EXCUTE **$3B**

Execute command(s) from Edit Line buffer.

FOR **$3C**

FOR command.

STOP **$3D**

STOP command. Does RST 8 with error number 9.

NEXT **$3E**

NEXT command.

READ **$3F**

READ command.

DATA **$40**

DATA statement.

RESTBC **$41**

RESTORE command. Line number in BC.

RAND **$42**

RANDomize command. Sets seed for random number generator based on parameter on calculator stack. If parameter is non-zero, value is loaded to SEED. If zero, value in FRAMES is loaded to SEED.

CONT **$43**

CONT command. Loads values from OLDPPC and OSPPC to NEWPPC and NSPCC and returns. Inside the BASIC interpreter, this results in executing from line number in NEWPPC, statement number in NSPPC.

JUMP **$44**

Jump to line. Loads line number from calculator stack to NEWPPC, sets NSPPC to 0 and returns.

FIX_U1 **$45**

Converts floating point number on calculator stack to a single byte unsigned binary value in A (uses FP-TO-A). Error B if the number is out of range.

FIX_U **$46**

Converts floating point number on calculator stack to a 2-byte unsigned binary value in BC (uses FP2BC). Error B if number is out of range.

CLEAR **$47**

CLEAR command. Processes parameter on calculator stack to value in BC for CLR_BC.

CLR_BC **$48**

Value in BC is new RAMTOP. Deletes variables, clears screen and calculator stack.

GO_SUB **$49**

GO_SUB command. Inserts a 3 byte GO_SUB block into the machine stack above the two most recent entries. The block consists of the current line number (2 bytes) and statement number (1 byte) to be used when RETURN is executed. Calls JUMP to process GO_SUB parameter and returns. At return to caller, machine stack consists of top of stack at point GO_SUB was called, followed by 3 byte entry (line number MSB, line number LSB, statement number).

CHK_SZ **$4A**

Checks if there is room for BC + 80 ($50) bytes between STKEND and RAMTOP. Addition of 80 bytes is left over from Spectrum to guarantee minimum machine stack where the stack was at the top of RAM. Error 4 if not enough room.

RETURN **$4B**

RETURN command. Retrieves most recent GO_SUB block from machine stack (SP + 4), loads data to NEWPPC and NSPPC and returns. Error 7 if MSB of line number is $3E (end of stack marker).

PAUSE **$4C**

PAUSE command. Processes parameter on calculator stack to BC then waits BC frames or until key is depressed. Uses HALT instruction, so interrupts must be enabled.

BREAK? **$4D**

Reads keyboard directly to see if both CAPS SHIFT and SPACE (BREAK) are being pressed. Returns NC if it is pressed and ON ERROR if it is not active.

DEF **$4E**

Define Function.

K_LPR **$4F**

LPRINT - selects channel 3 and processes items in LPRINT statement for output via WRCH.

K_PRIN **$50**

PRINT - selects channel 2 and processes items in PRINT statement for output via WRCH (same code used for K_LPR).

P_SEQ **$51**

Code used by K_LPR and K_PRIN to process output data and controls in BASIC statement. Address in CH_ADD.

INPUT **$52**

INPUT command. Selects channel 1 and processes I/O for keyboard/lower screen using a buffer at WORKSP for input.

I_SEQ **$53**

Code used by INPUT to process input items and controls in BASIC statement. Address in CH_ADD.

NOTKB? **$54**

Returns Z if current channel is keyboard/lower screen (device specification "K").

COLOR **$55**

Adjusts system variables ATTR_T, MASK_T and P_FLAG for color code in D (0-9). Enter with C to set INK or NC to set PAPER. Error K if D is invalid.

HIFLSH **$56**

Adjusts system variables (ATTR_T and MASK_T) for flash/bright code in D (0, 1 or 8). Enter with C for FLASH, NC for BRIGHT. Error K if D is invalid.

SCRMBL **$57**

Returns the primary display file address for the pixel with coordinates in BC (B=Y, C=X) in HL. Returns the bit number (0-7) where 0=lefthand or most significant bit in A. Error B if Y is greater than 175.

PLOT **$58**

PLOT command. Processes X/Y parameters on the calculator stack to BC for plotting of pixel via PLOTBC.

PLOTBC **$59**

Deals with pixel for coordinates in BC (B=Y, C=X). Processes using P_FLAG for INVERSE and OVER attributes. Updates attribute file and sets COORDS=BC.

GET_XY **$5A**

Converts a pair of numbers from the calculator stack to 2 single byte numbers. Top number goes to B and second to C. D=sign of B and E=sign of C (+1 or -1). Used by PLOT and other routines.

CIRCLE **$5B**

CIRCLE command. Calculates successive plot positions from the parameters in the BASIC statement.

DRAW **$5C**

DRAW command. Calculates successive plot positions from the parameters in the BASIC statement.

DRAW_L **$5D**

Plots a straight line from current position (COORDS) based on parameters from calculator stack (X, Y).

EXPRN **$5E**

Evaluates expression in BASIC program line (CH_ADD), putting value on calculator stack.

F_SCRN **$5F**

SCREEN$ function. Matches screen line/column number (parameters on calculator stack) against standard ASCII character set. Returns BC=0 if nothing is found. BC=1 and DE points to character code byte if match found.

F_ATTR **$60**

ATTR function. Returns attribute byte value controlling screen pixel position based on stack parameters on calculator stack (X, Y).

RND **$61**

RND function. Uses value in SEED to generate a pseudo-random number which is placed on the calculator stack (floating point number).

F_PI **$62**

PI function. Places value of PI on calculator stack.

F_INKY **$63**

INKEY$ function. Scans keyboard and puts character code byte in WORKSP if key detected. If any case, pushes registers AEDCB onto calculator stack. BC=0 if no input, =1 if character code stored; DE=address of character code byte.

FIND_N **$64**

Parse and find a specifed variable. Searches variables area for match against identifier pointed to by CH_ADD. Adjusts bit NO of FLAGS (bit 6) for type (1=numeric, 0=string). Also used to find formal parameters for user defined functions.

PSHSTR **$65**

Push string. Clears bit NO of FLAGS and pushes registers AEDCB on to calculator stack, adjusting STKNXT upwards. DE contains address of string, BC contains length.

PAEDCB **$66**

Same code as for PSHSTR but preserves state of bit NO of FLAGS (bit 6).

LET **$67**

LET command. Processes existing or creates new variables.

POPSTR **$68**

Pop string. Pops end of calculator stack (STKNXT-1 through STKNXT-5) to registers BCDEA, adjust ing STKNXT downwards.

DIM **$69**

DIM statement. Creates or initializes numeric or string arrays.

STKUSN **$6A**

Stack unsigned number. Inputs a floating point number on to the calculator stack from a series of ASCII characters address by CH_ADD. The first character is already in A (either decimal point, binary token or digit).

STK_A **$6B**

1-byte unsigned integer in A to top of calculator stack (binary to floating point). Loads 0 to B an A to C, then executes STACK-BC.

STK_BC **$6C**

2-byte unsigned integer in BC to top of calculator stack (binary to floating point).

ININT **$6D**

Converts a series of ASCII digits pointed to by CH_ADD into an unsigned floating point integer on the calculator stack. First character is in A on entry. Terminates when non-digit is found.

FP2BC **$6E**

Pops top of calculator stack (floating point number) and puts it in BC, rounded to nearest integer. Returns NZ if value is negative. Returns C if number exceeded maximum 2-byte value (65535). Range -65535 to 65535.

FP2A **$6F**

Pops top of calculator stack (floating point number) and puts in A, rounded to nearest integer. Returns NZ if value is negative. Returns C if number exceeded maximum 1-byte value (255). Range: -255 to 255.

OUTPUT **$70**

Outputs number on top of calculator stack to currently selected channel via WRCH. Converts from floating point to ASCII.

SUB **$71**

Subtract floating point format numbers (HL) minus (DE). (DE) assumed to be (HL) +5.

ADD **$72**

Add (HL) to (DE). See SUB.

MULT **$73**

Integer multiply HL * DE. Returns C if overflow.

TIMES **$74**

Floating point multiply (HL) * (DE).

DIVIDE **$75**

Floating point divide (HL)/(DE).

TRUNC **$76**

Truncates a floating point number (HL) towards zero to an integer. Assumes (DE) = (HL) + 5.

FLOAT **$77**

Converts number (HL) to floating point format. Assumes HL points to an integer in 5-byte format.

INTDIV **$78**

Replaces top two numbers on calculator stack (X and Y) with X mod Y and the integer quotient INT (X/Y). Returns with DE and HL = calculator stack pointers.

INT **$79**

Replaces the top of the calculator stack with its integer part. Returns with HL = top of calculator stack and DE = next free space.

EXP **$7A**

Replaces the top of the calculator stack, X, with EXP(X). Returned with DE and HL = calculator stack pointers.

LN **$7B**

Replaces top of the calculator stack with its natural logarithm. Returns DE and HL = calculator stack pointers.

ANGLE **$7C**

Replaces the top of the calculator stack (X) with Y where Y is greater than or equal to -1 and less than or equal to +1 and SIN X = SIN(PI/2 *).

COS **$7D**

Replaces the top of the calculator stack with its cosine.

SIN **$7E**

Replaces the top of the calculator stack with its sine.

TAN **$7F**

Replaces the top of the calculator stack with its tangent.

ATN **$80**

Replaces the top of the calculator stack with its inverse tangent.

ASN **$81**

Replaces the top of the calculator stack with its inverse sine.

ACS **$82**

Replaces the top of the calculator stack with its inverse cosine.

ROOT **$83**

Replaces the top of the calculator stack with its square root.

TO_THE **$84**

Replaces the top two numbers on the calculator stack (X, Y) with X**Y.

RDCH **$85**

Wait for character from currently selected channel (calls INCH). Returns character code in A. If the current device is a finite device and the endow file arrives, then RDCH aborts with Report Code 8.

SENDCH **$86**

Write character whose code is in A to currently selected output channel.

WRCH **$87**

Write character.

K_SCAN **$88**

Keyboard scan.

P_LFT **$89**

Backspace. Sets current column position back 1 for selected device. System variable updated is SPOSN, SPOSNL or PPOSN for Screen, Lower Screen or Printer, respectively.

P_RT **$8A**

Outputs a space to currently selected device.

P_NL **$8B**

End-of-line. Sets current position to start of next line if screen or outputs printer buffer if printer.

PUTMES **$8C**

Output message to currently selected device. DE points to base of message table which contains variable length ASCII coded messages. The first byte of the table and the last byte of each message must have the most significant bit set. Register A contains the message number, from 0 up.

K_CLS **$8D**

CLS command. Executes both CLS and CLLHS.

SCRL **$8E**

Scrolls entire screen (primary display file) up 1 line.

F_PNT **$8F**

POINT function. Processes X,Y parameters from calculator stack to BC. Returns unsigned integer value = 0 or 1 on calculator stack reflecting state of pixel at coordinates X/Y.

