PRELIMINARY

CONFIDENTIAL

PROPERTY OF TCC

TIMEX SINCLAIR 2000 THIRD PARTY SOFTWARE GUIDE

TECHNICAL PUBLICATION NO. 335-879200-01
MAY 1983

REVISION CHANGE PAGE RECORD

PUBLICATION NO.	DATE	PUBLICATION NO.	DATE
335-879200-01	May 1983		

LIST OF CHANGE PAGES

PAGE	DATE	PAGE	DATE
Cover	May 1983		
A	May 1983		
Preface	May 1983		
ii through viii	May 1983		
1-1 through 1-10	May 1983		
2-1 through 2-2	May 1983		
3-1 through 3-17	May 1983		
4-1 through 4-4	May 1983		
5-1 through 5-8	May 1983		
6-1 through 6-32	May 1983		
7-1	May 1983		
8-1 through 8-4	May 1983		

PREFACE

TIMEX COMPUTER CORPORATION has made every effort possible to insure that inaccuracies or discrepancies do not exist within our manuals or software. Our aim is to provide each user with a top-quality and reliable product.

However, we assume no liability for the misuse or misapplication of our manuals or systems or for inaccuracies or omissions within our manuals or programs.

Please note that the specifications in this manual are subject to change without prior notice.

For additional information regarding our products and/or services contact TIMEX CORPORATION, P.O.BOX 2655, WATERBURY, CONNECTICUT 06725; PHONE 1-800-24-TIMEX.

TABLE OF CONTENTS

TABLE OF CONTENTS (CON'T)

TABLE OF CONTENTS (CON'T)

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

PURPOSE OF MANUAL

The TIMEX Sinclair 2000 personal computer is a full-feature personal color-computer. Developed from the Sinclair Spectrum, the T/S 2000 has many features that are not available on the Spectrum. These include joysticks, 3-channel sound synthesis, bank switching, cartridge-software support, a function dispatcher, additional commands, etc.

The operating system within the T/S 2000 is a derivative of the one that the Spectrum uses. The addresses of any useful routines in the Spectrum ROM have changed. These routines, and others are available through the function dispatcher.

The purpose of this manual is two-fold. First, it will enable the veteran Spectrum programmer to convert his Spectrum programs to a form runable on the T/S 2000. It will also permit him to add functions to his programs that take advantage of the new features available on the T/S 2000. Second, this manual will guide the new T/S 2000 programmer through the features that the T/S 2000 offers.

This manual is specifically intended for software programmers who would like to submit software to TIMEX for marketing purposes. And, while we can not guarantee that all submissions will be accepted, following the guidelines outlined in this manual will improve your chances for having your submission accepted.

HOW TO USE THIS MANUAL

The organization of this manual makes it easy to find the information you need. To locate particular subject matter, use the Table of Contents to determine the page on which the information appears.

Read this reference manual before attempting to write or submit software for TIMEX evaluation. Sufficient familiarity with the contents of this manual will help to structure your program in the recommended manner. In addition, this manual should be used with the SPECTRUM manual as a guide to the T/S 2000 computer.

REVISION PROTOCOL

Periodically, new documentation is released to inform the user of revised procedures or product enhancements, etc. This documentation may take the form of revised manuals, change pages, or technical bulletins.

When change pages are received, immediately remove and discard the old pages and substitute the new ones. If it is necessary to issue an entirely new page, the page number will be followed by a letter in alpha sequence (e.g. 2-4a). In this instance, insert the additional page into your manual (e.g. between pages 2-4 and 2-5). A mark on the right-hand margin of the new page will indicate the place, line, or paragraph, where the documentation has been changed. Additionally, the date of the change will be found on the lower right-hand corner of the changed page.

Technical bulletins, which are sequentially numbered, should be kept in the binder with your manual.

Following this technique will insure that your manual is always up-to-date.

QUERIES

Should you have any questions regarding the contents of this manual, please direct your comments to TIMEX.

SECTION I

MEMORY

1.1 INTRODUCTION

SECTION I deals with the memory of the T/S 2000. It covers the system memory map, bank switching, the bank switching controller chip, the Home Bank layout, the ROS bank layout, AROS/LROS, and expansion Banks.

1.2 SYSTEM MEMORY

The system memory is configured as illustrated by Figure 1-1.

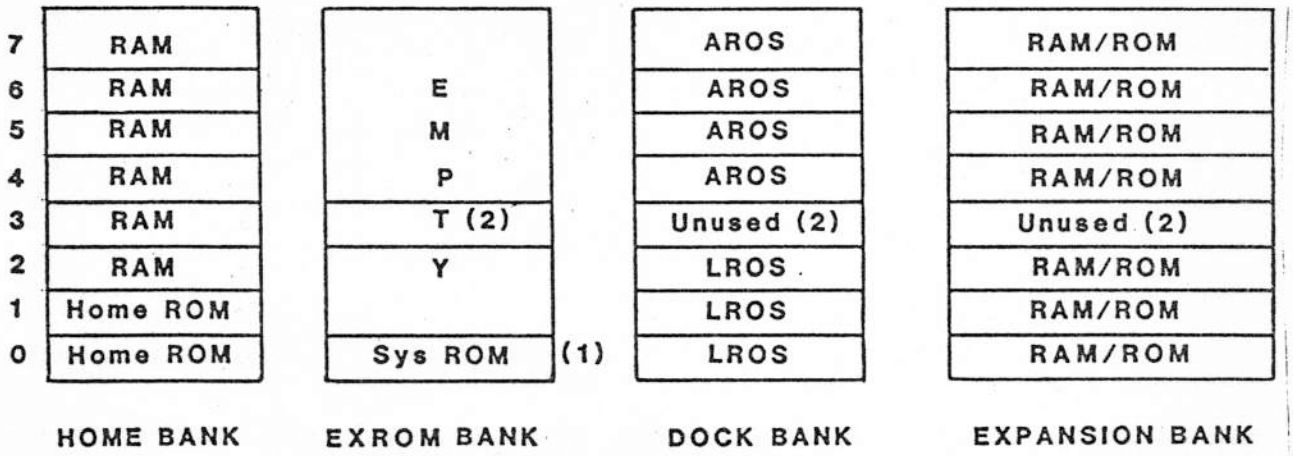


Figure 1-1. SYSTEM MEMORY MAP

[1] ROM contains the T/S 2000 System Software enhancements.

[2] Home Bank Chunk #3 is permanently enabled to provide interrupt handling and a stack.

1.3 BANK SWITCHING

Since the T/S 2000 is a Z-80 based computer and can only address 64K of memory directly, a bank selection mechanism is provided. Memory is selected in 8K "Chunks" which are identified by bank number (horizontal index) and Chunk number (vertical index), as illustrated by Figure 1-2.

The Home Bank is the bank which is selected by default, other banks must be explicitly selected. All banks are identified by a number, Ø - 255. Bank Ø is an AROS and/or LROS (i.e. DOCK Bank), banks 1 through 253 are peripherals, bank 254 is the Home ROM Extension, (i.e. EXROM Bank) and bank 255 is the Home Bank. Bank selection services are provided by the system software (refer to Section VI). The following paragraphs will describe the architecture of the banks in the system.

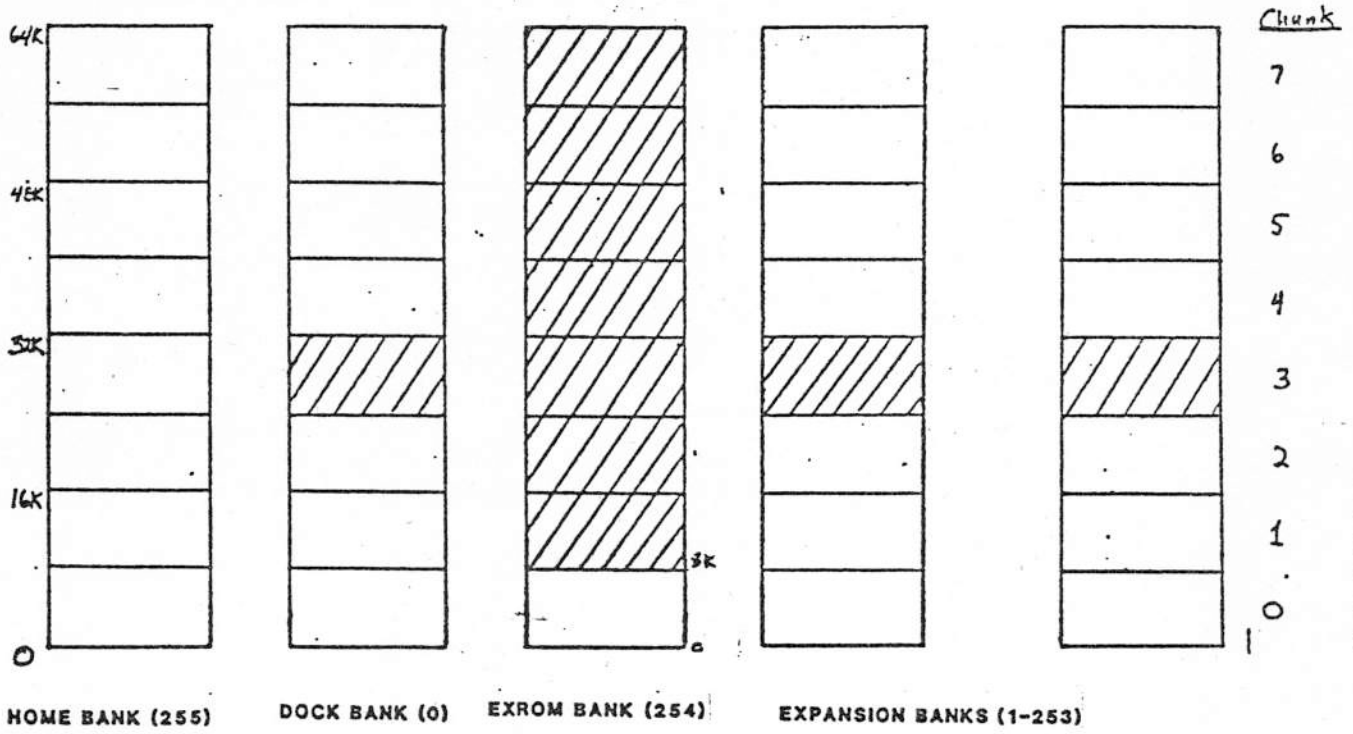


Figure 1-2. SYSTEM MEMORY ARCHITECTURE

The memory space on the TS 2000 consists of a 64K Home Bank, a 64K DOCK Bank, an 8K EXROM Bank and a plurality of 64K Expansion Banks. The memory map is shown in Figure 1-1. Each 64K bank is divided into eight 8K "Chunks". The T/S 2000 hardware provides an ability to enable eight Chunks under software control which determine the actual 64K address space of the Z80 at any one time. At most one Chunk along the horizontal axis may be enabled at any one time (i.e., if Chunk #1 is enabled in the DOCK Bank, it must/will be disabled in all other banks). Details of the software control of bank switching are contained in paragraph 6.9.

NOTE

The Home Bank has the lowest priority, thus its Chunks are enabled by default when no other bank has the same Chunks enabled. The DOCK Bank has the next highest priority, thus its Chunks are enabled when no expansion Bank has the same Chunks enabled. The expansion Banks have the highest priority. The Home ROM Extension Bank (EXROM) has the same priority as the DOCK Bank.

1.3.1 Reserved Chunks

Note that the machine stack, function dispatcher and bank switching communication facilities must always be available to the system. These routines will always be in either Chunk 3 or Chunk 7 of the Home Bank. One of these Chunks must always be enabled. The one that must be enabled is the one that contains the code listed above.

A system variable called VIDMOD (location 23741) determines where these blocks of code are at any given time. If VIDMOD is zero, then they are in Chunk 3. If VIDMOD is non-zero, then they are in Chunk 7. These routines are moved up to Chunk 7 if the secondary display file (D_FILE_2) is requested for use.

If you are operating a 16K machine, then these routines will always be in Chunk 3 because there is no memory to hold them in Chunk 7.

Whichever Chunk or Chunks are utilized to hold these essential routines, it/they should always be enabled within the Home Bank (i.e. disabled in the DOCK Bank and all Expansion Banks). Thus, Chunk 3 (and/or 7 if the second display file is used) should not be enabled in any bank other than the Home Bank.

1.4 BANK SWITCHING CONTROL

The bank switching communication services access the bank switching hardware by reading and writing to the following I/O ports:

- DKHSPT = F4H: DOCK horizontal select port
- BDATPT = FCH: Expansion Bank "data" port
- BCMDPT = FDH: Expansion Bank "address" port
- HREXPT = FFH: Home ROM Extension select port (bit 7)

These ports read from and/or write to the following bank switching hardware registers:

- DKHSPT writes to: DOCK horizontal select register
- BDATPT writes to: Registers Ø-3
- BCMDPT writes to: Address register (selects registers Ø-3)

In addition there are the following registers:

- HOLD : Temporary holding register
- ABN : Assigned bank number register (one for each Expansion Bank)
- BNA : Bank number accessed register
- HS : Expansion Bank horizontal select register (one for each Expansion Bank)
- STATUS : Status nybble whose bits have the following interpretation:

  bit Ø - Set to Ø if bank caused an interrupt (maskable)
  bit 1 - Not used
  bit 2 - Set to Ø if bank is responding to memory read/write
  bit 3 - Not used

The bank switching controller chip uses a four-bit data path (DØ-D3).

The DOCK horizontal select register specifies which of the DOCK Bank Chunks are enabled (active high). The hold register is used to latch data in and out of the ABN, BNA, and HS registers. The BNA register contains the bank number of the bank whose status is currently being changed or queried. There is only one hold register and one BNA register. There are ABN and HS registers for each expansion Bank. The ABN register contains the bank number assigned to a particular bank. The HS register specifies which Chunks in the expansion Banks are currently enabled (active high).

The DOCK horizontal select port accesses the DOCK horizontal select register. This port is both read from and written to. The expansion Bank "address" port is used to specify which register to write to, or read from. The "data" port is used to read/write data to the designated register within the bank switching controller. The addresses are listed in Table 1-1..

Table 1-1
EXPANSION BANK CONTROLLER REGISTERS

ADDRESS	READ DATA PORT	WRITE DATA PORT
Ø	Read status	Write cmd ls nybble
1	None	Write cmd ms nybble
2	Read HS ls nybble	Write hold reg. ls nybble
3	Read HS ms nybble	Write hold reg. ms nybble

The two command (cmd) ports have the functions listed in Table 1-2.

Table 1-2
CMD REGISTER FUNCTIONS

Command -- Least significant nybble TYPE I

BIT	ACTIVE	FUNCTION
Ø 14	LOW	Reset controller -- prepare for initialization
1 13	LOW	Start interrupt REG sequence.
2 11	LOW	Initialization done. Move to next bank in daisy chain.
3 7	LOW	Reset interrupt flag.

Command -- Most significant nybble TYPE II

Ø	LOW	Dump hold REG to ABN.
1	LOW	Dump hold REG to BNA.
2	LOW	Dump hold REG to HS.
3	-	Not used.

1.5 HOME BANK LAYOUT

The Home Bank is illustrated by Figure 1-3. Area 2 is a 16K Home ROM which contains: a) a BASIC interpreter, b) a set of routines which provide basic input and output (display text and graphics, keyboard input, printer, access to the sound chip and joysticks), and c) channeled I/O to dumb devices or intelligent devices both of which may be attached to the Expansion Bus. Area 1 is the Home RAM (either 16K or 48K) which contains the Display Files, System Variables, BASIC Program, etc.

NOTE

Area bracketed in text refers to
its counterpart in the illustration.

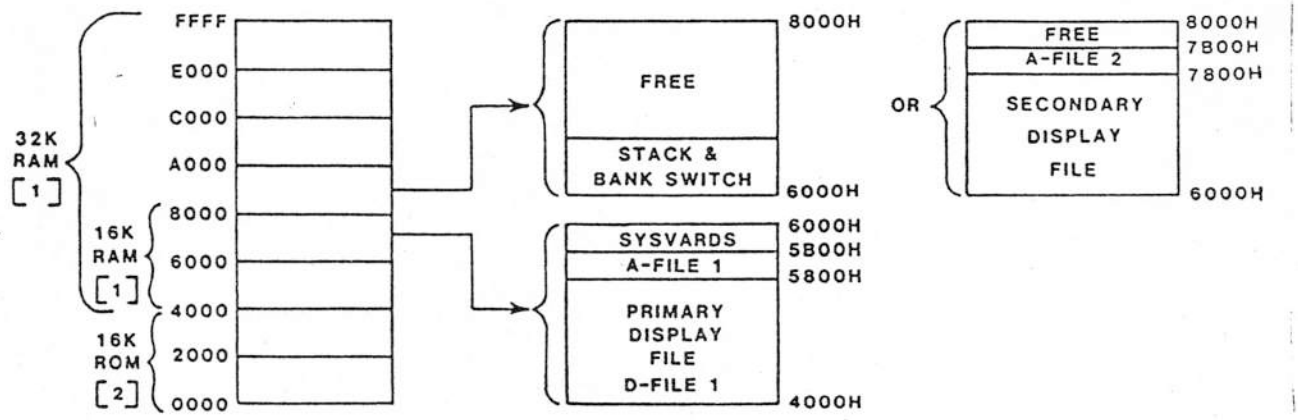


Figure 1-3. HOME BANK ARCHITECTURE

1.6 EXROM BANK LAYOUT

The Home Bank has associated with it an 8K ROM. This ROM together with the 16K ROM make up 24K of System ROM. The 8K ROM is located in a separate bank called the EXROM Bank. (This 8K ROM is called the EXROM.) Contained within the ROM are: the cassette tape I/O routines, the bank switching code, and the system initialization routines.

This ROM is accessed by a software switch. Bit 7 of I/O address FFhex designates the selection status of the EXROM. If this bit is set, the EXROM will overlay the lowest 8K Chunk (Chunk Ø) in the DOCK Bank.

Thus, to access this ROM, you must set bit 7 in port FFhex, and set at least bit Ø in the DHS register (I/O port F4h). You must also insure that no external banks have their first Chunk selected, as external banks have higher priority than the DOCK Bank and would override the system's access to the DOCK Bank, and thus the EXROM.

1.7 ROM ORIENTED SOFTWARE (ROS): AROS/LROS

A connector is present on the T/S 2000 base unit and on each Bus Expansion Unit (BEU) (refer to paragraph 1.9.1) to allow the plug-in of ROM Oriented Software. The ROS component may be one of two types or both:

1. Application ROM Oriented Software (AROS), and
2. Language ROM Oriented Software (LROS).

The AROS contains an application program which eliminates the cassette loading step and makes all of the system RAM available for data. The LROS allows the T/S 2000 to take on a personality other than that of the BASIC programming language (i.e.; other languages such as LOGO, etc.). The system software's architecture supports the ability of both an AROS and an LROS to be used at the same time.

It should be noted that both the AROS and the LROS share the same 56K of address space. An individual ROS may use up all or part of that 56K of space (the DOCK Bank). If LROS and AROS are to be used simultaneously, their address spaces must not overlap. Therefore, if an LROS is designed to accept an AROS, the LROS's designers must trade off maximum LROS size against desired sizes for their AROS. Typically the LROS will be a maximum of 24K and AROS will be a maximum of 32K bytes.

NOTE

If you plan on submitting a BASIC Program for possible use by TIMEX in an AROS, it must not utilize user-defined functions. They are not supported in AROS's.

1.7.1 LROS

LROS will typically map into the address space between the ØØØØ and 5FFF (24K). If the second display file is to be used instead of the first, then the LROS may extend to 7FFF (32K). If the system initialization routine detects the presence of an LROS, a jump to the starting address (contained in ØØØ2H) is performed, thereby passing control to the LROS. The following overhead bytes must appear in every LROS (all addresses are in hex):

ØØØØ - Not used (one byte)

ØØØ1 - Cartridge Type (one byte)
1 = LROS

ØØØ2 - Starting address (two bytes)
Address to be jumped to after operating system initialization is complete.