DRAWLN **$90**

Same as DRAW_L but enter with BC register containing coordinates. B=Y and C=X.

PUT_LN **$91**

Output line number as 4 digits, right aligned and space filled to current selected output channel. HL points to MSB of 2-byte line number.

DISPATCH
Performs a CALL or JP from any bank to some HOME and EXROM bank routines.
Push 00 for input and output parameters and the service code to the stack before calling.

Routine address lookup table only points to low-memory addresses for RAM-resident code routines. Do not try to use the Function Dispatcher to access another RAM-resident routine if it is located in high memory.

```
M6200   LD IX,0                 Zero IX.
        ADD IX,SP               Set IX to the stack pointer. This holds the position
                                of the parameters passed while all the registers
                                are saved.
        PUSH BC                 Reserve a word on the stack.
        PUSH AF                 Save the registers.
        PUSH BC
        PUSH DE
        PUSH HL
        LD E,(IX+2)             Copy service code (SVC_CODE) to DE.
        LD D,(IX+3)
        XOR A                   Clear A.
        SLA E                   Shift bit 7 (SVC_CODE Jump Flag) of E to carry.
        RLD                     DE = 2*DE
        RLA                     A = Jump Flag
        LD HL, LAST_EXT_SVC
        SLA L
        RL H                    HL = 2*HL
        AND A
        SBC HL,DE               COMPARE HL AND DE
        JR NC,D_EXT             IF DE <= HL
        LD HL,LAST_RAM_SVC
        SLA L
        RL H
        AND A
        SBC HL,DE
        JR C,D_HOME
        LD B,255                Here for RAM-based services.
        CALL  GET_STATUS        Get status of HOME BANK.
        LD B,255                BC=HOME BANK/HORIZ SELECT
        JR D_SAVE
```

D_EXT
```
M6238   LD B,254                Here for EXT ROM-based services.
        LD C,$FE
        JR D_SAVE
```

D_HOME
```
M623E   LD B,255                Set BANK_ENABLE parameters for HOME
        LD C,0
```

D_SAVE

```
M6242   PUSH AF
        PUSH BC          Save jump flag and BANK_ENABLE parameters.
        LD HL,JMPTBL     Calculate address of table entry.
        SCF
        SBC HL,DE
        LD B,254
        CALL GET_WORD    Read table entry.
        EX DE,HL
        POP BC
        POP AF           Restore jump flag etc.
        AND A
        JR Z,D_CALL
        LD (IX+(-2)),C   Put bank # and horizontal
        LD (IX+(-1)),B   select on stack.
        LD L,(IX+0)      Save return address.
        LD H,(IX+1)
        LD (IX+3),H      Put return address back on stack.
        LD (IX+2),L
        LD (IX+1),D      Set up stack for GOTO_BANK.
        LD (IX+0),E      Put address on stack.
        POP HL           Restore registers.
        POP DE
        POP BC
        POP AF
        CALL GOTO_BANK   Go here if jump flag is not set.
```

D_CALL

```
M6274   LD L,(IX+0)      Set up stack for CALL_BANK
        LD H,(IX+1)      Put return address in proper location.
        PUSH HL
        LD L,(IX+4)
        LD H,(IX+5)
        LD (IX+(-2)),L
        LD (IX+(-1)),H
        LD L,(IX+6)      Put PARM_OUT in proper location.
        LD H,(IX+7)
        LD (IX+0),L
        LD (IX+1),H
        LD (IX+2),C      Put bank number, Horizontal Select
        LD (IX+3),B      on stack.
        LD (IX+4),E      Put address on stack.
        LD (IX+5),D
        POP HL
        LD (IX+6),L
        LD (IX+7),H
        POP HL           Restore registers.
        POP DE
        POP BC
```

```
        POP AF
        CALL CALL_BANK          Go here if jump flag not set.
        RET
```

RAM Interrupt Handler

Link to the interrupt handler in HOME ROM.

The user must enter with bank status and Z80 registers intact, with address from point of interruption on the stack. This handler saves state, including memory selection, enables the HOME bank, updates the frame counter, call the keyboard scan routine in the HOME ROM, restores state and returns to the interrupted process.

This handler is used whenever the interrupt occurs while the Extension ROM is enabled. This technique can be used for interruption processing in another bank: for example, if an LROS wanted to use the standard system ROM keyboard scanning routines.

X38INT

```
M62AE   PUSH AF                 Save state.
        PUSH HL
        PUSH IX
        LD HL,0
        ADD HL,SP
        PUSH DE
        LD A,(BS_MAX_BANK)
        LD E,A
        LD D,0
        INC DE
        INC DE
        AND A
        SBC HL,DE
        EX DE,HL
        LD IX,0
        ADD IX,DE
        POP DE
        LD SP, IX
        CALL SAVE_STATUS
        PUSH BC
        LD B,$FF
        CALL GET_STATUS
        LD B,$FF
        LD A,C
        AND $F8
        LD C,A
        CALL BANK_ENABLE
        POP BC
        LD HL,(FRAMES)          Increment FRAMES counter.
        INC HL
        LD (FRAMES),HL
        LD A,H
        OR L
```

```
        JR NZ,LIT3
        INC (IY-(FRAMES2+Y))
```

LIT3

```
M62ED   PUSH BC
        PUSH DE
        CALL KEYBOARD
        POP DE
        POP BC
```

PHLAF

Jump here to POP HL, POP AF, enable interrupts and return.

```
M62F4   LD IX,0
        ADD IX,SP
        CALL RESTORE_STATUS
        INC IX
        LD SP,IX
        POP IX
        POP HL
        POP AF
        EI
        RET
```

NMI

Copy of the NMI handler in HOME ROM. Not used.

```
M6307   PUSH AF
        PUSH HL
        LD HL,(NMIADD)
        LD A,H
        OR L
        JR NZ,LNI3
        JP (HL)
```

LNI3

```
M6311   POP HL
        POP AF
        RETN
```

External Bank Communication

The bank communication routines below use memory location registers to get and push information to banks connected to the external bus. The routines contains portions to do standard bank switching, portions to access the expansion hardware and enough "smarts" to know when to do either.

Brzozowski theorized that each expansion bank would have its own version of an SCLD with 20 or 28 pins, depending on function, that communicated with the computer. He suggested that by communicating data in nybbles, only four data lines were necessary.

Similarly, only 3 address lines (A13-A15) are needed to decode the registers.

BS_MAX_BANK

```
M6315   DEFB $FF        Copy of MAX_BANK system variable. Allows
                        access to this parameter, even when HOME bank
                        does not control chunk 2.
```

GET WORD Subroutine

Performs LD HL, (HL) where HL is an address in bank B. Returns in HL the word from the address in HL in the bank specified in B.

GET_WORD

```
M6316   PUSH AF         Save registers.
        PUSH BC
        PUSH DE
        CALL GET_NUMBER Get bank number of owner of address.
        PUSH AF
        LD D,B
        LD B,A
        CALL GET_STATUS Get status of owner.
        PUSH BC
        CALL GET_CHUNK  Set Horizontal Select for getting at address
        CPL             and put in active low format.
        LD B,D
        LD C,A
        CALL BANK_ENABLE Enable address.
        LD E,(HL)       Read the word.
        INC HL
        LD D,(HL)
        DEC HL
        EX DE,HL
        POP BC
        POP AF
        LD B,A
        CALL BANK_ENABLE Re-enable owner of address.
        POP DE          Restore registers.
        POP BC
        POP AF
        RET
```

PUT WORD Subroutine

Writes the word in DE to the address in HL in the bank specified in B. Equivalent to LD (HL), DE where (HL) is an address in bank B.

This routine contains bugs. The technical manual gives adequate corrections which will relocate properly.

```
PUT_WORD
M633B   PUSH AF                 Save registers.
        PUSH BC
        CALL GET_NUMBER         Get bank number of owner of address.
        PUSH AF
        LD D,B
        LD B,A
        CALL GET_STATUS         Get status of owner.
        PUSH BC
        CALL GET_CHUNK          Set Horizontal Select for getting at address
        CPL                     and put in active low format.
        LD B,D
        LD C,A
        CALL BANK_ENABLE        Enable address
        LD (HL),E               Write the word.
        INC HL
        LD (HL),D
        DEC HL
        POP BC
        POP AF
        CALL BANK_ENABLE        Re-enable owner of address.
        POP BC
        POP AF
        RET
```

WRITE Bank Status Register Subroutine

This routine was intended to write data to the bus expansion unit (BEU) via bank status registers. D contains the register address; data is in E. Data is shifted out to the BEU via the external expansion bus with the IOA5 line low (active).

```
WRITE_BS_REG
M635C   PUSH AF                 Save these registers.
        PUSH BC
        PUSH HL
        LD H,D                  Put the register address in H.
        LD L,0                  Set L to zero. HL holds memory mapped bank
                                register/address.
```

Before using memory locations as registers, preserve the memory contents at those locations.

```
        LD A,(LOWNYB)           Get the value at LOWNYB ($C000).
```

```
PUSH AF                 Save it to the stack.
LD A,(HL)               Put value at HL in A.
PUSH AF                 Save it to the stack.
```

Save the contents of the sound chip registers that will be altered.

```
LD A,7                  Save value of AY-3-8910 register 7.
OUT (SADDPT),A
IN A,(SDATPT)
LD B,A                  Store the value in B.
LD A,$0E                Save value of AY-3-8910 register 14.
OUT (SADDPT),A
IN A,(SDATPT)
LD C,A                  Store the value in C.
```

Set IOA channel to output (to use IOA5 on expansion bus).

```
LD A,7                  Register 7, AY-3-8910
OUT (SADDPT),A
LD A,$40
OUT (SDATPT),A
LD A,$0E
OUT (SADDPT),A
XOR A                   Send 00 to the AY-3-8910 output port and
OUT (SDATPT),A          drive IOA5 low on the rear edge connector.
LD A,2                  Load 2 in A to
LD (LOWNYB),A           reset nybble steering logic.
LD A,E                  Put data in E, then
LD (HL),A               write least significant nybble of data to
SRA A                   the register address.
SRA A
SRA A
SRA A
LD (HL),A               Write the most significant nibble of the
LD A,7                  data. Restore AY-3-8910 registers to their
OUT (SADDPT),A          original values.
LD A,B
OUT (SDATPT),A
LD A,$0E
OUT (SADDPT),A
LD A,C
OUT (SDATPT),A
POP AF                  Restore the memory data back to the
LD (HL),A               locations used for registers.
POP AF
LD (LOWNYB),A
POP HL                  Restore registers.
POP BC
POP AF
```

```
        RET
```

READ Bank Status Register Subroutine

Reads nybbles from the bank switching registers in D (least significant nybble and E (most significant nybble), then packs both in to E.
LSN_ADDR: D, MSN_ADDR: E, BYTE_DATA: C

```
READ_BS_REG
M63AD   PUSH AF             Save registers.
        PUSH BC
        PUSH HL
        LD H,D              Put the register address in H.
        LD L,0              Set L to zero. HL holds memory mapped bank
                            register/address.
        LD A,(LOWNYB)       Save the data in the memory locations
        PUSH AF             that will be used as registers.
        LD A,(HL)
        PUSH AF
        LD A,7              Save the contents of the sound chip
        OUT (SADDPT),A      registers that will be altered
        IN A,(SDATPT)       (registers $07 and $0E).
        LD B,A
        LD A,$0E
        OUT (SADDPT),A
        IN A,(SDATPT)
        LD C,A
        PUSH BC             Push the values to the stack.
        LD A,7              Set IOA channel to output.
        OUT (SADDPT),A
        LD A,$40
        OUT (SDATPT),A
        LD A,$0E
        OUT (SADDPT),A
        XOR A
        OUT (SDATPT),A
        LD A,2              Load 2 in A to
        LD (LOWNYB),A       reset nybble steering logic.
        LD A,(HL)           Read least significant nybble of register D.
        AND $0F
        LD C,A              Move to C.
        LD H,E
        LD A,(HL)           Read most significant nybble of register E.
        SLA A
        SLA A
        SLA A
        SLA A
        OR C                Combine A and C.
        LD E,A              Return byte data in E
        POP BC              Get saved AY-3-8910 data and
```

```
        LD A,7              restore sound registers.
        OUT (SADDPT),A
        LD A,B
        OUT (SDATPT),A
        LD A,$0E
        OUT (SADDPT),A
        LD A,C
        OUT (SDATPT),A
        POP AF              Restore the memory data back to the
        LD (HL),A           locations used for registers.
        POP AF
        LD (LOWNYB),A
        POP HL              Restore registers.
        POP BC
        POP AF
        RET
```

Get Bank Status Register Subroutine

Returns current memory selection (horizontal select byte - low active) in C for the bank specified in B. Preserves bank number in B for HOME, EXROM or DOCK. This routine is meant to be called once for each bank. The HOME bank will appear to "own" chunks allocated to expansion banks and this will be misleading without the information on the expansion banks.

GET_STATUS

```
M6405   PUSH AF             Save registers.
        PUSH DE
        LD A,B              Move bank number to A.
        CP $FE              Is it EXROM?
        JR Z,GS_EXT         Yes, jump ahead.
        CP $FF              Is it HOME bank?
        JR Z,GS_HOME        Yes, jump ahead.
        AND A               Reset flags.
        JR Z,GS_DOCK        Jump ahead for DOCK bank.
        LD D,BNA            Send bank number to register $80 (BNA).
        LD E,B
        CALL WRITE_BS_REG
        LD D,HS_LSN         Read the $40/$80 register pair (HS).
        LD E,HS_MSN
        CALL READ_BS_REG
        LD A,E              Make the Horizontal Select
        CPL                 to low=true
        LD C,A              and put it in C.
        LD D,STA_L          Read the $A0/$C0 register
        LD E,STA_O          pair (bank status) and
        CALL READ_BS_REG
        LD B,E              put in B.
        JR GS_EXIT
```

```
GS_HOME
M642D   LD BC,0          Return 0 for HOME bank status.
        JR GS_EXIT

GS_DOCK
M6432   IN A,(HSR)       Return DOCK bank status.
        CPL
        LD B,A
        LD C,0
        JR GS_EXIT

GS_EXT
M643A   IN A,(DECR)      Get the value from display enhancement register.
        AND $80          Clear all bits but bit 7.
        CPL
        RLCA             Put active bit in bit zero
        LD B,A           Put it in B.
        IN A,(HSR)       Return DOCK bank status.
        CPL              Complement it.
        AND 1            Test bit 1.
        OR B             And see how it compares to B.
        LD B,A           Put the intersection in B.
        LD C,0

GS_EXIT
M644A   POP DE           Restore registers and return.
        POP AF
        RET
```

GET_CHUNK Subroutine

Computes the chunk for a given address. Returns a single byte mask in A with all bits 0 except for the one corresponding to the chunk for the address in HL.

```
GET_CHUNK
M6442   PUSH BC          Save B.
        LD A,H           Chunk number=high 3 bits of
        LD B,5           H so shift right 3 bits.

GC_SHIFT
M6451   SRL A
        DJNZ GC_SHIFT
        INC A            Create mask by rolling A 1 left.
        LD B,A           Chunk number+1 times
        XOR A            the flag comes from the carry.
        SCF
```

```
GC_ROLL
M6459   RLA
        DJNZ GC_ROLL
        POP BC                  Restore B.
        RET
```

GET_BANK_NUMBER Subroutine

Returns the bank number currently controlling the address in HL in A. Handles shortcomings in GET_STATUS by checking expansion banks first.

Contains a bug, per the technical manual. Should not be used if there's a chance EXROM is in use.

```
GET_NUMBER
M645E   PUSH BC                 Save registers.
        PUSH DE
        CALL GET_CHUNK
        LD C,A
        LD A,(BS_MAX_BANK)      Get largest bank number.
        AND A
        JR Z,GN_RD_DOCK         Jump if no exp banks.
        LD B,A

GN_CHECK
M646B   LD E,B                  Search all exp banks.
        CALL GET_STATUS
        AND C
        JR Z, GN_EXP            Found the chunk, so exit the loop.
        DJNZ GN_CHECK

GN_RD_DOCK
M6474   IN A,(HSR)              Return DOCK bank status.
        CPL
        AND C
        JR Z,GN_DOCK
        DEC C                   If chunk>1, then can't be in ext.
        JR NZ,GN_HOME
        IN A,(DECR)             Check if in ext bank.
        AND $80
        LD D,A
        IN A,(HSR)
        AND 1
        RRCA
        AND D
        JR Z,GN_HOME            Not in ext bank.
        LD A,$FE                In ext bank, return 254.
        JR GN_EXIT
```

```
GN_HOME
M648E   LD A,$FF                In HOME bank, return 255.
        JR GN_EXIT

GN_DOCK
M6492   XOR A                   Found in DOCK, return 0.
        JR GN_EXIT

GN_EXP
M6495   LD A,B                  Return exp bank number.

GN_EXIT
M6496   POP DE                  Restore registers.
        POP BC
        RET
```

BANK_ENABLE Subroutine

Enables the memory selected (horizontal select byte, low active) in the specified bank. Bank number in B, horizontal select byte in C.

Contains bugs, per technical manual. Manual fix is adequate. Better fix is changing the byte at M649A to F3 and byte at M651B to FB.

Errors in EXROM relocation table prevent this routine from relocating to high memory properly.

```
BANK_ENABLE
M6499   PUSH AF                 Save all the registers to the stack.
        PUSH BC
        PUSH DE
        PUSH HL
        LD H,B                  Put the bank number in H.
        LD A,(BS_MAX_BANK)      Get the largest bank number available.
        AND A                   Clear the flags.
        JR Z, BE_SKIP           Jump ahead if there are no expansion banks.
        LD D, BNA               Send 0 to Bank Number Access register.
        LD E, $0                Since the lowest expansion bank number
        CALL WRITE_BS_REG       is $01, make sure no expansion banks access.
        LD D, HSP               Send inverted Horizontal Select byte to
        PUSH AF                 Assigned Bank Number register.
        LD A,C                  Deselects the chunks that are about
        CPL                     to be allocated.
        LD E, A
        POP AF
        CALL WRITE_BS_REG
```

BE_SKIP

```
M64B5   LD A,B              Compare bank number DOCK bank ($00).
        AND A               Is it the DOCK Bank?
        JR NZ,BE_NTDOCK     If no, jump ahead.
        LD A,C              Put the memory select address in A.
        CP $FF              Are we allocating any chunks?
        JR Z, BE_EXT_OK     No, jump ahead.
        IN A,(DECR)         Get current value of display enhancement
        RES 7,A             register. Enable DOCK bank.
        OUT (DECR),A        Set it.
```

BE_EXT_OK

```
M64C4   LD A,C              Send inverted Horizontal Select byte
        CPL                 to enable the DOCK.
        OUT (HSR),A
        JR BE_EXIT
```

BE_NTDOCK

```
M64CA   LD A,B              Check if bank number is
        CP $FE              EXROM bank.
        JR NZ,BE_NTEXT      No? Jump ahead.
        IN A,(DECR)         Get the display enhancement register
        RLA                 value. Check to see if chuck 0
        RR C                is assigned in the bank.
        CCF
        RRA
        OUT (DECR),A
        BIT 7, A            If chunk 0 is assigned to the bank,
        JR NZ,BE_SET        then jump ahead to assign EXROM.
        IN A,(HSR)          Otherwise, assign chunk 0 to
        RES 0,A             the HOME dock.
        OUT (HSR),A
        JR BE_EXIT
```

BE_SET

```
M64E4   IN A,(HSR)          Assign chunk 0 to the EXROM bank.
        SET 0,A
        OUT (HSR),A
        JR BE_EXIT
```

BE_NTEXT

```
M64ED   IN A,(HSR)          Combine the Horizontal Select byte
        CPL                 with the present contents of port $F4.
        LD E,A              All chunks specified in C will be added
        LD A,C              to those controlled by the HOME bank,
        CPL                 unless overridden by the expansion
        OR E                bank.
        CPL
        OUT (HSR),A
        BIT 0,C             Is chunk 0 being reassigned?
```

```
        JR NZ,BE_CHK_HOME       No, jump ahead.
        IN A,(DECR)             Assign port 0 to HOME bank.
        RES 7,A
        OUT (DECR),A
        IN A,(HSR)
        RES 0,A                 Give port $F4 to the DOCK bank.
        OUT (HSR),A

BE_CHK_HOME
M6506   LD A,B                  Check if  the HOME bank is being
        CP $FF                  assigned.
        JR Z,BE_EXIT            Yes, done.
        LD D,BNA                Send the bank number
        LD E,B                  to the BNA register.
        CALL WRITE_BS_REG
        LD D,HS                 Invert the C register to
        LD A,C                  make the bits high
        CPL                     and send to Horizontal Select
        LD E,A                  register.
        CALL WRITE_BS_REG

BE_EXIT
M6516   POP HL                  Restore registers and return.
        POP DE
        POP BC
        POP AF
        RET
```

Save Bank Status Subroutine

Used internally to save bank information before making temporary Horizontal Select changes. Pushes the status of all banks onto a stack pointed to by IX.

```
SAVE_STATUS
M651E   PUSH AF                 Save registers.
        PUSH BC
        PUSH DE
        IN A,(DECR)             Get the status of EXROM/DOCK.
        NOP
        NOP
        LD (IX),A               Move status to IX location.
        INC IX                  Increment to next location.
        IN A,(HSR)              Get value of Horizontal Select register.
        LD (IX),A               Move HSR value to IX location.
        INC IX                  Increment to next location.
        LD A,(BS_MAX_BANK)      Get maximum number of banks.
        AND A                   Clear flags.
        JR Z,SS_EXIT            If BS_MAX_BANK is 0, exit.
        LD B,A                  Set up bank counter.
```

```
SS_LOOP
M6538   LD E,B              Move bank number to E.
        CALL GET_STATUS     Get status of bank number B.
        LD (IX),C           Move C to IX location.
        INC IX              Increment to next location.
        LD B,E              Put E in B.
        DJNZ SS_LOOP        Loop until B = 0.

SS_EXIT
M6544   DEC IX              Back IX up one location.
        POP DE              Restore these registers.
        POP BC
        POP AF
        RET
```

Restore Bank Status Subroutine

Used internally to put all banks back as they were before SAVE_STATUS was called. Restores the status of all banks from a stack pointed to by IX.