ØØØ4 - Chunk specification (one byte)
Bits Ø = 7 represent Chunks Ø - 7 respectively as follows:
Ø = if not in use
1 = if in use

1.7.2 AROS

AROS will typically map into the address space between 8ØØØ and FFFF (32K). If the second display file is to be used by the system, then the AROS will map into the address space between 8ØØØ and DFFF (24K). If the initialization routine detects the presence of an LROS, control is passed to the LROS upon completion of the initialization as described above. The LROS is then responsible for RUNning any installed AROS. If just an AROS is present, it is treated as a BASIC/machine language program with system RAM used for program variables. The following overhead bytes must appear in every AROS (all addresses are in hex):

8ØØØ - Language type (one byte)
1 = BASIC and machine language
2 = Machine language only
3 = LOGO
4 = PASCAL

8ØØ1 - Cartridge type (one byte)
2 = AROS

8ØØ2 - Starting address (two bytes) - for language type:
1 = Beginning of BASIC program
2 = Starting address of machine language program
3 = To be determined
4 = To be determined

8ØØ4 - Chunk specification (one byte)
Bits Ø - 7 represent Chunks Ø - 7 respectively as follows:
Ø = If not in use
1 = If in use

8ØØ5 - Autostart Specification (Language type 1 or 2) (one byte)
0 = If ROS not to be autostarted
1 = If ROS is to be autostarted

8ØØ6 - Number bytes for user area (two bytes)

1.8 DOCK BANK

The architecture of the DOCK Bank is illustrated by Figure 1-4. Either an AROS, an LROS, or both may be present in the bank. An AROS alone could contain a BASIC and/or machine code program. BASIC code in the AROS would be interpreted by the Home ROM, but the entire program would not be loaded into the Home RAM. Machine code in an AROS would be executed in the AROS. An AROS together with an LROS could be a program in some language (such as LOGO, and possibly including some machine code) supported by the LROS. An AROS or LROS alone may be up to 56K in size, where size is measured in 8K increments. An AROS and an LROS together must not add up to more than 56K and their address spaces must not overlap.

Note the hatched Chunk in Figure 1-4. This indicates a Chunk which is on the same horizontal slice as the machine stack in the Home Bank. This Chunk should not be used by the DOCK Bank.

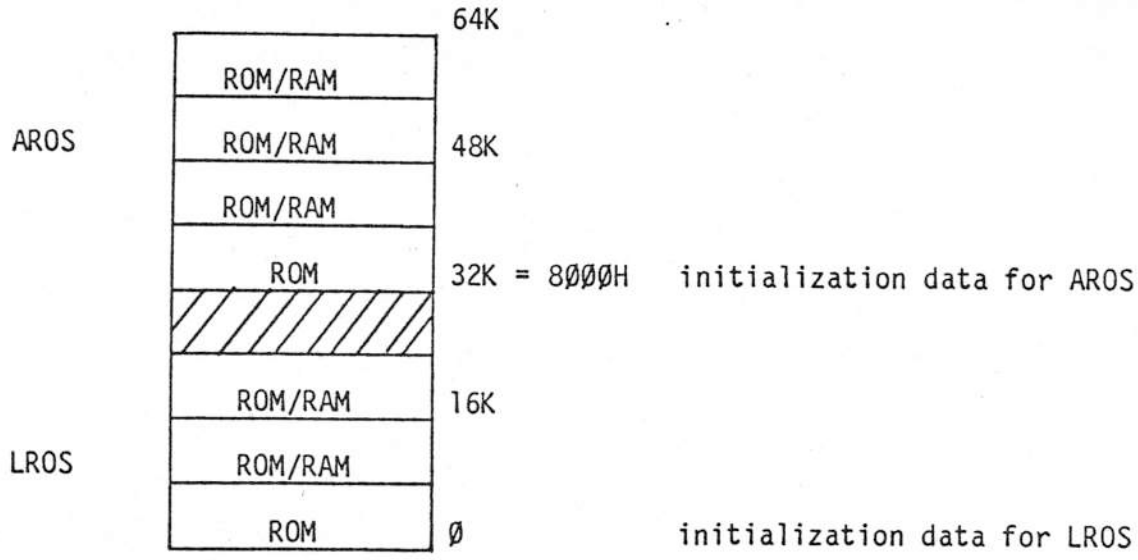


Figure 1-4. DOCK BANK ARCHITECTURE WITH D_FILE_2 OFF

1.9 EXPANSION BANKS AND BUS EXPANSION UNIT

Paragraph 1.9.1 describes Expansion Banks and the Bus Expansion Unit.

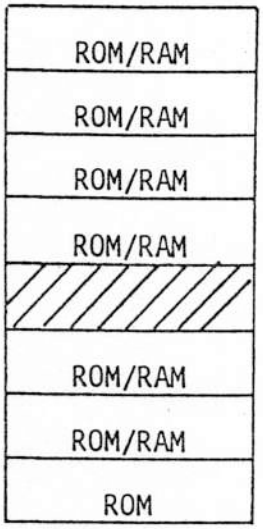


Figure 1-5. EXPANSION BANK ARCHITECTURE WITH D_FILE_2 OFF

1.9.1 Peripheral Expansion Bus

The Expansion Bus connector allows the attachment of one or more peripherals to the T/S 2000. Up to two peripherals may be attached directly to the Expansion Bus connector. If more than two peripherals are desired, a Bus Expansion Unit which contains four connectors is attached first. These connectors may be used to attach multiple peripherals. An additional Bus Expansion Unit may be attached to the first one, allowing a maximum of seven devices to be attached to the T/S 2000. Each of the devices may also contain one or more Expansion Banks. An Expansion Bank is a bank of up to 64K of RAM/ROM. The hardware and software architecture of the T/S 2000 supports up to 253 Expansion Banks. The Bus Expansion Unit is invisible to software. The BEU also contains one additional DOCK connector.

The Expansion Banks are used for controlling intelligent devices or for memory expansion. They may contain ROM and/or RAM as shown in the memory maps (refer to Figures 1-1, 1-2 and 1-5). Except for RAM banks, each Expansion Bank must contain the following overhead information to be properly recognized by the system software:

ØØØØ - Device Specification (1 byte) (Refer to Table 1-3 for a list of device specifications.)
ØØØ1 - Address of device OPEN routine (2 bytes)
ØØØ3 - Address of device CLOSE routine (2 bytes)
ØØØ5 - Address of device SELECT routine (2 bytes)
ØØØ7 - Address of device INPUT routine (2 bytes)
ØØØ9 - Address of device OUTPUT routine (2 bytes)
ØØØB - Address of disk command handler routine (2 bytes) (see Device Type (bit 1) below)
ØØØD - Address of device Interrupt Handler routine 92 bytes)
ØØØF - Address of device Initialization routine (2 bytes) (cold start)
ØØ11 - Address of device RESET routine (2 bytes) (warm start)
ØØ13 - Device type (1 byte)
Bit Ø = Ø Bootable
1 Initializable
Bit 1 = Ø Non-storage device
1 Storage device (capable of handling disk commands)
ØØ14 - Device Boot Priority (1 byte) (Ø - 255)
ØØ15 - Device interrupt priority (1 byte) (Ø - 255)

Table 1-3
DEVICE SPECIFICATIONS

BYTE	DEVICE
'T'	Telecommunications Device
'F'	Stringy Floppy
'D'	Floppy Disk
'H'	Hard Disk
'R'	RS232 Interface
'C'	Centronics Interface
'P'	Printer (80-column)
'L'	Local Area Network
'R'	Ram Insertion
'P'	Printer
'K'	Keyboard
'S'	Screen
'M'	Reserved

SECTION II

SYSTEM BLOCK DIAGRAM

2.1 INTRODUCTION

The T/S 2000 is illustrated in block form by Figure 2-1. The components are as follows:

SCLD	--	Standard Cell Logic Design
Z80	--	Microprocessor
16K RAM	--	16K Home RAM
32K RAM	--	Optional 32K Home RAM
16K ROM	--	System Software
8K ROM	--	System Software
MUX	--	Multiplexer

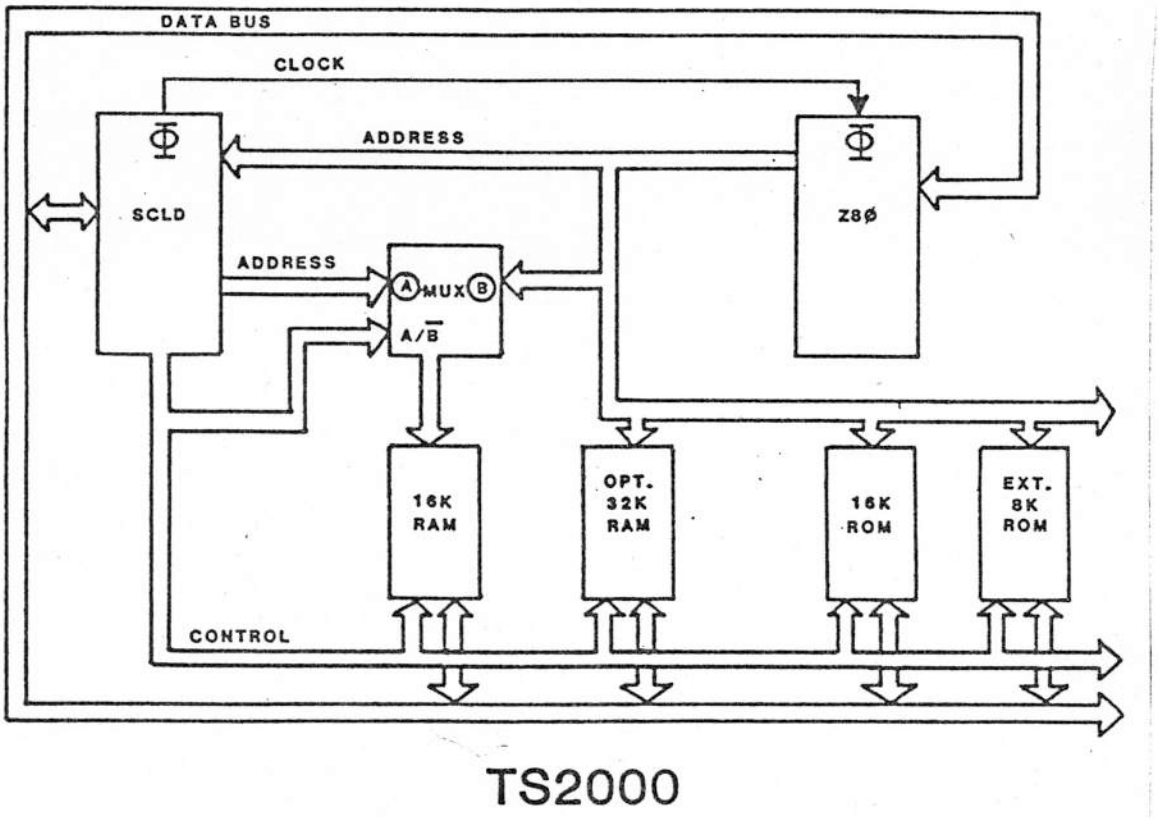


Figure 2-1. SYSTEM BLOCK DIAGRAM

SECTION III

INPUT/OUTPUT FACILITIES

3.1 INTRODUCTION

The T/S 2000 supports many input/output facilities.

3.2 I/O PORT LAYOUT MAP

Ports are configured as illustrated in Figure 3-1 and mapped below.

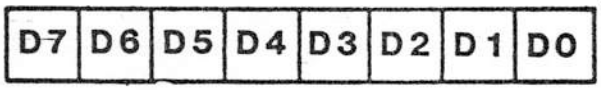


Figure 3-1. BYTE LAYOUT

FFH: Read/Write

DØ:	1 Enables D/FILE/2 (secondary display file)
D1:	1 Enables ultra-high-resolution color (expanded attributes)
D2:	1 Enables 64-column display (requires D1 to be set)
D3, D4, D5:	Paper color for 64-column display
D6:	1 Disables the keyboard interrupt (refer to paragraph 5-4)
D7:	1 Enables Extension ROM in the EXROM Bank

FEH: Read

DØ, D1, D2, D3, D4:	State of the five keyboard outputs (refer to the SPECTRUM users' manual Chapter 23)
D5:	Not used
D6:	Tape input
D7:	Not used

FEH: Write

DØ, D1, D2:	Border color
D3:	Tape output
D4:	Sound toggle (BEEP)
D5, D6, D7:	Not used

FDH: Bank selection ADDR port
FCh: Bank selection DATA port
FBh: Printer read/write
FAh: Printer read/write
F9h: Printer read/write
F8h: Printer read/write
F7h: Reserved
F6h: Sound chip register data read/write
F5h: Sound chip register address read/write
F4h: DOCK horizontal select read/write

EFh: Reserved

TIMEX modem uses DF, D7, CF, C7

TIMEX reserves ports 78h - FFH

Ports Øh - 77h are available*

* It is requested that vendors contact TIMEX for port assignment to prevent conflicts with other vendors. Please direct inquiries to:

TIMEX Computer Corporation
1579 Straits Turnpike
Middlebury, CT. 06720

Nineteen ports are illustrated by Figure 3-2

FF	Display Modes, Extension ROM, Interrupt Control
FE	Keyboard, Border Color, Tape, Sound
FD	Bank Controller - Address
FC	Bank Controller - Data
FB	Printer
FA	Printer
F9	Printer
F8	Printer
F7	Note B
F6	Sound Chip - Data
F5	Sound Chip - Address
F4	DOCK Horizontal Select
F3	Printer
F2	Printer
F1	Printer
F0	Printer
EF	Note B Reserved
EE	
ED	

Figure 3-2. T/S 2000 I/O PORTS

NOTES:

A) TIMEX reserves I/O addresses 78 - FFhex; outside world will use ØØ - 77hex.
B) Reserved
C) Modem will be DF, D7, CF, C7
D) Spectraside uses 78h - 7FH

3.3 DUMB DEVICES

A number of dumb devices are incorporated within the system.

3.3.1 Keyboard

The keyboard is a 5 x 8 matrix. The eight-inputs are address lines A8 to A15 which are sequentially lowered and an I/O read is performed to I/O address FEH as each address line is lowered. This uses the same routine as the BASIC IN command. Bits Ø to 4 of I/O port FEH contain the state of the five keyboard outputs (refer to Chapter 23 of the SPECTRUM BASIC manual). Refer to paragraph 5.2.2 for a description of the differences between the SPECTRUM and the T/S 2000's keyboard.

3.3.2 Tape

An I/O read to FEH pulls in the cassette input on bit 6.

An I/O write to FEH bit 3 controls the tape output. There is no provision for controlling the tape recorder's motor.

3.3.3 Sound

Simple Sound: (BEEP) Sound is generated under software control by toggling bit 4 of I/O address FEH.

Full Sound: The sound chip registers are set by writing the register number to I/O address F5h and the value for that register to I/O address F6h.

3.3.4 Video

Display file selection is accomplished by writing to bit Ø of I/O address FFH. A Ø enables D/FILE/1, a 1 enables D/FILE/2. D/FILE/1 is the default.

Ultra-high Resolution color is enabled by writing a 1 to I/O address FFH.

64-column display is enabled by writing a 6 to I/O address FFH. Paper color for this mode is specified by writing the appropriate value into bits 3 through 5 of I/O address FFH. A contrasting color ink will be chosen by the hardware; for example:

```
LD   A,ØØØ1Ø11Ø  ; select 64-character
OUT  (FF),A      ; mode w/red paper
```

Refer to paragraph 6.4 for full details.

3.3.5 Printer

The printer is accessed by an I/O request to any I/O address with A7 high, and A2 low.

3.3.6 Joysticks

The joystick port is selected by writing a 14 to I/O address F5h. The joystick is then read by reading I/O address F6h with the player number (1 or 2) on the upper half of the address bus., i.e.,

```
ld    a, 14
out   a, (ØF5H)
ld    a, PLAYER
in    a, (ØF6H)
```

The joysticks used are industry-standard. They connect to a 9-pin "D-type" connector.

The byte read is interpreted as follows:

1 DØ:	Ø indicates stick up
2 D1:	Ø indicates stick down
4 D2:	Ø indicates stick left
8 D3:	Ø indicates stick right
16-64 (D4, D5, D6):	Not used (all ones)
128 D7:	Ø indicates pushbutton depressed

3.3.7 Sound Chip

The basic registers in the PSG (Programmable Sound Generator -- GI 8912) which produce the programmed sounds include:

Tone Generators:	Produce the basic square wave tone frequencies for each channel (A, B, C).
Noise Generator:	Produces a frequency modulated pseudo random pulse width square wave output.
Mixers:	Combine the outputs of the Tone Generators and the Noise Generator. One for each channel (A, B, C).
Amplitude Control:	Provides the D/A Converters with either a fixed or variable amplitude pattern. The fixed amplitude is under direct CPU control; the variable amplitude is accomplished by using the output of the Envelope Generator.
Envelope Generator:	Produces an envelope pattern which can be used to amplitude modulate the output of each Mixer.
D/A Converters:	The three D/A Converters each produce up to a 16-level output signal as determined by the Amplitude Control.

NOTE

All register numbers are in octal.

An additional register is shown in the PSG Block Diagram (Figure 3-3) which has nothing directly to do with the production of sound -- this is the I/O Port (A). Data to/from the CPU may be read/written to/from the 8-bit I/O Port without affecting any other function of the PSG. The T/S 2000 uses the I/O port to access the joystick.

REGISTER \ BIT		B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-BIT Fine Tune A							
R1		4-BIT Coarse Tune A							
R2	Channel B Tone Period	8-BIT Fine Tune B							
R3		4-BIT Coarse Tune B							
R4	Channel C Tone Period	8-BIT Fine Tune C							
R5		4-BIT Coarse Tune C							
R6	Noise Period	5-BIT Period Control							
R7	$\overline{\text{Enable}}$	$\overline{\text{IN/OUT}}$		$\overline{\text{Noise}}$			$\overline{\text{Tone}}$		
		IOB	IOA	C	B	A	C	B	A
R10	Channel A Amplitude			M	L3	L2	L1	L0	
R11	Channel B Amplitude			M	L3	L2	L1	L0	
R12	Channel C Amplitude			M	L3	L2	L1	L0	
R13	Envelope Period	8-BIT Fine Tune E							
R14		8-BIT Coarse Tune E							
R15	Envelope Shape/Cycle	CONT.	ATT.	ALT.	HOLD				
R16	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							

Figure 3-3. PSG REGISTER ARRAY

3.3.7.1 Tone Generator Control (Registers R0, R1, R2, R3, R4, R5). - The frequency of each square wave generated by the three Tone Generators (one each for Channels A, B, and C) is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 12-bit Tone Period value. Each 12-bit value is obtained in the PSG by combining the contents of the relative Coarse and Fine Tune registers, as illustrated by Figure 3-4.

Coarse Tune Register	Channel	Fine Tune Register
R1	A	R0
R3	B	R2
R5	C	R4

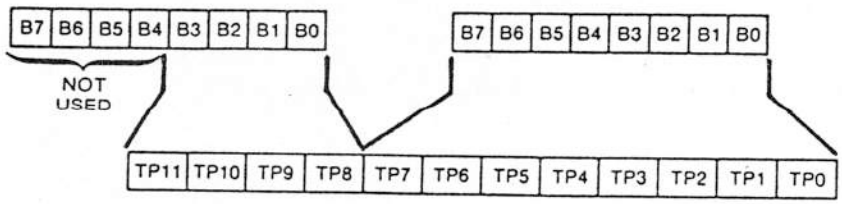


Figure 3-4. 12-BIT TONE PERIOD (TP) TO TONE GENERATOR

Note that the 12-bit value programmed in the combined Coarse and Fine Tune registers is a period value -- the higher the value in the registers, the lower the resultant tone frequency.

Note also that due to the design technique used in the Tone Period countdown, the lowest period value is ØØØØØØØØØØØ1 (divide by 1) and the highest period value is 111111111111 (divide by 4,Ø95).