This will undo any video mode changes made after calling SAVE_STATUS as well as other port $FF control bits. Contains a bug; fix is in technical manual.

```
RESTORE_STATUS
M654A   PUSH AF             Save these registers for later.
        PUSH BC
        PUSH DE
        LD A,(IX)           Get EXROM status from location IX.
        OUT (DECR),A        Write it to the display enhancement register.
        INC IX              Increment to next location.
        LD A,(IX)           Get dock bank status from location IX.
        OUT (HSR),A         Write it to the Horizontal Select register.
        INC IX              Increment to next location.
        LD A,(BS_MAX_BANK)  Get maximum number of banks.
        AND A               Clear flags.
        JR Z, RS_EXIT       If zero, exit.
        LD B,A              Set up bank counter.

RS_LOOP
M6562   LD C,(IX)           Get status of bank from location IX.
        CALL BANK_ENABLE    Write bank status of bank number B.
        INC IX              Increment to next location.
        DJNZ RS_LOOP        Loop until B = 0.

RS_EXIT
M656C   DEC IX              Back IX up one location.
        POP DE              Restore these registers.
        POP BC
        POP AF
        RET
```

GOTO_BANK Routine

Transfers control to the specified address after enabling the memory selected in the specified bank. Parameters passed on stack by pushing target address, then bank number and memory select prior to calling. Return address is discarded.

GOTO_BANK

```
M6572   LD IX,0              Set IX to the stack pointer. This is done to
        ADD IX,SP            get access to items on the stack without popping
                             them off the stack (preserving the stack).
        LD (IX+0),C          Save BC and trash caller's return address.
        LD (IX+1),B
        LD C,(IX+2)          Set parameters for BANK_ENABLE.
        LD B,(IX+3)
        CALL BANK_ENABLE
        POP BC               Restore BC.
        POP IX               Trash parameters to GOTO_BANK.
        POP IX               Get jump address.
```

JMPIX

```
M658C   JP (IX)              Go to that address.
```

Bank Switch Stack

Used to exchange values and parameters when switching between HOME and EXROM. Each time CALL_BANK is run, the return and PRM_IN go here.

BS_STACK

```
M658E   DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
        DEFB $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
```

BS_SP

Bank switch stack pointer.

```
M65CE   DEFW $FFFF
```

CALL_BANK

Like GOTO_BANK except saves current bank status, calls target address, and restores status prior to returning to user. Two additional parameters are passed on stack prior to doing call out to CALL_BANK. These are PRM_OUT (16 bits) followed by PRM_IN (16 bits), as described for the Function Dispatcher.

Clobbers IX.

CALL_BANK

```
M65D0   EX (SP),HL          Get return address (the last value pushed to the
                            stack); put HL in its place on the stack.
        LD IX,(BS_SP)       Put value of the bank switching stack pointer in
                            IX.
        DEC IX              Subtract one (to get to the bottom value of the
                            bank switch stack).
        LD (IX+0),H         Push return address (value in HL) to
        DEC IX              BS_STACK.
        LD (IX+0),L
        POP HL              Restore HL from the stack.
        EX (SP),HL          Get PRM_IN; push HL to the stack.
        DEC IX              Push PRM_IN (value in HL) to BS_STACK.
        LD (IX+0),H
        DEC IX
        LD (IX+0),L
        LD (BS_SP),IX       Update BS_SP with the IX, pointer to current
                            location in BS_STACK.
        PUSH DE             Save these registers.
        PUSH BC
        PUSH AF
        LD HL,0             Zero HL.
        ADD HL,SP           Set HL to the stack pointer.
        LD D,H              Copy stack pointer to DE.
        LD E,L
        LD A, (BS_MAX_BANK) Get the maximum number of banks.
        LD C,A              Save that number to C.
        LD B,0              Zero B.
        INC BC
        INC BC              BC = BS_MAX_BANK+2
        AND A               Clear carry flag.
        SBC HL,BC           Subtract BC+CF from HL, store result in HL.
                            If max bank number is 0, HL = SP.
        LD SP,HL            Move HL to stack pointer.
        LD IX,0             Reset IX.
        ADD IX,DE           Put DE in IX.
        EX DE,HL            DE, HL now contain destination and src pointers
                            for a block move.
        LD C,(IX+PRM_OUT)   Put PRM_OUT in BC.
        LD B,(IX+PRM_OUT)   This should have been IX+PRM_OUT+1.
        LD A, 0E            14 spaces.
        ADD A,C             Add PRM_OUT low byte.
        LD C,A              Put that in C.
        JR NC, CB_NC1       BC = PRM_OUT + 14. If A+C < 256, jump ahead
                            to CB_NC1.
        INC B               A+C > 255, 256 to PRM_OUT.
```

CB_NC1

```
M6618   LDIR                    Make room for bank status.
        PUSH DE                 Move value in DE (base of the status stack)
        POP IX                  to IX.
        CALL SAVE_STATUS        Save the status of all banks.
        LD IX, 0                Reset IX.
        ADD IX,SP               Put stack pointer in IX.
        LD C,(IX+10)            Get Horizontal Select byte and
        LD B,(IX+11)            bank number for BANK_ENABLE.
        CALL BANK_ENABLE        Enable destination bank.
        POP AF                  Restore these registers.
        POP BC
        POP DE
        POP HL
        POP IX                  Discard params CALL_BANK and get
        POP IX                  address.
        POP IX
        CALL JMPIX
        PUSH AF
        PUSH BC
        PUSH DE
        PUSH HL
        LD IX,(BS_SP)
        LD C,(IX+0)
        INC IX
        LD B,(IX+0)
        INC IX                  Pop PRM_IN off BS_STACK.
        LD (BS_SP),IX           Update BS_SP.
        LD IX,0
        ADD IX,SP
        LD A,8
        ADD A,C
        LD C,A
        JR NC,CB_NC2            BC = PRM_IN + 8
        INC B
```

CB_NC2

```
M665F   ADD IX,BC               IX = SP + PRM_IN +9
        PUSH IX
        POP HL                  HL=IX
        DEC HL                  Set HL to source pointer for block move.
        CALL RESTORE_STATUS
        PUSH IX
        POP DE                  Set DE to destination pointer for block
        LDDR                    move and deallocate space for bank
        EX DE,HL                status.
        INC HL
        LD SP,HL                Restore stack pointer.
        LD IX,(BS_SP)
```

```
        LD C,(IX+0)
        INC IX
        LD B,(IX+0)
        INC IX                  POP return address of BS_STACK.
        LD (BS_SP),IX           Update BS_SP.
        PUSH BC
        POP IX
        POP HL
        POP DE                  Restore registers.
        POP BC
        POP AF
        PUSH IX
        RET
```

MOVE_BYTES Routine

Used only as a subroutine to XFER_BYTES, and is intended to transfer bytes between banks when source and destination chunks overlap and the transfer is between two different banks. Bytes to move is in DE, direction is in A.

Contains numerous bugs that are not documented in the technical manual. Due to the programmer's misunderstanding of the subtleties of LDIR and LDDR, and the differences in their usage, some counters are not properly updated, and some intermediate transfers can be made to the wrong part of the stack, destroying critical information. Major bugs can be tolerated by putting $73 at M66DF, and $72 at M66E2. This routine can only be used in the LDIR mode; fortunately, the LDDR case is not needed where this is used.[5]

MOVE_BYTES

```
M668C   PUSH HL
        PUSH DE
        PUSH BC
        LD C,B
        LD B,(IX+SRC_BANK)
        CALL BANK_ENABLE
        LD B,D
        LD C,E
        LD E,(IX+BUF_PTR)
        LD D,(IX+BUF_PTR+1)
        LD L,(IX+SRC_ADDR)
        LD H,(IX+SRC_ADDR+1)
        RLCA
        RRCA
        JR C,MB_DOWN1
        LDIR
        ADD HL,BC
        JR MB_UP1
```

[5] "Mystery of the Missing 253" series.

MB_DOWN1
```
M66AD   LDDR
        AND A
        SBC HL,BC
```

MB_UP1
```
M66B2   LD (IX+SRC_ADDR),L
        LD (IX+SRC_ADDR+1),H
        POP BC
        POP HL
        PUSH HL
        PUSH BC
        LD B,(IX+DEST_BANK)
        CALL BANK_ENABLE
        LD B,H
        LD C,L
        LD E,(IX+DEST_ADDR)
        LD D,(IX+DEST_ADDR+1)
        LD L,(IX+BUF_PTR)
        LD H,(IX+BUF_PTR+1)
        RLCA
        RRCA
        JR C,MB_DOWN2
        LDIR
        ADD HL,BC
        JR MB_UP2
```

MB_DOWN2
```
M669D9  LDDR
        AND A
        SBC HL,BC
```

MB_UP2
```
M66DE   LD (IX+DEST_ADDR),L
        LD (IX+DEST_ADDR+1),H
        POP BC
        POP DE
        POP HL
        RET
```

CREATE_BITMAP Subroutine

Used only as a subroutine to XFER_BYTES, and is intended to produce a low-true "Horizontal Select" byte for all the chunks involved in either the source or destination bank for a data transfer. Address is in HL, returns with bitmap is in A.

Contains undocumented bugs. Due to improper computation of first and/or last bytes in a data transfer, this may give an improper result, when the error in computation straddles a chunk boundary. These can be corrected by inserting at 66F3 and following: $0B, $3C, $2B.

```
CREATE_BITMAP
M66E8   LD D,H              Save start address.
        LD E,L
        LD C,(IX+LENGTH)
        LD B,(IX+LENGTH+1)
        LD A,(IX+DIRECTION)
        RLCA                Calculate end address.
        RRCA
        JR C,CB_SUB         If A<0
        ADD HL,BC
        JR CB_CONT

CB_SUB
M66FA   SBC HL,BC

CB_CONT
M66FC   CALL GET_CHUNK      Get end chunk bit.
        CPL
        LD B,A
        EX DE,HL
        CALL GET_CHUNK      Get start chunk bit.
        CPL
        LD C,A
        XOR B
        JR Z,CB_EXIT        Go here if start and end chunks are not the
        LD A,C              same.
        LD B,A              Put start and end bits together and fill
        LD C,0              between them with zeros.
        SCF

CB_NB1
M6710   LD A,B              Test next bit.
        RL C
        AND C               Jump if we have not found
        JR NZ,CB_NB1        first zero.

CB_NB2
M6716   LD A,B              Test next bit.
        RL C
        AND C
        JR Z,CB_EXIT        Found the last zero.
        XOR B               Otherwise, update bitmap.
        LD B,A
        JR CB_NB2

CB_EXIT
M6720   LD A,B
        RET                 Return bitmap.
```

XFER_BYTES Routine

Intelligent transfer routine to move data between banks, but also intended to allow transfer within a single bank, whether or not all the necessary chunks are enabled.

Copies n bytes from specified source to specified destination in either ascending or descending order. Source and destination can be in the same or different banks and can be in shadowing chunk, but neither source nor destination can pass any "chunk" (8K) boundary since only the chunks containing the starting source and destination addresses are explicitly enable.

Parameters passed on stack by pushing:
- Source Bank/Destination Bank
- Source Address
- Destination Address
- Length (number of bytes)
- Direction (0 ascending/-1 descending)

Contrary to the description in the technical manual, this routine was intended to be able to do transfers between larger memory areas than just a single source chunk and a single destination chunk. The limitation was probably intended to mask one of the problems in this routine. Also, this routine does not relocate properly to high memory.

This routine requires that the machine stack be in its proper location in the RAM-resident code. It was not intended to be able to transfer data into or out of the chunk that currently contains the RAM-resident code. If the stack is nearly full, the transfer will be aborted, without notifying the CALLing routine. A status flag, intended to perform such a warning, is corrupted before completion.

The technical manual documents only one bug, but several changes are needed to get this working properly. Location M6722 gets $00, M672B gets $0B, M676A gets $5F, M67C2 gets $2C, and M67FF gets $00. Because this code would normally be used to initialize expansion banks, it's possible that this routine would have to be fixed before they could be debugged.

With these fixes, AF is no longer preserved, and A is now returned with a status code. It will contain 00 if the transfer was successful, and 01 if it was aborted due to insufficient stack space.

Because of the bug in MOVE_BYTES, using this routine in the LDDR mode can crash the system in some cases. The only time the LDDR mode may really be needed would be certain times when the source and destination areas overlap within the same bank. This case does not cause the problem, so if the use of the LDDR mode is limited to this case, there will be no trouble.

XFER_BYTES

```
M6722   PUSH AF                 Save registers.
        PUSH BC
        PUSH DE
        PUSH HL
        LD HL,0
        ADD HL,SP
        LD DE,10
        ADD HL,DE
        EX DE,HL                DE points to start of params.
        LD A,(BS_MAX_BANK)
        LD C,A
        LD B,0
        LD HL,0
        ADD HL,SP
        AND A
        SBC HL,BC
        DEC HL
        DEC HL                  HL=SP-MAX_BANK-2
        PUSH HL
        POP IX                  IX points to location to save status.
        LD SP,IX
        CALL SAVE_STATUS        Save bank's status.
        PUSH DE
        POP IX                  IX points to params.
        LD L,(IX+SRC_ADDR)
        LD H,(IX+SRC_ADDR+1)
        CALL CREATE_BITMAP      Get source bitmap.
        PUSH AF                 Save on stack temporarily.
        LD L,(IX+DEST_ADDR)
        LD H,(IX+DEST_ADDR+1)
        CALL CREATE_BITMAP      Get destination bitmap.
        LD C,A                  Set C to destination bitmap.
        POP AF
        LD B,A                  Set C to source bitmap.
        LD A,(IX+SRC_BANK)
        LD D,(IX+DEST_BANK)
        CP D                    Compare src and dest bank numbers.
        JR NZ,XB_DIFF_BANKS     Jump if bank numbers diff.
        LD A,B
        AND C
        LD B,A                  B= union OR SRC and DEST bitmaps.
        JR XB_DO_MOVE
```

XB_DIFF_BANKS

```
M676D   LD A,B              Check for overlap between src and
        OR C                dest chunks.
        CP $FF
        JR NZ,XB_OVERLAP    Jump if overlap.
        LD E,B
        LD B,D
        CALL BANK_ENABLE    Select destination bank.
```

XB_DO_MOVE

```
M6778   LD B,(IX+SRC_BANK)
        LD C,E
        CALL BANK_ENABLE    Select source bank.
        LD L,(IX+SRC_ADDR)
        LD H,(IX+SRC_ADDR+1)
        LD E,(IX+DEST_ADDR)
        LD D,(IX+DEST_ADDR+1)
        LD C,(IX+LENGTH)
        LD B,(IX+LENGTH+1)
        LD A,(IX+DIRECTION)
        RLCA
        RRCA
        JR C,XB_REVERSE     IF A<0
        LDIR                Move low address first.
        JR XB_EXIT
```

XB_REVERSE

```
M779C   LDDR                Move high address first.
        JR XB_EXIT
```

XB_OVERLAP

```
M67A0   LD HL,MSTBOT
        PUSH BC
        LD B,255
        CALL GET_WORD
        POP BC
        LD DE,STKSZ
        AND A
        SBC HL,DE           HL=address of stack limi
        LD DE, $0020        32 bytes.
        ADD HL,DE
        EX DE,HL            DE=SP_NEW
        LD HL,0
        ADD HL,SP           HL=SP_OLD
        INC DE              Compare SP_OLD and SP_NEW
        AND A
        SBC HL,DE
        JR NC,XB_SPACE      If SP_OLD-SP_NEW > 0
        LD A,1              Return error code.
```

```
        JR XB_EXIT

XB_SPACE
M67C3   DEC DE
        EX DE,HL
        LD SP,HL                Set SP TO SP_NEW
        INC DE                  DE=BUF_SZ
        LD A,(IX+DIRECTION)
        LD (IX+BUF_PTR),L
        LD (IX+BUF_PTR+1),H
        LD L,(IX+LENGTH)
        LD H,(IX+LENGTH+1)

XB_MOVE_LOOP
M67D6   AND A                   HL = bytes left to move.
        SBC HL,DE               DE = bytes to move this time.
        JR C,XB_LAST_MOVE       If less than BUF_SZ BYTES left
        CALL $5200+MOVE_BYTES
        JR XB_MOVE_LOOP

XB_LAST_MOVE
M67E5   ADD HL,DE
        EX DE,HL
        CALL $5200+MOVE_BYTES
        EX DE,HL
        LD L,(IX+BUF_PTR)
        LD H,(IX+BUF_PTR+1)
        ADD HL,DE               HL = BUF_PTR+BUF_SZ
        LD SP,HL                Restore stack pointer.

XB_EXIT
M67EE   XOR A                   Return code for successful completion.
        LD IX,0
        ADD IX,SP
        CALL RESTORE_STATUS     Restore state and return zero code.
        INC IX
        LD SP,IX
        POP HL                  Restore registers.
        POP DE
        POP BC
        POP AF
        POP IX                  Clean up PARAMS.
        EX (SP),IX
        POP IX
        EX (SP),IX
        POP IX
        EX (SP),IX
        POP IX
        EX (SP),IX
        POP IX
```

```
        EX (SP),IX
        RET
```

GOTO_EXT_INIT

Intended for use only during initialization. Does an effective JP (HL) to the EXROM. Would not work properly if expansion banks were enabled in chunk 0, hence its suitability only for system initialization. Address in HL.

```
GOTO_EXT
M6815   POP IX              Discard return address.
        PUSH AF
        IN A,(DECR)         Get the value of the display enhancement
        SET 7,A             register.
        OUT (DECR),A
        LD A,1
        OUT (HSR),A
        POP AF
        JP (HL)
```

Blank space

BLK1	$1624 to $17FF	$00
BLK2	$1800 to $1BFF	$FF
BLK3	$1C00 to $1CFF	$00

FIXTBL

Fix-up table for video mode changer. This table defines the locations in RAM which must be updated when the dispatcher is moved from chunk 3 to chunk 7 or vice-versa. Addresses in the table are defined as chunk 3 addresses.

This table contains errors that prevent the code from being properly corrected after its moved.

DISPATCH	$6200
X38INT	$62AE
GET_WORD	$6316
PUT_WORD	$633B
GET_STATUS	$6405
GET_NUMBER	$645E
BANK_ENABLE	$6499
SAVE_STATUS	$651E
RESTORE_STATUS	$654A
BS_STACK	$658E
BS_SP	$65CE
GOTO_BANK	$6572
CALL_BANK	$65D0
MOVE_BYTES	$668C
CREATE_BITMAP	$66E8

	XFER_BYTES	$6722	
M1D00	DISPATCH+$32	$6232	
	DISPATCH+$4D	$624D	
	DISPATCH+$72	$6272	
	DISPATCH+$AB	$62AB	
	INT+$0A	$62B8	
	INT+$1F	$62CD	
	INT+$25	$62D3	
	INT+$2E	$62DC	
	INT+$4D	$62FB	
	GET_WORD+$04	$631A	
	GET_WORD+$0A	$6320	
	GET_WORD+$0E	$6324	
	GET_WORD+$14	$632A	
	GET_WORD+$1F	$6335	
	PUT_WORD+$03	$633E	
	PUT_WORD+$09	$6344	
	PUT_WORD+$0D	$6348	
	PUT_WORD+$13	$634E	
	PUT_WORD+$1C	$6357	
	GET_STATUS+$12	$6417	
	GET_STATUS+$19	$641E	
	GET_STATUS+$23	$6428	
	GET_NUMBER+$03	$6461	
	GET_NUMBER+$07	$6465	
	GET_NUMBER+$0F	$646D	
	BANK_ENABLE+$06	$649F	
	BANK_ENABLE+$13	$64AC	
	BANK_ENABLE+$1A	$64B3	
	BANK_ENABLE+$75	$650E	Should be $650F.
	BANK_ENABLE+$7D	$6516	Should be $6517.
	SAVE_STATUS+$14	$6532	
	SAVE_STATUS+$1C	$653A	
	RESTORE_STATUS+$12	$655C	
	RESTORE_STATUS+$1C	$6566	
	BS_SP	$65CE	
	GOTO_BANK+$13	$6585	
	CALL_BANK+$03	$65D3	
	CALL_BANK+$1D	$65ED	
	CALL_BANK+$29	$65F9	
	CALL_BANK+$4E	$661E	
	CALL_BANK+$5D	$662D	
	CALL_BANK+$6A	$663A	
	CALL_BANK+$72	$6642	
	CALL_BANK+$80	$6650	
	CALL_BANK+$96	$6666	
	CALL_BANK+$A2	$6672	
	CALL_BANK+$B0	$6680	
	MOVE_BYTES+$08	$6694	