The equations describing the relationship between the desired output tone frequency and the input clock frequency and Tone Period value are:

(a) $fT = \frac{fCLOCK}{16TP_{10}}$ (b) $TP_{10} = 256CT_{10} + FT_{10}$

Where:
- f_T = Desired tone frequency
- fCLOCK = Input clock frequency
- TP_{10} = Decimal equivalent of the Tone Period bits TP11 -- TPØ
- CT_{10} = Decimal equivalent of the Coarse Tune register bits B3--BØ (TP11--TP8)
- FT_{10} = Decimal equivalent of the Fine Tune register bits B7--BØ (TP7--TPØ)

From the above equations it can be seen that the tone frequency can range from a low of $\frac{fCLOCK}{65,520}$ (wherein: $TP_{10} = 4,095_{10}$) to a high of $\frac{fCLOCK}{16}$ (wherein: TP_{10}=1). Using a 2-MHz input clock, for example, would produce a range of tone frequencies from 30.5-Hz to 125-kHz. The T/S 2000 uses a 1.75-MHZ input clock. 14.112 MHz ÷ 1.764 MHz

To calculate the values for the contents of the Tone Period Coarse and Fine Tune registers, given the input clock and the desired output tone frequencies, we simply rearrange the above equations, yielding:

(a) $TP_{10} = \frac{fCLOCK}{16f_T}$ (b) $CT_{10} + \frac{FT_{10}}{256} = \frac{TP_{10}}{256}$

Example 1: $f_T = 1kHz$, $f_{CLOCK} = 2MHz$

$$TP_{10} = \frac{2x10^6}{16(1x10^3)} = 125$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{125}{256}$$

$\therefore CT_{10} = 0 = 0000$ (B3--B0)
$FT_{10} = 125_{10} = 01111101$ (B7--B0)

Example 2: $f_T = 100Hz$, $f_{CLOCK} = 2MHz$

$$TP_{10} = \frac{2x10^6}{16(1x10^2)} = 1250$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{1250}{256} = 4 + \frac{226}{256}$$

$\therefore CT_{10} = 4_{10} = 0100$ (B3--B0)
$FT_{10} = 226_{10} = 11100010$ (B7--B0)

3.3.7.2 Noise Generator Control (Register R6). -- The frequency of the noise source is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 5-bit Noise Period value. This 5-bit value consists of the lower 5 bits (B4--BØ) of register R6, as illustrated by Figure 3-5.

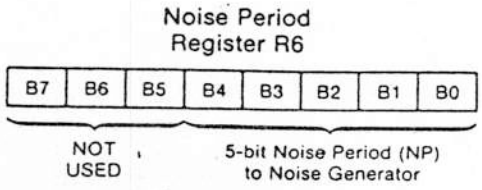


Figure 3-5. NOISE PERIOD REGISTER R6

Note that the 5-bit value in R11 is a period value -- the higher the value in the register, the lower the resultant noise frequency. Note also that, as with the Tone Period, the lowest period value is ØØØØ1 (divide by 1); the highest period value is 11111 (divide by 31_{10}).

The noise frequency equation is:

$$fN = \frac{fCLOCK}{16\ NP_{10}}$$

Where:

f_N = Desired noise frequency
fCLOCK = Input clock frequency
NP_{10} = Decimal equivalent of the Noise Period register bits B4--BØ.

From the above equation it can be seen that the noise frequency can range from a low of $\frac{fCLOCK}{486}$ (wherein: $NP_{10}=31_{10}$) to a high of $\frac{fCLOCK}{16}$ (wherein: $NP_{10}=1$). Using a 2-MHz input clock, for example, would produce a range of noise frequencies from 4-kHz to 125-kHz.

To calculate the value for the contents of the Noise Period register, given the input clock and the desired output noise frequencies, we simply rearrange the above equation, yielding:

$$NP_{10} = \frac{fCLOCK}{16f_N}$$

3.3.7.3 Mixer Control I/O enable (Register R7). -- Register 7 is a multi-function Enable register which controls the three Noise/Tone Mixers and the two general purpose I/O Ports.

The Mixers, as previously described, combine the noise and tone frequencies for each of the three channels. The determination of combining neither/either/both noise and tone frequencies on each channel is made by the state of bits B5--BØ of R7.

The direction (input or output) of the two general purpose I/O Ports (IOA and IOB) is determined by the state of bits B7 and B6 of R7. Note that in the T/S 2000 there is no second I/O Port IOB.

These functions are illustrated by Figure 3-6 and Tables 3-1 and 3-2

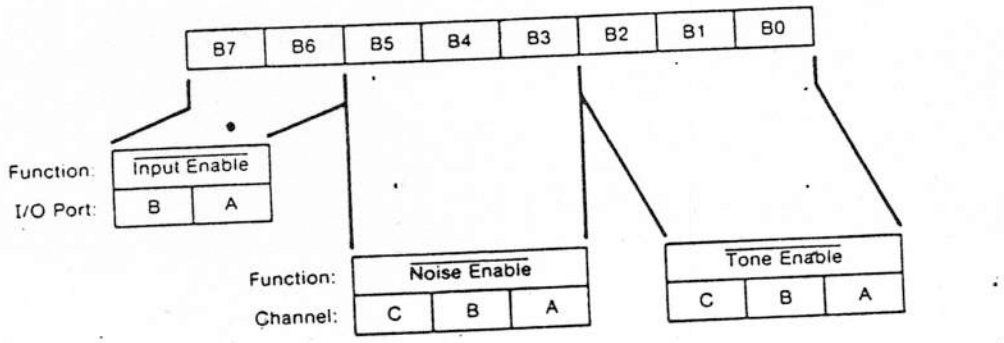


Figure 3-6. MIXER CONTROL-I/O ENABLE REGISTER R7

Table 3-1
I/O ENABLE

Noise Enable Truth Table:		Tone Enable Truth Table:	
R7 Bits B5 B4 B3	Noise Enabled on Channel	R7 Bits B2 B1 B0	Tone Enabled on Channel
Ø Ø Ø	C B A	Ø Ø Ø	C B A
Ø Ø 1	C B -	Ø Ø 1	C B -
Ø 1 Ø	C - A	Ø 1 Ø	C - A
Ø 1 1	C - -	Ø 1 1	C - -
1 Ø 0	- B A	1 Ø Ø	- B A
1 Ø 1	- B -	1 Ø 1	- B -
1 1 Ø	- - A	1 1 Ø	- - A
1 1 1	- - -	1 1 1	- - -

Table 3-2
I/O PORT TRUTH TABLE

R7 Bits	I/O Port Status
B6	IOA
Ø	Input
1	Output

NOTE

Disabling noise and tone does not turn off a channel. Turning a channel off can only be accomplished by writing all zeroes into the corresponding Amplitude Control register, RIØ, R11, or R12 (refer to paragraph 3.3.7.4).

3.3.7.4 Amplitude Control (Register R10, R11, R12). -- The amplitudes of the signals generated by each of the three D/A Converters (one each for Channels A, B, and C) is determined by the contents of the lower 5 bits (B4--BØ) of registers RIØ, R11, and R12 as illustrated by Figure 3-7.

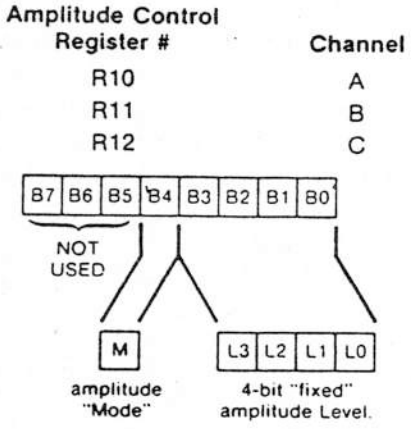


Figure 3-7. D/A CONVERTED SIGNAL GENERATION

The amplitude "mode" (bit M) selects either fixed level amplitude (M=Ø) or variable level amplitude (M=1). It follows then that bits L3--LØ, defining the value of a "fixed" level amplitude, are only active when M=Ø. When fixed level amplitude is selected, it is "fixed" only in the sense that the amplitude level is under the direct control of the system processor (via bits D3--DØ). Varying the amplitude when in this "fixed" amplitude mode requires in each instance the direct intervention of the system processor via an address latch/write data sequence to modify the D3--DØ data.

When M=1 (select "variable" level amplitudes), the amplitude of each channel is determined by the envelope pattern as defined by the Envelope Generator's 4-bit output E3 E2 E1 EØ.

The amplitude "mode" (bit M) can also be thought of as an "envelope enable" bit, (i.e., when M=Ø the envelope is not used, and when M=1 the envelope is enabled). A full description of the Envelope Generator function follows in Paragraph 3.3.7.5.

Figure 3-8 illustrates all combinations of the 5-bit Amplitude Control.

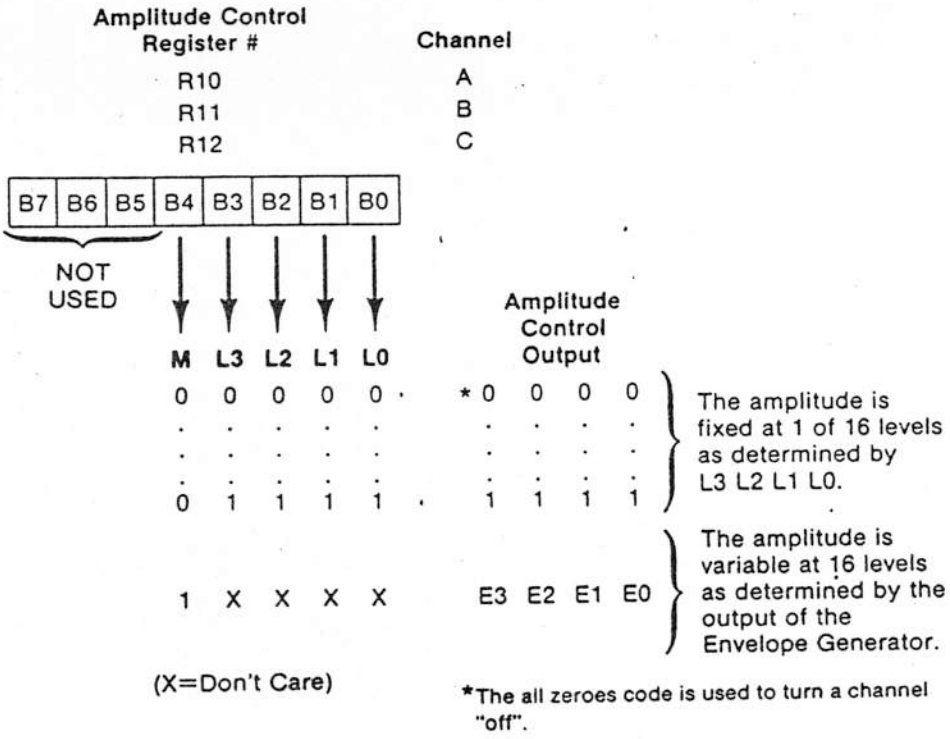


Figure 3-8. FIVE-BIT AMPLITUDE CONTROL

Figure 3-9 graphically illustrates a selection of variable level (envelope-controlled) amplitude where the 16 levels directly reflect the output of the Envelope Generator. A fixed level amplitude would correspond to only one of the levels shown, with the level directly determined by the decimal equivalent of bits L3 L2 L1 LØ.

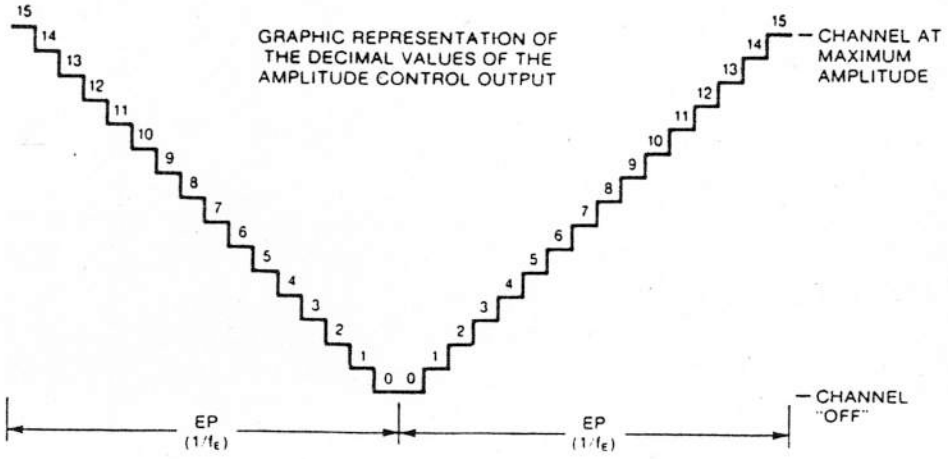


Figure 3-9. VARIABLE AMPLITUDE CONTROL (M=1)

3.3.7.5 Envelope Generator Control (Register R13, R14, R15). -- To accomplish the generation of fairly complex envelope patterns, two independent methods of control are provided in the PSG: first, it is possible to vary the frequency of the envelope using registers R13 and R14; and second, the relative shape and cycle pattern of the envelope can be varied using register R15. The following paragraphs explain the details of the envelope control functions, describing first the envelope period control and then the envelope shape/cycle control.

3.3.7.6 Envelope Period Control (Register R13, R14) -- The frequency of the envelope is obtained in the PSG by first counting down the input clock by 256, then by further counting down the result by the programmed 16-bit Envelope Period value. This 16-bit value is obtained in the PSG by combining the contents of the Envelope Coarse and Fine Tune registers, as illustrated by Figure 3-10.

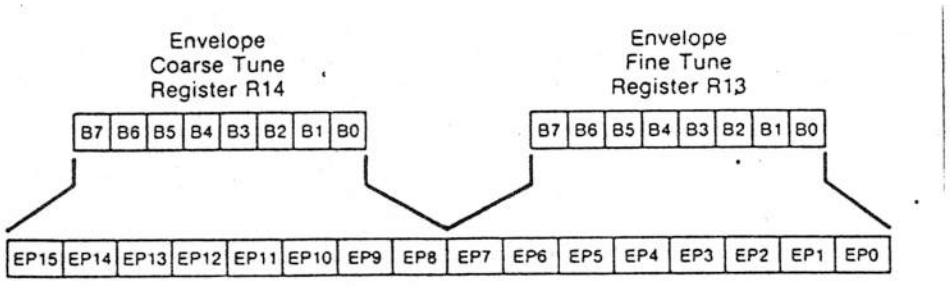


Figure 3-10. 16-BIT ENVELOPE PERIOD (EP) TO ENVELOPE GENERATOR

Note that the 16-bit value programmed in the combined Coarse and Fine Tune registers is a period value - the higher the value in the registers, the lower the resultant envelope frequency.

Note also, that as with the Tone Period, the lowest period value is ØØØØØØØØØØØØØØØ1 (divide by 1); the highest period value is 1111111111111111 (divide by $65,535_{10}$).

The envelope frequency equations are:

(a) $f_E \frac{fCLOCK}{256\ EP_{10}}$ (b) $EP_{10} = 256\ CT_{10} + FT_{10}$

Where:

f_E = Desired envelope frequency
fCLOCK = Input clock frequency
EP_{10} = Decimal equivalent of the Envelope Period bits EP15--EP0
CT_{10} = Decimal equivalent of the Coarse Tune register bits B7--BØ (EP15--EP8)
FT_{10} = Decimal equivalent of the Fine Tune register bits B7--BØ (EP7--EPØ)

From the above equation it can be seen that the envelope frequency can range from a low of $\frac{fCLOCK}{16,766,960_{10}}$ (wherein: $EP_{10}=65,535_{10}$) to a high of $\frac{fCLOCK}{256}$ (wherein: $EP_{10}=1$). Using a 2-MHz clock, for example, would produce a range of envelope frequencies from 0.12-Hz to 7812.5-Hz.

To calculate the values for the contents of the Envelope Period Coarse and Fine Tune registers, given the input clock and the desired envelope frequencies, we rearrange the above equations, yielding:

(a) $EP_{10} = \frac{f_{CLOCK}}{256 f_E}$ (b) $CT_{10} + \frac{FT_{10}}{256} = \frac{EP_{10}}{256}$

Example: $f_E = 0.5$ Hz
$f_{CLOCK} = 2$ MHz

$$EP_{10} = \frac{2 \times 10^6}{256(0.5)} = 15,625$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{15,625}{256} = 61 + \frac{9}{256}$$

$CT_{10} = 61_{10} = 00111101$ (B7--B0)
$FT_{10} = 9_{10} = 00001001$ (B7--B0)

3.3.7.7 Envelope Shape/Cycle Control (Register R15). -- The Envelope Generator further counts down the envelope frequency by 16, producing a 16-state per cycle envelope pattern as defined by its 4-bit counter output, E3 E2 E1 EØ. The particular shape and cycle pattern of any desired envelope is accomplished by controlling the count pattern (count up/count down) of the 4-bit counter and by defining a single-cycle or repeat-cycle pattern. This envelope shape/cycle control is contained in the lower 4 bits (B3--BØ) of register R15. Each of these 4 bits controls a function in the envelope generator, as by Figure 3-11.

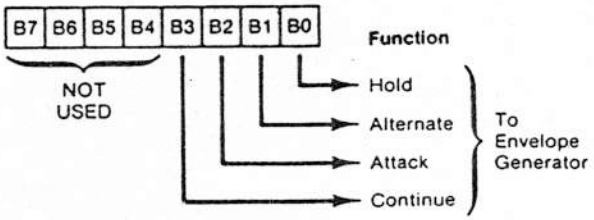


Figure 3-11. ENVELOPE SHAPE/CYCLE CONTROL REGISTER (R15)

The definition of each function is as follows:

Hold: When set to logic "1", limits the envelope to one cycle, holding the last count of the envelope counter (E3--EØ=ØØØØ or 1111, depending on whether the envelope counter was in a countdown or count-up mode, respectively).

Alternate: When set to logic "1", the envelope counter reverses count direction (up-down) after each cycle.

NOTE

When both the Hold bit and the Alternate bit are ones, the envelope counter is reset to its initial count before holding.

Attack: When set to logic "1", the envelope counter will count up (attack) from E3 E2 E1 EØ=ØØØØ to E3 E2 E1 EØ=1111; when set to logic "Ø", the envelope counter will count down (decay) from 1111 to ØØØØ.

Continue: When set to logic "1", the cycle pattern will be as defined by the Hold bit; when set to logic "Ø", the envelope generator will reset to ØØØØ after one cycle and hold at that count.

To further describe the above functions could be accomplished by numerous charts of the binary count sequence of E3 E2 E1 EØ for each combination of Hold, Alternate, Attack and Continue. However, since these outputs are used (when selected by the Amplitude Control registers) to amplitude modulate the output of the Mixers, a better understanding of their effect can be accomplished via a graphic representation of their value for each condition selected, as illustrated in Figures 3-12 and 3-13.