MOVE_BYTES+$34	$66C0
CREATE_BITMAP+$15	$66FD
CREATE_BITMAP+$1B	$6703
XFER_BYTES+$0E	$6730
XFER_BYTES+$2D	$674F
XFER_BYTES+$2E	$6750
XFER_BYTES+$38	$675A
XFER_BYTES+$54	$6776
XFER_BYTES+$5B	$677D
XFER_BYTES+$85	$67A7
XFER_BYTES+$BA	$67DC
XFER_BYTES+$C1	$67E3
XFER_BYTES+$D4	$67F6

DISPATCH

JMPTBL

```
M1EDC  DEFW $1795  PUT_LN
       DEFW $2813  DRAWLN
       DEFW $2624  F_PNT
       DEFW $0939  SCRL
       DEFW $08A6  K_CLS
       DEFW $073F  PUTMES
       DEFW $0566  P_NL
       DEFW $0554  P_RT
       DEFW $053A  P_LFT
       DEFW $02B0  K_SCAN
       DEFW $0010  WRCH
       DEFW $11ED  SENDCH
       DEFW $11CF  RDCH
       DEFW $3C6C  TO_THE
       DEFW $3C65  ROOT
       DEFW $3C5E  ACS
       DEFW $3C4E  ASN
       DEFW $3BFD  ATN
       DEFW $3BF5  TAN
       DEFW $3BD0  SIN
       DEFW $3BC5  COS
       DEFW $3B9E  ANGLE
       DEFW $3B2E  LN
       DEFW $3ADF  EXP
       DEFW $3ACA  INT
       DEFW $3ABB  INTDIV
       DEFW $3656  FLOAT
       DEFW $35D3  TRUNC
```

```
DEFW $356E  DIVIDE
DEFW $3489  TIMES
DEFW $3468  MULT
DEFW $33D3 ADD
DEFW $33CE  TSSUB
DEFW $31A1  OUTPUT
DEFW $3193  FP2A
DEFW $3160  FP2BC
DEFW $30F9  ININT
DEFW $30E9  STK_EC
DEFW $30E6  STK_A
DEFW $3059  STKUSN
DEFW $2FC0 DIM
DEFW $2FAF  POPSTR
DEFW $2EBD LET
DEFW $2E74  PAEDCB
DEFW $2E70  PSHSTR
DEFW $2C70 FIND_N
DEFW $29F2  F_INKY
DEFW $29E5  F_PI
DEFW $29B6 RND
DEFW $28D7 F_ATTR
DEFW $288E  F_SCRN
DEFW $2854  EXPRN
DEFW $2810  DRAW_L
DEFW $26DB DRAW
DEFW $2679  CIRCLE
DEFW $2660  GET_XY
DEFW $263E  PLOTBC
DEFW $2635  PLOT
DEFW $2603  SCRMBL
DEFW $241D HIFLSH
DEFW $23DE COLOR
DEFW $2380  NOTKBQ
DEFW $226B ISEQ
DEFW $222B INPUT
DEFW $217E  P_SEQ
DEFW $2159  K_PRIN
DEFW $2155  K_LPR
DEFW $201D DEF
DEFW $2009  BREAKQ
DEFW $1FEB  PAUSE
DEFW $1FD4  RETURN
DEFW $1FBB  CHK_SZ
DEFW $1F99  GO_SUB
DEFW $1F39  CLR_BC
DEFW $1F36  CLEAR
DEFW $1F23  FIX_U
DEFW $1F1E  FIX_Ul
```

```
DEFW $1EF1  JUMP
DEFW $1EE4  CONT
DEFW $1ED4  RAND
DEFW $1ECA  RESTBC
DEFW $1E82  DATA
DEFW $1D97  READ
DEFW $1D55  NEXT
DEFW $1C59  STOP
DEFW $1C78  FOR
DEFW $1AD8  EXCUTE
DEFW $1A27  SYNTAX
DEFW $1788  PUT_BC
DEFW $1750  DELREC
DEFW $1720  RECLEN
DEFW $16F0  SUBLIN
DEFW $16D6  FIND_L
DEFW $160D  FLASHA
DEFW $25D0  MOVE
DEFW $25CC  FORMAT
DEFW $25D4  ERASE
DEFW $25C8  CAT
DEFW $1465  OPCHAN
DEFW $142A  OPEN
DEFW $13BE  CLCHAN
DEFW $139F  CLOSE
DEFW $1354  RESET
DEFW $12BB  INSERT
DEFW $1230  SELECT
DEFW $11E1  INCH
DEFW $0D31  INIT
DEFW $0D1D  K_NEW
DEFW $0D0D  DESLUG
DEFW $0A4A  PRSCAN
DEFW $0A23  DUMPPR
DEFW $08EA  CLS
DEFW $08A9  CLLHS
DEFW $0888  R_ATTS
DEFW $0710  ATTBYT
DEFW $05B2  SET_AT
DEFW $0500  SEND_TV
DEFW $0A02  K_DUMP
DEFW $0436  BEEP
DEFW $03F3  PARP
DEFW $02E1  UD_K start of HOME ROM services
DEFW $FFFF  RESERVED24
DEFW $FFFF  RESERVED23
DEFW $FFFF  RESERVED22
DEFW $FFFF  RESERVED21
DEFW $FFFF  RESERVED20
```

```
DEFW $6721  XFER_BYTES?
DEFW $6722  RESERVED19
DEFW $65CF  CALL_BANK?
DEFW $65CF  RESERVED18
DEFW $6571  GOTO_BANK?
DEFW $6571  RESERVED17
DEFW $6499  BANK_ENABLE
            RESERVED16
DEFW $645E  GET_NUMBER GET_NUMBER
DEFW $6405  GET_STATUS GET_STATUS
DEFW $FFFF  RESERVED13
DEFW $FFFF  RESERVED12
DEFW $FFFF  RESERVED11
DEFW $FFFF  RESERVED10
DEFW $00E5  W_BORD
            RESERVED9
DEFW $0EA3  RESERVED8
DEFW $0851  SAVE
            RESERVED7
DEFW $06E5  RESERVED6
DEFW $05CC  LOAD
            RESERVED5
DEFW $01AB  SLVM
            RESERVED4
DEFW $018D  R_EDGE
            RESERVED3
DEFW $0189  RD_BIT
            RESERVED2
DEFW $00FC  RD_TAPE
            RESERVED1 EXROM services
DEFW $0068  W_TAPE
```

Appendices

Appendix A

System Variables

RAM beginning at 23552 ($5C00) is dedicated to the BASIC System Variables as defined in Appendix D of the TS 2068 User Manual and in Appendix B of the Technical Reference. The area from the end of the defined variables (STRMNM - $5CCB) to $5EE9H is reserved for expansion of the system variables, but is not used by the operating system.

All values are $0 by default unless otherwise noted.

KSTATE **$5C00**

Used to read the keyboard. Every time the Z80 receives an interrupt (60 times per second in 2068), one of the things the computer does is read the keyboard and save the result in these eight bytes. These bytes have different uses, and not all are useful for a programmer. To understand how it works, we can consider that we have two blocks: the upper one (4 bytes from 5C00/23552 to 5C03/23555) and the lower one (4 bytes from 5C04/23556 to 5C07/23559). Normally the lower one is used. The upper is used when, while holding a key, a new key is pressed without releasing the first one. The logic is as follows:

If there is no key pressed:

- (KS_A2) = 255
- (KS_C2) = 0

One key is pressed:

- (KS_A2) = key value in uppercase, for example 65 if A was pressed.
- (KS_C2) = 5
- (KS_D2) = countdown, varying over time, to automatically repeat the key.
- (KS_B2) = ASCII code of the key pressed.

If there is a single key pressed or none:

- (KS_A1) = 255
- (KS_C1) = 0

If there is more than one key pressed:

- (KS_A1) = key value in uppercase.
- (KS_C1) = 5
- (KS_D1) = countdown.
- (KS_B1) = ASCII code of the key pressed, taking into account SHIFT or SYMBOL.

KS_A1 **$5C00**

Bit 7=1 means no key stored

KS_C1 **$5C01**

Debounce count 0.

KS_D1 **$5C02**

Repeat count 0.

KS_B1 **$5C03**

Character 0.

KS_A2 **$5C04**

Bit 7=1 means no key stored.

KS_C2 **$5C05**

Debounce count 1.

KS_D2 **$5C06**

Repeat count 1.

KS_B2 **$5C07**

Character 1.

LASTK **$5C08**

Stores newly pressed key. If you are using the system ROM keyboard routines from an LROS or autostart machine code AROS and accessing the input character code via LASTK, you must initialize the TS 2068 to "L" mode by setting MODE ($5C41) to zero and bit 3 of FLAGS ($5CBH) to 1. See page 126 of the original technical manual.

REPDEL **$23** **$5C09**

Time, in 60ths of a second, that a key must be held before it repeats. Default is 35; can be set.

REPPER **$05** **$5C0A**

Key repeat delay. Delay, in 60ths of a second, between successive repeats of a key held down. Default is 5; can be set.

DEFADD **$5C0B**

Address of first argument letter of the user-defined function if one is being evaluated.

KDATA **$5C0D**

Store information of the last color change of the line being edited.

TVDATA **$5C0E**

Bytes of color, AT and TAB controls going to display.

STRMS **$5C10**

Addresses of channels attached to streams. This is a 38 byte table (2 bytes each for the 19 channels). The table is accessed using (channel # * 2) + $16) as an index added to $5C00. The 2-byte value in the table is an index in to the CHANS area of memory, which contains the address of the I/O routines for the selected channel. If the 2-byte value is zero, the channel is closed. This table is modified via the OPEN # and CLOSE # commands. When a channel is opened, the device specification is used to obtain the 2-byte value to be inserted. This value is taken from the STRMINIT table in the HOME ROM. When channels 0 - 3 are closed, the values are restored to those on power-up. All others are cleared to zero.

CH_M3 **$100** **$5C10**

Chan -3: Channel for keyboard and lower screen.

CH_M2	**$600**	**$5C12**
Chan -2: Main screen.		
CH_M1	**$B00**	**$5C14**
Chan -1: RAM write.		
CH_0	**$100**	**$5C16**
Chan 0: Output to lower screen.		
CH_1	**$100**	**$5C18**
Chan 1: INPUT command		
CH_2	**$600**	**$5C1A**
Chan 2: PRINT/LIST		
CH_3	**$1000**	**$5C1C**
Chan 3: LPRINT/LLIST		
CH_4		**$5C1E**
User chan 1 (4)		
CH_5		**$5C20**
User chan 2 (5)		
CH_6		**$5C22**
User chan 3 (6)		
CH_7		**$5C24**
User chan 4 (7)		
CH_8		**$5C26**
User chan 5 (8)		
CH_9		**$5C28**
User chan 6 (9)		
CH_10		**$5C2A**
User chan 7 (10)		
CH_11		**$5C2C**
User chan 8 (11)		
CH_12		**$5C2E**
User chan 9 (12)		
CH_13		**$5C30**
User chan 10 (13)		
CH_14		**$5C32**
User chan 11 (14)		
CH_15		**$5C34**
User chan 12 (15)		

CHARS **$3C00** **$5C36**

Start address of the characters definition byte, minus 256. 256 is subtracted because the first printable code (space) is number 32. 32*8 = 256.