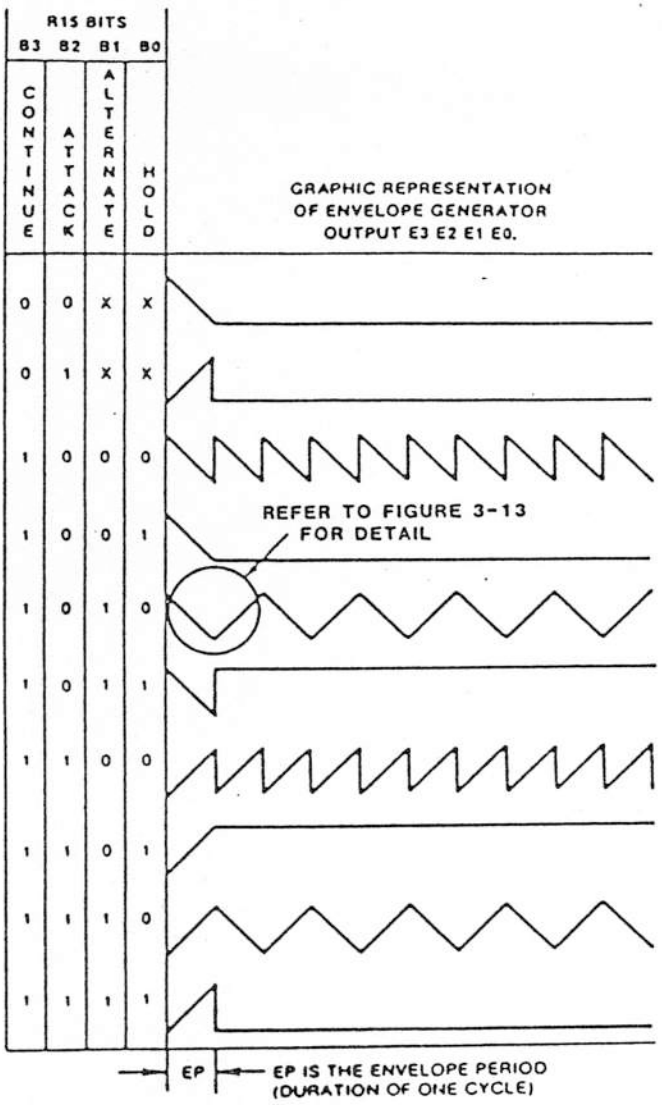


Figure 3-12. ENVELOPE SHAPE/CYCLE CONTROL

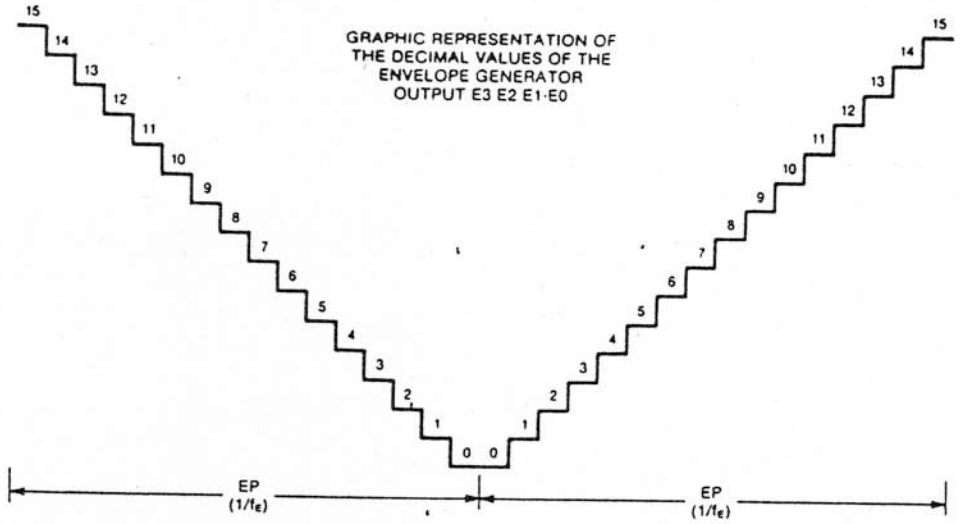


Figure 3-13. DETAIL OF TWO CYCLES OF Figure 3-12

3.3.7.8 I/O Port Data Store (Register R16). -- Register R16 functions as intermediate data storage registers between the PSG/CPU data bus (DAØ--DA7) and the I/O port (IOA7--IOA0). This port is available for reading the joystick. Using register R16 for the transfer of I/O data has no effect at all on sound generation.

To output data from the CPU bus to a peripheral device connected to I/O Port A would require only the following steps:

1. Latch address R7 (select Enable register)
2. Write data to PSG (setting B6 to R7 in "1")
3. Latch address R16 (select IOA register)
4. Write data to PSG (data to be output on I/O Port A)

To input data from I/O Port A to the CPU bus would require the following:

1. Latch address R7 (select Enable register)
2. Write data to PSG (setting B6 in R7 to "Ø")
3. Latch address R16 (select IOA register)
4. Read data from PSG (data from I/O Port A)

Note that once loaded with data in the output mode, the data will remain on the I/O port until changed either by loading different data, by applying a reset (grounding the Reset pin), or by switching to the input mode.

Note also that when in the input mode, the contents of register R16 will follow the signals applied to the I/O port. However, transfer of this data to the CPU bus requires a "read" operation as described above.

3.3.7.9 DHS

The horizontal select register for the DOCK Bank is accessed through I/O address F4h. It is read/write. It is called DOCK Horizontal Select. Each bit is active high. This means that if bit 3 of DHS is set high, then Chunk 3 of the DOCK Bank is mapped into the CPU's address space.

3.4 BANK CONTROLLER I/O PORTS

The expansion Banks on the T/S 2000 each use a controller IC (integrated-circuit). This controller uses two I/O ports. They are:

FCh - Data Port
FDH - Address Port

3.5 CHARACTER OUTPUT

Character output on the T/S 2000 is accomplished by calling a single routine in the system ROM which sends the designated character to the current output channel. (i.e., TV or printer, etc.). The routine is accessed via the Z80's RST 1Ø instruction. The character to be output, is passed to the routine in the Accumulator (A register). This routine uses the following registers: A, F, B', C', D', and E'; preserving BCDEHL

SECTION IV

CONNECTOR SPECIFICATIONS

4.1 INTRODUCTION

In this section, specifications are provided for the DOCK connector, edge connector and joystick connectors.

4.2 DOCK CONNECTOR (ROS, etc.)

The T/S 2000 DOCK uses a 36-pin connector. The layout and function of each of these pins is listed below in Table 4-1 and Figure 4-1.

Table 4-1
DOCK CONNECTOR SIGNAL ASSIGNMENT

PIN #	SIGNAL NAME	DESCRIPTION
1	A14B	Address Bus Bit 14. Buffered.
2	+5V	+ 5V DC.
3	A12	Address Bus Bit 12.
4	A13B	Address Bus Bit 13. Buffered.
5	D0	Data Bus Bit Ø.
6	D7	Data Bus Bit 7.
7	D1	Data Bus Bit 1.
8	A0	Address Bus Bit Ø.
9	D2	Data Bus Bit 2.
10	A1	Address Bus Bit 1.
11	D6	Data Bus Bit 6.
12	A2	Address Bus Bit 2.
13	D5	Data Bus Bit 5.
14	A3	Address Bus Bit 3.
15	D3	Data Bus Bit 3.
16	A15	Address Bus Bit 15.
17	D4	Data Bus Bit 4.
18	$\overline{\text{MREQB}}$	Memory Request (active low). Buffered.
19	$\overline{\text{IORQB}}$	I/O Request (active low). Buffered.
20	A7RB	Refresh Address Bit 7. Buffered.
21	$\overline{\text{RDB}}$	Read (active low). Buffered.
22	$\overline{\text{M1}}$	CPU M1 State (active low).
23	$\overline{\text{WRB}}$	Write (active low). Buffered.
24	A8	Address Bus Bit 8.
25	A7	Address Bus Bit 7.
26	A9	Address Bus Bit 9.
27	A6	Address Bus Bit 6.
28	A10	Address Bus Bit 10.
29	A5	Address Bus Bit 5.
30	A11	Address Bus Bit 11.
31	A4	Address Bus Bit 4.
32	$\overline{\text{RFSHB}}$	Refresh (active low). Buffered.
33	$\overline{\text{BE}}$	Bank Enable (active low).
34	$\overline{\text{EXROM}}$	Extension ROM Enable (active low).
35	$\overline{\text{ROSCS}}$	ROS (ROM Oriented Software) ship Select (active low).
36	GND	Ground

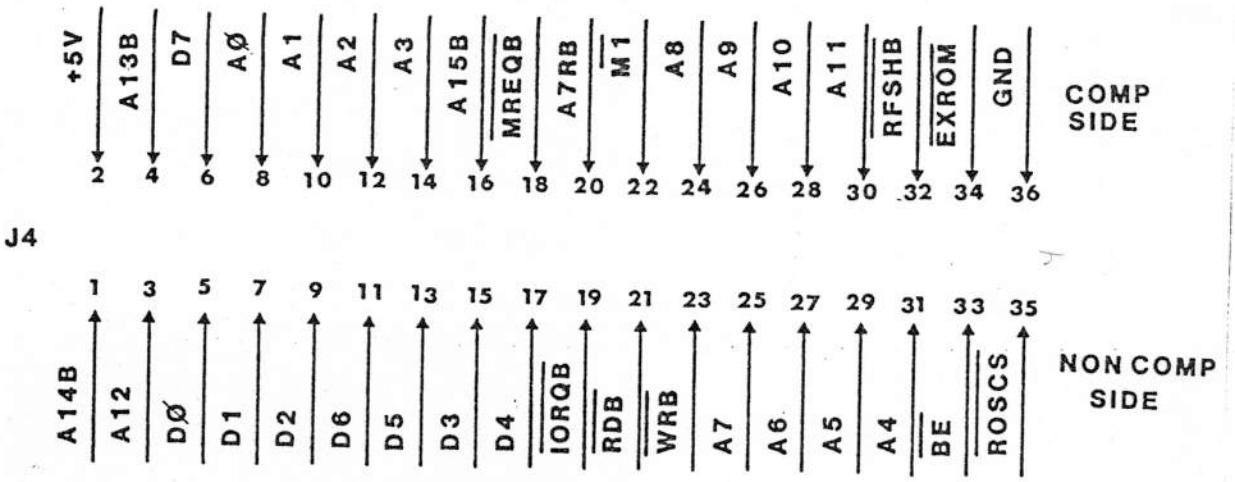


Figure 4-1. DOCK CONNECTOR

4.3 EDGE CONNECTOR SPECIFICATION

The T/S 2000 uses a 64-pin edge connector. The layout and function of each of these pins are specified in Table 4-2 and Figure 4-2.

Table 4-2
EDGE CONNECTOR SIGNAL ASSIGNMENT

PINS #	SIGNAL NAME	DESCRIPTION
1A	Tape OUT/SPKR	Tape/Speaker Output
1B	EAR	EAR Input
2A	DZOUT	Daisy out.
2B	DZIN	Daisy in.
3A	A7RB	Refresh Address Bit 7. Buffered.
3B	+15V	+15V DC.
4A	D7	Data Bus Bit 7.
4B	+5V	Plus 5 Volts.
5A	Not used	- - - - - - -
5B	Not used	- - - - - - -
6A	Slot	- - - - - - -
6B	Slot	- - - - - - -
7A	D0	Data Bus Bit Ø.
7B	GND	Signal Ground.
8A	D1	Data Bus Bit 1.
8B	GND	Signal Ground.
9A	D2	Data Bus Bit 2.
9B	Φ	CPU Clock (inverted).
10A	D6	Data Bus Bit 6.
10B	A0	Address Bus Bit Ø.
11A	D5	Data Bus Bit 5.
11B	A1	Address Bus Bit 1.
12A	D3	Data Bus Bit 3.
12B	A2	Address Bus Bit 2.
13A	D4	Data Bus Bit 4.
13B	A3	Address Bus Bit 3.
14A	$\overline{INT}$	Interrupt request (active low).

Table 4-2 (CON'T)

PINS #	SIGNAL NAME	DESCRIPTION
14B	A15B	Address Bus Bit 15. Buffered.
15A	$\overline{\text{NMI}}$	Non Maskable Interrupt (active low).
15B	A14B	Address Bus Bit 14. Buffered.
16A	$\overline{\text{HALT}}$	CPU HALT Indicator Signal (active low).
16B	A13B	Address Bus Bit 13. Buffered.
17A	$\overline{\text{MREQB}}$	Memory Request (active low). Buffered.
17B	A12	Address Bus Bit 12.
18A	$\overline{\text{IORQB}}$	I/O Request (active low). Buffered.
18B	A11	Address Bus Bit 11.
19A	$\overline{\text{RDB}}$	Read (active low). Buffered.
19B	A10	Address Bus Bit 1Ø.
20A	$\overline{\text{WRB}}$	Write (active low). Buffered.
20B	A9	Address Bus Bit 9.
21A	$\overline{\text{BUSAK}}$	Bus Acknowledge (active low).
21B	A8	Address Bus Bit 8.
22A	$\overline{\text{WAIT}}$	CPU Wait (active low).
22B	A7	Address Bus Bit 7.
23A	$\overline{\text{BUSRQ}}$	Bus Request (active low).
23B	A6	Address Bus Bit 6.
24A	$\overline{\text{RESET}}$	CPU Reset (active low).
24B	A5	Address Bus Bit 5.
25A	$\overline{\text{M1}}$	CPU M1 State (active low).
25B	A4	Address Bus Bit 4.
26A	$\overline{\text{RFSHB}}$	Refresh (active low).
26B	NOT USED	Not used.
27A	$\overline{\text{EXROM}}$	Extension ROM Enable (active low).
27B	R	Color Signal - Red.
28A	$\overline{\text{ROSCS}}$ OUTPUT	ROS (ROM Oriented Software) ship select.
28B	G	Color Signal - Green.
28A	$\overline{\text{BE}}$ INPUT	Bank Enable (active low).
29B	B	Color Signal - Blue. TTL POSITIVE
30A	GND	Ground.
30B	GND	Ground.
31A	SOUND	Analog Sound Signal Output (Ø-5 V).
31B	VIDEO	Composite Video Signal Output. 1V
32A	GND	
32B	GND	

NOTE

All A pins are on component side signal of board
All B pins are on soldering side signal of board

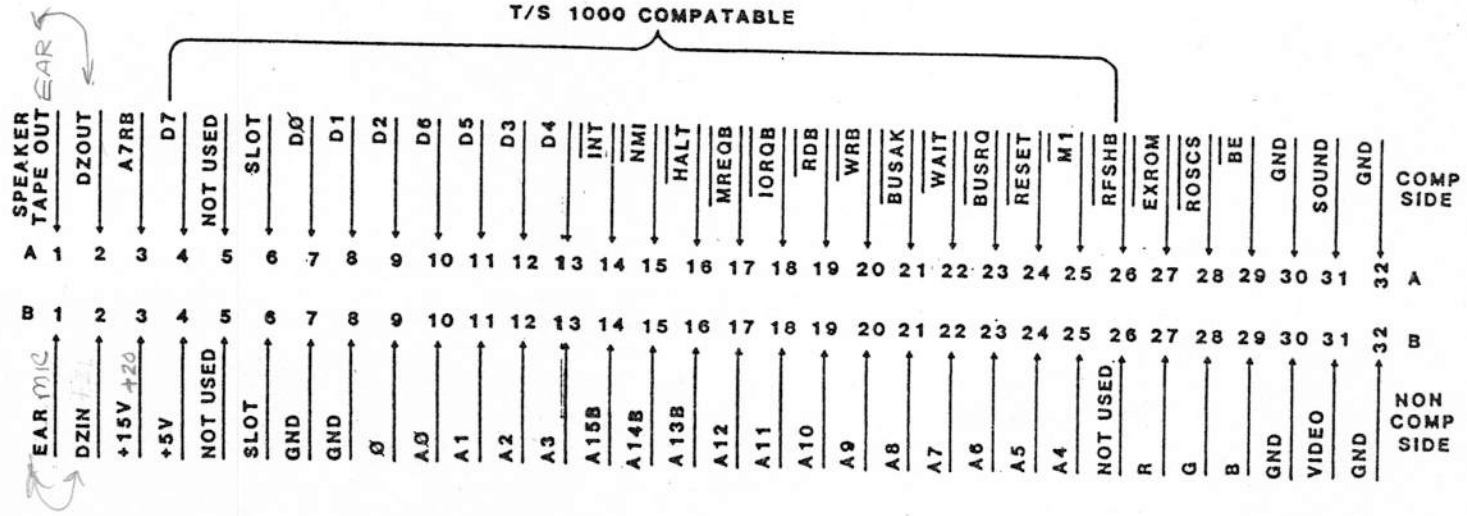


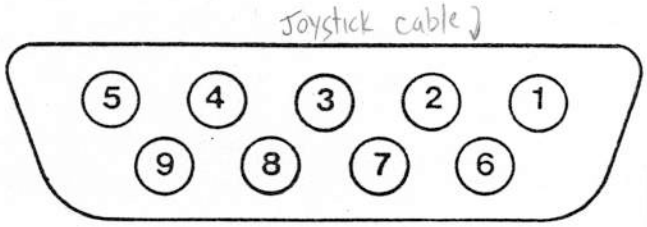
Figure 4-2. EDGE CONNECTOR

4.4 JOYSTICK CONNECTORS

The T/S 2000 has the built-in capability to use two eight-position joysticks. These joysticks are industry-standard. The connectors are industry-standard 9-pin "D" type connectors. The layout of the connector, and the function of each pin is given below in Table 4-3 and Figure 4-3.

Table 4-3
JOYSTICK CONNECTOR SIGNAL ASSIGNMENT

PINS	SIGNAL NAME	FUNCTION
1	$\overline{\text{DIR1}}$ UP	Direction One (active low).
2	$\overline{\text{DIR2}}$ DOWN	Direction Two (active low).
3	$\overline{\text{DIR3}}$ LEFT	Direction Three (active low).
4	$\overline{\text{DIR4}}$ RIGHT	Direction Four (active low).
5	-	Not used.
6	$\overline{\text{BUTTON}}$	Button Input (active low).
7	-	Not used.
8	$\overline{\text{RDSTB}}$ 3 µS	Read strobe. active low - no pullup)
9	-	Not used.



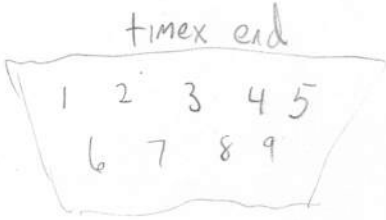


Figure 4-3. JOYSTICK CONNECTOR

SECTION V

CHARACTER SET AND KEYBOARD

5.1 INTRODUCTION

For the T/S 2000, this section provides a description of the keyboard and details the character set.

5.2 DESCRIPTION OF KEYBOARD

The T/S 2000 keyboard is not the same as the SPECTRUM keyboard, and has the following features:

- Hard key typewriter style keyboard with 42-keys, including two shift keys, and a SPACE bar
- Full upper and lower-case characters (with descenders)
- 16-standard predefined graphics keys and 21-user definable graphics keys
- Auto repeat on all keys
- Audible keyclick (tone) provided for each key
- Single key entry of BASIC command keywords and tokens

The T/S 2000's entire character set may be redefined by altering the system variable CHARS. This variable points to 256-bytes before the beginning of the bit map describing the characters in the character set. The user may thus create alternate character fonts, or foreign language characters sets.

5.2.1 Tokens

The T/S 2000 uses a system of tokens to increase effective memory utilization. Instead of storing, for example, the keyword PRINT as a string of 5-characters, the T/S 2000 represents PRINT with a single byte. The T/S 2000 has reserved characters 165-255 to represent the tokens used in the computer.

When you output, for example, CHR$ 166, the computer's output routine expands the token, and actually sends six (6) characters (I, N, K, E, Y, and $) to the current output device. This technique creates a two-fold benefit: firstly, typing time is reduced, and secondly, memory space is conserved.

5.2.2 Spectrum vs T/S 2000 Keyboard

The following list describes changes to the Spectrum keyboard:

1) In CAPS LOCK mode SPACE depressed = SPACE

2) Both CAPS SHIFT and SPACE depressed = BREAK

3) Copyright symbol = RESET

4) Vertical bar = STICK

5) Left curly bracket = ON ERR

6) Right curly bracket = SOUND

7) Tilde = FREE

8) DELETE is context dependent. In K-mode, the keyword DELETE is assumed for the command described in paragraph 6.10. Otherwise the DELETE key functions in the usual manner deleting the last character typed.

5.3 T/S 2000 CHARACTER SET

The T/S 2000 uses ASCII. With a few minor exceptions, the T/S 2000's character set is fully compatible with ASCII. Table 5-1 is a listing of the T/S 2000 character set.

Table 5-1
T/S 2000 CHARACTER SET

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
Ø	Not used	ØØ	nop	rlc b	
1	Not used	Ø1	ld bc, NN	rlc c	
2	Not used	Ø2	ld (bc), a	rlc d	
3	Not used	Ø3	inc bc	rlc e	
4	Not used	Ø4	inc b	rlc h	
5	Not used	Ø5	dec b	rlc l	
6	PRINT Comma	Ø6	ld b, N	rlc (hl)	
7	EDIT	Ø7	rlca	rlc a	
8	Cursor Left	Ø8	ex af, af'	rrc b	
9	Cursor Right	Ø9	add hl, bc	rrc c	
10	Cursor Down	ØA	ld a, (bc)	rrc d	

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
11	Cursor Up	ØB	dec bc	rrc e	
12	DELETE	ØC	inc c	rrc h	
13	ENTER	ØD	dec c	rrc l	
14	Number (slug)	ØE	ld c, N	rrc (hl)	
15	Not used	ØF	rrca	rrc a	
16	INK Control	1Ø	djnz DIS	rl b	
17	PAPER Control	11	ld de, NN	rl c	
18	FLASH Control	12	ld (de), a	rl d	
19	BRIGHT Control	13	inc de	rl e	
20	INVERSE Control	14	inc d	rl h	
21	OVER Control	15	dec d	rl l	
22	AT Control	16	id d, N	rl (hl)	
23	TAB Control	17	rla	rl a	
24	Not used	18	jr DIS	rr b	
25	Not used	19	add hl, de	rr c	
26	Not used	1A	ld a, (de)	rr d	
27	Not used	1B	dec de	rr e	
28	Not used	1C	inc e	rr h	
29	Not used	1D	dec e	rr l	
30	Not used	1E	ld e, N	rr (hl)	
31	Not used	1F	rra	rr a	
32	Space	2Ø	jr nz, DIS	sla b	
33	!	21	ld hl, NN	sla c	
34	"	22	ld (NN), hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h, N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z, DIS	sra b	
41	)	29	add hl, hl	sra c	
42	*	2A	ld hl, (NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld, l, N	sra (hl)	
47	/	2F	cpl	sra a	
48	Ø	3Ø	jr nc, DIS		
49	1	31	ld sp, NN		
50	2	32	ld (NN), a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl), N		
55	7	37	scf		

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
56	8	38	jr c, DIS	srl b	
57	9	39	add hl, sp	srl c	
58	:	3A	ld a, (NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a, N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	4Ø	ld b, b	bit 0, b	in b, (c)
65	A	41	ld b, c	bit 0, c	out(c), b
66	B	42	ld b, d	bit 0, d	sbc hl, bc
67	C	43	ld b, e	bit 0, e	ld(NN), bc
68	D	44	ld b, h	bit 0, h	neg
69	E	45	ld b, l	bit 0, l	retn
70	F	46	ld b, (hl)	bit 0,(hl)	im 0
71	G	47	ld b, a	bit 0, a	ld i, a
72	H	48	ld c, b	bit 1, b	in c, (c)
73	I	49	ld c, c	bit 1, c	out(c), c
74	J	4A	ld c, d	bit 1, d	adc hl,bc
75	K	4B	ld c, e	bit 1, e	ld bc,(NN)
76	L	4C	ld c, h	bit 1, h	
77	M	4D	ld c, l	bit 1, l	reti
78	N	4E	ld c, (hl)	bit 1,(hl)	
79	O	4F	ld c, a	bit 1, a	ld r, a
80	P	5Ø	ld d, b	bit 2, b	in d, (c)
81	Q	51	ld d, c	bit 2, c	out(c), d
82	R	52	ld d, d	bit 2, d	sbc hl, de
83	S	53	ld d, e	bit 2, e	ld (NN), de
84	T	54	ld d, h	bit 2, h	
85	U	55	ld d, l	bit 2, l	
86	V	56	ld d, (hl)	bit 2, (hl)	im 1
87	W	57	ld d, a	bit 2, a	ld a, i
88	X	58	ld e, b	bit 3, b	in e, (c)
89	Y	59	ld e, c	bit 3, c	out(c), e
90	Z	5A	ld e, d	bit 3, d	adc hl, de
91	[	5B	ld e, e	bit 3, e	ld de, (NN)
92	/	5C	ld e, h	bit 3, h	
93	]	5D	ld e, l	bit 3, l	
94	↑	5E	ld e, (hl)	bit 3, (hl)	im 2
95		5F	ld e, a	bit 3, a	ld a, r
96	£	6Ø	ld h, b	bit 4, b	in h, (c)
97	a	61	ld h, c	bit 4, c	out(c), h
98	b	62	ld h, d	bit 4, d	sbc hl, hl
99	c	63	ld h, e	bit 4, e	ld (NN), hl
100	d	64	ld h, h	bit 4, h	

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
101	e	65	ld h, l	bit 4, l	
102	f	66	ld h, (hl)	bit 4, (hl)	
103	g	67	ld h, a	bit 4, a	rrd
104	h	68	ld l, b	bit 5, b	in l, (c)
105	i	69	ld l, c	bit 5, c	out(c), l
106	j	6A	ld l, d	bit 5, d	adc hl, hl
107	k	6B	ld l, e	bit 5, e	ld hl, (NN)
108	l	6C	ld l, h	bit 5, h	
109	m	6D	ld l, l	bit 5, l	
110	n	6E	ld l, (hl)	bit 5, (hl)	
111	o	6F	ld l, a	bit 5, a	rld
112	p	7Ø	ld (hl), b	bit 6, b	in f, (c)
113	q	71	ld (hl), c	bit 6, c	
114	r	72	ld (hl), d	bit 6, d	sbc hl, sp
115	s	73	ld (hl), e	bit 6, e	ld (NN), sp
116	t	74	ld (hl), h	bit 6, h	
117	u	75	ld (hl), l	bit 6, l	
118	v	76	halt	bit 6, (hl)	
119	w	77	ld (hl), a	bit 6, a	
120	x	78	ld a, b	bit 7, b	inc a, (c)
121	y	79	ld a, c	bit 7, c	out(c), a
122	z	7A	ld a, d	bit 7, d	adc hl, sp
123	{ (ONERR)	7B	ld a, e	bit 7, e	ld sp, (NN)
124	STICK	7C	ld a, h	bit 7, h	
125	} (SOUND)	7D	ld a, l	bit 7, l	
126	FREE	7E	ld a, (hl)	bit 7, (hl)	
127	© (RESET)	7F	ld a, a	bit 7, a	
128		8Ø	add a, b	res 0, b	
129		81	add a, c	res 0, c	
130		82	add a, d	res 0, d	
131		83	add a, e	res 0, e	
132		84	add a, h	res 0, h	
133		85	add a, l	res 0, l	
134		86	add a, (hl)	res 0, (hl)	
135		87	add a, a	res 0, a	
136		88	adc a, b	res 1, b	
137		89	adc a, c	res 1, c	
138		8A	adc a, d	res 1, d	
139		8B	adc a, e	res 1, e	
140		8C	adc a, h	res 1, h	

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
141	▙	8D	adc a, l	res 1, l	
142	▟	8E	adc a, (hl)	res 1, (hl)	
143	▛	8F	adc a, a	res 1, a	
144	(a)	9Ø	sub b	res 2, b	
145	(b)	91	sub c	res 2, c	
146	(c)	92	sub d	res 2, d	
147	(d)	93	sub e	res 2, e	
148	(e)	94	sub h	res 2, h	
149	(f)	95	sub l	res 2, l	
150	(g)	96	sub (hl)	res 2, (hl)	
151	(h)	97	sub a	res 2, a	
152	(i)	98	sbc a, b	res 3, b	
153	(j) user	99	sbc a, c	res 3, c	
154	(k) graphics	9A	sbc a, d	res 3, d	
155	(l)	9B	sbc a, e	res 3, e	
156	(m)	9C	sbc a, h	res 3, h	
157	(n)	9D	sbc a, l	res 3, l	
158	(o)	9E	sbc a, (hl)	res 3, (hl)	
159	(p)	9F	sbc a, a	res 3, a	
160	(q)	AØ	and b	res 4, b	ldi
161	(r)	A1	and c	res 4, c	cpi
162	(s)	A2	and d	res 4, d	ini
163	(t)	A3	and e	res 4, e	outi
164	(u)	A4	and h	res 4, h	
165	RND	A5	and l	res 4, l	
166	INKEY$	A6	and (hl)	res 5, (hl)	
167	PI	A7	and a	res 4, a	
168	FN	A8	xor b	res 5, b	ldd
169	POINT	A9	xor c	res 5, c	cpd
170	SCREEN$	AA	xor d	res 5, d	ind
171	ATTR	AB	xor e	res 5, e	outd
172	AT	AC	xor h	res 5, h	
173	TAB	AD	xor l	res 5, l	
174	VAL$	AE	xor (hl)	res 5, (hl)	
175	CODE	AF	xor a	res 5, a	
176	VAL	BØ	or b	res 6, b	ldir
177	LEN	B1	or c	res 6, c	cpir
178	SIN	B2	or d	res 6, d	inir
179	COS	B3	or e	res 6, e	otir
180	TAN	B4	or h	res 6, h	

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
181	ASN	B5	or l	res 6, l	
182	ACS	B6	or (hl)	res 6, (hl)	
183	ATN	B7	or a	res 6, a	
184	LN	B8	cp b	res 7, b	lddr
185	EXP	B9	cp c	res 7, c	cpdr
186	INT	BA	cp d	res 7, d	indr
187	SQR	BB	cp e	res 7, e	otdr
188	SGN	BC	cp h	res 7, h	
189	ABS	BD	cp l	res 7, l	
190	PEEK	BE	cp (hl)	res 7, (hl)	
191	IN	BF	cp a	res 7, a	
192	USR	CØ	ret nz	set 0, b	
193	STR$	C1	pop bc	set 0, c	
194	CHR$	C2	jp nz, NN	set 0, d	
195	NOT	C3	jp NN	set 0, e	
196	BIN	C4	call nz, NN	set 0, h	
197	OR	C5	push bc	set 0, l	
198	AND	C6	add a, N	set 0, (hl)	
199	<=	C7	rst 0	set 0, a	
200	>=	C8	ret z	set 1, b	
201	<>	C9	ret	set 1, c	
202	LINE	CA	jp z, NN	set 1, d	
203	THEN	CB	- - - - -	set 1, e	
204	TO	CC	call z, NN	set 1, h	
205	STEP	CD	call NN	set 1, l	
206	DEF FN	CE	adc a, N	set 1, (hl)	
207	CAT	CF	rst 8	set 1, a	
208	FORMAT	DØ	ret nc	set 2, b	
209	MOVE	D1	pop de	set 2, c	
210	ERASE	D2	jp nc, NN	set 2, d	
211	OPEN#	D3	out (N), a	set 2, e	
212	CLOSE#	D4	call nc, NN	set 2, h	
213	MERGE	D5	push de	set 2, l	
214	VERIFY	D6	sub N	set 2, (hl)	
215	BEEP	D7	rst 16	set 2, a	
216	CIRCLE	D8	ret c	set 3, b	
217	INK	D9	exx	set 3, c	
218	PAPER	DA	jp c, NN	set 3, d	
219	FLASH	DB	in a, (N)	set 3, e	
220	BRIGHT	DC	call c, NN	set 3, h	
221	INVERSE	DD	prefixes instructions using ix	set 3, l	

Table 5-1 (CON'T)

CODE	CHARACTER	HEX	Z80 ASSEMBLER	-AFTER CB	-AFTER ED
222	OVER	DE	sbc a, N	set 3, (hl)	
223	OUT	DF	rst 24	set 3, a	
224	LPRINT	EØ	ret po	set 4, b	
225	LLIST	E1	pop hl	set 4, c	
226	STOP	E2	jp po, NN	set 4, d	
227	READ	E3	ex (sp, hl	set 4, e	
228	DATA	E4	call po, NN	set 4, h	
229	RESTORE	E5	push hl	set 4, l	
230	NEW	E6	and N	set 4, (hl)	
231	BORDER	E7	rst 32	set 4, a	
232	CONTINUE	E8	ret pe	set 5, b	
233	DIM	E9	jp (hl)	set 5, c	
234	REM	EA	jp pe, NN	set 5, d	
235	FOR	EB	ex de, hl	set 5, e	
236	GO TO	EC	call pe, NN	set 5, h	
237	GO SUB	ED	- - - -	set 5, l	
238	INPUT	EE	xor N	set 5, (hl)	
239	LOAD	EF	rst 40	set 5, a	
240	LIST	FØ	ret p	set 6, b	
241	LET	F1	pop af	set 6, c	
242	PAUSE	F2	jp p, NN	set 6, d	
243	NEXT	F3	di	set 6, e	
244	POKE	F4	call p, NN	set 6, h	
245	PRINT	F5	push af	set 6, l	
246	PLOT	F6	or N	set 6, (hl)	
247	RUN	F7	rst 48	set 6, a	
248	SAVE	F8	ret m	set 7, b	
249	RANDOMIZE	F9	ld sp, hl	set 7, c	
250	IF	FA	jp m, NN	set 7, d	
251	CLS	FB	ei	set 7, e	
252	DRAW	FC	call m, NN	set 7, h	
253	CLEAR	FD	prefixes instructions using iy	set 7, l	
254	RETURN	FE	cp N	set 7, (hl)	
255	COPY	FF	rst 56	set 7, a	

5.4 SCANNING THE KEYBOARD

The T/S 2000's Z80 microprocessor is interrupted every 1/60th of a second to scan the keyboard. This feature may be disabled by setting bit 6 in I/O port FFH. The routine used to perform this function is called UPD_K, and may be accessed via the function dispatcher. It returns with bit 5 of FLAGS set if a key is depressed, and both the accumulator and LAST K contain the character code for the key depressed.

Thus the T/S 2000's keyboard may be scanned in one of two ways. First, you may simply perform a HALT instruction. Then look at bit 5 of FLAGS, and LAST K. Second, you may use the function dispatcher to access UPD-K directly, and then look at bit 5 of FLAGS and the accumulator. Remember not to execute a HALT instruction if interrupts are disabled either by software (a DI instruction) or by hardware as noted above.

SECTION VI

SYSTEM SOFTWARE

6.1 INTRODUCTION

This section provides a description of system software, data structures, system variables, display modes, interrupts, function dispatcher, I/O channels, initialization, bank switching, commands unique to the T/S 2000 and mass storage devices. Appropriate explanations are found in this section.

6.2 DATA STRUCTURES

Figure 6-1 illustrates the layout of the data structures which are utilized by the system software and Figures 6-2 through 6-9 supplement this diagram.

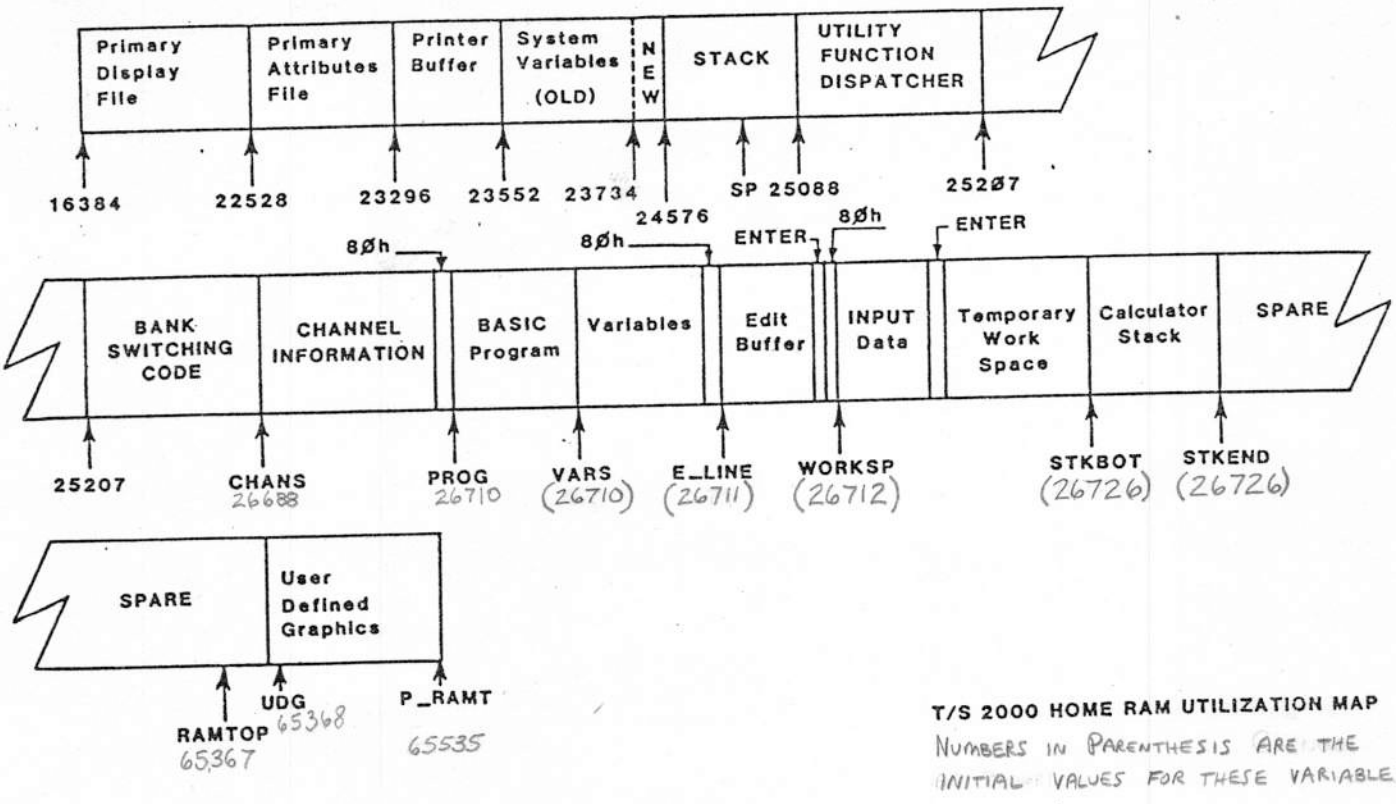


Figure 6-1. DATA STRUCTURE LAYOUT

Each line of BASIC program has the form illustrated by Figure 6-2.

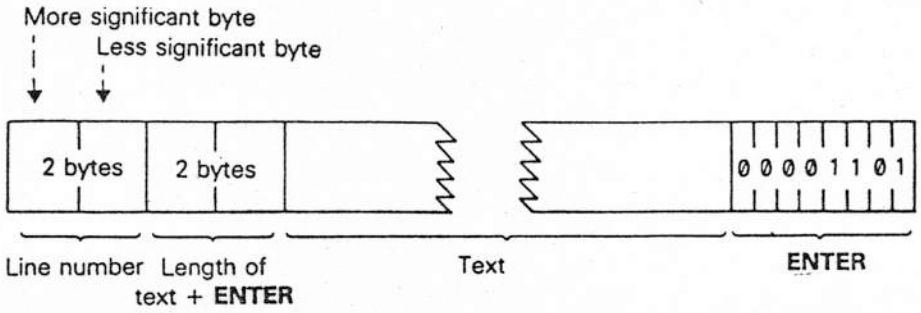


Figure 6-2. BASIC PROGRAM LINE LAYOUT

Note that in contrast with all other cases of two-byte numbers in the system, the line number here is stored with its most significant byte first: that is to say, in the order that you write them down in.

A numerical constant in the program is followed by its binary form, using the character CHR$ 14 followed by five bytes for the number itself.

The variables have different formats according to their different natures. The letters in the names should be imagined as starting off in lower case. This order is illustrated by Figure 6-3.

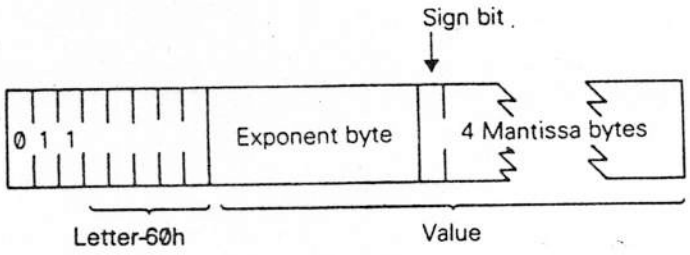


Figure 6-3. NUMBER WHOSE NAME IS ONE LETTER ONLY

Figure 6-4 illustrates a situation when a number whose name is longer than one letter is used:

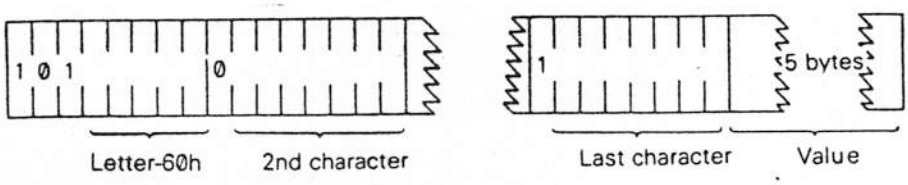


Figure 6-4. LONGER NAME DATA STRUCTURE

An array of numbers is illustrated by Figure 6-5:

The order of the elements is:
first, the elements for which the first subscript is 1
next, the elements for which the first subscript is 2
next, the elements for which the first subscript is 3
and so on for all possible values of the first subscript.