It is possible to change the value of this variable to point to another address in RAM, where you define your own character set.

RASP **$40** **$5C38**

Length of warning buzz that sounds when the editing buffer is full.

PIP **$5C39**

Length of keyboard click.

ERR_NR **$FF** **$5C3A**

1 less than the report code. Error number (IY points here) Y =ERR_NR.

FLAGS **$05** **$5C3B**

Various flags to control BASIC.

SPC	0	Suppress space before tokens
PR	1	Print to printer, not TV
LMODE1	2	L mode, not K at current character
LMODE2	3	L mode, not K at cursor
TOKEN	4	Token mode: unique to TS 2068.[6]
KEYHIT	5	A key has been pressed
NUM	6	Expression is numerical, not string
INTPT	7	Interpret rather than check syntax

TVFLAG **$05** **$5C3C**

Flags associated with the TV.

LHS	0	Printing to lower half of screen
EDIT	1	Outputting line for edit or number for string
ECHREQ	3	Echo requested if inputting from keyboard
TVLIST	4	Outputting and automatic listing
CLHS	5	Clear lower half screen when key pressed

ERRSP **$61FC** **$5C3D**

Address of item on machine stack to be used as error return.

LISTSP **$61FC** **$5C3F**

Address of return address from automatic listing.

MODE **$5C41**

Specifies the K, L, C, E, or G cursor. 0 = K or L, 1 = F, 2 = G

NEWPPC **$17FC** **$5C42**

Line to be jumped to after GOTO/GOSUB or RUN.

[6] Timex used this flag, which is unused in the ZX Spectrum, as a shortcut to indicate whether this computer was processing tokens or characters.

NSPPC **$01** **$5C44**

Statement number in line to be jumped to. Poking first NEWPPC and then NSPPC forces a jump to a specified statement in a line. Bit 7 off forces jump.

PPC **$17FC** **$5C45**

Line number being interpreted.

SUBPPC **$01** **$5C47**

Number within line of statement being interpreted.

BORDCR **$38** **$5C48**

Border color*8. Also contains the attributes normally used for the lower half of the screen.

EPPC **$5C49**

Line number of current line in listing (with program cursor).

VARS **$6856** **$5C4B**

Starting address of definition of variables (last is 1 byte of $80).

DEST **$6856** **$5C4D**

Address of variable in assignment. Matched by template code 1 or 4 (text or record).

CHANS **$6840** **$5C4F**

Address of the 21-byte table initialized on power-up or execution of a NEW command. Supports stream I/O to the four standard system devices (K, S, R and P). Each table entry is 5 bytes long and is indexed by the value obtained from the STRMS table added to (CHANS)-1. Each entry has the following format: output routine address for WRCH (2 bytes), input routine address for WAIT-KEY (2 bytes), device type (1 byte). and where appropriate, a file name, additional data and a buffer.

CURCHL **$6845** **$5C51**

Address of information current being used for input and output. Pointer to data for current channel.

PROG **$6856** **$5C53**

Address where BASIC program starts in memory.

NXTLIN **$6857** **$5C55**

Address of next line in program.

DATADD **$6855** **$5C57**

Address of terminator of last data item.

ELINE **$6857** **$5C59**

Address of command being typed in (line being edited).

KCUR **$009D** **$5C5B**

Address of cursor (current character in input buffer).

CH_ADD **$9A25** **$5C5D**

Address of the next character to be interpreted. The character after the argument of a PEEK, or the NEWLINE at the end of a POKE statement. Current character when syntax checking, etc.

X_PTR **$00BB** **$5C5F**

Address of the character after the ? marker: 1st char not syntactically OK (0 if all OK). Also stores (CH_ADD) during read and input.

WORKSP **$6859** **$5C61**

Address of temporary workspace.

STKBOT **$6859** **$5C63**

Address of bottom of calculator stack.

STKEND **$6859** **$5C65**

Address of next free place on calculator stack.

STKNXT **STKEND**

Alias for STKEND.

BREG **$2B** **$5C67**

Calculator's B register.

MEM **$5C92** **$5C68**

Address of area used for calculator's memory. Usually MEMBOT but not always. If you are using the system ROM calculator routines or any ROM routines that use them from an LROS or autostart machine code AROS, you must initialize this variable. See page 126 of the original technical manual.

FLAGS2 **$08** **$5C6A**

More flags.

ALOS	0	Automatic listing on screen
PRLEFT	1	Printer buffer not empty
L_STR	2	Inside string when doing KB mode in LISTCH
CAPS_L	3	Capitals shift lock on
RETPOS	4	Retype possible after syntax error
DELREP	5	Delete key repeat (key held down)

DFSZ **$02** **$5C6B**

Number of lines (including one blank line) in lower part of screen.

STOP **$0000** **$5C6C**

Line number of the top program line in automatic listings.

OLDPPC **$105E** **$5C6E**

Line to which CONTINUE jumps.

OSPCC **$01** **$5C70**

Statement number within line for OLDPPC.

FLAGX **$5C71**

Flags associated with assignment

FLEX	0	Flexible length assignment required
UNFND	1	Destination of assignment not found
INPLN	5	Req input value rather than line of program
NO	6	Reqd type is numeric
LINPLN	7	Input (input line) rather than straight input

STRLEN **$C663** **$5C72**

Length of destination when string type.

TADDR **$197E** **$5C74**

Address of next item in syntax table.

SEED **$FCFE** **$5C76**

Seed for RND. Set with RANDOMIZE.

FRAMES **$5C78**

Least significant byte of 3 byte frame counter. Incremented every 16ms.

FRAMES2 **$5C7A**

Most significant byte of 3 byte frame counter.

UDG **$FF58** **$5C7B**

Address of first user-defined graphic (UDG).

XCOORD **$5C7D**

X coordinate of last point plotted.

YCOORD **$5C7E**

Y coordinate of last point plotted.

PPOSN **$21** **$5C7F**

33-column number of printer position.

PRCC **$5C80**

Least significant ($5C80) and most significant ($5C81) bytes of address of next character (in printer buffer) for printer.

ECHOE **$1721** **$5C82**

33-column number and 24-line number (in lower half) of end of keyboard input buffer.

DFCC **$50C0** **$5C84**

Address in display file of PRINT position.

DFCCL **$50E0** **$5C86**

Like DFCC, for lower half screen.

SPOSNCOL **$21** **$5C88**

33-column number for PRINT position.

SPOSNLIN **$2** **$5C89**

24-row number for PRINT position.

SPOSNLCOL **$17** **$5C8A**

Column for next screen output on lower half screen.

SPOSNLLIN **$21** **$5C8B**

Line for next screen output on lower half screen.

SCRCT **$0A** **$5C8C**

Scroll count. Is always 1 more than the number of scrolls that will be done before stopping with **scroll?**. Decremented for each scroll.

ATTRP **$38** **$5C8D**

Current printing attributes, as set by color statements.

FOREG	0	Least sig bit of foreground color (ink).
BLUEFG	0	
REDFG	1	
GRNFG	2	
BACKG	3	Least sig bit of background color
BLUEBG	3	(paper).
REDBG	4	
GRNBG	5	
HILITE	6	BRIGHT.
FLASH	7	FLASH.

MASKP **$38** **$5C8E**

Use for transparent colors. Any bit that is 1 takes the corresponding attribute bit from the screen, not from ATTRP.

ATTRT **$38** **$5C8F**

Current temporary printing attributes (see ATTRP for bit definitions).

MASKT **$38** **$5C90**

Current temporary printing attributes mask.

PFLAG **$5C91**

Additional flags for printing: temporary flags in even bits, permanent flags in odd bits.

XOR_CH	0	New chars XORed (OVER) into old.
INV_CH	2	New characters inverted.
F_CB	4	Foreground = complement of bckgd.
B_CF	6	Background = complement of foregd.

MEMBOT **$5C92**

Calculator's memory area. Used to store number that cannot conveniently be put on the calculator stack. 6 numbers.

MEM0	$5C92
MEM1	$5C97
MEM2	$5C9C
MEM3	$5CA1
MEM4	$5CA6
MEM5	$5CAB

NMIADD (USRNMI) **$5CB0**

Address of NMI service routine.

RAMTOP **$E100** **$5CB2**

Address of last byte of BASIC system area.

PRAMT **$FFFF** **$5CB4**

Last byte of physical RAM.

ERRLN **$0900** **$5CB6**

Pointer to ONERROR line number for GOTO.

ERRC **$5CB8**

Line number in which error occurred.

ERRS **$5CBA**

Statement number in which error occurred.

ERRT **$5CBB**

Error number (report code).

SYSCON **$5EEA** **$5CBC**

Pointer to the system configuration table.

MAXBNK **$5CBE**

Number of expansion banks in system.

CRCBN **$5CBF**

Current channel bank number.

MSTBOT **$6200** **$5CC0**

Address of location above machine stack.

VIDMOD **$5CC2**

Video mode. Non zero if the second display file is open for use.

ARSBUF **$5CC4**

Pointer to AROS buffer.

ARSFLG **$5CC6**

AROS flag.

BANKCHN	0	I/O through a bank channel.
AROSQTE	1	Quoted string in AROS.
AROSKCUR	2	K_CUR pointing to AROS.
AROSDTA	3	DATADD pointing to AROS.
AROSNXT	4	NXTLIN pointing to AROS.
AROSDST	6	DEST pointing to AROS.
AROS	7	1 indicates AROS present.

ADATLN **$5CC7**

Pointer to the start of the current data line (AROS only).

DTLNLN **$5CC8**

Length of the current data line (AROS only).

STRMN **$5CCB**

Current stream number. Used for bus expansion unit devices.

BNADDR **$5FE9**

Highest priority bank number address. Used for bus expansion unit memory.

BANKP **$5FEB**

Highest bank priority. Used for bus expansion unit memory.

Internal Variables, Equates and Registers

Assigned Bank Number (ABN) **$A0**

In setup mode: receives the assigned bank number for the bank presently selected by the daisy chain.

In normal mode: receives the universal deselect byte. Chunks are hi-active. As an input, least significant nybble is bank status for "presently accessed bank".

BANK **11**

BCMDPT **$FD**

Expansion bank "address" port.

BDATPT **$FC**

Expansion bank "data" port.

BLKLEN **$0B**

Data block length, used in load and save routines.

Bank Number Acceseds (BNA) **$80**

As an output register, sets the "presently accessed bank". On input, it is the most significant nybble for register $40.

Command Register (CMD) **$C0**

As an output register, four commands are known:

- 00 Reset daisy chain and enter the setup mode.
- 01 Step the daisy chain to the next bank.
- 02 Reset the nybble steering logic.
- 04 End the setup mode and enter the normal mode.

No more than one bit is ever set simultaneously.

As an input register, it is the most significant nybble for register $A0.

DATTYPE **$00**

Describes the kind of data being saved/loaded. 03 is binary/code.

Display Enhancement Control Register (DECR) **$FF**

The display enhancement control register, in the SCLD, controls selection of enhanced video modes, ink selection for 64-column mode, enable/inhibit of the 17ms interrupt and selection of EXROM or DOCK.[7] Also referred to as HREXPT in the Timex Sinclair 2068 Third Party Software Guide.

- Bit 0: 1 enables D_FILE_2 (secondary display file).
- Bit 1: 1 enables ultra-high-resolution color mode (expanded attributes).
- Bit 2: 1 enables 64-column display.
- Bit 3, 4, 5: Paper color for 64-column display.

[7] Section 2.1.8.1 of the technical manual.

- Bit 6: 1 disables the keyboard interrupt.
- Bit 7: 1 enables extension ROM in the EXROM bank.

HOLD

Temporary holding register used for bank switching.

HOMEROM **$FF**

Value used in preparing to switch to the HOME ROM.

Horizontal Select (HS) **$40**

As an output register, it receives the horizontal select byte (hi-active) for the "presently accessed bank".

As an input register, it is the least significant nybble of horizontal select for "presently accessed bank".

Horizontal Select Register (HSR) **$F4**

Hardware register in the SCLD that determines which chunks are selected in HOME and EXROM/DOCK memory space. Also referred to as DKHSPT in the Timex Sinclair 2068 Third Party Software Guide.

Universal Deselect Byte (HSP) **$A0**

Bank select/switch register A0 when accessed in normal mode. When the system is in "normal mode," a write to register A0 sends the "Universal Deselect Byte" to all expansion banks. This looks a bit like a Horizontal Select byte, but has important differences. Each bit represents a memory chunk, just like a Horizontal Select byte, but if a particular bit contains a zero, each Horizontal Select register will leave its corresponding bit alone. If a particular deselect bit contains a one, then if ANY Horizontal Select byte has a one in that location, it RESETS it. As such, the Universal Deselect byte tells all banks which chunks they must give up.[8]

LAST_EXT_SVC **13**

LAST_RAM_SVC **24**

LOWNYB **$C000**

PRM_OUT **8**

SADDPT **$F5**

SDATPT **$F6**

[8] "Mystery of the Missing 253" series.

STA_L **ABN**

STA_O **CMD**

STATUS

Nybble whose bit have the following states:

- Bit 0: Set to 0 if bank cause a maskable interrupt.
- Bit 1: Not used.
- Bit 2: Set to 0 if bank is responding to memory read/write.
- Bit 3: Not used.

STKSZ **$200**

Size of the stack.

SVC_CODE

16 bit quantity. Bit 15 is used as a jump flag: if set, the dispatcher will do a GOTO_BANK to the specified routine, otherwise it will do a CALL_BANK.

IY offsets

OERRNR	ERR_NR	$00
OFLAGS	FLAGS	$01
OTVFLAG	TVFLAG	$02
OMODE	MODE	$07
ONSPPC	SPPC	$0A
OSUBPPC	SUBPPC	$0D
OBORDCR	BORDCR	$0E
OEPPC	EPPC	$0F
OKCUR	KCUR	$21
OXPTR	X_PTR	$25
OBREG	BREG	$2D
OFLAGS2	FLAGS2	$30
ODFSZ	DFSZ	$31
OOSPCC	OSPCC	$36
OFLAGX	FLAGX	$37
OSTRLEN	STRLEN	$38
OFRAMES	FRAMES	$40
OPPOSN	PPOSN	$45
OPRCC	PRCC	$46
OSPOSNLIN	POSNLIN	$4F
OSPOSNLCOL	POSNLCOL	$50
OSCRCT	SCRCT	$52
OPFLAG	PFLAG	$57
OERRLN	ERRLN	$7C
OPIP	PIP	$FF
ORASP	RASP	$FE

System Configuration Table (SYSCON)

The area from $5EEA to $5FFF is reserved for the System Configuration Table (SYSCON).

This table is built during system initialization and contains an 8 byte entry for AROS, a 4 byte entry for LROS, followed by eleven 24-byte entries for proposed expansion banks and an End-Of-Table marker.

The actual usage of this table is limited to the 12 bytes for software cartridge identification.

AROS (SYSCON, $5EEA)

The first 8 bytes describe an AROS, if one is available.

Byte(s)	Description
00	Language type (1 = BASIC, 2 = Machine code)
01	Cartridge type (2 = AROS)
02-03	Starting address. BASIC AROS, address of first program line. Machine CODE AROS, address of first Z80 instruction.
04	Memory chunk specification. Bits 0-7 represent chunks 0-7, respectively, in the DOCK banks in low active format. 0 if in use, 1 if not in use. Note: Bits 0-3 must be set to 1 (marking chunks 0-3 as not in use) for proper execution.
05	Autostart specification. 0 = no autostart, 1 = autostart.
06-07	Number of byte of RAM to reserve for machine code variables. In LSB/MSB format.

LROS

The next 4 bytes describe an LROS.

Byte(s)	Description
00	Not used.
01	Cartridge type (1 = LROS).
02-03	Starting address (LSB/MSB). Address to jump after operating system is complete.
04	Memory chunk specification. Bits 0-7 represent chunks 0-7, respectively, in the DOCK banks in low active format. 0 if in use, 1 if not in use. Note: Bits 3 should be set to 1 (marking chunks 3 as not in use) for the JP to address on 02-03 to work.

Only the two entries above are used by the 2068. The entry below describes the format of an expansion bank, had they been produced.

EXPANSION BANK ENTRY

Byte(s)	Description
00	01=ROM 02=RAM 00=Inactive
01	Bank #. MSB is set if bank is not yet renumbered

The following is copied from 0000-0015 of ROM expansion banks

02	For RAM - Chunks available - High true For ROM - Channel specifier, if this bank controls a channel. This will be an ASCII character, and the initialization software resets bit 5, insuring that the letter will be uppercase.
03-04	Address of OPEN routine for the channel. (Alternately, 02-04 could have a residual JP instruction, which does no good to the SYSCON table, but allows RST 0 to work in the expansion bank, since the JP is also at location 0000 of that bank.)
05-06	Address of the CLOSE routine, if the bank controls a channel. Call with RAM Res Code with PRM_OUT=2, and stream number on the stack.
07-08	Timex called this the address of the SELECT routine. It could have been used in initialization, and to attach the current channel to this bank.
09-0A	An I/O device INPUT routine address.
0B-0C	An I/O device OUTPUT routine address. (Alternately, 0A-0C could have contained a residual JP that would have been intended to allow RST 08 to work in the ROM bank.)
0D-0E	Address of Disk Command Handler routine
0F-10	Address of device interrupt handler (92 bytes)
11-12	Address of device initialization code (cold start)
13-14	Address of device reset routine (warm start) (Alternately, 12-14 could have contained a residual JP that would have been used in the ROM bank to make RST 10 work.)
15	Device type. Bit 0 = 0 if bootable,1 if initializable. Bit 1 = 0 if non storage device, 1 if storage (disk commands).
16	Boot up priority. Low # =high priority. Home bank=80.
17	Interrupt Priority. RAM banks get 255. ROM get lower value, which means higher priority.

NOTE: The Timex document gives this list as ROM addresses, rather than SYSCON entries, as given here. The SYSCON displacements must always differ by two from the ROM addresses. This difference is not an error.

Bibliography

"A Practical Study of System Variables: Put Them to Use with BASIC." *Update Magazine*, April 1988.

Charles Bothner. "Extra Memory." *Update Magazine*, October 1988.

Wes Brzozowski. "The Mystery of the Missing 253." *Time Designs Magazine*, August 1986.

Wes Brzozowski. "More About ... the Mystery of the Missing 253." *Time Designs Magazine*, October 1986.

Wes Brzozowski. "The Mystery of the Missing 253, Part Three." *Time Designs Magazine*, December 1986.

Wes Brzozowski. "The Mystery of the Missing 253, Part Four." *Time Designs Magazine*, February 1987.

Wes Brzozowski. "The Mystery of the Missing 253: Conclusion." *Time Designs Magazine*, March/April 1987.

V. C Corcoran and M. H Branigin. *Timex Sinclair 2068 Personal Color Computer: Technical Reference Manual*. Waterbury, CT: Timex Computer Corp., 1984. https://archive.org/stream/TimexSinclair2068Manuals/TimexSinclair2068_TechnicalManual.

———. *Timex Sinclair 2068 Personal Color Computer: Technical Reference Manual*. Second Edition. Colton, OR: Time Designs Magazine, 1986. https://archive.org/stream/TimexSinclair2068Manuals/Timex%20Sinclair%202068%20Technical%20Manual%20%28best%29.

Lloyd Dreger. The Timex 2068 ROM Manuscript. Smugware Conversions, 1986, https://archive.org/details/lloyd-dreger.

Bill Jones. "A Utility That Uses System Variables." *Update Magazine*, April 1988.

Bill Jones. "Extra Memory Programming." *Update Magazine*, January 1988.

Bob Orrfelt. Timex Sinclair 2068 Memory Map and ROM Disassembly. Gesso Products, 1985, https://archive.org/stream/bob-orrfelt/TS2068%20Memory%20Map%20and%20ROM%20Disassembly%20-%20Bob%20Orrfelt.

William J. Pedersen. "Bank Switching... The Timex/Sinclair 2068 Computer, Part I." T-S Horizons, no. 19, June 1986, pp. 20–23, https://archive.org/stream/TSHorizons/T-sHorizonsN19#page/n19/mode/2up.

Nazir A. Pashtoon. "Comparative ROM Atlas: From ZX Spectrum to 2068." Syntax, vol. 5, no. 10, Oct. 1984, pp. 3, 5, 7, 9, 11, https://archive.org/stream/SyntaxMagazine/Syntax%20v5n10#page/n2.

Routine Index

The Timex/Sinclair 2068 was the Timex Computer Corporation's third and final home computer sold in the United States market. Based on the Sinclair ZX Spectrum, Timex added many new features to the TS 2068.

The TS 2068 included new BASIC commands and a dedicated sound chip, cartridge port, support for up to 16MB of RAM, disk drives, printers, and more. These impressive features resulted in a powerful home computer that was no longer compatible with the ZX Spectrum.

This disassembly shows how Timex programmers changed the ZX Spectrum source code to support the new features and planned expansions.

This book reveals the inner workings of the TS 2068 operating system. It is a valuable resource for the serious machine language programmer.

www.timexsinclair.com