The elements with a given first subscript are ordered in the same way using the second subscript, and so on down to the last.

As an example, the elements of the 3*6 array b are stored in the order b(1,1) b(1,2) b(1,3) b(1,4) b(1,5) b(1,6), b(2,1) b(2,2) ... b(2,6) b(3,1) b(3,2) ... b(3,6)

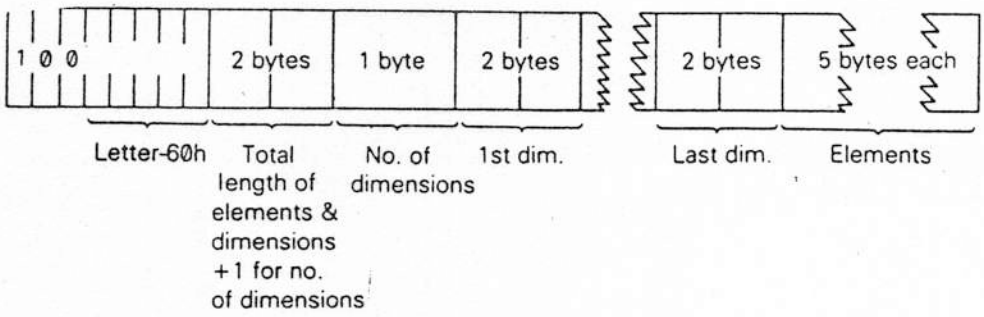


Figure 6-5. ARRAY DATA STRUCTURE

Structure of a control variable for a FOR - NEXT loop is illustrated by Figure 6-6.

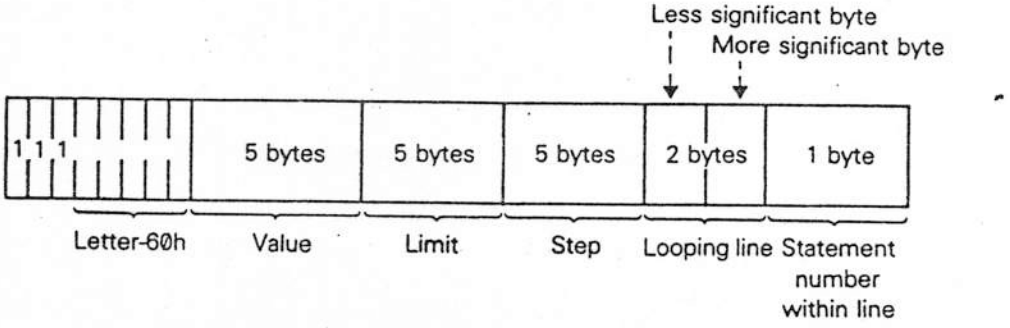


FIGURE 6-6. FOR NEXT LOOP DATA STRUCTURE

String structures are illustrated by Figure 6-7:

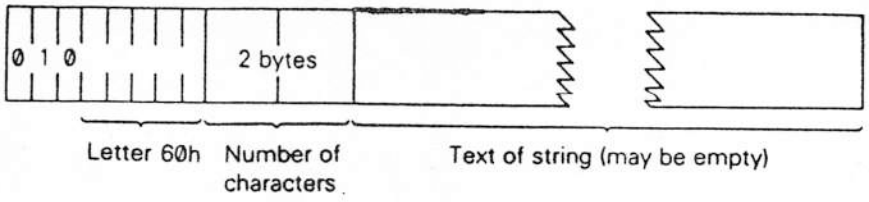


Figure 6-7. STRING DATA STRUCTURE

Figure 6-8 illustrates the data structure of an array of characters:

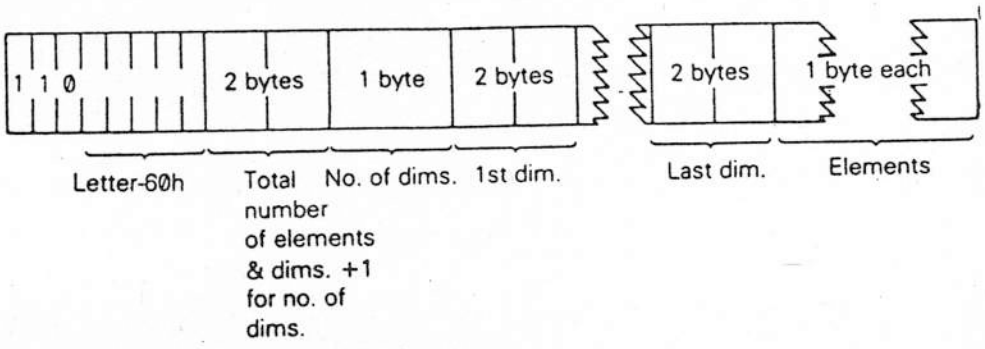


Figure 6-8. CHARACTER ARRAY DATA STRUCTURE

The calculator is the part of the BASIC system that deals with arithmetic, and the numbers on which it is operating are held mostly on the calculator stack.

The spare part contains the space so far unused.

The machine stack is the stack used by the Z80 processor to hold return addresses and so on.

Any number (except Ø) can be written uniquely as

$$\pm m * 2^{e}$$

where + is the sign
m is the mantissa, and lies between 1/2 and 1(it cannot be 1),
and e is the exponent, a whole number (possibly negative).

Suppose you write m in the binary scale. Because it is a fraction, it will have a binary point (like the decimal point in the scale of ten) and then a binary fraction (like a decimal fraction): so in binary,

a half is written .1
a quarter is written .Ø1
three quarters is written .11
a tenth is written .ØØ11ØØ11ØØ11ØØ11ØØ11 ... and so on. With our number m, because it is less than 1, there are no bits before the binary point, and because it is at least 1/2, the bit immediately after the binary point is a 1.

To store the number in the computer, we use five bytes, as follows:

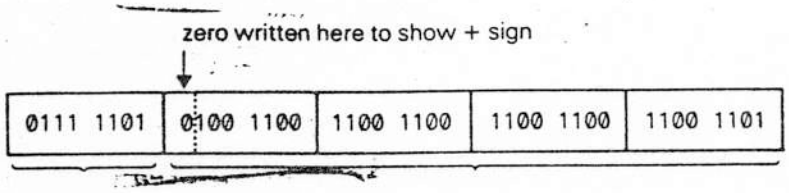
1. Write the first eight bits of the mantissa in the second byte (we know that the first bit is 1), the second eight bits in the third byte, the third eight bits in the fourth byte and the fourth eight bits in the fifth byte,

2. Replace the first bit in the second byte - which we know is 1 - by the sign: Ø for plus, 1 for minus.

3. Write the exponent + 128 in the first byte. For instance, suppose our number is 1/1Ø

$$1/1Ø = 4/5 \times 2^{-3}$$

Thus the mantissa m is .11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ in binary (since the 33rd bit is 1, we shall round the 32nd up from Ø to 1), and the exponent e is -3. Applying our three rules gives the five bytes illustrated in Figure 6-9.



-3+128 Mantissa 4/5 except that the first bit should be 1 for exponent.

Figure 6-9. ZERO WRITTEN HERE TO SHOW + SIGN

There is an alternative way of storing whole numbers between -65535 and +65535:

1. The first byte is Ø.
2. The second byte is Ø for a positive number, FFH for a negative one.
3. The third and fourth bytes are the less and more significant bytes of the number (or the number +131072 if it is negative).
4. The fifth byte is Ø.

6.3 SYSTEM VARIABLES

Refer to the list of system variables outlined in Table 6-1.

X The variable should not be poked because the system might crash.
N Poking the variable will have no lasting effect.

The number in column 1 is the number of bytes in the variable. For two bytes, the first one is the less significant byte - the reverse of what you might expect. So to poke a value v to a two-byte variable at address n, use

```
POKE n,v - 256*INT (v/256)
POKE n+1,INT (v/256)
```

and to peek its value, use the expression

```
PEEK n+256*PEEK (n+1)
```

Table 6-1
SYSTEM VARIABLES

NOTES	ADDRESS	NAME	CONTENTS
N8	23552	KSTATE	Used in reading the keyboard.
N1	23560	LAST K	Stores newly pressed key.
1	23561	REPDEL	Time - in 60this of a second that a key must be held down before it repeats. This starts off at 35, but you can POKE in other values.
1	23562	REPPER	Delay - in 60this of a second between successive repeats of a key held down: initially 5.
N2	23563	DEFADD	Address of arguments of user-defined function if one is being evaluated; otherwise Ø.
N1	23565	K DATA	Stores 2nd byte of color controls entered from keyboard.
N2	23566	TVDATA	Stores bytes of color, AT and TAB controls going to television.
X38	23568	STRMS	Addresses of channels attached to streams.
2 (3CØØ)	23606	CHARS	256 less than address of character set (which starts with space and carries on to the copyright symbol). Normally in ROM, but you can set up your own in RAM and make CHARS point to it.
1	23608	RASP	Length of warning buzz.
1	23609	PIP	Length of keyboard click.
1	23610	ERR NR	1 less than the report code. Starts off at 255 (for -1) so PEEK 23610 gives 255.
X1	23611	FLAGS	Various flags to control the BASIC system.
X1	23612	TV FLAG	Flags associated with the television.
X2	23613	ERR SP	Address of item on machine stack to be used as error return.
N2	23615	LIST SP	Address of return address from automatic listing.
N1	23617	MODE	Specifies K, L, C, E or G cursor.
2	23618	NEWPPC	Line to be jumped to.
1	23620	NSPPC	Statement number in line to be jumped to. Poking first NEWPPC and then NSPPC forces a jump to a specified statement in a line.
2	23621	PPC	Line number of statement currently being executed.
1	23623	SUBPPC	Number within line of statement being executed.
1 (38)	23624	BORDCR	Border color * 8; also contains the attributes normally used for the lower half of the screen.
2	23625	E PPC	Number of current line (with program cursor).
X2 (6856)	23627	VARS	Address of variables.
N2	23629	DEST	Address of variable in assignment.

Table 6-1 (CON'T)

NOTES	ADDRESS	NAME	CONTENTS
X2 (6840)	23631	CHANS	Address of channel data.
X2	23633	CURCHL	Address of information currently being used for input and output.
X2 (6856)	23635	PROG	Address of BASIC program.
X2	23637	NXTLIN	Address of next line in program.
X2	23639	DATADD	Address of terminator of last DATA item.
X2	23641	E LINE	Address of command being typed in.
2	23643	K CUR	Address of cursor.
X2	23645	CH ADD	Address of the next character to be interpreted: the character after the argument of PEEK, or the NEWLINE at the end of a POKE statement.
2	23647	X PTR	Address of the character after the [?] marker.
X2	23649	WORKSP	Address of temporary work space.
X2	23651	STKBOT	Address of bottom of calculator stack.
X2	23653	STKEND	Address of start of spare space.
N1	23655	BREG	Calculator's b register.
N2	23656	MEM	Address of area used for calculator's memory. (Usually MEMBOT, but not always).
1	23658	FLAGS2	More flags.
X1	23659	DF SZ	The number of lines (including one blank line) in the lower part of the screen.
2	23660	S TOP	The number of the top program line in automatic listings.
2	23662	OLDPPC	Line number to which CONTINUE jumps.
1	23664	OSPCC	Number within line of statement to which CONTINUE jumps.
N1	23665	FLAGX	Various flags.
N2	23666	STRLEN	Length of string type destination in assignment.
N2	23668	T ADDR	Address of next item in syntax table.
2	23670	SEED	The seed for RND. This is the variable that is set by RANDOMIZE.
3	23672	FRAMES	3-byte (least significant first), frame counter. Incremented every 16ms.
2	23675	UDG	Address of 1st user-defined graphic. You can change this for instance to save space by having fewer user-defined graphics.
1	23677	COORDS	x-coordinate of last point plotted.
1	23678		y-coordinate of last point plotted.
1	23679	P POSN	33-column number of printer position.
1	23680	PR CC	Less significant byte of address of next position for LPRINT to print at (in printer buffer).
1	23681		Not used.
2	23682	ECHO E	33-column number and 24-line number (in lower half) of end of input buffer.

Table 6-1 (CON'T)

NOTES	ADDRESS	NAME	CONTENTS
2	23684	DF CC	Address in display file of PRINT position.
2	23686	DFCCL	Like DF CC for lower part of screen.
X1	23688	S POSN	33-column number for PRINT position.
X1	23689		24-line number for PRINT position.
X2	23690	SPOSNL	Like S POSN for lower part.
1	23692	SCR CT	Counts scrolls: it is always 1 more than the number of scrolls that will be done before stopping with scroll?. If you keep poking this with a number bigger than 1 (say 255), the screen will scroll on and on without asking you.
1	23693	ATTR P	Permanent current colors, etc. (as set up by color statements).
1	23694	MASK P	Used for transparent colors, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from ATTR P, but from what is already on the screen.
N1	23695	ATTR T	Temporary current colors, etc. (as set up by color items).
N1	23696	MASK T	Like MASK P, but temporary.
1	23697	P FLAG	More flags.
N30	23698	MEMBOT	Calculator's memory area; used to store numbers that cannot conveniently be put on the calculator stack.
2	23728		Not used.
2 FF57	23730	RAMTOP	Address of last byte of BASIC system area.
2	23732	P-RAMT	Address of last byte of physical RAM.
2	23734	ERRLN	Line number to GOTO on error.
2	23736	ERRC	Line number in which error occurred.
1	23738	ERRS	Statement number within line in which error occurred.
1	23739	ERRT	Error number (Report Code).
2X	23740	SYSCON	Pointer to the System Configuration Table
1X	23742	MAXBNK	Number of Expansion Banks in System
1X	23743	CURCBN	Current Channel Bank Number
2X	23744	MSTBOT	Address of location above machine stack.
1X ϕ	23746	VIDMOD	Video Mode. Non-zero if the second display file is open for use.

6.4 DISPLAY MODES

The T/S 2000 has available to its user, four display modes.

There are two display files and two attribute files. Their addresses are shown below.

Display File One	4ØØØ - 57FF
Attribute File One	58ØØ - 5AFF
Display File Two	6ØØØ - 77FF
Attribute File Two	78ØØ - 7AFF

The second set of files (D_FILE_2/A_FILE_2) are usually overlaid by some of the system software's data structures. The function dispatcher has a function that is used to switch display modes, and these data structures are moved out of the way. Refer to paragraph 6-6.

The display modes are listed below:

Display Mode 1 -- Normal (D_FILE_1)
24-rows of 32-characters
256-pixels horizontal x 192-pixels vertical
One attribute defines one character position (of 8x8 pixels)
Primary attributes page (A_FILE_1)

Display Mode 2 -- 64-Column Mode (D_FILE_1 and D_FILE_2)
24-rows of 64-characters
512-pixels horizontal x 192-vertical
Only paper color is selectable. High brightness
,no flashing, and ink color are chosen
by hardware.

PAPER	yields	INK
black		white
blue		yellow
red		cyan
magenta		green
green		magenta
cyan		red
yellow		blue
white		black

Display Mode 3 -- Secondary Screen Page (D_FILE_2)
24-rows of 32-characters
256-pixels horizontal by 192-pixels vertical
Secondary attributes page (A_FILE_2)

Display Mode 4 -- Ultra High Color Resolution (D_FILE_1 and D_FILE_2)
Like Display Mode 1. Each byte in first display file defines a given eight-pixels. Each corresponding byte in the second display byte defines the attributes for the given pixel data. This means that for every pixel row of eight-pixels (within a character position), one attribute can be defined for brightness, flashing, ink and paper colors.

6.5 INTERRUPT HANDLING

The following information details the processes used for handling interrupts on the T/S 2000.

Non-maskable interrupts are to be handled in the following fashion:

The occurrence of an NMI causes a return address to be pushed to the RAM location pointed to by SP and a restart to location 66H occurs. The interrupt handler (in whichever bank is selected) services the interrupt and returns. Since the state of the system includes the bank selection status of all the banks, all interrupt handlers must save and restore these statuses as part of their operation.

Maskable interrupts are to be handled in the same fashion as NMI's. There are interrupt fielders at location 38H in each bank other than the Home Bank, and an interrupt handler at location 38H in the Home Bank.

The interrupt handlers for both NMI's and INT's determine whether the interrupting device supplies its own interrupt handling. If this is the case, then control is passed to the address and bank specified for the device as specified in the System Configuration Table. Maskable and non-maskable interrupt handling is available in all banks.

When an interrupt is received, the system will poll each expansion Bank to see which bank caused the interrupt. If the interrupt was caused by an expansion Bank, the interrupt handler for that bank will be called. Please note that all banks should be queried, as, when an interrupt is received, it may have been caused by several banks simultaneously.

6.6 FUNCTION DISPATCHER

Located at location 6200H (or FA5ØH if D/FILE/2 is open), is a function dispatcher which will, when passed two arguments on the machine stack, perform various functions for applications and systems programmers.

The arguments passed to the function dispatcher are contained in a single word. Bits Ø-14 contain the service code for the function you are requesting, and bit 15 is a jump/call selection flag. If bit 15 is set, the dispatcher will jump to the function you have requested. If bit 15 is reset, the dispatcher will call the routine and then return to its (the dispatcher's) caller.

Table 6-2 is a list of functions currently provided by the dispatcher.

Table 6-2
T/S 2000 FUNCTIONS

FUNCTION	SERVICE CODE	ACTION
W_TAPE	Ø	Write a block to tape.
R_TAPE	1	Read a block from tape.
RD_BIT	2	Read a bit from tape.
R_EDGE	3	Read an edge from tape.
SLVM	4	General tape routine.
LOAD	5	Load.
MERGE	6	Merge.
SAVE	7	Save.
CHNG_VID	8	Change video mode.
W_BORD	9	Write border color.
	10	Reserved.
	11	Reserved.
	12	Reserved.
	13	Reserved.
GET_STATUS	14	
GET_NUMBER	15	
BANK_ENABLE	16	Refer to paragraph 6.9.
GOTO_BANK	17	
CALL_BANK	18	
XFER_BANK	19	
	20	Reserved.
	21	Reserved.
	22	Reserved.
	23	Reserved.
	24	Reserved.
UPD_K	25	Scan keyboard.
PARP	26	Sound routine.
BEEP	27	BEEP command.
K_DUMP	28	COPY command.
SENDTV	29	Send char. to screen.
SETAT	30	Set print position.
STTBYT	31	Fix attribute byte.

Table 6-2 (CON'T)

FUNCTION	SERVICE CODE	ACTION
R_ATTS	32	Temp. Atts., Perm. Atts.
CLLHS	33	Clear lower half of screen.
CLS	34	Clear entire screen.
DUMPPR	35	Printer buffer sent to print.
PRSCAN	36	Send scan to printer.
DESLUG	37	Remove slugs from line buffer.
K_NEW	38	NEW command.
INIT	39	Initialize.
INCH	4Ø	Input character.
SELECT	41	Select current stream.
INSERT	42	Insert bytes.
RESET	43	Reset calculator stack.
CLOSE	44	CLOSE command.
CLCHAN	45	Close channel.
OPEN	46	OPEN command.
OPCHAN	47	Open channel.
CAT	48	CAT command.
DELETE	49	DELETE command.
FORMAT	5Ø	FORMAT command.
MOVE	51	MOVE command.
FLASHA	52	Flash char. in A to screen.
FINDL	53	Find BASIC line.
SUBLIN	54	Find sub-line.
RECLEN	55	Record length.
DELREC	56	Delete record.
PUT_LN	57	Send line to output.
SYNTAX	58	Check syntax.
EXECUTE	59	Execute line.
FOR	6Ø	FOR command.
STOP	61	STOP command.
NEXT	62	NEXT command.
READ	63	READ command.
DATA	64	DATA command.
RESTBC	65	RESTORE bc.
RAND	66	RAND command.
CON'T	67	CON'T command.
JUMP	68	Jump to line.
FIX_U1	69	Fix 1 byte # from calc. stack.
FIX_U	7Ø	Fix 2 byte # from calc. stack.
CLEAR	71	CLEAR command.
CLR_BC	72	CLEAR bc.
GO_SUB	73	GOSUB command.
CHKUSR	74	Free space left.
RETURN	75	RETURN command.
PAUSE	76	PAUSE command.
BREAK?	77	Break key pressed?
DEF	78	DEF command.
K_LPR	79	LPRINT command.
K_PRIN	8Ø	PRINT command.
P_SEQ	81	Print sequence.
INPUT	82	INPUT command.

Table 6-2 (CON'T)

FUNCTION	SERVICE CODE	ACTION
I_SEQ	83	Input sequence.
NOTKB?	84	Test CURCHL=KB.
COLOR	85	Adjust attributes sysvars.
HIFLSH	86	Adjust attributes sysvars.
SCRMBL	87	Screen address calculator.
PLOT	88	PLOT command.
PLOTBC	89	Plot c, b.
GET_XY	9Ø	Get x and y.
CIRCLE	91	CIRCLE command.
DRAW	92	DRAW command.
DRAW_L	93	Draw line.
EXPRN	94	Expression Evaluator.
F_SCRN	95	Run time action for SCREEN $
F_ATTR	96	Run time action for ATTR.
RND	97	RND action.
F_PI	98	PI action.
F_INKY	99	INKEY action.
FIND_N	1ØØ	Find variable.
PSHSTR	1Ø1	Push string.
PAEDCB	1Ø2	Push A, E, D, C, B.
LET	1Ø3	LET command.
POPSTR	1Ø4	Pop string.
DIM	1Ø5	DIM command.
STKUSN	1Ø6	Stack unsigned number.
STK_A	1Ø7	Stack A.
STK_BC	1Ø8	Stack BC.
ININT	1Ø9	Read/Stack Integer.
FP2BC	11Ø	Get 16 bit #.
FP2A	111	Get 8 bit #.
OUTPUT	112	Output number on stack.
SUB	113	Subtract.
ADD	114	Add.
MULT	115	Multiple (int).
TIMES	116	Multiply (F.P.).
DIVIDE	117	Divide.
TRUNC	118	Truncate.
FLOAT	119	Force Int F.P.
INTDIV	12Ø	Integer Divide.
INT	121	INT.
EXP	122	EXP.
LN	123	LN.
ANGLE	124	Angle calculator.
COS	125	COS.
SIN	126	SIN.
TAN	127	TAN.
ATN	128	ATN.
ASN	129	ASN.
ACS	13Ø	ACS.
ROOT	131	SQR calculator.
TO_THE	132	Exponentiation.
RDCH	133	Read character.
SENDCH	134	Send character.

Table 6-2 (CON'T)

FUNCTION	SERVICE CODE	ACTION
WRCH	135	Write character
K_SCAN	136	Keyboard scan
P_LFT	137	Print cursor left
P_RT	138	Print cursor right
P_NL	139	Print newline
PUTMES	14Ø	Print message
K_CLS	141	CLS command
SCRL	142	Scrolling routine
F_PNT	143	Point action.

The following paragraph describes some of the more useful functions provided by the dispatcher.

6.6.1 W_TAPE

Write a block of data from memory to cassette tape.

This routine will write the number of bytes specified in the DE register 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:

```
Usually  Ø for data
        -1 for header
```

The format of data on tape is a leader at 806.5-Hz, one cycle of 2040-Hz, and then data. The data is MS bit first. A zero bit is 1-cycle of 2040-Hz. A one bit is 1-cycle of 1020 Hz. Lastly comes the parity byte. This is the XOR of all data bytes -- formatted as a data byte. All frequencies are nominal.

This routine aborts with an Report Code D if the BREAK key is pressed.

6.6.2 R_TAPE

Read a block of data to memory from cassette tape.

This routine reads in the number of bytes specified in the DE register from tape. The routine should be entered with the A register equal to the block type:

```
 Ø  = data
-1  = header
```

The IX register should point to the memory location into which the first data byte should be loaded.

The routine returns carry set if there were no errors. It returns carry reset if the checksums did not match, or the block consisted of fewer than DE bytes or the verify failed or the block type was wrong.

The routine will abort with an Report Code D if the BREAK key is pressed.

6.6.3 CHNG_VID

Change video mode.

This function is used to "open" and "close" the second display file (and its corresponding attributes file - D_FILE_2 and A_FILE_2). If the second display file is closed, it should not be used by the applications programmer because data structures used by the Operating System currently overlay the needed memory area.

To access this routine, you merely pass it a value in the accumulator. Valid values are shown in Table 6-3.

Table 6-3
CHANGE VIDEO MODE ACCESS CODES

VALUE	FUNCTION
Ø	Normal video mode (D/FILE/1)
1	Second display page (D_FILE_2)
2	Ultra-high-resolution color mode (D_FILE_1 and D_FILE_2)
6	64-column mode - paper black.
8+6	64-column mode - paper blue.
16+6	64-column mode - paper red.
24+6	64-column mode - paper magenta.
32+6	64-column mode - paper green.
40+6	64-column mode - paper cyan.
48+6	64-column mode - paper yellow
56+6	64-column mode - paper white.

A system variable VIDMOD, keeps track of the current display mode (see Table 6-3 above).

Note that using the second display file means that the function dispatcher, and system stack will move to the top of memory. This means that you should only open the second display file if you are operating a 48K machine.

6.6.4 BREAK?

Read the BREAK key.

This routine looks directly at the keyboard to determine if both CAPS SHIFT and SPACE are being depressed (i.e.; BREAK). It returns carry reset if the BREAK key is depressed.

6.6.5 FIND_N

Parse and find a specified variable.

This routine will skip CHADD over the identifier (variable name) specified. It will adjust bit 6 of FLAGS for the type of the variable as follows:

Set = Numeric
Reset = String

It will return with NC if the system is checking syntax, or if the variable was found when FIND_N searched the variables area (VARS) for the specified variable.

It will return with Z if array. (i.e.; '(' present if variable not found or checking syntax and array or string if running and variable found).

If running (i.e., not checking Syntax), Then HL⟶ :

Last character of identifier in record if found (NC),
First character of name in text if not found (C),

C := First character of name with bits 5 and 6 right, and bit 7 OFF.

If checking syntax, C gets bits Ø - 4 OFF, 5 and 6 correct (as above), and bit 7 OFF.

6.6.6 RDCH

Wait for a character from current channel.

This routine waits for a character from the current input stream. It places the character in A.

If the current device is a finite device, and an end of file arrives, then RDCH aborts with an Report Code 8.

6.6.7 INCH

Get character from current channel without waiting.

This routine reads a character from the current input stream without waiting. It returns C if the character is OK. It returns NC, NZ if end of file (for a finite device; i.e.: one like a disk file for which it can always be told if there are any more characters to come).

It will return NC, Z if no character yet (for an infinite device, like the keyboard).

6.6.8 SELECT

Select the current input stream.

The routine sets the current channel to be that for stream A.

Specified Streams

STREAM	ID NO.	CONNECTION
HID-K	-3	Keyboard
HID-S	-2	Screen
HID-R	-1	RAM insertion
COM-ST	Ø	Stream for commands
INP-ST	1	Stream for input data
PR-ST	2	Stream for PRINT
LPR-ST	3	Stream for LPRINT

Streams whose ID numbers are less than zero, are referred to as hidden. They are tied unalterably to specific channels.

6.6.9 SCRMBL

Screen address calculator.

This routine will set HL pointing at a byte in the primary display file, given BC holding PLOT-type coordinates. B should be holding the y-coordinate, and C should be holding the x-coordinate. The accumulator will hold the bit number in (HL) where the pixel is stored. (Zero for left-hand = MS end, one for right-hand = LS end.)

The routine preserves DE.

6.6.10 PLOTBC

Plot a point on the screen.

This routine performs the equivalent of PLOT c,b. It sets COORDS to the coordinates specified.

There are four options depending on two bits of PFLAG, as listed by Table 6-4. Bit Ø is XOR-CH. Bit 2 is INV-CH.

Table 6-4
PLOTBC OPTIONS

XOR-CH	INV-CH	FUNCTION
Ø	Ø	Set bit.
Ø	1	Reset bit.
1	Ø	Invert bit.
1	1	Leave bit.

This routine adjusts the proper attributes byte for the primary display file.

6.7 I/O CHANNELS

The T/S 2000 uses a system of I/O channels (or "streams") to create a structured environment under which input and output in and around the system may be performed.

The following paragraph describes how the T/S 2000 handles these streams. (Also see Character Output, paragraph 3.5.

6.7.1 I/O Channel Tables

The T/S 2000 uses a number of tables located in the Home ROM and RAM. These tables contain initialization and other data for the "permanent" (dumb) devices (keyboard, screen, and printer) and initialization data for I/O streams.

SELTAB is a table containing offsets to the device dependent initialization routines for the channels 'K', 'S', and 'P'. The appropriate routine is invoked whenever a channel is selected (i.e., when a PRINT # is done). This table is illustrated by Figure 6-10.

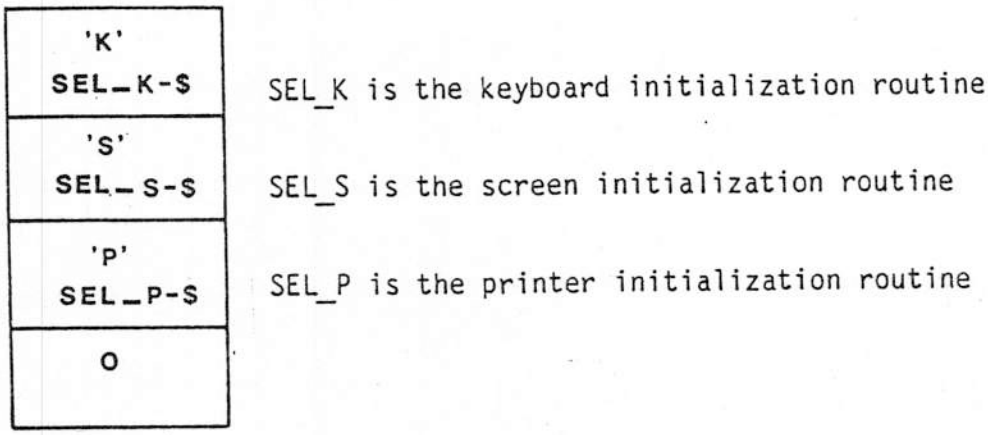
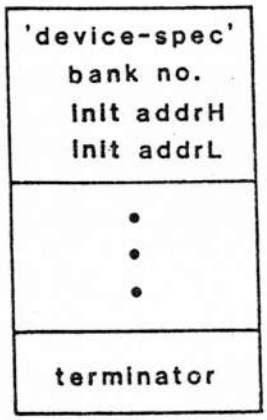


Figure 6-10. SELTAB TABLE

Any other dumb devices must also have entries in this table.

A new table (to reside in Home RAM) is needed to contain similar information for the intelligent devices. The data for this table is obtained from the Expansion Banks during initialization (refer to paragraph 6.8.4).). This table is integrated into the SCT (System Configuration Table) and has the structure illustrated by Figure 6-11.

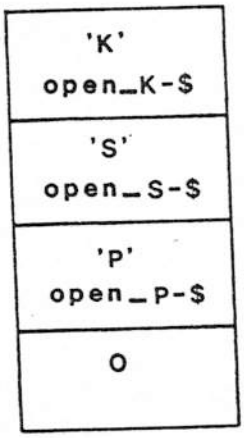


high order byte of init routine address
low order byte of init routine address

Figure 6-11. SELTAB INTELLIGENT DEVICE TABLE

This table must be searched whenever an intelligent device is selected.

SPEC_T is a table containing offsets to the stream open routines for the channels 'K', 'S', and 'P'. The appropriate routine is invoked whenever a stream is opened using OPEN #. This table has the structure illustrated by Figure 6-12.



OPEN_K is the keyboard stream open routine

OPEN_S is the screen stream open routine

OPEN_P is the printer stream open routine

Figure 6-12. SPEC_T TABLE

Any other dumb devices must also have entries in this table.

A new table (to reside in Home RAM) is needed to contain similar information for the intelligent devices. The stream open data for this table is obtained from the Expansion Banks during initialization (refer to paragraph 6.8.4). This table is integrated into the SCT and has the structure illustrated by Figure 6-13.

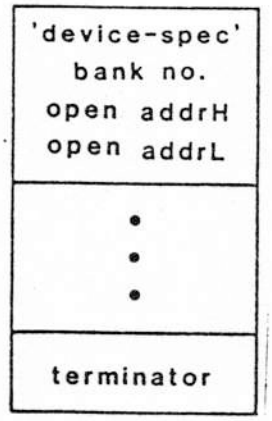


Figure 6-13. SPEC_T INTELLIGENT DEVICE TABLE

Whenever SPEC_T is searched and a match is not found, the new table must also be searched.

CL_TAB is a table containing offsets to the stream close routines for the channels 'K', 'S', and 'P'. The appropriate routine is invoked whenever a stream is closed using CLOSE #. This table has the structure illustrated by Figure 6-14.

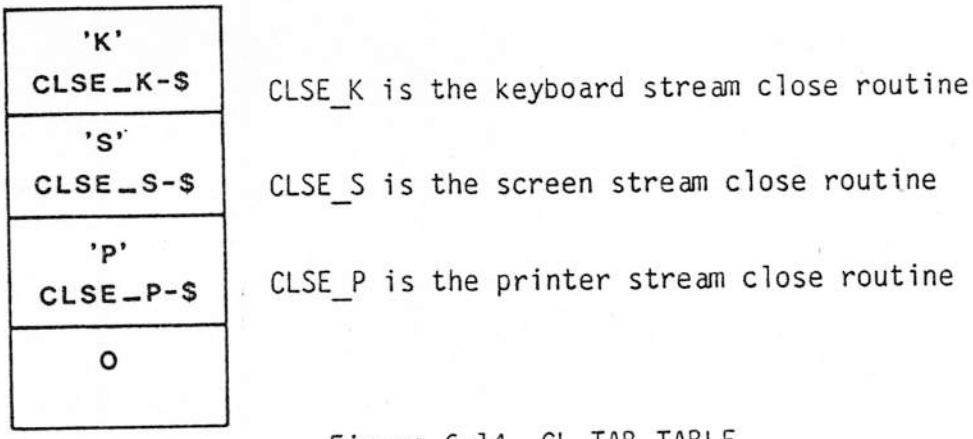
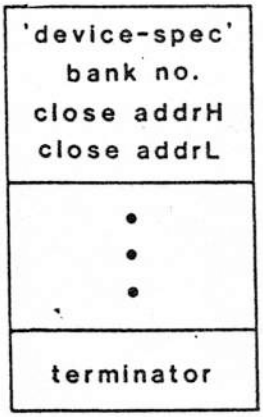


Figure 6-14. CL_TAB TABLE

Any other dumb devices must also have entries in this table.

A new table (to reside in Home RAM) is needed to contain similar information for the intelligent devices. The stream close data for this table is obtained from the Expansion Banks during initialization (refer to paragraph 6.8.4). This table is integrated into the SCT and has the structure illustrated by Figure 6-15.

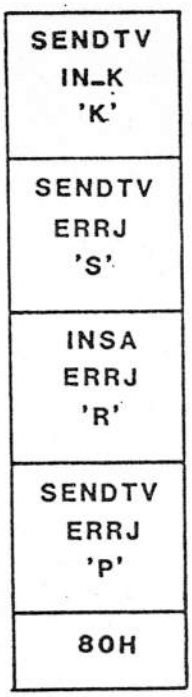


high order byte of close routine address
low order byte of close routine address

Figure 6-15. CL_TAB INTELLIGENT DEVICE TABLE

This table is searched whenever an intelligent device is closed.

CHINIT is a table containing I/O addresses for channels 'K', 'S', 'R', and 'P'. This table has the structure illustrated by Figure 6-16.

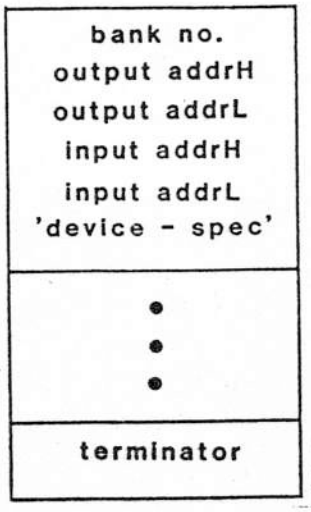


address of the output routine
address of the input routine
device specification

Figure 6-16. CHINIT TABLE

Any other dumb devices must also have entries in this table. This information is loaded into Home RAM (at location CHANS) during system initialization (refer to paragraph 6.8.2).

A new table is needed to contain channel information for the intelligent devices. The channel data for this table is obtained from the Expansion Banks during initialization (refer to paragraph 6.8.4). This table is integrated into the SCT and has the structure illustrated by Figure 6-17.



high order byte of output routine address
low order byte of output routine address
high order byte of input routine address
low order byte of input routine address

Figure 6-17. CHINIT INTELLIGENT DEVICE TABLE

6.8 INITIALIZATION PROCEDURES

For operation of the T/S 2000 to commence, the system, Home Bank, DOCK Bank, and the peripheral expansion Banks must be initialized.

6.8.1 System Initialization

The System Configuration Table is built upon powerup. This table contains information describing the configuration of the TS 2000 system and is used by the initialization routines to determine what action to take. Table 6-5 lists the Configuration Table.

Table 6-5
CONFIGURATION TABLE

AROS	LROS	EXP. BANK 1	• •	EXP. BANK n
LANGUAGE TYPE CARTRIDGE TYPE START ADDR CHUNK SELECT AUTOSTART # BYTES OF VAR SPACE	CARTRIDGE TYPE START ADDR CHUNK SELECT	PRESENCE FLAG BANK NUMBER DEVICE SPEC (CHUNK SPEC IF RAM) OPEN ROUTINE ADDR CLOSE ROUTINE ADDR SELECT ROUTINE ADDR INPUT ROUTINE ADDR OUTPUT ROUTINE ADDR DISK CMD HANDLER ADDR INT HANDLER ADDR INIT ROUTINE ADDR RESET ROUTINE ADDR DEVICE TYPE DEVICE BOOT PRIORITY DEVICE INT PRIORITY	• •	PRESENCE FLAG BANK NUMBER DEVICE SPEC (CHUNK SPEC IF RAM) OPEN ROUTINE ADDR CLOSE ROUTINE ADDR SELECT ROUTINE ADDR INPUT ROUTINE ADDR OUTPUT ROUTINE ADDR DISK CMD HANDLER ADDR INT HANDLER ADDR INIT ROUTINE ADDR RESET ROUTINE ADDR DEVICE TYPE DEVICE BOOT PRIORITY DEVICE INT PRIORITY

All fields, except for the presence flag, bank number and Chunk specification for RAM banks are read by the initialization routine from the bank in question. Note that only intelligent devices, and RAM banks appear in the System Configuration Table. Configuration information for dumb devices is stored in the channel I/O tables described in paragraph 6.7.2. The presence flag takes on the following values: Ø if nothing present; 1 if intelligent device is present; and 2 if RAM is present.

6.8.2 Home Bank Initialization

If there are neither AROS nor LROS attached to the system, the initialization of the Home Bank proceeds as it currently does in the Sinclair Spectrum, with the following exceptions:

After loading the permanent device channel information, load the channel information for each intelligent device (if any).

Load the bank selection code and function dispatcher from the Home ROM Extension to the Home RAM in Chunk 3.

Build (in Home RAM) new channel I/O table and the stream I/O select, open, and close tables (as described in paragraph 4.3.2). If there are no intelligent devices, then these tables are empty.

Display, on the screen, the copyright messages.

6.8.3 ROS Bank Initialization

The initialization procedure for the ROS Bank is as follows:

```
If LROS present Then
        If LROS present Then
            perform basic system initialization and pass control to the LROS
            boot address

        Else
            If AROS present Then
                 If AROS language = BASIC Then
                          If to be autostarted Then
                              Perform basic system initialization and
                              Run the AROS

                          Else
                              Perform basis system initialization

                 Else
                      If AROS language = Machine code Then
                              perform basic system initialization and pass
                              control to the AROS boot address if it is to be
                              autostarted.

                      Else
                              Report Code - AROS contains code for a language
                              without the proper LROS

            Else
                 No ROS present.
```

6.8.4 Peripheral Expansion Banks Initialization

The initialization procedure for intelligent devices can be described as follows:

```
For each Expansion Bank Do:
          If not RAM Then
               read configuration information into the System Configuration
               Table.
               If the initialization type indicates that the device is to be
               initialized, call the initialization routine in the Expansion
               bank.
          Else
               Set the presence flag for RAM
               Set the Chunk spec
               Load interrupt handler into Chunk Ø at location 38H
          Endif
          Assign a bank number
Endfor
Boot the device with the highest book priority.
```

6.9 BANK SWITCHING COMMUNICATION ROUTINES

Communication between banks is accomplished by the transfer of control from one bank to another. Two layers of software are provided to programmers for this purpose. These layers are:

Indirect Communication
Direct Communication

Services provided by these layers are described by giving a high level procedural interface and a description of the processing performed. These services are available to all banks. Also, advanced I/O features provided by high level languages are build on these software layers.

6.9.1 Indirect Communication

The Indirect Communications layer consists of a set of assembly language routines which provide call/return and goto types of transfer of control and a routine for the transfer of data. For each of these routines there exists, in each bank other than the Home Bank, an interface routine. The interface routines mediate between the calling code and the code actually performing the service required in a transparent fashion. There is only one copy (in the Home ROM) of the actual service code for the Indirect Communications routines. These routines have the following syntax and semantics:

GOTO_BANK (addr, bank, horizontal-select)

addr -	The memory address to which control is to be transferred.
bank -	The number of the bank in which the memory address resides.
horizontal-select -	A bit pattern describing which Chunks are to enabled (Ø indicates enable, 1 indicates disable).

This routine causes an unconditional transfer of control without return. The specified bank and the Chunk in which the specified memory address resides are selected.

CALL_BANK (addr, bank, horizontal-select, param-in, param-out)

addr -	the memory address to which control is to be transferred.
bank -	the number of the bank in which the memory address resides.
horizontal-select -	a bit pattern describing which Chunks are to enabled (Ø indicates enable, 1 indicates disable).
param-in	number of bytes passed to called routine on the stack.
param-out	number of bytes returned on the stack to the calling routine.

This routine causes an unconditional transfer of control with return. The specified bank and the Chunk in which the specified memory address resides are selected.

XFER_BYTES (src-addr, src-bank, dest-addr, dest-bank, length, direction)

src-addr -	the memory address from which the data is to be taken.
scr-bank -	the number of the bank from which the data is to be taken.
dest-addr -	the memory address to which the data are to be transferred.
length -	the number of bytes to be transferred.
direction -	a flag indicating the direction of the transfer: -1 for high address to low or 1 for low address to high.

This routine causes the specified string of bytes to be moved to the specified location within the specified bank. The status of both the source and destination banks are left unchanged by this operation.

6.9.2 Direct Communication

This layer of software provides a lower level of services to the systems programmer. The routines provided by this layer allow the selection status of any bank or Chunk to be read or modified. This layer provides the following routines:

BANK_ENABLE (bank, horizontal-select)

bank -	the number of the bank to be selected.
horizontal-select -	a bit pattern describing which Chunks are to enabled (Ø indicates enable, 1 indicates disable).

This routine allows the selection status of any bank or Chunk to be modified.

GET_BANK_STATUS (bank, status-ret, hor-sel-ret)

bank -	the number of the bank whose status is to be read.
status-ret -	where the status information is to be returned.
hor-sel-ret-	where the horizontal select value is to be returned.

This routine allows the status and horizontal select (which Chunks are selected) of any bank to be read from the bank selection hardware.

GET_BANK_NUMBER (bank-no-ret, addr)
bank-no-ret - where the bank number is to be returned.
addr an address

This routine returns the number of the bank which currently owns the Chunk in which addr resides.

6.9.3 Restrictions

The above routines may be used as desired by programmers with the following restrictions:

Any service routine (i.e., CALL_BANK) which manipulates Chunks (selecting and/or deselecting) must return the changed Chunk statuses to their original values.

Any code running in a Chunk which "shadows" a RAM Chunk in another bank must access data in that RAM Chunk via the indirect data transfer routine XFER BYTES (refer to paragraph 6.9.1).

Any bank wishing to enable a Chunk shadowed by Chunk 3 of the Home Bank should do the following: 1) Move the machine stack to the Chunk to be enabled (this implies the presence of RAM in this Chunk). If the value of the stack pointer is to be modified, its old value should be saved. 2) The bank selection code running in Chunk 3 of the Home Bank should be copied into the shadowed Chunk, at the same address at which it resides in the Home Bank. Otherwise, no bank will be able to access the bank selection software. 3) The stack must be restored to its original state when the shadowed Chunk is disabled and Chunk 3 in the Home Bank must be re-enabled. (If the second display file is in use, then the above applies to Chunk 7.)

You must never turn on two Chunks on the same horizontal level. The bank switching routines will always avoid this situation because when you request a given Chunk to be turned on, the bank switching routines will turn off that Chunk in any other bank in which it is selected.

6.10 COMMANDS IMPLEMENTED ON THE TIMEX SINCLAIR 2000, BUT NOT ON THE ZX SPECTRUM.

The following commands and functions are additions to those which are already provided on the Spectrum.

6.10.1 SOUND reg, value; reg, value; etc.

This command allows the programmer to manipulate the 15-registers of the General Instrument Ay-3-8912 Programmable Sound Generator. The semantics of the values assigned to the registers are described in the General Instrument Programmable Sound Generator Data Manual, and in paragraph 3.3.7.

The valid values for reg are 1-15, and are specified in decimal. The valid values for value are Ø-255. If any of the values is found to be out of range the Report Code B Integer Out of Range, is given.

6.10.2 STICK (device type, player)

This function allows the programmer to read the status of devices attached to the joystick port where:

```
device type = 1 for joystick
              2 for pushbutton
player      = 1 or 2
```

```
Value returned for joystick = Ø  -  joystick on center
                              1  -  joystick up
                              2  -  joystick down
                              4  -  joystick left
                              8  -  joystick right
```

```
Values returned for pushbutton  =  Ø  -  pushbutton not depressed
                                =  1  -  pushbutton depressed
```

If the specified values are not within the ranges given above the Report Code A (Invalid Argument) is displayed.

NOTE

Combinations are allowed (e.g., joystick diagonally up and to the left yields a value of 1 + 4 = 5)

```
DELETE  m, n
DELETE  ,n
DELETE  m,
```

This command allows the programmer to delete a sequence of lines from a program. The 'm, n' specification causes the sequence of lines from m to n inclusive to be deleted from the program. The ',n' specification causes the lines from the beginning of the program to n inclusive to be deleted. The 'm,' specification causes the lines from m to the end of the program inclusive to be deleted. It should be noted that this causes the DELETE key to become context dependent as described in paragraph 6.9.

If the range n,m does not include at least one line in the BASIC program or n is greater then m, the Report Code A (Invalid Argument) is displayed.

6.10.3 FREE

This function returns the number of bytes of free space currently available in the Home RAM for either program or variables.

6.10.4 RESET (#c) (*)

This command causes the device associated with the specified stream to be reinitialized by performing a call to its initialization (warm start) address (refer to paragraph 6.8.1). If a channel number is not provided, the system initializes any new devices it finds (as it normally does during powerup).

If the stream number is specified and it has not been previously opened using the OPEN command (refer to paragraph 6.10), the Report Code 0 (Invalid Stream) is given. The RESET * command does the equivalent of turning the machine off, and then on again.

ON ERR GOTO linenumber
ON ERR CON'T
ON ERR RESET

These statements allow the programmer to disable automatic program termination upon encountering an error condition. The ON ERR GOTO linenumber allows the programmer to cause the transfer to the specified linenumber to handle the encountered error. The error number and linenumber on which it occurred are available by PEEKing the locations (23739) and (23736). The statement number within the line that caused the error is stored in location (23738). The ON ERR CON'T statement causes the program to resume execution at the statement in which the error originally occurred. If an ON ERR CON'T statement is encountered and an error has not occurred, then the command is ignored. The ON ERR RESET command disables this feature causing the program to report errors and terminate in the usual manner.

6.10.5 Stringy Floppy/Floppy Disk Commands

The following commands are used for accessing disk and disk-like storage devices which are present on an Expansion Bank. A one-bit specifier is present in the Expansion Bank memory space which determines whether or not the device attached to that bank can handle disk-like commands (CAT, FORMAT, SAVE, LOAD, ERASE, MOVE, VERIFY, MERGE). All Expansion Banks, except for RAM banks, are expected to be able to handle the OPEN, CLOSE, PRINT # and INPUT # commands.

If one of the disk-like commands is attempted to a non-disk-like device, the Report Code J (Invalid I/O Device) is displayed. Execution of these commands is handled by the Expansion Bank itself. The Home ROM is responsible for parsing the command and calling the appropriate routine in the specified Expansion Bank. The Expansion Bank handles those operations which are device dependent (special syntax, extra parameters, etc.).

6.10.6 OPEN #c, "m", "filespec" (, "file type spec")

Opens a stream identified by c and creates a new file or opens an existing file as specified by "filespec". A filespec is formed by the concatenation of a "volume spec" and a "filename". The "volume spec" specifies the device to be used and is optional. If it is not specified, a default device is selected. "File type spec" is device dependent and may be used to specify such things as random or sequential access, record length, etc.

6.10.7 CLOSE #c

Closes the file associated with the specified stream after flushing the appropriate buffers where necessary.

6.10.8 CAT "m", "volspec"

Causes a list of files on the specified volume to be displayed on the screen.

6.10.9 FORMAT "m", "volspec"

Formats the media on the specified volume and gives it the specified volume name.

6.10.10 SAVE * (Note '*' distinguishes from Tape SAVE)

SAVE * "m", "filespec"
Saves the current program and variables to the specified file on the specified volume.

SAVE * "m", "filespec" LINE
Saves the current program and variables to the specified file on the specified volume. When it is loaded, it automatically starts at the beginning of the program.

SAVE * "m", "filespec" LINE linenumber
Saves the current program and variables to the specified file on the specified volume. When it is loaded, it automatically starts at the specified linenumber.

SAVE * "m", "filespec" DATA arrayname ()
Saves the specified numeric data array to the specified file on the specified volume.

SAVE * "m", "filespec" DATA arrayname $()
Saves the specified string data array to the specified file on the specified volume.

SAVE * "m", "filespec" CODE start, len
Saves the byte array starting at address 'start' for a length of 'len' bytes to the specified file on the specified volume.

SAVE * "m", "filespec" SCREEN$
Saves the contents of the screen to the specified file on the specified volume. This is equivalent to:
SAVE * "m", "filespec" CODE 16384, 6912

6.10.11 LOAD * (Note: '*' distinguishes from Tape LOAD)

LOAD * "m", "filespec"
Loads the current program and variables from the specified file on the specified volume.

LOAD * "m", "filespec" DATA arrayname ()
Loads the specified numeric data array from the specified file on the specified volume.

LOAD * "m", "filespec" DATA arrayname $()
Loads the specified string data array from the specified file on the specified volume.

LOAD * "m", "filespec" CODE [start [, len]]
Loads the byte array into memory starting at address 'start' for a length of 'len' bytes from the specified file on the specified volume. If either 'start' or 'len' is not specified, the parameters with which it was saved are used.

LOAD * "m", "filespec" SCREEN$
Loads the contents to the screen from the specified file on the specified volume.

6.10.12 ERASE "m", "filespec"
Removes the specified file from the specified volume. See OPEN above for a description of "filespec".

6.10.13 MOVE "m", "old filespec", "new filespec"
Causes the file on the volume specified in "old filespec" to be renamed with "new filespec". If "new filespec" specifies a volume, it must be identical to that specified or implied (default) in "old filespec".

6.10.14 VERIFY * (Note: '*' distinguishes from Tape VERIFY)
The following commands have the same semantics as the corresponding LOAD commands except that the data is not loaded into memory but compared with what is already there. An error message is displayed if they are not identical.

```
VERIFY  *  "m", "filespec"
VERIFY  *  "m", "filespec"  LINE
VERIFY  *  "m", "filespec"  LINE  linenumber
VERIFY  *  "m", "filespec"  DATA  arrayname ()
VERIFY  *  "m", "filespec"  DATA  arrayname $()
VERIFY  *  "m", "filespec"  CODE  start, len
VERIFY  *  "m", "filespec"  SCREEN$
```

6.10.15 PRINT #c, var list
Causes the variable list to be written to the file opened through stream c.

6.10.16 INPUT #c, var list
Causes the variable list to be read from the file opened through stream c.

6.10.17 MERGE * "m", "filespec" (Note: '*' distinguishes from Tape MERGE)
Loads the specified file from the specified drive, causing it to be merged with the BASIC program which is already in memory.

Notational Conventions:

c = stream #

m = device spec

[..] = optional item

volspec = volume only
filename = filename only
filespec = volume name and filename

6.11 MASS STORAGE

The T/S 2000 has been designed with maximum expansion flexibility in mind. To support fast mass storage devices, the device and its controller, simply need to be placed under the direction and control of a collection of bank switching logic. The functions of this logic are described in paragraph 1.3. The structure of the expansion Bank in which the mass storage device would reside is described in paragraph 1.9.1.

SECTION VII

DEVELOPMENT SYSTEM INTERFACES

7.1 SPECTRASIDE

TIMEX will have available a development system interface termed SPECTRASIDE.

SPECTRASIDE is a software development tool designed to allow developers to create programs on their existing development systems and debug the programs on the T/S 2000. It is intended to be used by a person knowledgeable in Z-80 coding and to a limited degree knowledgeable in Z-80 hardware.

With the SPECTRASIDE system installed on a T/S 2000, memory may be examined and modified and machine language programs executed. Programs may be executed in real time or single stepped. The Z-80 registers may be examined and modified. The hardware break point circuit allows a break point at a RAM or ROM location conditionally on instruction fetch, any read, any write or any access. Through an external RS-232 port, code may be up-loaded or down-loaded to or from a Host Software Development System with programmer interaction through the external port or via the T/S 2000 keyboard and display. SPECTRASIDE occupies only 128 bytes of memory employing a shadow ROM technique and occupies I/O ports 78H - 7FH.

SPECTRASIDE installs by plugging a connector onto the back of the T/S 2000. Connection to the Host Development System is by a female DB-25 connector with the pinout of a RS-232C modem, ready to talk to a terminal device.

If your development environment is not supported by the system described above please contact TIMEX.

SECTION VIII

SUBMITTED SOFTWARE

8.1 GUIDELINES FOR SUBMITTED SOFTWARE

In order to be considered for possible inclusion in the TIMEX Computer product software line, the submission of software must adhere to the following guidelines.

All submitted software must be in finished product form. We are interested only in looking at what you consider to be the final version of your code. After examining your software, TIMEX may recommend that certain changes be made to increase the utility or versatility of the program, but it is expected that the initial submission be as bug-free as possible.

All documentation must be included in the initial submission. The required documentation includes:

1) A signed copy of the condition statement (refer to paragraph 8.2) -- no software will be examined until this document has been received.

2) A completed copy of the software check sheet (refer to paragraph 8.3) -- the information on this sheet is vital to our evaluation process.

3) Instructions to the user -- the instructions for use of a program are as important as the code itself.

Other requirements:

1) Initial submissions should include one copy of the software. Upon acceptance, three copies (on separate cassettes) will be requested.

2) The program should be able to be readily duplicated to aid in our evaluation.

3) Your copyright notice need not be displayed, but it must appear somewhere in your code.

4) No software should require that it be loaded via LOAD "(NAME)" CODE. If your program consists entirely of machine code, it must be preceded on the tape by a BASIC program to LOAD the code:

```
10 LOAD "(NAME)" CODE
20 PRINT USR X
```

5) Software should autostart wherever possible.

8.2 COPY OF CONDITION FORM

CONDITIONS FOR EVALUATING PROGRAMS SUBMITTED BY NON-TIMEX AUTHORS AND OWNERS FOR THE TIMEX SINCLAIR PERSONAL COMPUTER

The TIMEX Computer Corporation (hereinafter referred to as "TIMEX") agrees to receive and evaluate, upon request, a computer program submitted you, but only after you have agreed to the following terms and conditions.

1. You agree to submit a copy of the computer program entitled _____ (Program) to TIMEX for the purpose of evaluation for possible marketing and use by TIMEX. You represent that you are the owner of and have the unqualified right to make the Program available to TIMEX.

2. The Program must be the result of your original work or owned by you, must be complete and fully operational on TIMEX Sinclair 2000 Series computers, and must be sufficiently documented in the English language to permit the user to operate the Program without unreasonable difficulty. However, you must not send additional Program information or material unless later requested by TIMEX.

3. Since TIMEX does not wish to receive or hold any voluntarily submitted materials in confidence, you understand that TIMEX cannot treat your Program as secret or confidential, and no confidential relationship of any kind will exist between you and TIMEX as a result of your Program submission. TIMEX agrees to only review and evaluate your voluntarily submitted Program and to indicate its interest, if any, therein. However, TIMEX will use reasonable efforts to avoid any disclosure which might prejudice any confidential aspect of the Program. Nevertheless, you must protect your proprietary interest in your Program before you disclose it to TIMEX. TIMEX will not be liable to you in the event of any disclosure by TIMEX unless it is due to the willful and malicious act of TIMEX.

5. TIMEX may receive from other sources computer programs that are similar to each other and to yours. TIMEX will be free to use, without obligation to you, subject to any valid patent and federal copyright rights that you may have in your Program, similar programs and ideas which have been developed independently and submitted by others. Also, TIMEX shall be free to use similar programs and ideas which become publicly know either before or after the submission of your program.

6. In the event that TIMEX accepts your submitted program for possible use and marketing, you agree that you will enter into a Program Acceptance Agreement with TIMEX in the form provided herewith without alternation. You understand that even though your program may be accepted and subsequently marketed by TIMEX, TIMEX cannot make any commitment as to the amount of Program sales based on any marketing effort.

7. If following TIMEX's evaluation of your Program it is determined that TIMEX has no present or foreseeable future interest therein, such conclusion shall be transmitted to you in writing. However, all such material submitted in connection with the Program shall be placed in a permanent file separated from TIMEX's own internally generated computer programs. The Program and all such material submitted therewith is retained so that TIMEX will have a record of what was disclosed to it.

8. Should any dispute occur between you and TIMEX arising out of or related to these Conditions For Evaluating Programs, or your Program submission, you agree that Two Thousand Dollars ($2,000) shall be the absolute limit of TIMEX liability.

9. TIMEX reserves the right at any time, without notice and without obligation to terminate its policy of evaluating submitted programs, to change the conditions under which programs are submitted, and to stop receiving programs from persons outside the TIMEX organization.

10. TIMEX cannot be held responsible for any delay in evaluating your Program and replying to your submission. Therefore, you should pursue whatever course of action would be beneficial to you while waiting for our response based on the evaluation of your submitted program.

11. This agreement constitutes the entire understanding between you and TIMEX with regard to the evaluation of your Program and it applies to TIMEX and all of its affiliates and may not be amended or modified except in writing, agreed to by both you and duly authorized officers or representatives of TIMEX.

12. If you wish to submit your computer program to TIMEX on the foregoing basis, kindly sign below indicating that you have read and thoroughly understand the foregoing and that you agree with each of the various terms and conditions stated therein.

Please send one signed copy to:

Mr. George Grimm
TIMEX Computer Corporation
Waterbury, CT. 06720

Name:_____
Date:_____

Please provide your mailing address

_____

_____

_____

8.3 COPY OF CHECK SHEET

TIMEX COMPUTER CORP.
SOFTWARE CHECK SHEET

TITLE: _____

COPYRIGHT: _____

K-RAM: _____

PROGRAMS:

SEQUENCE	NAME	LOAD NAME	LOAD TIME

BRIEF PROGRAM DESCRIPTIONS:

COMMENTS: