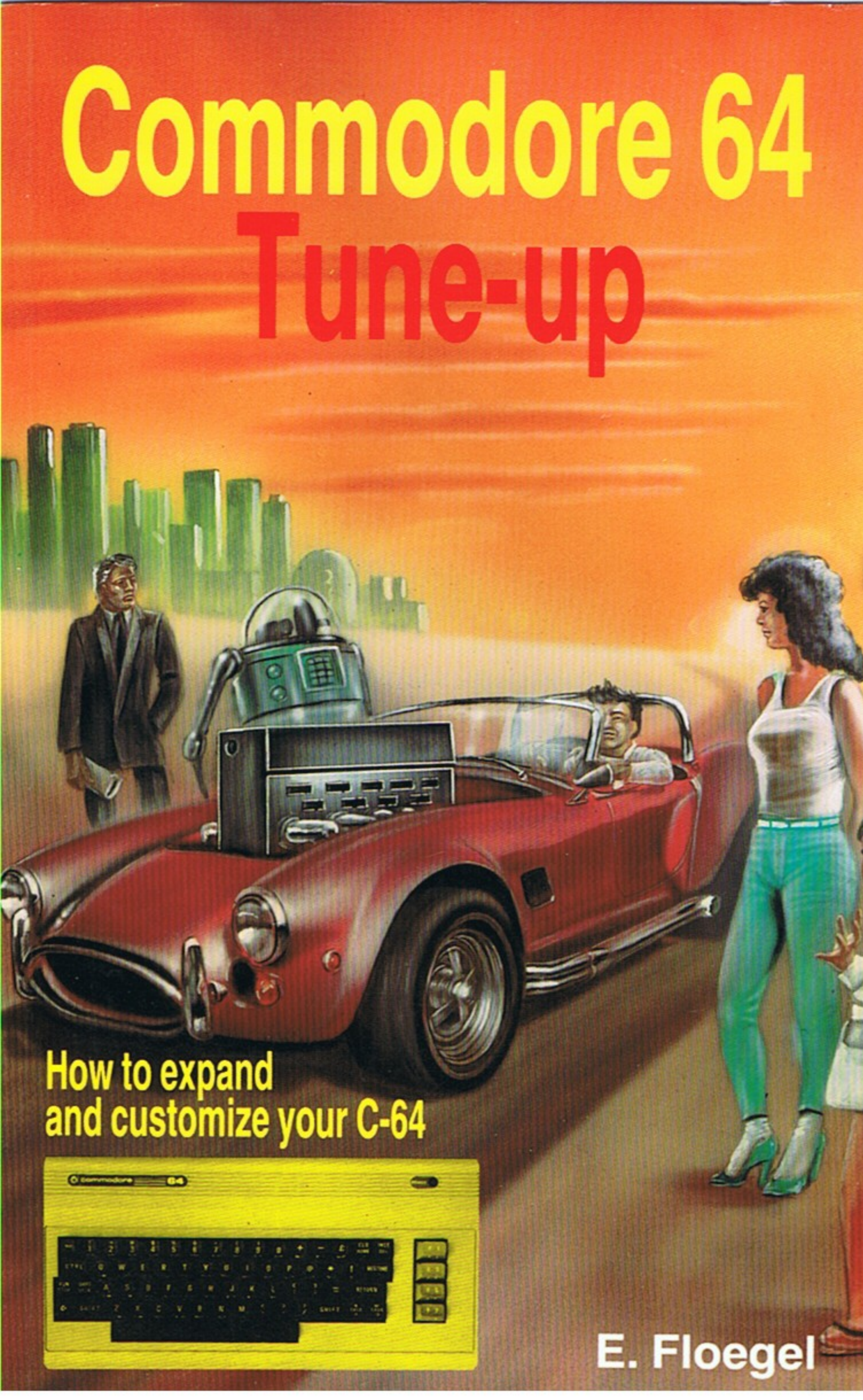
# Commodore 64 Tune-up



How to expand and customize your C-64

E. Floegel

E. Floegel

# Commodore 64
# Tune-up

How to expand

and customize your C-64

# PREFACE.

A computer is mostly used for data processing. These data comes very often from economics and is entered via the keyboard into the computer.

A different kind of data comes from measurement. These data, wich origine in an analog world must be adapted by transducers to the digital world.

Therefore it is not only necessary to know all about computers, but a good knowledge of how to process analog signals is needed.

The experiments described in this book must be regarded only as examples, because every real application in measurement must have a special, distinct solution.

The Commodore C64 is an ideal computer for measurement. Data can be easily accessed by the USER- or the Expansion port. The Time of Day clock can be used for real time applications.

Have fun in controlling the analog world.

Los Angeles, Spring 1984 Ekkehard Floegel

# Table of Contents

Table of Contents.

# 1 Introduction

## 1.Introduction.

While computers are used mostly to make calculations and for Data Base management, there are also numerous other applications. A computer maybe used in a control application in such areas as energy conservation. Great savings in cost of gas or electricity can be realized by controlling the air conditioning. A computer can be made to reduce the temperature inside a building by comparing the external temperature. For the solution of such tasks, an expansion of the hardware of a computer is necessary. Sensors must be connected to the computer. The signals must be analyzed by a program. The results of the computation is used to control external devices.

For the completion of such tasks, one must have the knowledge to program a computer and the ability to build the analog part and the analog digital interface. Figure 1-1 shows the basic circuit to use a computer for data acqusition and control applications.

The first part of the measuring equipment is the sensor. In many cases it is very difficult to find the right transducer for the measurement. If one is found, the output of the transducer has to be converted in input signals for the computer. For this, amplifiers and analog-digital converters are used. An analog-digital converter converts analog voltage into a digital number. The computer analyzes the incoming data and reacts with a corresponding output signal. These output signals are bit patterns which are issued via the ports of the computer. They control mechanical devices such as motors or relays. In an air conditioning control, a motor is used to open or close a valve. Opto isolators are often used between the computer and the external device.

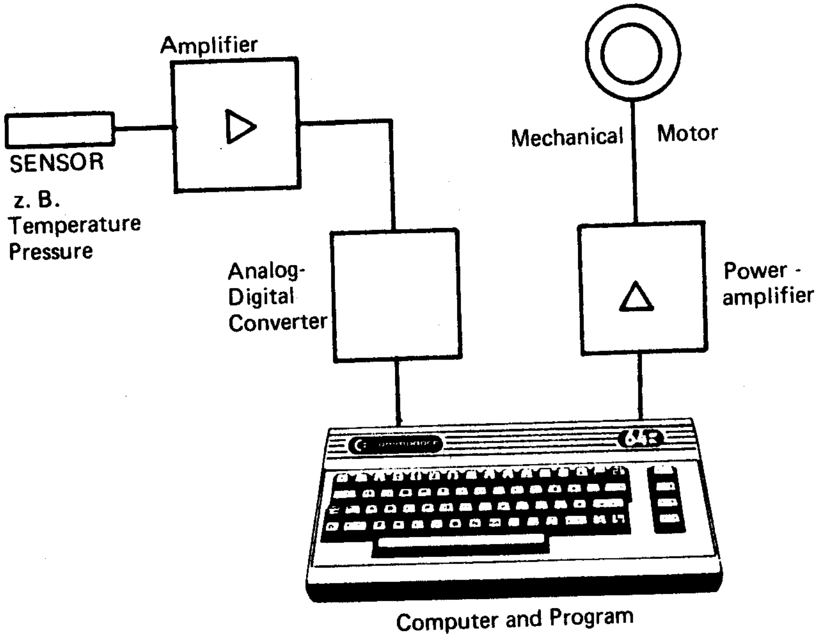


Figure 1-1: Data Acquisition and Control with a Computer.

Without a knowledge of basic electronics, the above applications would prove to difficulty to attempt. In this instance it would be better for an individual to stay within the realm of computer games and more conventional applications.
For experimentation, a minimum of equipment is needed. This is shown in Figure 1-2.

The circuit can be mounted on a solderless experimenter board. A Multimeter is used to measure voltage, current and resistance. This does not have to be a precision Multimeter, because it is mostly used for "On and Off" measurements. Other tools are small diagonal wire cutters or front-cutting 'nippers', needlenose pliers for bending leads, and a pair of tweezers. As the integrated circuits become smaller, and smaller a pocket lens is often necessary to read the printing on an IC.

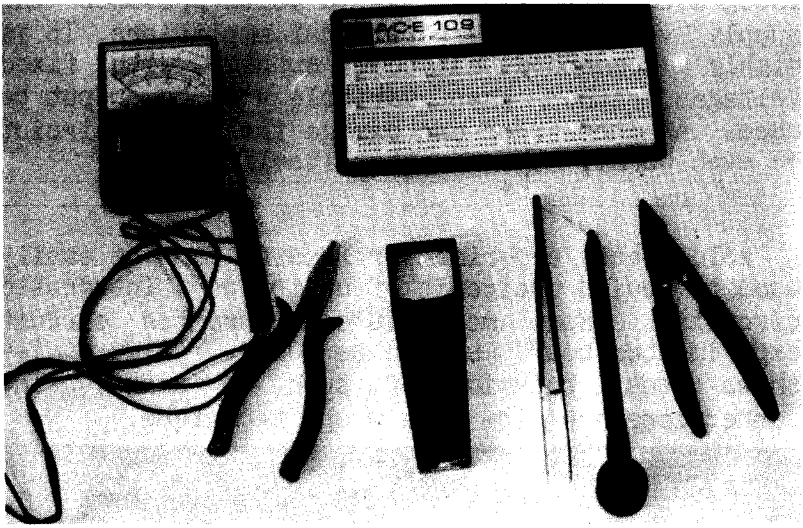


Figure 1-2: Tools needed for Experimentation.

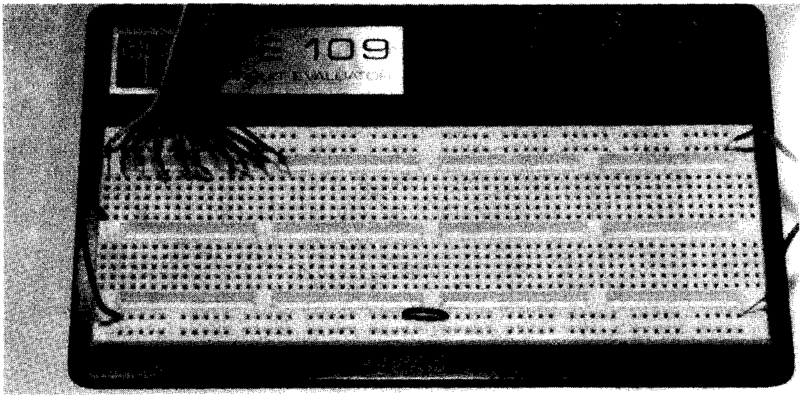


Figure 1-3: Solderless Experimenter Board.

An external power supply should be used for the circuits. It should provide the following voltages:

+5 volts, 1 amp
+12 volts, 0.5 amp
-12 volts, 0.5 amp

The +5 volts are used to supply normal TTL or MOS integrated circuits. The +- 12 volts provides the supply voltage for operational amplifiers. It is highly recommended that power supplies with fixed voltages rather than variable voltage output be used. A minor error can damage the whole circiut or even worse, the computer itself.

A solderless experimenter board can be easily used to build a circuit of up to four integrated circuits. For more IC's a printed circuit experimenter board should be used. Figure 1-4 shows the backside of such a board. The connections are made in a wrap technique.

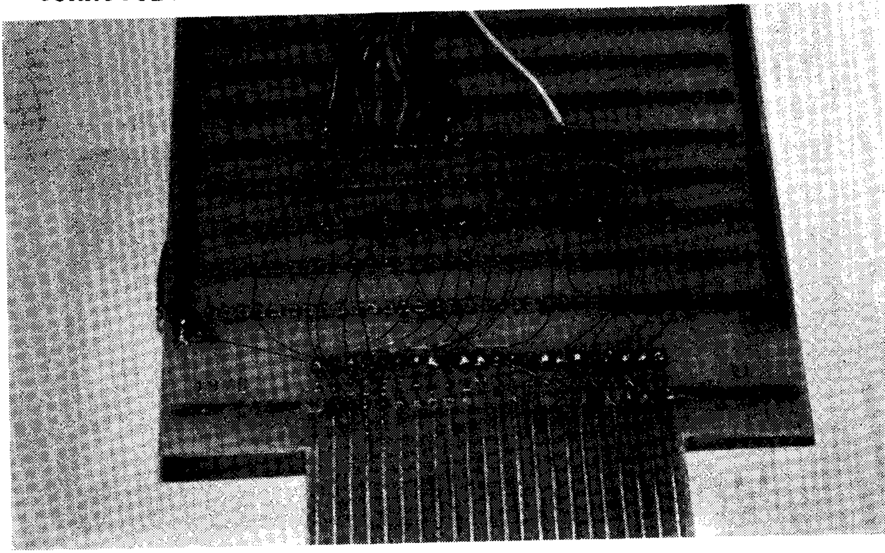


Figure 1-4: Wrap Technique.

The tool shown in Figure 1-2 between the wire cutter and the pliers is used for this wrap technique. An isolated wire is wrapped around the pins of an integrated circuit and then soldered. The heat of the solder iron melts the isolation of the wire and makes a good connection between pin and wire. A temperature controlled solder iron should be used.

This is the minimum of equipment used for experimentations. It can be extended with other tools. One extension is the Logic-Probe shown in Figure 1-5.

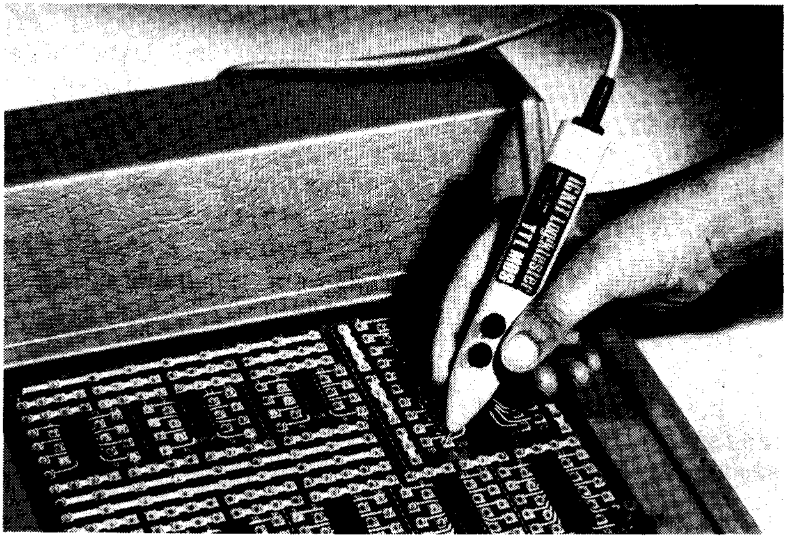


Figure 1-5: Logic-Probe

Two LED's show the state of an output. One LED is on, if the output voltage is larger then 2.4 volts, the other if the voltage is less than 0. 8 volts. In this book the following abbreviations are used:

H=1= voltage level between 2.4 and 5 volts.
L=0= voltage level between 0.0 and 0.8 volts.

One of the advantages of a logic-probe is that it can detect short pulses. Even though it is usually very difficult to show short pulses on an oscilloscope. An oscilloscope must still be part of the extended equipment. It schould be a dual trace with a bandwidth of at least 25 Mhz. The minimum resolution time should be 0.1 us/ 1 cm.

Extreme care must be taken when working with CMOS circuits. Static electricity must be discharged from the human body before touching any of the integrated circuits. This is best done by soldering one end of an 1M resistor to a ground line and then touching the other end. This discharges the static electricity from the body without electric shock. The computer should also be turned off, when a connector is plugged in or off the USER- or Expansion Port.

We have covered briefly the hardware changes necessary for data acquisition and control by a computer. Now, a few words about the software.

BASIC is the most popular programming language. It has only one disadvantage: it is too slow for most control and real time applications. Programs dealing with control and real time applications should be written in assembler language. Because the reaction time of the computer is increased so greatly over that of the external devices, waiting loops must be inserted in the program to allow equality among the computer and the devices.

Another possibilty is FORTH. It is slower than Assembler, but much faster than BASIC. It has one tremendous advantage over the two other languages:In BASIC or Assembler the program is a sequence of commands and subroutines and is started by a command (i.g. RUN in BASIC) and continues until an END command is found, while in FORTH, programs consist of WORDS. These words are stored in a vocabulary. To start a program in FORTH, you must call a word. This word may contain other words which have been defined earlier and stored in the vocabulary. There is a free access to every word in the vocabulary. This makes it possible to enter the following sequence:

```
MOTOR 2 ON TURN VALVE 5 SLOWLY OFF
```

The word MOTOR selects the control lines for the motors. 2 ON outputs a starting pulse on line 2. The word SLOWLY changes the parameter of a waiting loop. FORTH is widely used in military applications and also in movies that require special effects. In this book, the software for most of the examples is written in all three languages.

# NOTES

# 2 Hardware Extensions via the USER-PORT

2. Hardware Extensions via the USER-Port.

The USER-Port of the C64 can be used for hardware extensions. The I/O lines of this port are connected with a CIA 6526. The pin layout of the USER-Port is shown in Figure 2-1.

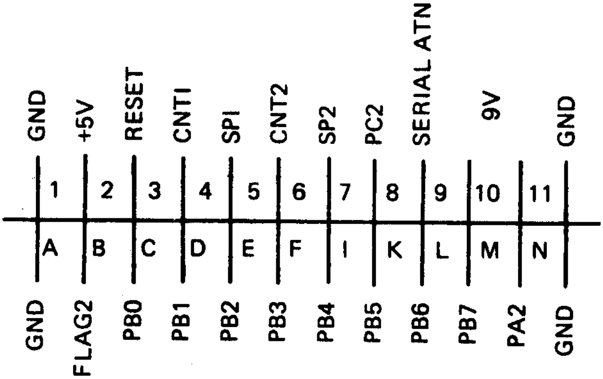


Figure 2-1: Pin Layout of the USER-Port.

A description of the pins is included in the description of the CIA 6526.

## 2.1 The CIA 6526.

The abbreviation "CIA" means Complex Interface

Adapter. This integrated circuit features:

16 individually programable I/O lines
8 or 16 bit handshaking for reading or writing
2 independent, linkable 16-Bit interval timers
24 hour time of day clock with programmable alarm
2 TTL load capability
CMOS compatible I/O lines

The pin layout of the 6526 is shown in Figure 2-2.

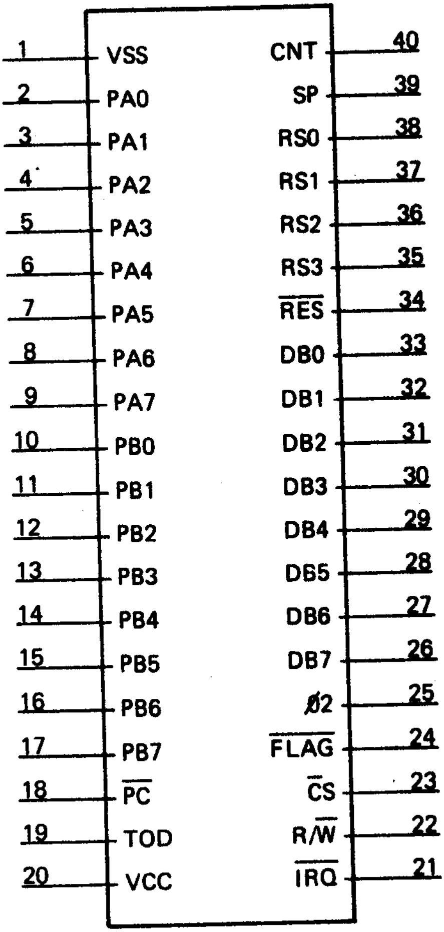


Figure 2-2: Pinlayout of the CIA 6526.

The lines RES, R/$\overline{W}$, $\overline{CS}$, Ø2, RS3, RS2, RS1 and RS0 are used for internal addressing and selection of registers. Figure 2-3 is a register map of the 6526. The C64 contains two CIA 6526's. The addresses contained in the register map are the addresses of the second 6526. From this chip, the lines PB0 through PB7 and PA2 are connected to the USER-Port.

| ADRESS | NAME | DESCRIPTION |
|---|---|---|
| DD00 | PORTA | PERIPHERAL DATA REG A |
| DD01 | PORTB | PERIPHERAL DATA REG B |
| DD02 | DDRA | DATA DIRECTION REG A |
| DD03 | DDRB | DATA DIRECTION REG B |
| DD04 | T1L | TIMER A LOW REGISTER |
| DD05 | T1H | TIMER A HIGH REGISTER |
| DD06 | T2L | TIMER B LOW REGISTER |
| DD07 | T2H | TIMER B HIGH REGISTER |
| DD08 | DOD10 | 10THS OF SECONDS REGISTER |
| DD09 | TODS | SECONDS REGISTER |
| DD0A | TODM | MINUTES REGISTER |
| DD0B | TODH | HOURS – AM/PM REGISTER |
| DD0C | SDR | SERIAL DATA REGISTER |
| DD0D | ICR | INTERRUPT CONTROL REGISTER |
| DD0E | CRA | CONTROL REG A |
| DD0F | CRB | CONTROL REG B |

Figure 2-3: Register Map of the CIA 6526.

## 2.11 Programming the I/O Ports.

Port A and B are programmed using the internal data direction registers DDRA and DDRB. If one bit is set to 1 in DDRA or DDRB, than the corresponding line in Port A or Port B, respectively, will be used for output. If the bit in DDRA or DDRB is 0, the corresponding line acts as an input.

The bit pattern 11000111 = $C7 = 199 sets the lines PB0 through PB3, PB6 and PB7 as output;

and the lines PB3 through PB5 as input. This is shown in Figure 2-4.

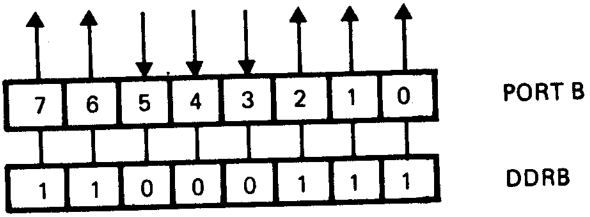


Figure 2-4: Port B and Data Direction Register DDRB.

The lines FLAG2 and PC2 can be used for a data transfer with handshaking. They are tied to the second CIA 6526. The line PC2 goes low for one clock cycle, if a read or write operation is done with Port B. The line FLAG2 acts as an input. A negative edge of a pulse at FLAG2 sets the interrupt flag bit.

## 2.12 Switching On and Off External Devices.

### 2.121 Switching a LED.

The lines of Port B are used to switch external devices on and off. Figure 2-5 shows the testing equipment. The circuit is mounted on a solderless experimenter board. This board is connected to the USER-Port of the C64 with a multi-flat cable. An external power supply is used to supply voltage.

If the I/O lines are used as an output, they may be loaded with 2 TTL loads. Only a few milliamperes will be available for power. If the external device needs more power, transistors or relays must be used. Figure 2-7 shows the control of a LED with a transistor. The NPN transistor is used in an emitter circuit. A current limiting resistor of 180 Ohms is connected in series with the LED.

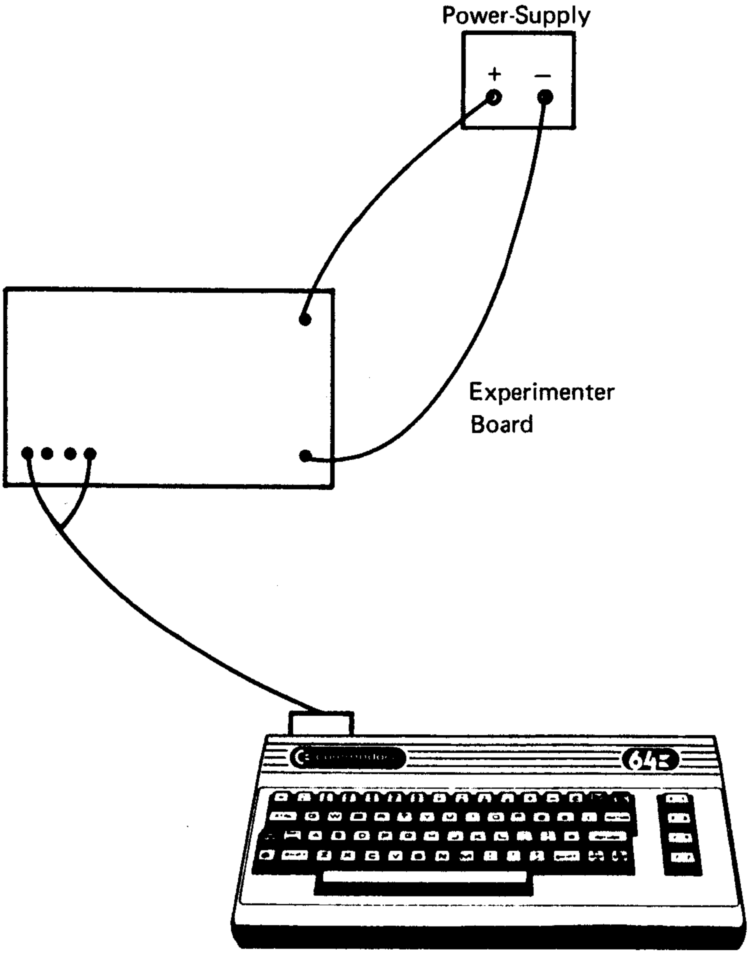


Figure 2-5: Experimentation Circuit.

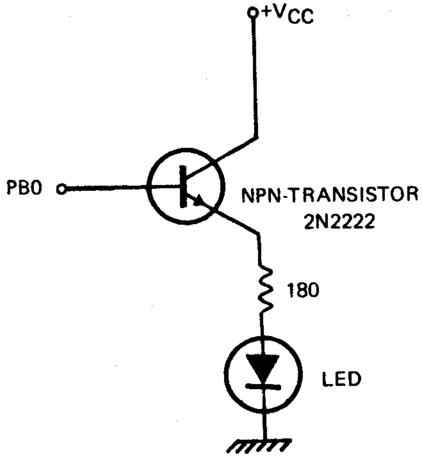


Figure 2-6: Control of a LED.

In this circuit, PB0=1 switches the LED on and PB0=0 switches it off. The program is shown in Figure 2-8. As mentioned earlier, most of the programs will be shown in BASIC, Assembler and FORTH. For the programs in this book the following agreements are made:

BASIC:

```
10 A=56576
20 PB=A+1
30 DB=A+3
40 L1=A+4
50 H1=A+5
60 L2=A+6
70 H2=A+7
80 CA=A+14
90 CB=A+15
```

ASSEMBLER:

```
PORTB     EQU $DD01
DDRB      EQU $DD03
T1        EQU $DD04
T2        EQU $DD06
CRA       EQU $DD0E
CRB       EQU $DD0F
```

FORTH:

```
   SCR # 10
     0 ( I/O                           9.11.  EF)
     1 HEX
     2 DD01 CONSTANT PORTB
     3 DD03 CONSTANT DDRB
     4 DD04 CONSTANT T1
     5 DD06 CONSTANT T2
     6 DD0E CONSTANT CRA
     7 DD0F CONSTANT CRB
     8 DECIMAL
     9  ;S
    10
```

Figure 2-7: Agreements.

BASIC:

```
          POKE DB,1
          POKE PB,1
          POKE PB,0
```

ASSEMBLER:

```
                                ORG $C000
C000: A901      INIT            LDA #1
C002: 8D03DD                    STA DDRB ; PB0 OUTPUT
C005: 00                        BRK

C006: A901      ON              LDA #1
C008: 8D01DD                    STA PORTB ; LED ON
C00B: 00                        BRK

C00C: A900      OFF             LDA #0
C00E: 8D01DD                    STA PORTB ; LED OFF
C011: 00                        BRK
PHYSICAL ENDADDRESS: $C012
```

FORTH:

```
SCR # 19
  0 ( ON AND OFF                          11/9EF)
  1 HEX
  2 1 DDRB C!
  3 : ON 1 PORTB C! ;
  4 : OFF 0 PORTB C! ;
  5
```

Figure 2-8: Switching an LED On and Off.

In BASIC, the POKE command POKE DB, 1 makes PBO an output. POKE PB,1 switches the LED on, POKE PB,0 switches it off. In Assembler, three small programs must be written. The code starting at $C000 makes PBO an output. The code beginning at $C006 switches the LED on and the code at $C00C switches it off.

Compiling screen 19 in FORTH makes PBO an output. The word ON switches the LED on, the word OFF switches it off.

Figure 2-9 illustrates a collector circuit, which can also be used to control an LED. A collector circuit is used. The LED is now switched on with PBO=0 and switched off with PBO=1.

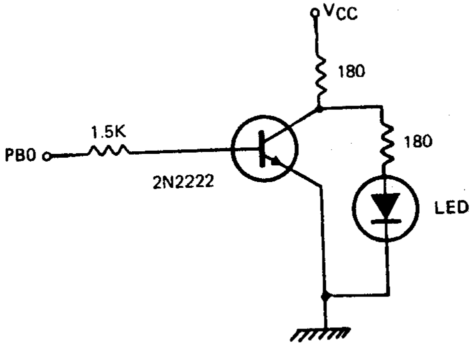


Figure 2-9: Controlling an LED with a Collector Circuit.

## 2.122 Controlling several LED's simultaneously.

In the programs that follow, 8 LED's are connected via the circuit in Figure 2-6 to the I/O lines PBO through PB7. Four examples are given: 1) the output of a bit pattern, 2) switching on a particular LED, 3) running lights and 4) a light bar.

### 1) Output of a Bit Pattern.

The program is shown in Figure 2-10.

BASIC:

```
100 POKE DB,255
110 INPUT"N=";N
120 IF N<0 THEN END
130 POKE PB,N
140 GOTO 110
```

ASSEMBLER: not implemented.

FORTH:

```
 6 ( BIT PATTERN                                )
 7
 8 FF DDRB C!
 9
10 : OUT ( N) PORTB C! ;
11
12 DECIMAL ;S
13
14
15
OK
```

Figure 2-10: Output of a Bit Pattern.

Line 100 in the BASIC program makes all I/O lines of Port B outputs. A number is entered in a loop and the lower 8 bits are output. The loop is left if a negative number is entered.

An Assembler program would be too long for implementation. For example the input of an ASCII character from the keyboard must be converted into a decimal number. As there is no need for speed in this program, an Assembler program was not implemented.

In FORTH, the word OUT is defined. It takes the top number of the stack and issues the lower 8 bits to Port B.

2) Switching a particular LED On an Off.

Eight LED's are numbered from 1 to 8, as shown in Figure 2-11. The LED with the number one is controlled by PB0. The LED with the number eight is controlled by PB7. The problem is to change only one LED while leaving the others unaffected.

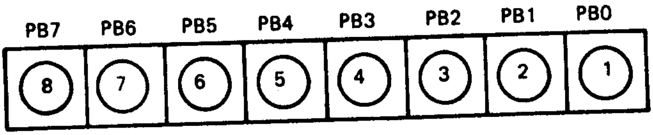


Figure 2-11: Eight LED's in a Row.

BASIC:

```
100 POKE DB,255
110 S=0
120 POKE PB,S
130 INPUT  "WHICH  LED ";N
135 IF N<0 THEN END
140 N=N-1: A=2^N
150 INPUT"I)N OR O)UT ";A$
```

```
160 IF LEFT$(A$,1)="I" THEN 200
170 IF LEFT$(A$,1)="O" THEN 220
180 GOTO 150
200 S=S+A: IF S<256 THEN POKE PB,S
210 GOTO 130
220 S=S-A: IF S>-1 THEN POKE PB,S
230 GOTO 130
```

ASSEMBLER: not implemented

FORTH:

```
: PINIT ( -N) FF DDRB C! 0 DUP
  PORTB C! ;
: NEW ( N-N') 1 SWAP DUP 1 = IF
  DROP ELSE 1 DO 2 * LOOP THEN ;

: ON ( NN'-N") NEW OR DUP PORTB
  C! ;
: OFF ( NN'-N") NEW XOR DUP
  PORTB C! ;
```

Figure 2-12: Switching a Certain LED.

In the BASIC program the data direction register is set and the variable S is set to zero. The state of the eight LED's is stored in S. In line 130 the number of the LED's is entered. The input of A$ in line 150 determines if the LED has to be turned on or off. The following is an example.

It is assumed that LED #5 is turned on. The value of S is then 16. LED #8 should be turned on. For N, the value 8 is entered. In line 140 a one is subtracted and A becomes 2^7=128. In line 200, A is added to S and stored in Port B. This is only done if S is less than 256. In the example, the value of S is 144. The LED's #5 and #8 are now turned on. In order to turn off an LED, the value of A is substracted from S. The program is not error free. If a burning LED is turned on more than once the state of the other LED's is changed. A check occures only if S is less than 256. If a negative number is entered for N, the entire program will become non-operational.

There was no Assembler program written for this application.

The FORTH program is similar to the BASIC. The word PINIT sets the data direction register, places a zero on the stack, and stores it in Port B. The word NEW determines the bit which has to be set for switching a particular LED. If N is greater than one, it is N-1 multiplied by two.

If the word ON is called, the number of the LED and the state of the other LED's has to be on the stack. After execution of the word, the new state remains on the stack. The turning on of an LED is done with the OR function. For turning off an LED the EXCLUSIVE-OR function is used. The sequence of words

PINIT

3 ON

turns on LED #3. Other LED's can be turned on or off with 5 ON, 3 OFF etc. At the end of playing with the LED's the stack must be emptied with DROP.

3) Running Light.

The program in Figure 2-13 turns the LED's on and off in sequence, one after the other. This simulates a running light.

BASIC:

```
100 POKE DB,255
110 POKE PB,0
120 A=1
130 POKE PB,A
140 GOSUB 200
150 A=A*2
160 IF A=256 THEN A=1
170 GOTO 130
200 FOR I=1 TO 50
210 NEXT I: RETURN
```

ASSEMBLER:

```
                              ORG  $C000
C000: A9FF                    LDA  #$FF
C002: 8D03DD                  STA  DDRB
C005: A900                    LDA  #00
C007: 8D01DD                  STA  PORTB
C00A: 38                      SEC
C00B: 2A          M           ROL
C00C: 2016C0                  JSR  WAIT
C00F: 8D01DD                  STA  PORTB
C012: 90F7                    BCC  M
C014: B0F5                    BCS  M

C016: A280        WAIT        LDX  #$80
C018: A0FF                    LDY  #$FF
C01A: 88          W           DEY
C01B: D0FD                    BNE  W
C01D: CA                      DEX
C01E: D0FA                    BNE  W
C020: 60                      RTS

PHYSICAL ENDADDRESS: $C021
```

FORTH:

```
SCR # 13
( I/O PORTB                      10.11.EF)
HEX
FF DDRB C!
DECIMAL
: WAIT ( N) 0 DO LOOP ;
: LL 1 BEGIN DUP PORTB C! 2 *
  DUP 256 = IF DROP 1 THEN
  2000 WAIT ?TERMINAL UNTIL ;
```

Figure 2-13: Running Light.

In the BASIC program, the starting value of variable A is one. This value is multiplied by two and stored in Port B. If A is greater than 256 it is reset to one. The program runs in an endless loop and has to be interrupted with the STOP key. The subroútine in lines 200 and 210 is a waiting loop.

In the Assembler program, the data direction register is set and a zero is stored in Port B. The Carry bit is set and the content of the accumulator is rotated one time to the left. This is done by the ROL instruction. After running through a waiting loop the accumulator is stored in Port B. Instead of shifting the accumulator and storing it, Port B could be shifted by itself with ROL PORTB.

The FORTH program uses the same algorithm as the BASIC program. The number in the top of the stack is multiplied by two and stored in Port B. If this number is greater than 256 it is reset to one. The word WAIT is a waiting loop. The program runs until a key is pressed.

4) Lightbar.

The program in Figure 2-14 simulates a lightbar. The LED's are turned on one after another, until all are lit. They are then all turned off together and the cycle repeated.

BASIC:

```
100 POKE DB,255
110 POKE PB,0
120 A=1:B=1
130 POKE PB,B
140 GOSUB 200
150 A=A*2:B=B+A
160 IF A=256 THEN 120
170 GOTO 130
200 FOR I=1 TO 50
210 NEXT I: RETURN
```

ASSEMBLER:

```
                                ORG $C000
C000: A9FF                      LDA #$FF
C002: 8D03DD                    STA DDRB
C005: A900    M                 LDA #00
C007: 8D01DD                    STA PORTB
C00A: 38      M1                SEC
C00B: 2A                        ROL
C00C: 2016C0                    JSR WAIT
C00F: 8D01DD                    STA PORTB
C012: 90F6                      BCC M1
C014: B0EF                      BCS M

C016: A280    WAIT              LDX #$80
C018: A0FF                      LDY #$FF
C01A: 88      W                 DEY
C01B: D0FD                      BNE W
C01D: CA                        DEX
C01E: D0FA                      BNE W
C020: 60                        RTS

PHYSICAL ENDADDRESS: $C021
```

FORTH:

```
SCR # 14
  0 ( I/O PORTB                        10.11.EF)
  1 HEX
  2 FF DDRB C!
  3 DECIMAL
  4
  5 : LB 0 BEGIN DUP PORTB C! 2 * 1+
  6   DUP 255 > IF DROP 0 THEN 1000
  7   WAIT ?TERMINAL UNTIL ;
  8
  9
```

Figure 2-14: Lightbar.

The programs are similar to those in Figure 2-13. In the BASIC program, the variable A is multiplied by two and added to B. The value of B is stored in Port B.

In the Assembler program, the Carry bit is set prior to every shift instruction.

In FORTH, the number sequence 1, 3, 7, 15 etc. is created by the word LB and stored in Port B.

### 2.123 Controlling of a Relay.

Relays are widely used to switch devices with a large power consumption. They are also used to isolate the computer from the consumer. The computer controls the coil of the relay. This is galvanically isolated from the contacts of the relay. This is depicted in Figure 2-15.

Figure 2-16 illustrates a practical circuit. The NPN transistor drives a Reed relay. The supply voltage of this relay is 12 volts. The diode 1N4148 is used to suppress inductive spikes which could damage the transistor. In most cases, one transistor can switch a relay. For higher currents and voltages, a Darlington transistor should be used.

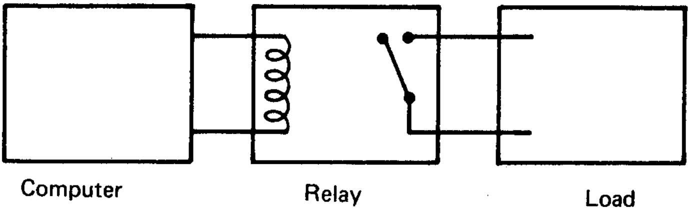


Figure 2-15: Galvanic Isolation of computer and Consumer.

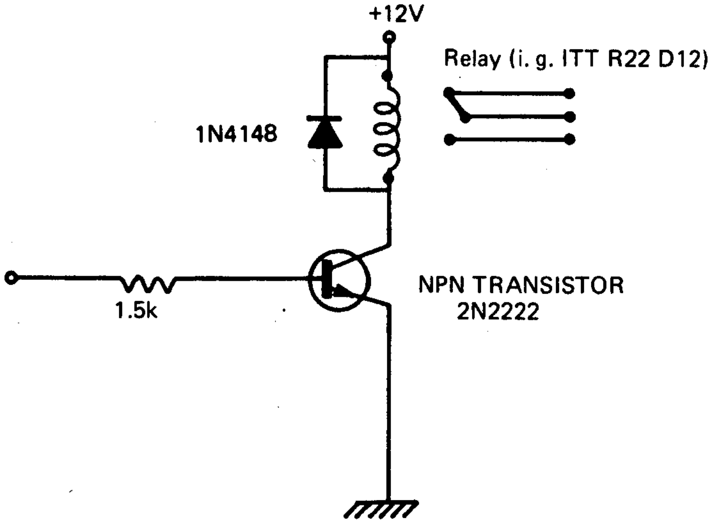


Figure 2-16: Controlling of a Relay.

2.124 Control of an Opto-Isolator.

There are two disadvantages in using relays for switching. The power consumption may be high and the switching time will be extended due to the mechanical inertia of the contacts. For fast switching, Opto-isolators should be used. A dual-in-line package contains an LED and a light sensitive transistor. The pin layout is shown in Figure 2-17. The LED is between pins 1 and 2, and the light sensitive transistor is between pins 4, 5 and 6.

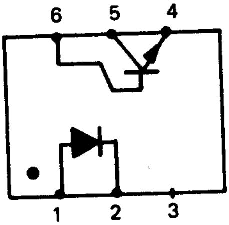


Figure 2-17: Pin Layout of an Opto-Isolator.

A practical circuit is shown in Figure 2-18. The LED is controlled by PB0. A transistor is used to switch the LED. The power supply of the computer is used for this part of the circuit. On the other side a TTL NAND gate is controlled by the light sensitive transistor. The power for this part of the circuit comes from an external power supply.

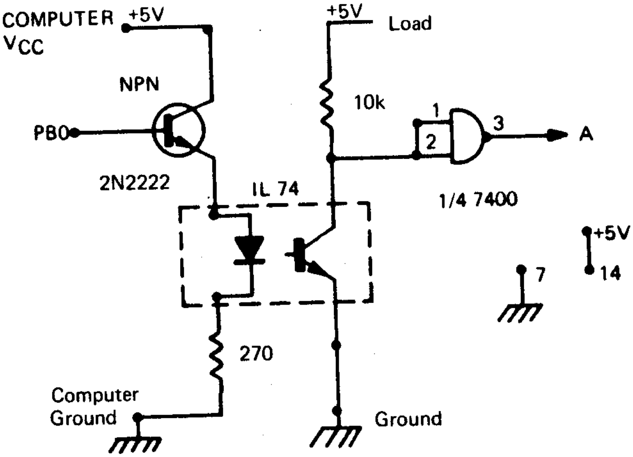


Figure 2-18: Controlling an Opto-Isolator.

This galvanic isolation between computer and consumer is necessary if Triacs or Thyristors are used to switch alternating current.

## 2.13 Using the USER-Port for the Input of Data.

If the I/O lines of the USER-Port are programmed as inputs data can be entered into the computer. The simplest example is to test the state of a key.

### 2.131 Key-Input.

Figure 2-19 shows the connection of a key to the USER-Port. If the key is open, PBO is grounded via the 1.5k resistor. If the key is closed, the potential at PBO is +5 volts.

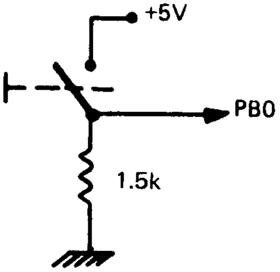


Figure 2-19: Key-Input.

The Program is shown in Figure 2-20.

BASIC:

```
110 A=PEEK(PB)
120 IF A/2=INT(A/2) THEN 110
130 PRINT" KEY PRESSED"
```

ASSEMBLER:

```
                     AUX       EPZ $02
                     BSOUT     EQU $FFD2

                               ORG $C000
C000: 2026C0                   JSR MAIN
C003: 00                       BRK

C004: 68             PRINT     PLA
C005: 8502                     STA AUX
C007: 68                       PLA
C008: 8503                     STA AUX+1
C00A: A200                     LDX #00
C00C: E602           P1        INC AUX
C00E: D002                     BNE *+4
C010: E603                     INC AUX+1
C012: A102                     LDA (AUX,X)
C014: 297F                     AND #$7F
C016: 20D2FF                   JSR BSOUT
C019: A200                     LDX #0
C01B: A102                     LDA (AUX,X)
C01D: 10ED                     BPL P1
C01F: A503                     LDA AUX+1
C021: 48                       PHA
C022: A502                     LDA AUX
C024: 48                       PHA
C025: 60                       RTS

C026: A900           MAIN      LDA #00
C028: 8D03DD                   STA DDRB
C02B: AD01DD         M         LDA PORTB
C02E: 2901                     AND #%00000001
C030: F0F9                     BEQ M
C032: 2004C0                   JSR PRINT
C035: 4B4559                   ASC \KEY PRESSED\
C038: 205052
C03B: 455353
C03E: 45C4
C040: 60                       RTS

PHYSICAL ENDADDRESS: $C041
```

FORTH:

```
SCR # 19
  ( INPUT                       EF]
  HEX
  00 DDRB C!
  : MES ." KEY PRESSED" ;
  : INKEY BEGIN PORTB C@ 1 AND 1=
    UNTIL MES ;

  DECIMAL ;S

```

Figure 2-20: Program Key-Input.

In the programs, it is assumed that PB0 is 1 if the key is closed. In BASIC, it is not possible to mask the input with the AND function. Line 120 proves if an even number is entered at the USER-Port. This is true as long as the key is not pressed. In Assembler and FORTH, the state of PB0 is determined by AND #%00000001, or 1 AND respectivly.

For the printout of a message in Assembler, the subroutine PRINT is used. The text to be printed is entered directly after the subroutine call. Bit 8 of the last character to be printed must be 1. This stops the printing. The program continues after the message.

## 2.132 Light Sensor.

The circuit in Figure 2-21 is used to detect light. The current through a light sensitive diode is compared with the currents through resistors R1 and R2. The values of the resistors are chosen to allow light from a lamp to switch on the amplifier, but not normal sunlight. The output A of the amplifier is high, if light falls on the diode.

The program determines the state of this output and prints the message "Light is on" or "Light is off".

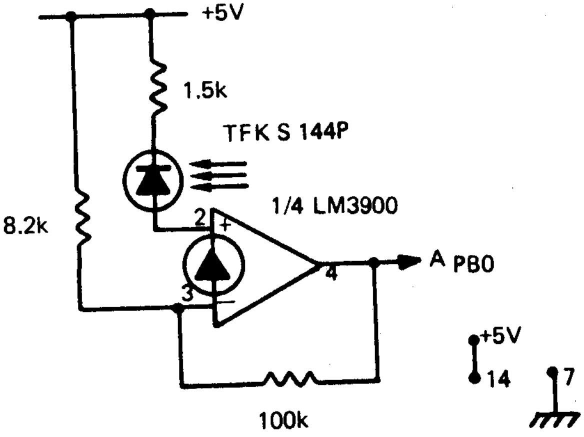


Figure 2-21: Light Sensor.

BASIC:

```
110 A=PEEK(PB)
120 IF A/2=INT(A/2) THEN 150
130 PRINT "LIGHT IS ON"
140 END
150 PRINT "LIGHT IS OFF"
160 END
```

ASSEMBLER:

```
                     AUX       EPZ $02
                     BSOUT     EQU $FFD2

                               ORG $C000
C000: 2026C0                   JSR MAIN
C003: 00                       BRK

C004: 68             PRINT     PLA
C005: 8502                     STA AUX
```

```
C007: 68                    PLA
C008: 8503                  STA AUX+1
C00A: A200                  LDX #00
C00C: E602     P1           INC AUX
C00E: D002                  BNE *+4
C010: E603                  INC AUX+1
C012: A102                  LDA (AUX,X)
C014: 297F                  AND #$7F
C016: 20D2FF                JSR BSOUT
C019: A200                  LDX #0
C01B: A102                  LDA (AUX,X)
C01D: 10ED                  BPL P1
C01F: A503                  LDA AUX+1
C021: 48                    PHA
C022: A502                  LDA AUX
C024: 48                    PHA
C025: 60                    RTS

C026: A900     MAIN         LDA #00
C028: 8D03DD                STA DDRB
C02B: AD01DD                LDA PORTB
C02E: 2901                  AND #%00000001
C030: F00F                  BEQ M
C032: 2004C0                JSR PRINT
C035: 4C4947                ASC \LIGHT IS ON\
C038: 485420
C03B: 495320
C03E: 4FCE
C040: 60                    RTS
C041: 2004C0 M              JSR PRINT
C044: 4C4947                ASC \LIGHT IS OFF\
C047: 485420
C04A: 495320
C04D: 4F46C6
C050: 60                    RTS

PHYSICAL ENDADDRESS: $C051
```

FORTH:

```
    SCR # 15
      0 ( I/O PHOTODIODE            11.11.EF)
      1 HEX
```

```
00 DDRB C! DECIMAL
: MES1 ." LIGHT IS ON" ;
: MES2 ." LIGHT IS OFF" ;

: E/A  PORTB C@ 1 AND 0=
  IF MES2 ELSE MES1 THEN ;

```

Figure 2-22: Program Light Sensor.

Figure 2-23 shows the circuit of an optical limit switch. Model OPB 813 is used as an Opto-sensor. On the left is an LED which emits ultra violet light. On the right is a light sensitive diode. If the beam is interrupted, a signal is generated at the output A. This circuit is used in the next chapter to measure the period of a pendulum.

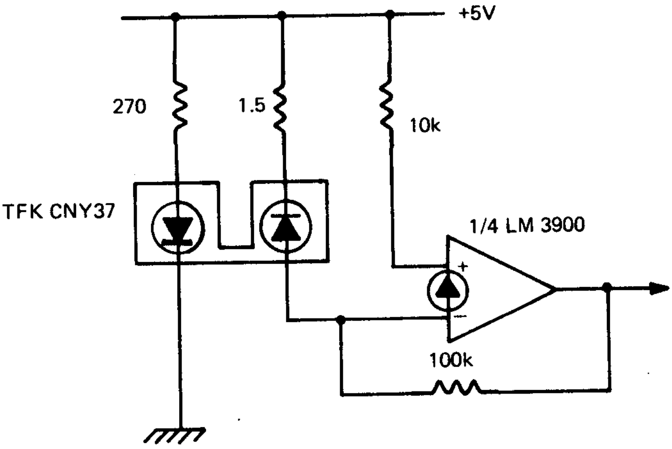


Figure 2-23: Optical Limit Switch.

2.133 An Acoustic Sensor.

Figure 2-24 shows an acoustic sensor. Using an acoustic sensor can lead to many problems. The microphones available on the market have quite different sensitivities and characteristics. In general, an acoustic sensor consists of an amplifier, a rectifier and a switch.

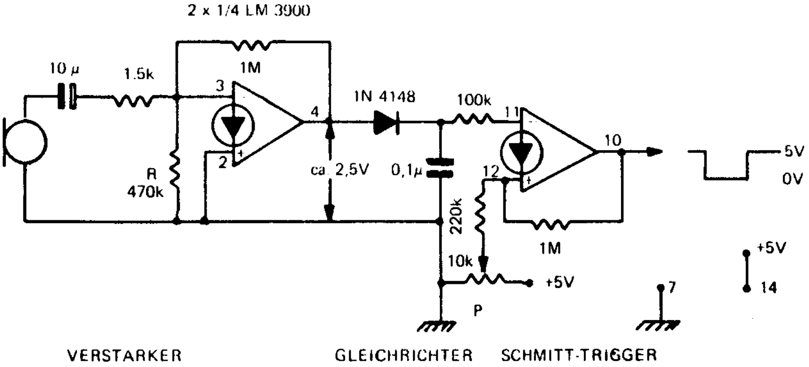


Figure 2-24: Acoustic Sensor.

The circuit in Figure 2-24 is dimensioned in such a manner, that a word spoken with normal loudness generates a pulse at the output. A cassette recorder microphone is used. The amplifier and the switch is built with the Norton amplifier LM3900, because it needs only one supply voltage. The first stage is an AC amplifier with a gain of approximatly 700. The resistor R has to be dimensioned in such a manner that the DC output level of the amplifier is aproximatly 2. 5 volts. The rectifier is followed by a Schmitt trigger. The feedback resistor of 1M adds a hysteresis to avoid oscillations. The potentiometer P has to be adjusted so that the output voltage without a signal is equal to the supply voltage.

## 2.2 Programming the Timer.

The CIA 6526 has two timers. Each consits of a 16-

bit, read only counter and a 16-bit, write only latch. Data is written into the latch and read from the counter. Both timers can be used independently or linked together. The mode of the timer is controlled by the two control registers CRA and CRB. Figures 2-25 and 2-26 show the meaning of the single bit in the registers.

Bit CRA0 starts and stops the Timer A. The I/O line PB6 can be used as an output. If CRA1 is 1, PB6 acts as an output. This overwrites the programming of the data direction register. At PB6, a square wave or a pulse can be generated. With CRA2=1, the polarity at PB6 is reversed with every zero crossing of the Timer A. If CRA2 is 0, a positive pulse with the length of one clock cycle is generated. Bit CRA3 determines if this is done continuously or in the one shot mode. With CRA3=0, the Timer A works continously. Everytime the Timer A reaches zero, the content of the latch is stored in the counter. If the timer has to be set immediatly to a new value, this has to be written to the latch with CRA4=1. This forces the counter to be set to a new value. If not, with CRA4=0, the new value is entered into the counter with the next zero crossing. Bit CRA5 determines whether the internal clock Ø2 or an external clock at CNT has to be used for counting. The last two bits of Control Register A determine the mode of the serial shift register and which frequency is used for the Time Of Day clock.

Bit CRB0 through CRB4 have the same meaning as the Bits of CRA. PB7 is used as output for Timer B. Bits CRB5 and CRB6 determine the input clock for Timer B.

CRB6=0 CRB5=0 Count internal Ø2 clock.
CRB6=0 CRB5=1 Count positive pulses at CNT.
CRB6=1 CRB5=0 Count Timer A zero crossings.
CRB6=1 CRB5=1 Count Timer A zero crossings, while CNT is positive.

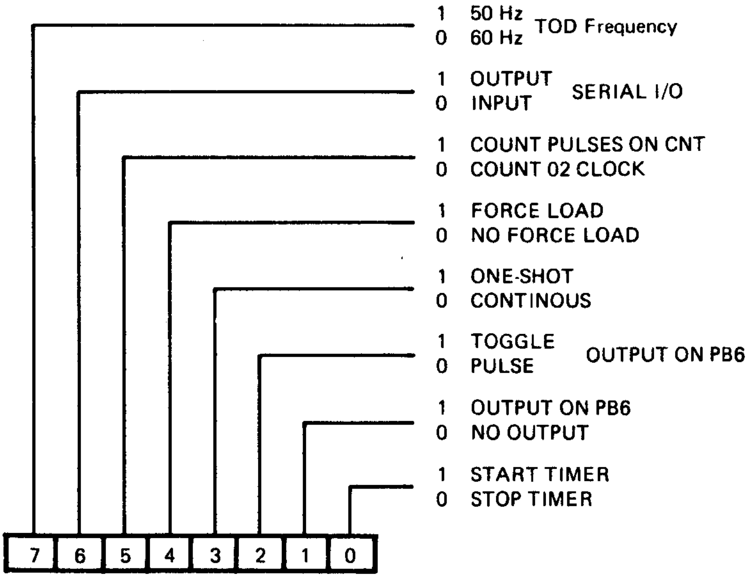


Figure 2-25: Control Register A.

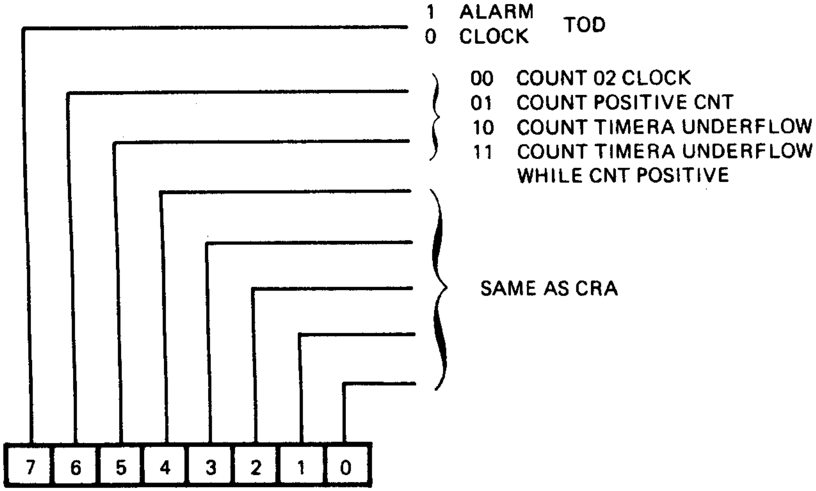


Figure 2-26: Control Register B.

## 2.21 Square Wave at PB6.

In the next program shown in Figure 2-27, the Timer A is used to generate a square wave. The timer is programmed in the continous mode (CRA3=0), to toggle the output (CRA2=1) at PB6 (CRA1=1). With CRA0=1, the square wave is started and with CRA0=0 it is stopped. The corresponding number for the frequency is stored in the timer latches T1L and T1H. During counting, a new value may be written into the latches to change the frequency.

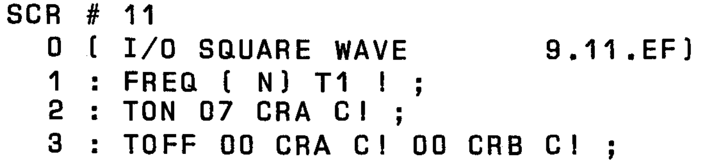
BASIC:

```
100 GOSUB 200
110 POKE CA,7
120 GOSUB 200:GOTO 120
200 INPUT"N=";N
210 IF N<0 THEN POKE CA,0:END
220 V=INT(N/256)
230 C=INT((N/256-V)*256)
235 POKE L1,C:POKE H1,V
240 RETURN
```

ASSEMBLER:

```
                 AUX       EPZ $02

                           ORG $C000

C000: A502       FREQ      LDA AUX
C002: 8D04DD               STA T1
C005: A503                 LDA AUX+1
C007: 8D05DD               STA T1+1
C00A: A907                 LDA #07
C00C: 8D0EDD               STA CRA      ;START
C00F: 00                   BRK           SQUARE WAVE

C010: A900                 LDA #0
C012: 8D0EDD               STA CRA      ;STOP
C015: 00                   BRK           SQUARE WAVE

PHYSICAL ENDADDRESS: $C016
```

FORTH:



```
SCR # 11
  0 ( I/O SQUARE WAVE          9.11.EF)
  1 : FREQ ( N) T1 ! ;
  2 : TON 07 CRA C! ;
  3 : TOFF 00 CRA C! 00 CRB C! ;
```

Figure 2-27: Square Wave at PB6.

In BASIC, the number is entered in the subroutine starting at line 200. It is divided by 256. The quotient is stored in T1H (H1) and the remainder in T1L (L1). Entering a number less zero stops the timer.

In the Assembler program, it is assumed that the number for the frequency is stored at location $02 and $03. The contents of these locations are written into the timer latches.

In FORTH, the following words are defined. The word FREQ ( N) determines the frequency. The number N must be on the stack. The ! (store) word in FORTH stores a 16 bit number in two consequtive memory cells. Thus N is stored in T1L and T1H. The word TON switches the timer on and the word TOFF switches it off.

Once the timer is started, it runs independently from the CPU. The computer can do other things and is only needed for changing the frequency or stopping the timer.

The number N is equal to half of the period of the frequency of the square wave. If the internal clock of the C64 is used, the corresponding frequency can be calculated by

f=fc/(2*N)

with fc as clock frequency of the computer. For a given f, we get N by

N=fc/(2*f)

The C64 has a clock frequency of about 1MHz. For a square wave of 1kHz, the number N is 500. To obtain exact values, the clock frequency of the computer must be measured by a digital frequency counter.

## 2. 22 Measuring the Duration of a Pulse and a Period.

The optical limit switch in Figure 2-23 is used to measure the speed of a moving object and the frequency of a mechanical pendulum. The measurement of the speed is equal to the duration of a pulse. When the object enters the light barrier, the counter is started, the counter is then turned off when the object leaves it.

In order to be independent of the clock period of the computer, an external oscillator with 1. 000000 MHz crystal frequency is used. The circuit is shown in Figure 2-28.

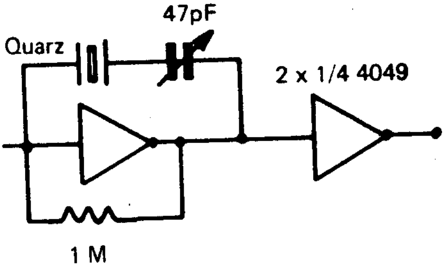


Figure 2-28: External Oscillator with 1 MHz Crystal Frequency.

The external clock is fed to Timer A via the CNT2 input of the USER-Port. It is programmed to produce a zero crossing every 1ms. Timer B is linked to Timer A and programmed to count the zero crossings of Timer A.

During the experiments with the timers, it was found that when using an external clock, N is no longer half of the period. N+1 now is the full period. The external frequency is also divided by two. N becomes 249 for a square wave of 0.5kHz ((N+1)*4=1000). The Figures 2-29 and 2-30 show the output of the Timers A and B for a ratio of 9 and 10.

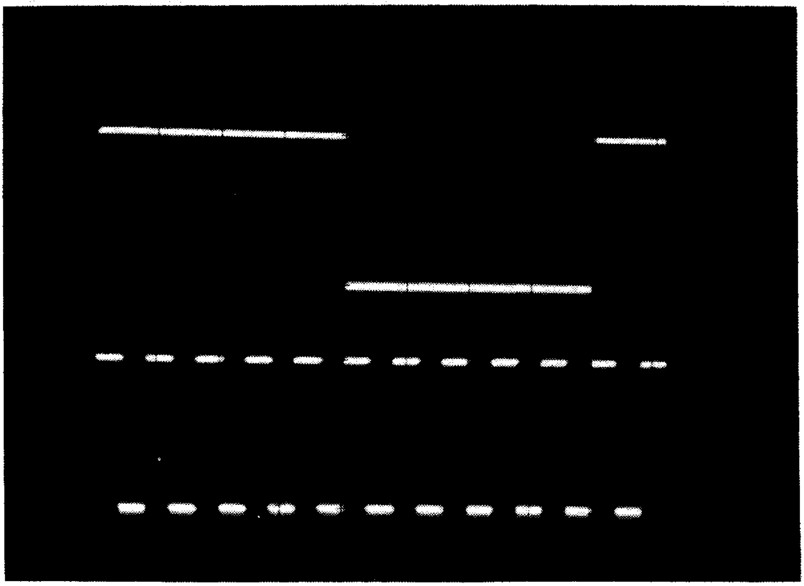


Figure 2-29: Ratio with N=9.

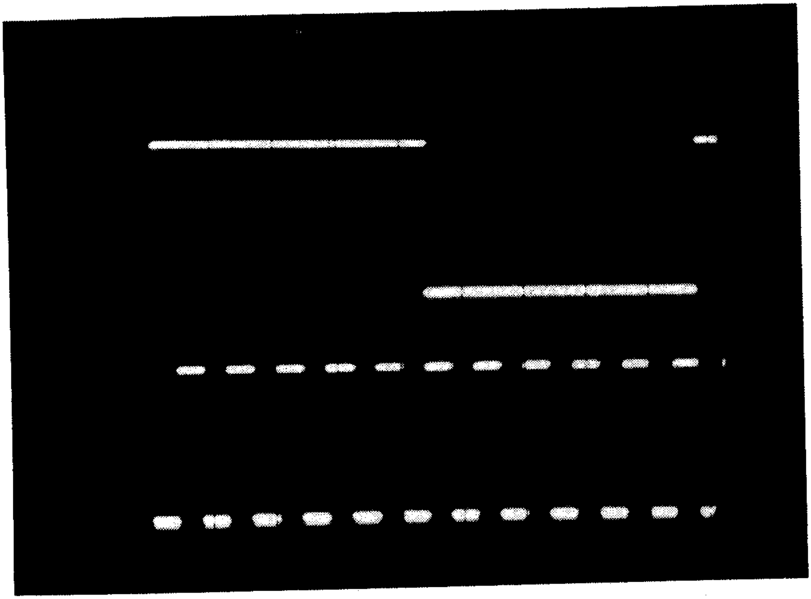


Figure 2-30: Ratio with N=10.

The program for measuring the length of a pulse is shown in Figure 2-31.

BASIC: Not illustrated. Running time is to slow.

ASSEMBLER:

```
                AUX        EPZ $02

                           ORG $C000

C000: A930      PULSE      LDA #$30
C002: 8502                 STA AUX
C004: 8D06DD               STA T2
C007: A975                 LDA #$75
C009: 8503                 STA AUX+1
C00B: 8D07DD               STA T2+1    ;30000 -->
C00E: A9F9                 LDA #249          T2,AUX
C010: 8D04DD               STA T1
```

```
C013: A900          LDA #0
C015: 8D05DD        STA T1+1   ;1MS --> T1
C018: AD01DD M      LDA PORTB
C01B: 2901          AND #%00000001
C01D: F0F9          BEQ M      ;POSITIVE EDGE?
C01F: A947          LDA #$47
C021: 8D0EDD        STA CRA
C024: 8D0FDD        STA CRB    ;START TIMER
C027: AD01DD M1     LDA PORTB
C02A: 2901          AND #%00000001
C02C: D0F9          BNE M1     ;NEGATIVE EDGE?
C02E: A900          LDA #0
C030: 8D0EDD        STA CRA
C033: 8D0FDD        STA CRB    ;TIMER STOP
C036: 38            SEC
C037: A502          LDA AUX
C039: ED06DD        SBC T2
C03C: 8502          STA AUX
C03E: A503          LDA AUX+1
C040: ED07DD        SBC T2+1
C043: 8503          STA AUX+1 ;CALCULATING
C045: 00            BRK                  TIME T

PHYSICAL ENDADDRESS: $C046
```

FORTH:

```
  4 : F1 ( N) T1 ! ;
  5 : F2 ( N) T2 ! ;
  6 : 1MS 249 F1 ;
  7 : FM 1MS 30000 F2  ;
  8 : CON FM BEGIN PORTB C@ 1 AND
  9   0=  NOT UNTIL 71 CRA C!
 10   71 CRB C! ;
 11 : COFF BEGIN PORTB C@ 1 AND
 12   0= UNTIL TOFF ;
 13 : DT 30000 T2 @ - CR . ."  MS" ;
 14  : PULSE CON COFF DT ;
 15
 OK
```

Figure 2-31: Measuring the Duration of a Pulse.

BASIC is to slow for this measurement. It can only be used with machine code subroutines.
In Assembler, the number 30000 is stored in memory locations AUX and AUX+1 and also in Timer B. The maximum length of a pulse is 30s. The number 249 is stored in Timer A. This generates zero crossings every 1ms with an external frequency of 1MHz. In the loop at label M, the program awaits a positive edged pulse. This signal is generated by interrupting the light barrier. Both timers are then started. The program will now await a negative edged pulse at loop M1. Both timers are stopped and the difference between the content of Timer B and 30000 is calculated.

The word PULSE measures one period of a pulse. CON stores the number in Timer A for 1ms and 30000 in Timer B. It then awaits a positive edged pulse and then starts the timer. COFF awaits the negative edged pulse and then stops the timer. The word DT calculates the difference between 30000 and the content of Timer B. This number is then printed on the screen.

Figure 2-33 illustrates the program for measuring the frequency of a mechanical pendulum. The circuit is shown in Figure 2-32. A small paper strip is mounted at the lower end of the pendulum. This will interrupt the light barrier when the pendulum swings. The Timers are started with the first interrupt. The next interruption is ignored and the timers are stopped on the third interruption.

In Assembler, the program awaits a positive edged in the subroutine PFL and a negative edged pulse in the subroutine NFL. The loop WAIT is needed, because the computer operates faster than the analog world.

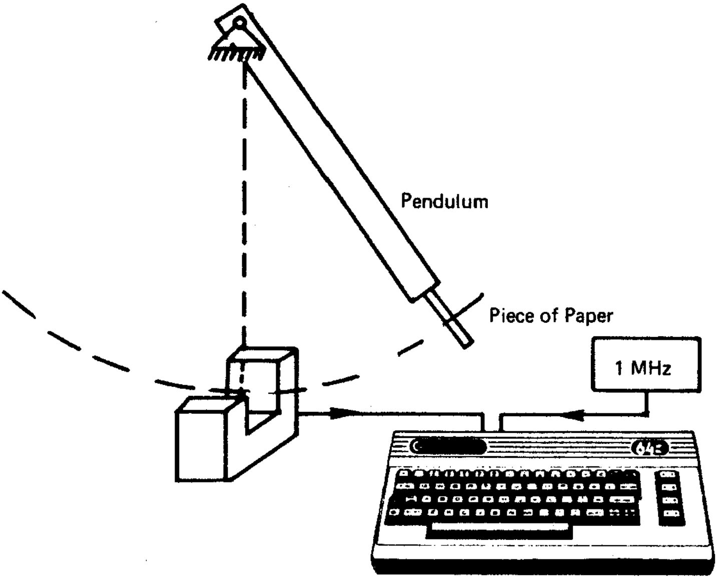


Figure 2-32: Circuit for the Measurement of the Periode of a Pendulum.

FORTH uses the words CON, TOFF, and DT from the last program. The word FL awaits a negative and then a positive edge. One measurement is done with the word ? T. CON awaits for the first positive edge. The timers are started. The measurement is finished after the third positive edge. DT calculates the time and prints it on the screen.

BASIC: Not illustrated. Running time to slow.

ASSEMBLER:

```
                 AUX        EPZ $02

                            ORG $C000

      C000: 201AC0          JSR MAIN
      C003: 00              BRK
```

```
C004: AD01DD PFL       LDA PORTB
C007: 2901             AND #%00000001
C009: F0F9             BEQ PFL
C00B: 60               RTS

C00C: AD01DD NFL       LDA PORTB
C00F: 2901             AND #$00000001
C011: D0F9             BNE NFL
C013: 60               RTS

C014: A280    WAIT     LDX #$80
C016: CA      W        DEX
C017: D0FD             BNE W
C019: 60               RTS

C01A: A930    MAIN     LDA #$30
C01C: 8502             STA AUX
C01E: 8D06DD           STA T2
C021: A975             LDA #$75
C023: 8503             STA AUX+1
C025: 8D07DD           STA T2+1    ;30000 -->
C028: A9F9             LDA #249        T2,AUX
C02A: 8D04DD           STA T1
C02D: A900             LDA #0
C02F: 8D05DD           STA T1+1    ;1MS --> T1
C032: 2004C0           JSR PFL
C035: A947             LDA #$47
C037: 8D0EDD           STA CRA
C03A: 8D0FDD           STA CRB     ;START TIME
C03D: 2014C0           JSR WAIT
C040: 200CC0           JSR NFL
C043: 2004C0           JSR PFL
C046: 2014C0           JSR WAIT
C049: 200CC0           JSR NFL
C04C: 2014C0           JSR WAIT
C04F: 2004C0           JSR PFL
C052: A900             LDA #0
C054: 8D0EDD           STA CRA
C057: 8D0FDD           STA CRB     ;TIMER STOP
C05A: 38               SEC
C05B: A502             LDA AUX
C05D: ED06DD           SBC T2
```

```
C060: 8502          STA AUX
C062: A503          LDA AUX+1
C064: ED07DD        SBC T2+1
C067: 8503          STA AUX+1  ;CALCULATING
C069: 00            BRK         TIME T

PHYSICAL ENDADDRESS: $C06A
```

FORTH:

```
   SCR # 16
     0 ( MEASUREMENT OF A PERIODE   EF)
     1 : FL BEGIN PORTB C@ 1 AND 0=
     2   UNTIL 10 0 DO LOOP
     3   BEGIN PORTB C@ 1 AND 1 =
     4   UNTIL ;
     5
     6 : ?T CON FL FL TOFF DT ;
     7
```

Figure 2-33: Measuring the Period of a Pendulum.

## 2.3 Programming the Time of the Day Clock.

The "Time of the Day Clock" of the 6526 is a general purpose, 24 hour (AM/PM) clock for real time applications. It is organized into four registers: 1/10 seconds, Seconds, Minutes and Hours. The AM/PM flag is bit 8 of the hour register. The readout of the registers is in BCD format. There are no problems with this in Assembler or in FORTH. If the clock is set in BASIC, a number conversion must be made. This is shown in the following example.

The register for the Minutes must be set to 54. The 5 goes into the upper four bits, the 4 into the 4 lower bits. The content of the register is in binary

01010100=$54

If 54 is entered into the register, the content would be

00110110=$36

The number 54 must be regarded as a hexadecimal number. It must be converted to a decimal number prior to entering it into the register. CRA7 determines if the clock is used with 50 or 60Hz. CRB7 determines if it is used as alarm or as normal clock.

BASIC:

```
 100 SS=A+8
 110 S= A+9
 120 M= A+10
 130 H= A+11
 200 INPUT "H= (0-23)";HS
 205 IF HS>11 THEN HS=HS-12:AM=1
 210 V=HS:GOSUB 300
 215 IF AM=1 THEN V=V+128
 218 POKE H,V
 220 INPUT "M= (0-59)";MI
 230 V=MI:GOSUB 300:POKE M,V
 240 INPUT "S= (0-59)";SE
 250 V=SE:GOSUB 300:POKE S,V
 260 INPUT"START (J)";A$
 270 POKE CA,128:POKE SS,0
 280 END
 300 V1=INT(V/10): V2=(V/10-V1)*10
 310 V=V1*16+V2
 320 RETURN
 500 HS=PEEK(H):MI=PEEK(M)
 510 SE=PEEK(S):S10=PEEK(SS)
 520 V=HS
 530 IF V>127 THEN PRINT " PM ";:V=V-128:
     GOTO 550
 540 PRINT " AM ";
 550 GOSUB 600
 560 V=MI:GOSUB 600
 570 V=SE:GOSUB 600
 580 END
```

```
600 V1=INT(V/16): V2=(V/16-V1)*16
610 V=V1*10+V2
620 PRINT"/";V;
630 RETURN
```

ASSEMBLER:

```
                      PORTB     EQU $DD01
                      DDRB      EQU $DD03
                      T1        EQU $DD04
                      T2        EQU $DD06
                      SS        EQU $DD08
                      SEK       EQU $DD09
                      MIN       EQU $DD0A
                      HRS       EQU $DD0B
                      CRA       EQU $DD0E
                      CRB       EQU $DD0F

                      AUX       EPZ $F8

                                ORG $C000

    C000: A980        MAIN      LDA #$80
    C002: 8D0EDD                STA CRA
    C005: A900                  LDA #00
    C007: 8D0FDD                STA CRB
    C00A: A5F8                  LDA AUX
    C00C: 8D0BDD                STA HRS
    C00F: A5F9                  LDA AUX+1
    C011: 8D0ADD                STA MIN
    C014: A5FA                  LDA AUX+2
    C016: 8D09DD                STA SEK
    C019: A900                  LDA #00
    C01B: 8D08DD                STA SS
    C01E: 00                    BRK


                                ORG $C100
    C100: AD0BDD                LDA HRS
    C103: 85F8                  STA AUX
    C105: AD0ADD                LDA MIN
    C108: 85F9                  STA AUX+1
```

```
C10A: AD09DD           LDA SEK
C10D: 85FA             STA AUX+2
C10F: AD08DD           LDA SS
C112: 00               BRK

PHYSICAL ENDADDRESS: $C113
```

FORTH:

```
SCR # 17
  0 ( TIME OF THE DAY CLOCK        EF)
  1 HEX DD08 CONSTANT 1/10
  2 DD09 CONSTANT SEC
  3 DD0A CONSTANT MIN
  4 DD0B CONSTANT STD
  5 0 CONSTANT AM
  6 1 CONSTANT PM DECIMAL
  7 : ./ 47 EMIT ;
  8 : ?TI STD C@ MIN C@ SEC C@
  9   1/10 C@ ;
 10 : ?TIME ?TI DROP >R >R DUP 127 >
 11   IF 128 - ." PM " ELSE ." AM "
 12   THEN HEX . ./ R> . ./ R> .
 13   DECIMAL  ;
 14    -->
 15

SCR # 18
  0 ( TIME OF THE DAY CLOCK CNTD EF)
  1 HEX 80 CRA C! DECIMAL
  2 : STI ( HMSZ) >R >R >R >R IF
  3   128 ELSE 0 THEN R> + STD C!
  4   R> MIN C! R> SEC C! R> 1/10
  5   C! DECIMAL ;
  6
  7
  8
```

Figure 2-34: Programming the Time of Day Clock.

When the clock is set, the values are stored in the registers starting with the Hour register. It begins to run if a value is stored in the 1/10 register. The time of the clock can be read by reading first the Hour register. The content of the other registers are stored in a latch. The latch is free for another time after reading the 1/10 register.

The BASIC program starts in line 200. Hours, minutes and seconds are entered one after another. The hexadecimal to decimal conversion is done in the subroutine starting in line 300. In line 260, the program awaits the starting the clock. Pressing any key starts the clock. The readout of the clock can be done with a GOTO 500.

The registers are read and converted to a hexadecimal number. The result is displayed on the screen. The GOTO 500 is only valid if the program has not been changed. If it has been changed, then a new program for reading only the clock must be used.

In the Assembler program, the memory locations $F8 to $FA are used to set the clock. This is done by starting the program at $C000. At $C100, the program reads the time back into the locations above.

In FORTH, the word ?TIME reads the registers and displays them on the screen in this manner:

```
PM 1 / 40 / 29
```

The word STI sets the clock. The input of

```
HEX AM 10 35 0 0 STI
```

sets the time to 10 Hours 35 Minutes 0 Seconds and 0 1/10 Seconds. This clock is a very powerful tool for real time applications.

## 2.4 Using the Analog Digital Converter uA9708.

The Analog Digital converter uA9708 is a converter with 6 analog channels, decoder, sample and hold circuit, integrator and voltage reference. The working of this converter is shown in Figure 2-35. A negative edge at RAMP START begins the charging of a capacitor to a voltage of Vin-0. 7 volts. After receiving the positive edge of the pulse, the capacitor is discharged by a constant current. The time for discharge depends on the reference voltage. If the voltage across the capacitor is less than a given voltage, the RAMP STOP signal goes low. The time between the positive edge of the RAMP START signal and the negative edge of the RAMP STOP signal is a measure for the unknown input voltage Vin.

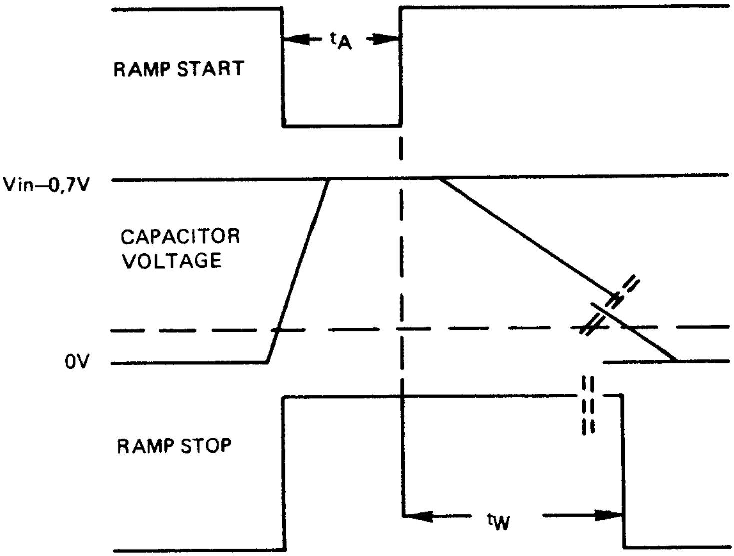


Figure 2-35: Working the ADC uA9708.

The converter uses only 5 I/O lines. These are the address lines A0, A1 and A2, the output RAMP

START and the input RAMP STOP. The pin layout of the ADC is shown in Figure 2-36 and the practical circuit in Figure 2-37.

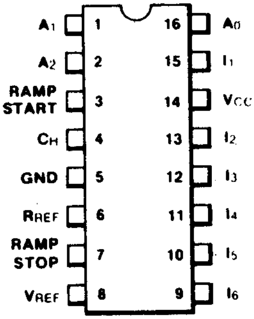


Figure 2-36: Pin Layout of the uA9708.

If the converter is used with the BASIC language, the two machine code routines starting at $C000 and $C020 must be called with the SYS command. The BRK command must be replaced by the RTS command. The routine INIT sets the Port B and stores 08 there. The Timer A is set to $FFFF. A A data conversion is started with

```
LDA #0X
STA PORTB
```

X is the number of the channel, channels are numbered 0 through 7. After a definite time delay for charging the capacitor, discharging the capacitor and the Timer A is started with the command

```
  LDA #0X+8
  STA PORTB
  LDA #07
  STA CRA
```

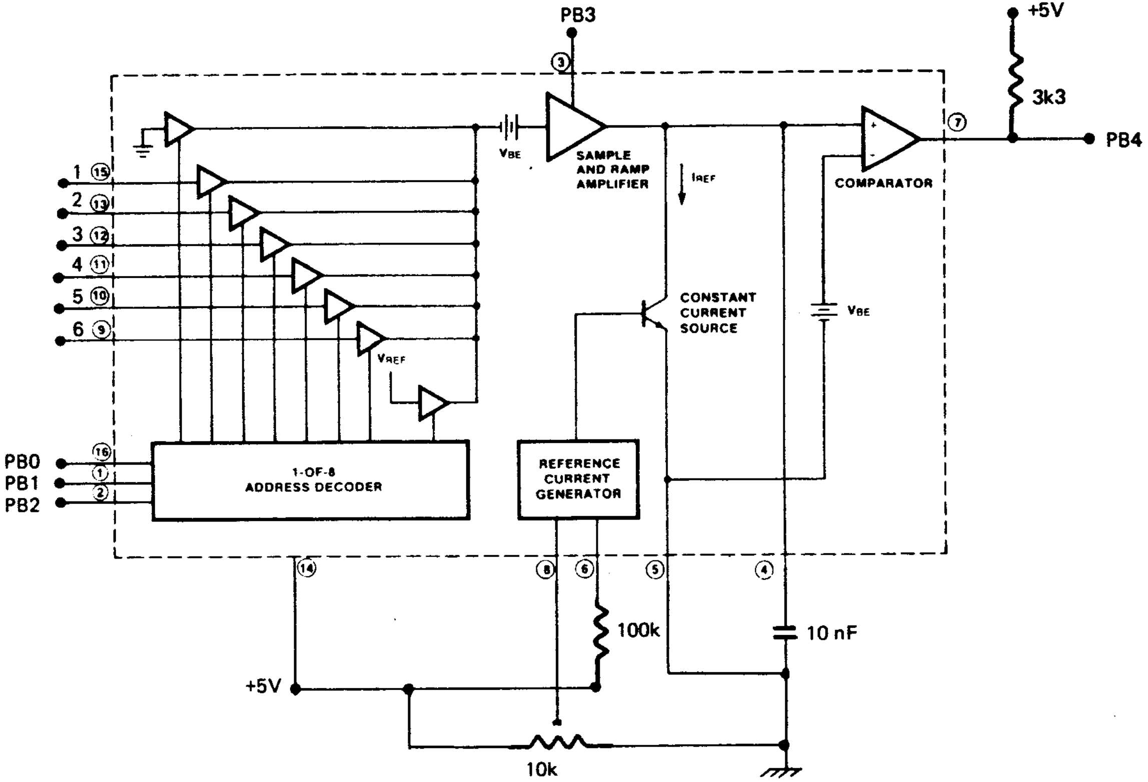


Figure 2-37: Schematic of the uA9708.

BASIC: not implemented.

ASSEMBLER:

```
                                ORG $C000

   C000: A90F    INIT           LDA #$0F
   C002: 8D03DD                 STA DDRB
   C005: A908                   LDA #08
   C007: 8D01DD                 STA PORTB
   C00A: A9FF                   LDA #$FF
   C00C: 8D04DD                 STA T1
   C00F: 8D05DD                 STA T1+1
   C012: 00                     BRK

                                ORG $C020
   C020: A9FF    ADW            LDA #$FF
   C022: 8D04DD                 STA T1
   C025: 8D05DD                 STA T1+1
   C028: A901                   LDA #01
   C02A: 8D01DD                 STA PORTB
   C02D: A230                   LDX #$30
   C02F: CA      AA             DEX
   C030: D0FD                   BNE AA
   C032: A909                   LDA #09
   C034: 8D01DD                 STA PORTB
   C037: A907                   LDA #07
   C039: 8D0EDD                 STA CRA'
   C03C: AD01DD  AB             LDA PORTB
   C03F: 2910                   AND #%00010000
   C041: D0F9                   BNE AB
   C043: A900                   LDA #00
   C045: 8D0EDD                 STA CRA
   C048: 00                     BRK

   PHYSICAL ENDADDRESS: $C049
```

FORTH:

```
 SCR # 21
   0 ( CODE ADW                        22.11.EF)
   1 CODE ADW XSAVE STX, HEX
   2 30 # LDA, T1 STA, 75 # LDA,
T1 1+ STA, DECIMAL
PORTB LDA, 7 # AND, PORTB STA,
48 # LDX, BEGIN, DEX, 0= UNTIL,
8 # ORA, PORTB STA,
7 # LDA, CRA STA,
BEGIN, PORTB LDA, 16 # AND,
0= UNTIL, 0 # LDA, CRA STA,
XSAVE LDX, DEX, DEX, T1 LDA,
BOT STA, T1 1+ LDY, BOT 1+ STY,
NEXT JMP, END-CODE


SCR # 20
( ADW TEST                    22.11.EF)
: TT BEGIN INIT 1 CHA ADW . CR
  ?TERMINAL UNTIL ;
0 VARIABLE TMI
: TMIN INIT 0 CHA ADW TMI ! ;
: MESS ( N) CHA ADW TMI @ SWAP
  - . ;

```

Figure 2-38: Program to Control the ADC uA9708.

Port B is now tested for the negative edge of RAMP STOP. After this signal, the timer is stopped. The difference between $FFFF and the content of the timer latches is the elapsed time between both signals. The input of channel 0 is grounded. The time measured with this channel equals an input voltage of 0 volts. The input of channel 7 is Vref. This is the maximum voltage and time, respectivly, which can be measured.

The word INIT in FORTH initializes the port. CHA ( N) stores the number of the wanted channel in Port B. The word ADW is written in Assembler. It is similar to the program ADW in the Assembler program. At the end of the conversion, it places the content of the timer latches on the stack. The word TMIN determines the conversion time for channel 0. The result is stored in the variable

TMI. The word MESS takes the number of the channel from the stack and makes one conversion. It then calculates the difference between the conversion time of channel 0 and the conversion time just measured. The result is displayed on the screen. A measurement always starts with TMIN. The voltage at channel 3 can then be measured with 3 MESS. The input of 7 MESS determines the time for a maximum input. This value can be used as a scale factor.

This ADC can be used for measuring the temperature at different places. A thermistor can be used as a sensor. The resistance of a thermistor depends on the temperature. Figure 2-39 shows the characteristic of a thermistor with a negative temperarure coefficient. The curve is linear between the temperatures T1 and T2. Because there are so many different varieties of thermistors, there are no numbers given in Figure 2-39. The best way to select one is to use the circuit in Figure 2-40 to connect the thermistor to the ADC. Trials have to be made to get the value of the resistor R for a given thermistor and a given temperature range.

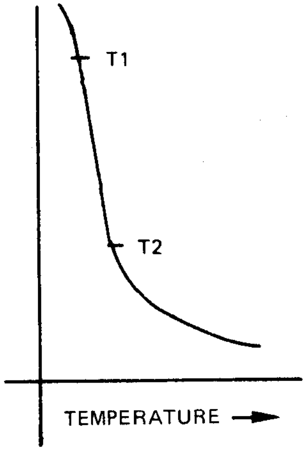


Figure 3-2: Pin Layout of the AD7574. ;tor.

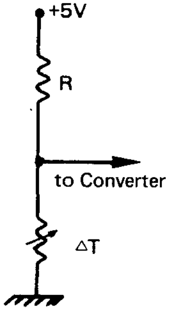
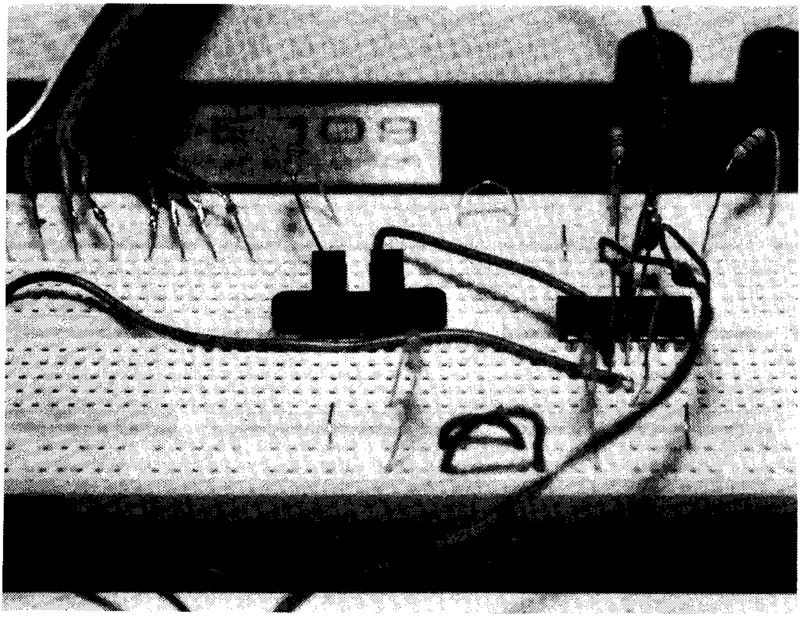


Figure 2-40: Connecting a Thermistor to the ADC.



Light Barrier

# 3 Hardware Extensions Using the Expansion Port

3. 0 Hardware Extensions Using the Expansion Port.

The Expansion Port of the C64 provides all address, data and control lines needed for the use of hardware extension. The pin layout of the Expansion Port is shown in Figure 3-1.

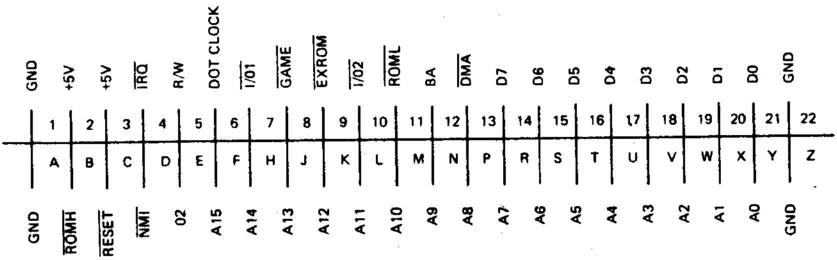


Figure 3-1: Pin Layout of the Expansion Port.

If the expansion port is used for the extension in conjunction with I/O Ports, a uP compatible DA/AD converter and certain other devices, a a certain address range must be decoded. The next chapter will show how this is decoding is done. In this chapter two decoded address ranges inside the C64 will be used. The line I/O1 goes low if an address between DE00 and DEFF is selected. The line I/O2 goes low if an address between DF00 and DFFF is selected. These two ranges will be used for expansion.

## 3.1 Connection of the ADC AD7574.

The Analog-Digital Converter AD7574 is an 8-bit converter in CMOS Technology, produced by Analog Device Inc. The pin layout is shown in Figure 3-2.

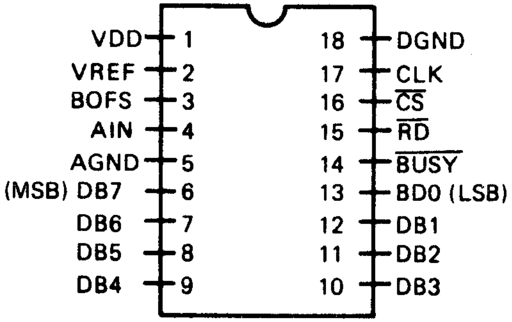


Figure 3-2: Pin Layout of the AD7574.

The ADC is selected with the CS input. With CS=L and RD=H, a conversion is begun. This conversion is done by successive approximation. This will be clarified shortly. During conversion, the BUSY output is low. The negative edge of a pulse at RD places the data on the output lines DB0 to DB7. The conversion time is controlled by an RC network and lasts about 15 to 20 us. The input voltage range depends on the reference voltage. When Vref=-10 volts, the input voltage ranges from 0 to +10 volts. The internal block diagram is shown in Figure 3-3.

The 2R/R resistor ladder can be connected either to ground or to the summation node of a comparator. The unknown input voltage is first compared with Vref/2. If the input voltage is larger than Vref/2, the switch DB7 stays on the summing node. Otherwise the 2R resistor is grounded. The next comparison takes place with either Vref/4 or Vref/2+Vref/4. The switch DB6 will stay in the summation node or will be grounded, in regard whether the input voltage is higher or not. In this way the unknown input voltage is succesively approximated.

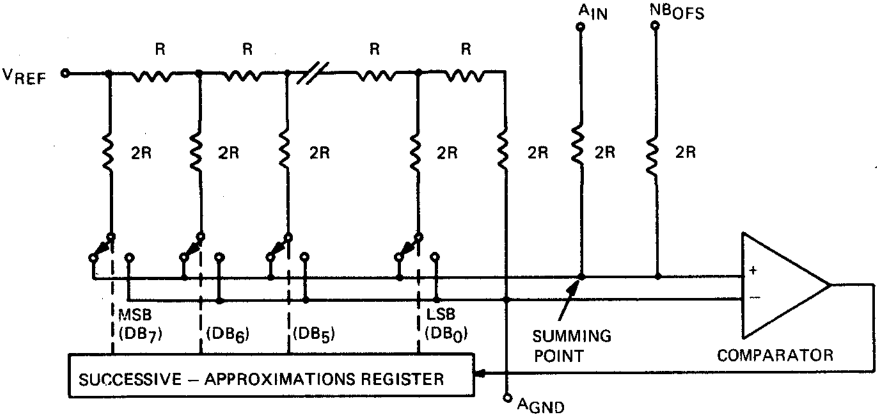


Figure 3-3: Internal Block Diagram of the AD7574.

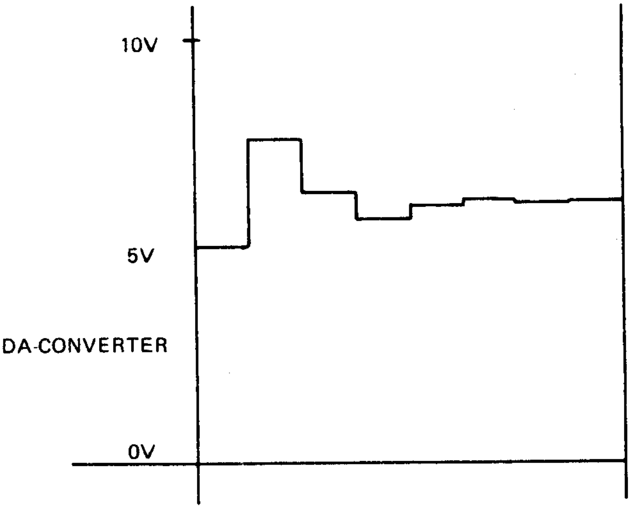
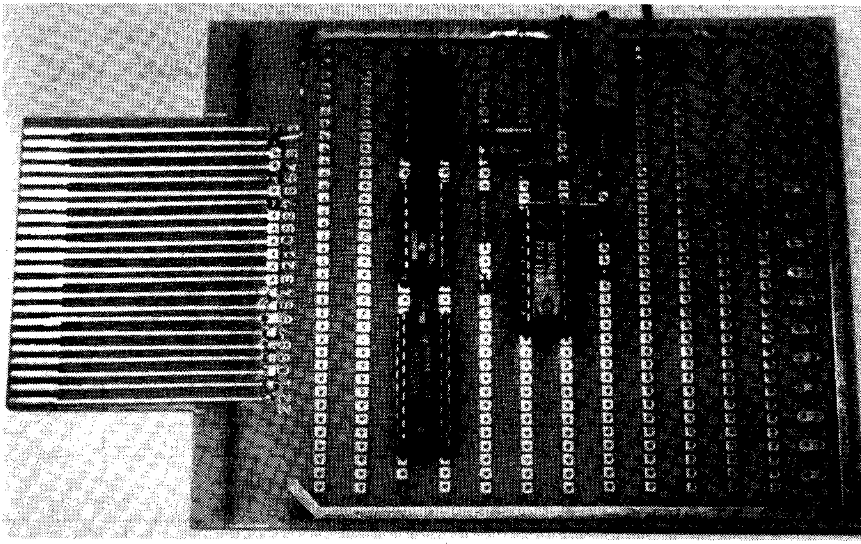


Figure 3-4: Succesive Approximation

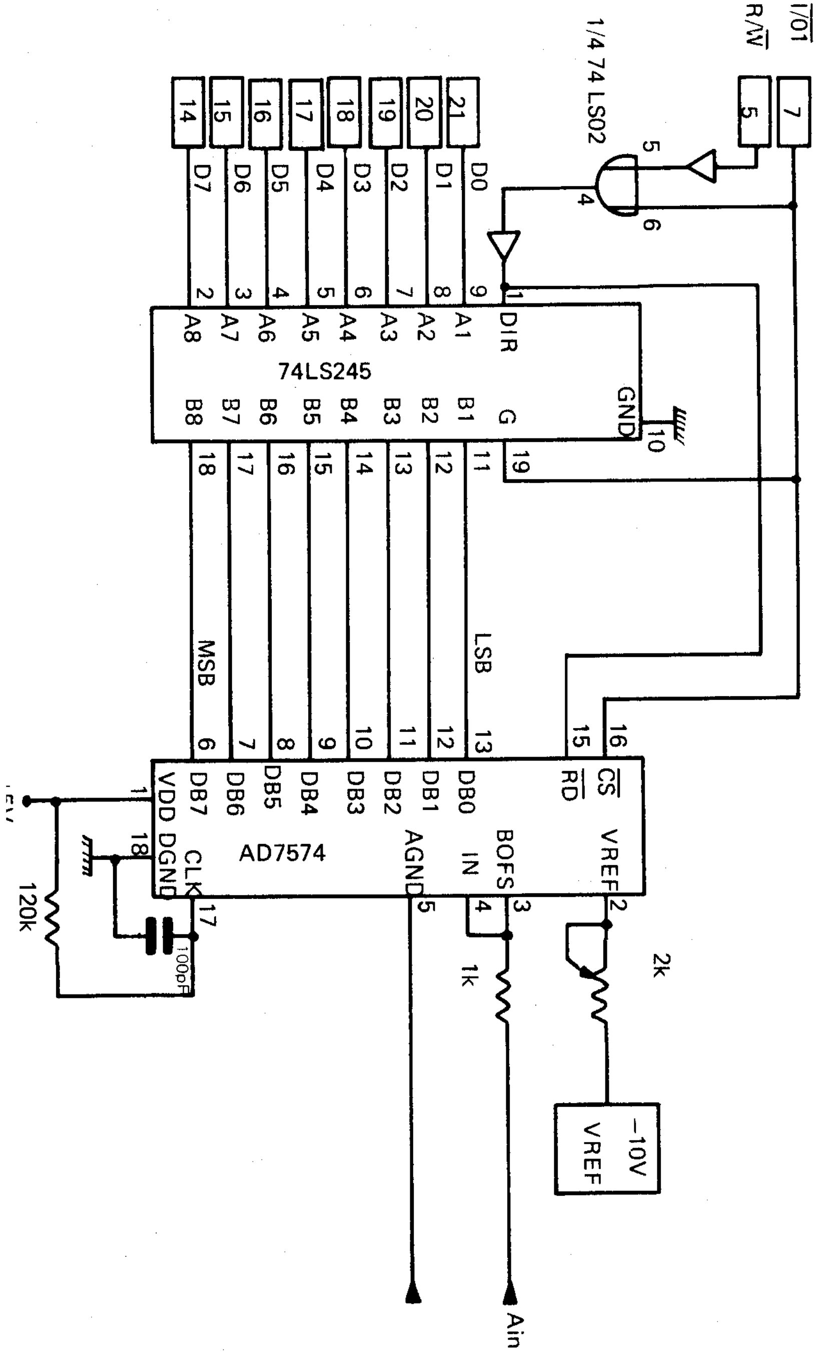
Figure 3-4 shows the succesive approximation of a unknown input voltage.



AD7574

The practical circuit is shown in Figure 3-5. A Bus Transceiver SN74LS245 is used to separate the data bus of the computer from the data outputs of the ADC. Such a bus transceiver should always be used if experiments are made with the data lines. It is less possible to destroy parts in the inside of the computer. Both integrated circuits are selected by the I/O1 line. The transceiver sends the data from B to A if RD is low. This signal is derived from the I/O1 and the R/W signals. The -10 volts can be provided by a Zener diode or the voltage reference diode AD584.

Great care should be taken when wiring the Ground lines of this circuit. All digital Ground lines must be wired to DGND and all analog Ground lines must be wired to AGND. A malfunction often occurs with bad ground connections.


I/O1
R/W
1/4 74 LS02
7
5
5
4
6
21
20
19
18
17
16
15
14
D0
D1
D2
D3
D4
D5
D6
D7
1
9
8
7
6
5
4
3
2
DIR
A1
A2
A3
A4
A5
A6
A7
A8
74LS245
GND
G
B1
B2
B3
B4
B5
B6
B7
B8
10
19
11
12
13
14
15
16
17
18
LSB
MSB
16
15
13
12
11
10
9
8
7
6
CS
RD
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
VDD
DGND
CLK
AD7574
VREF
BOFS
IN
AGND
1
18
17
2
3
4
5
2k
1k
100pF
120k
−10V
VREF
Ain

A conversion is begun with CS=L and RD=H. The R/W signal must therefore be low. This is done with a WRITE command to location $DE00=56832. The following program continously converts the voltage at the input into a number and prints it on the screen.

BASIC:

```
100 ADW=56832
110 POKE ADW,1:PRINT=PEEK(DAW)
120 GOTO 110
```

ASSEMBLER:

```
                      BSOUT     EQU $FFD2
                      ADW       EQU $DE00
                      AUX       EPZ $F8

                                ORG $C000

   C000: 2023C0                 JSR MAIN
   C003: 00                     BRK

   C004: 85F8         PRTBYT    STA AUX
   C006: 4A                     LSR
   C007: 4A                     LSR
   C008: 4A                     LSR
   C009: 4A                     LSR
   C00A: 2015C0                 JSR PRT
   C00D: A5F8                   LDA AUX
   C00F: 2015C0                 JSR PRT
   C012: A5F8                   LDA AUX
   C014: 60                     RTS

   C015: 290F         PRT       AND #$0F
   C017: C90A                   CMP #$0A
   C019: 18                     CLC
   C01A: 3002                   BMI P
   C01C: 6907                   ADC #$07
   C01E: 6930         P         ADC #$30
   C020: 4CD2FF                 JMP BSOUT
```

```
C023: 8D00DE MAIN     STA ADW
C026: A210            LDX #$10
C028: CA      M       DEX
C029: D0FD            BNE M
C02B: AD00DE          LDA ADW
C02E: 2004C0          JSR PRTBYT
C031: A90D            LDA #$0D
C033: 20D2FF          JSR BSOUT
C036: 4C23C0          JMP MAIN

PHYSICAL ENDADDRESS: $C039
```

FORTH:

```
 7
 8 HEX
 9 : CONV BEGIN 1 DE00 C! DE00 C@ .
10     ?TERMINAL UNTIL ;
11 DECIMAL
12
```

Figure 3-6: Continous Conversion of an Input Voltage.

In BASIC and also in FORTH, the time delay between the Write and the Read command is longer than 20 us. This is the time the converter needs to convert the voltage into a number. In Assembler a waiting loop has to be programmed. It is not possible to access the BUSY line with this circuit.

For an exact adjustment of the ADC, a digital voltmeter must be used. The following steps must be executed: The reference voltage has to be exactly -10. 000 volts. An input voltage of 39.1 mV (10.000/256) is applied to the input. Bit DBO must oscillate between L and H. The gain is adjusted in applying 9. 961 volts (FS-39mV) to the input. All bits should now be H, except bit DBO. The gain must be adjusted by potentiometer P, so that DBO registers between L and H. Some basic principles about AD conversions are described in Appendix B.

## 3.2 Measuring the Temperature with the Sensor STP 35.

The temperature transducer STP35 (Texas Instruments Inc) is a high precision sensor. It is a Zener diode whose Zener voltage is proportional to the absolute temperature. The sensor is linear over a wide range. If the temperature varies about 1K the output voltage changes about 10mV. At 25°C, the output voltage is 2. 98 volts. This may be adjusted as shown in Figure 3-7 (courtesy Texas Instruments).

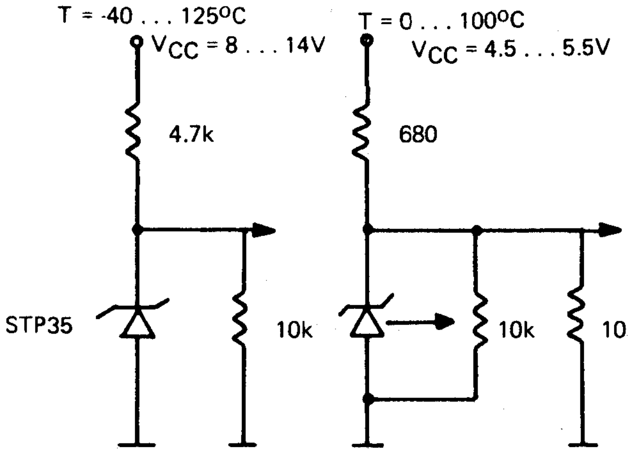


Figure 3-7: Schematic of the Temperature Transducer STP 35.

As an example, the temperature should be measured between 0°C and 100° C. The circuit shown in Figure 3-8 is an amplifier. The first stage is a differential amplifier with a gain of 1. The voltage from the transducer is fed to the negative input. A constant voltage is added at the positive input. At 20°C, the output voltage of the transducer is 2.93 volts and at 0°C, 2.73 volts. At a temperature of 20°C, the output B of the first stage is trimmed to -0. 2 volts. For this, a multiturn potentiometer P1 is used. The second stage is an amplifier with a gain of 10. This gain is adjusted with P2, also a multiturn potentiometer. The output voltage at C is 2.0 volts for 20° C. The sensifity of the transducer is now 100mV/1 K. The output of the second stage is connected to the input of the AD converter. The program is shown in Figure 3-9.

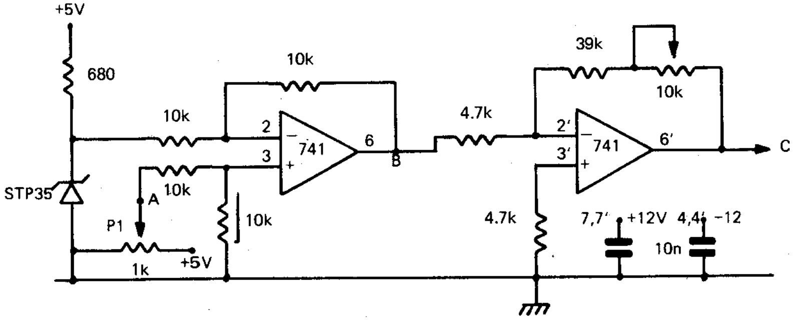


Figure 3-8: Schematic of the Temperature Measurement.

BASIC:

```
100 ADW=56832
110 POKE ADW,1:A=PEEK(DAW)
120 PRINT A
130 M=A*0.392:PRINT M
140 T=INT(M+0.5):PRINT T
```

ASSEMBLER:not implemented

FORTH:

```
SCR # 22
  ( TEMPERATURE MEASUREMENT    EF)
  HEX
  : CON ( -N) 1 DE00 C! DE00 C@ ;
  DECIMAL
  : ROUND SWAP 500 + 1000 / + ;
  : SCALE 392 1000 */MOD ROUND ;
  : M->T CON SCALE . CR ;
```

Figure 3-9: Program for Measuring a Temperature.

In the BASIC program, a measurement is made in line 110. The POKE starts the conversion, PEEK reads the ADC and places the value in the variable A. A is multiplied by 0.392 to get the temperature. For A=48, the value of M is 18. 816. Do not assume this is the exact temperature. The codewidth (see Appendix B) of our 8-bit converter is 0. 392. A measurement is always exact for +- 1 LSB. In this example, it is +- 0.4. Since the exact temperature lies somewhere between 18.4 and 19.2 ° C, the value of M is rounded to the next integer value.

The FORTH kernal has no floating point arithmetic. For data processing, this is not necessary. The accuracy depends on the codewidth of the Analog-Digital Converter.

For scaling, two words, */ and */MOD, can be used in FORTH. Both need 3 numbers on the stack. With

48 392 100 */ ,

the product, 48*392, is calculated. The result is a double precision (32 bit) product. This number is divided by 100 for single precision result. The word */MOD places the result and the remainder on the stack. The word . SCALE (print scale) is used to make some calculations.

```
: .SCALE ( N) 392 100 */ . ;
48 .SCALE 188
49 .SCALE 192
50 .SCALE 196
```

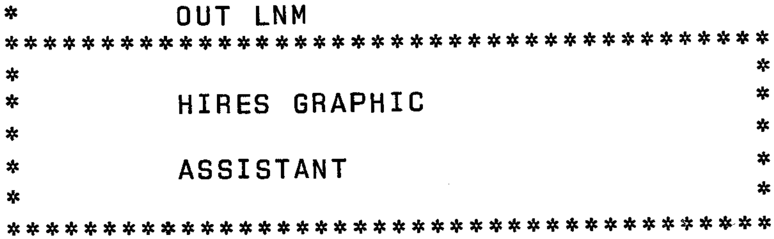
The number 192 has to be read as 19.2. The temperature lies somewhere between 18.8 and 19.6°C
In the FORTH program, the scaling is done with the word */MOD. The word ROUND is used to round to the next integer value. The word M->T makes one measurement and displayes the result on the screen.

In FORTH, it is very easy to add the Time of Day clock. If the words for the clock are compiled into the vocabulary and the clock has been started, only the word ?TIME must to be inserted into M->T for the time to be printed on every measurement.

If a Multitasking FORTH is used, the real time clock and the measurements may run in the background. The foreground is free for other tasks.

### 3.3 Drawing the Measured Values on the Screen.

In high resolution graphics, the C64 has 200 pixels in vertical direction and 320 pixels in horizontal direction. The resolution of an 8-bit ADC is higher than the resolution in the vertical direction. For plotting the points on the screen the program HIRES ASSISTANT from the book (3) is used.



```
*        OUT LNM
****************************************
*                                      *
*        HIRES GRAPHIC                 *
*                                      *
*        ASSISTANT                     *
*                                      *
****************************************
```

```
             XCOORD   EPZ $14.5
             SECADR   EQU $B9
             TEMP     EPZ $FD.E
             ADDRESS  EQU TEMP

             COLORLOW EQU $0400
             COLORHI  EQU $0800

             GRAPHICL EQU $2000
             GRAPHICH EQU $4000

             CHECKCOM EQU $AEFD
             GETBYTE  EQU $B79E
             GETCOORD EQU $B7EB
             GETPARAM EQU $E1D4

             BSOUT    EQU $FFD2
             LOAD     EQU $FFD5
             SAVE     EQU $FFD8

             VIDEO    EQU $D000

             FALSE    EQU 255
             TRUE     EQU 0

             CLS      EQU 19+128

                      ORG $C000

             * INIT HIRES GRAPHIC
             * SYS12*4096

C000: 4C18C0          JMP INIT

             * CLEAR HIRES SCREEN
             * SYS12*4096+3

C003: 4C33C0          JMP CLEAR

             * SET BACKGROUND COLOR
             * SYS12*4096+6,COLOR

C006: 4C4AC0          JMP COLOR
```

```
                  * PLOT X,Y (0 <= X < 320)
                  *          (0 <= Y < 200)

                  * SYS12*4096+9,X,Y

C009: 4C6BC0            JMP SET

                  * CLEAR X,Y
                  * SYS12*4096+12,X,Y

C00C: 4C67C0            JMP RESET

                  * SWITCH HIRES OFF AND
                  * BACK TO NORMAL MODE
                  * SYS12*4096+15

C00F: 4CD8C0            JMP SWTCHOFF

                  * SAVE HIRES GRAPHIC
                  * SYS12*4096+18,"NAME",DEVICE

C012: 4CE9C0            JMP SCREENSA

                  * LOAD HIRES GRAPHIC
                  * SYS12*4096+21,"NAME",DEVICE

C015: 4C00C1            JMP SCREENLO


                  * INIT HIRES SCREEN

C018: AD11D0 INIT       LDA VIDEO+17
C01B: 8D52C1            STA SCRATCH+1
C01E: AD18D0            LDA VIDEO+24
C021: 8D51C1            STA SCRATCH
C024: A93B              LDA #27+32
C026: 8D11D0            STA VIDEO+17
C029: A918              LDA #16+8
C02B: 8D18D0            STA VIDEO+24
C02E: A210              LDX #16
C030: 4C50C0            JMP COLOR1
```

```
                     * CLEAR HIRES SCREEN

C033: A000    CLEAR     LDY #0
C035: A920              LDA #GRAPHICL:H
C037: 84FD              STY TEMP
C039: 85FE              STA TEMP+1
C03B: 98      CLEAR1    TYA
C03C: 91FD    CLEAR2    STA (TEMP),Y
C03E: C8                INY
C03F: D0FB              BNE CLEAR2
C041: E6FE              INC TEMP+1
C043: A5FE              LDA TEMP+1
C045: C940              CMP #GRAPHICH:H
C047: D0F2              BNE CLEAR1
C049: 60                RTS

                     * SET BACK COLOR

C04A: 20FDAE  COLOR     JSR CHECKCOM
C04D: 209EB7            JSR GETBYTE

C050: A000    COLOR1    LDY #0
C052: A904              LDA #COLORLOW:H
C054: 84FD              STY TEMP
C056: 85FE              STA TEMP+1
C058: 8A      COLOR2    TXA
C059: 91FD    COLOR3    STA (TEMP),Y
C05B: C8                INY
C05C: D0FB              BNE COLOR3
C05E: E6FE              INC TEMP+1
C060: A5FE              LDA TEMP+1
C062: C908              CMP #COLORHI:H
C064: D0F2              BNE COLOR2
C066: 60      OUTRANGE  RTS


                     * (RE)SET DOT AT X,Y

C067: A9FF    RESET     LDA #FALSE
C069: D002              BNE SET1     A.T.
C06B: A900    SET       LDA #TRUE
C06D: 8D53C1  SET1      STA RSFLG
C070: 20FDAE            JSR CHECKCOM
```

```
C073:  20EBB7         JSR GETCOORD
C076:  E0C8           CPX #200
C078:  B0EC           BCS OUTRANGE
C07A:  A514           LDA XCOORD
C07C:  C940           CMP #320:L
C07E:  A515           LDA XCOORD+1
C080:  E901           SBC #320:H
C082:  B0E2           BCS OUTRANGE
C084:  8A             TXA
C085:  4A             LSR
C086:  4A             LSR
C087:  4A             LSR
C088:  0A             ASL
C089:  A8             TAY
C08A:  B90FC1         LDA MUL320,Y
C08D:  85FD           STA ADDRESS
C08F:  B910C1         LDA MUL320+1,Y
C092:  85FE           STA ADDRESS+1
C094:  8A             TXA
C095:  2907           AND #%00000111
C097:  18             CLC
C098:  65FD           ADC ADDRESS
C09A:  85FD           STA ADDRESS
C09C:  A5FE           LDA ADDRESS+1
C09E:  6900           ADC #0
C0A0:  85FE           STA ADDRESS+1
C0A2:  A514           LDA XCOORD
C0A4:  2907           AND #%00000111

C0A6:  A8             TAY
C0A7:  A514           LDA XCOORD
C0A9:  29F8           AND #%11111000
C0AB:  18             CLC
C0AC:  65FD           ADC ADDRESS
C0AE:  85FD           STA ADDRESS
C0B0:  A5FE           LDA ADDRESS+1
C0B2:  6515           ADC XCOORD+1
C0B4:  85FE           STA ADDRESS+1
C0B6:  A5FD           LDA ADDRESS
C0B8:  18             CLC
C0B9:  6900           ADC #GRAPHICL:L
C0BB:  85FD           STA ADDRESS
C0BD:  A5FE           LDA ADDRESS+1
```

```
C0BF:  6920              ADC  #GRAPHICL:H
C0C1:  85FE              STA  ADDRESS+1
C0C3:  A200              LDX  #0
C0C5:  A1FD              LDA  (ADDRESS,X)
C0C7:  2C53C1            BIT  RSFLG
C0CA:  1006              BPL  SET2
C0CC:  3949C1            AND  ANDMASK,Y
C0CF:  4CD5C0            JMP  SET3
C0D2:  1941C1  SET2      ORA  ORMASK,Y
C0D5:  81FD    SET3      STA  (ADDRESS,X)
C0D7:  60                RTS

               * SWITCH GRAPHIC OFF BACK
               * TO NORMAL MODE

C0D8:  AD52C1  SWTCHOFF  LDA  SCRATCH+1
C0DB:  8D11D0            STA  VIDEO+17
C0DE:  AD51C1            LDA  SCRATCH
C0E1:  8D18D0            STA  VIDEO+24
C0E4:  A993              LDA  #CLS
C0E6:  4CD2FF            JMP  BSOUT

               * SAVE HIRES SCREENMEMORY

C0E9:  20FDAE  SCREENSA  JSR  CHECKCOM
C0EC:  20D4E1            JSR  GETPARAM
C0EF:  A200              LDX  #GRAPHICH:L
C0F1:  A040              LDY  #GRAPHICH:H
C0F3:  A900              LDA  #GRAPHICL:L
C0F5:  85FD              STA  TEMP
C0F7:  A920              LDA  #GRAPHICL:H
C0F9:  85FE              STA  TEMP+1
C0FB:  A9FD              LDA  #TEMP
C0FD:  4CD8FF            JMP  SAVE

               * LOAD HIRES SCREENMEMORY

C100:  20FDAE  SCREENLO  JSR  CHECKCOM
C103:  20D4E1            JSR  GETPARAM
C106:  A961              LDA  #6*16+1
C108:  85B9              STA  SECADR
C10A:  A900              LDA  #0
```

```
C10C: 4CD5FF          JMP LOAD

             N        EQU 320

             * MULTIPLY TABLE

C10F: 000040 MUL320   DFW 0*N,1*N,2*N,3*N,4*N
C112: 018002
C115: C00300
C118: 05
C119: 400680          DFW 5*N,6*N,7*N,8*N,9*N
C11C: 07C008
C11F: 000A40
C122: 0B
C123: 800CC0          DFW 10*N,11*N,12*N,13*N,14*N
C126: 0D000F
C129: 401080
C12C: 11
C12D: C01200          DFW 15*N,16*N,17*N,18*N,19*N
C130: 144015
C133: 8016C0
C136: 17
C137: 001940          DFW 20*N,21*N,22*N,23*N,24*N
C13A: 1A801B
C13D: C01C00
C140: 1E

             * SET MASK FOR BIT WITHIN
             * THE SELECTED BYTE

C141: 80     ORMASK   DFB %10000000
C142: 40              DFB %01000000
C143: 20              DFB %00100000
C144: 10              DFB %00010000
C145: 08              DFB %00001000
C146: 04              DFB %00000100
C147: 02              DFB %00000010
C148: 01              DFB %00000001


             * CLEAR MASK FOR BIT WITHIN
             * THE SELECTED BYTE
```

```
C149: 7F      ANDMASK  DFB %01111111
C14A: BF               DFB %10111111
C14B: DF               DFB %11011111
C14C: EF               DFB %11101111
C14D: F7               DFB %11110111
C14E: FB               DFB %11111011
C14F: FD               DFB %11111101
C150: FE               DFB %11111110

C151: 0000    SCRATCH  DFW 0  SAVE OLD VALUES
C153: 00      RSFLG    DFB 0  SET OR RESET
C154: 00      YCOORD   DFB 0  YCOORDINATES
```

Figure 3-10: Program HIRES ASSISTANT.

The HIRES ASSISTANT uses a jump table for the various high resolution subroutines. With GRA=12*4096 there are the following entry points:

SYS GRA Initializes the high res graphics.

SYS GRA+3 Clears the high res screen.

SYS GRA+6,X Sets the background color X.

SYS GRA+9,X,Y Plots the point X, Y. 0<=X<=320 0<=Y<=200. The left upper edge of the screen has the coordinates X=0 and Y=0.

SYS GRA+12,X,Y Erases point X,Y.

SYS GRA+15 Switch back to normal mode.

SYS GRA+18,"NAME",DEVICE
Saves picture with NAME.

SYS GRA+21,"NAME",DEVICE
Loads picture NAME.

The program in Figure 3-11 converts an input voltage at the ADC into a number and plots the result on the screen. See the example in the last picture of this chapter.

BASIC:

```
100 ADW=56832
110 GRA=12*4096
200 SYS GRA:SYS GRA+3
300 FOR X=10 TO 300
310 POKE ADW,1:M=PEEK(ADW)
320 P=200-INT(M*200/256)
330 SYS GRA+9,X,P
340 FOR I=1 TO 10:NEXT I
350 NEXT X
360 INPUT A$
370 SYS GRA+15
```

ASSEMBLER: see HIRES ASSISTANT

FORTH:

```
SCR # 23
  0 ( DRAWING POINTS                      EF)
  1 ( WRITTEN IN ELCOMP GFORTH              )
  2
  3 : WAIT 1000 0 DO LOOP ;
  4 : SCALE ( N-N') 200 256 */
  5   200 SWAP - ;
  6 : DRAW HGR 311 10 DO CONV SCALE
  7   I PLOT WAIT LOOP KEY DROP
  8   TEX ;
  9
 10 ( HGR SELECTS HGR MODE                  )
 11 ( YX PLOT PLOTS POINT X,Y               )
 12 ( TEX SELECTS TEXT MODE                 )
 13
 14
```

Figure 3-11: Plotting Data on the Screen.

## 3-4 The Pressure Transducer SP10.

The SP10 is a piecoresistive pressure sensor using a small monolytic silicon chip in which a cavity is etched out to form a diaphragm. Four ion implantated resistors are integrated on the top side which is not in contact with the pressure medium and connected to a full bridge.

When an appropriate supply voltage is applied, the output voltage is proportional to the pressure. The sensitivity is 200mV/Bar.

The practical circuit is shown in Figure 3-12. The bridge of the transduser is connected to the inputs of an instrumentation amplifier. The gain of this amplifier is 10. For a pressure of one Bar, the output voltage is 2 volts.

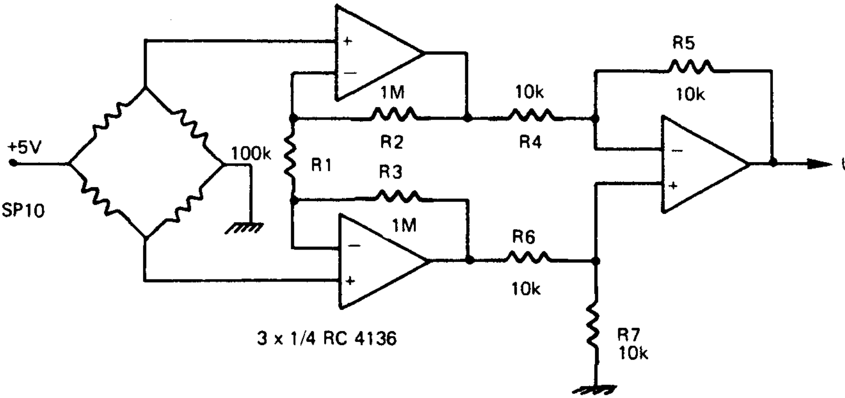


Figure 3-12: Measurement of Pressure with the SP10.

## 3.5 Connecting a CIA 6526 to the Expansion Bus.

Sometimes there are not enough lines available at the USER-Port to control external devices. To overcome this, it is yet very easy to connect an other 6526 to the Expansion Bus. This is shown in Figure 3-13. The registers are selected with the address lines A0 to A3. The CS input of the 6526 is tied to the I/01 output. The addresses of the registers then are DE00 to DE0F. If the I/02 signal is used, the addresses range from DF00 to DF0F.

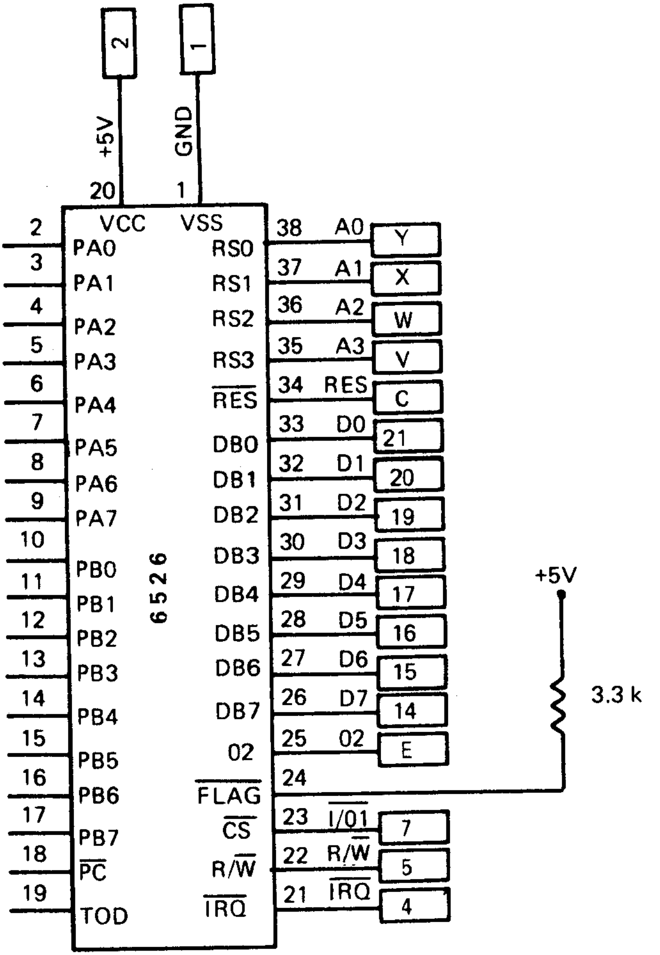


Figure 3-13: Connecting a CIA 6526 to the Expansion Port.

The FLAG input is connected with a pull-up resistor to the positive supply voltage. All other lines are connected directly to the Expansion Port. If the Time of the Clock is to be used, a 60 Hz square wave has to be applied at input TOD. Figure 3-14 shows a practical circuit ( courtesy COMMODORE Inc). The AC voltage can be obtained from pin 10 of the USER-Port.

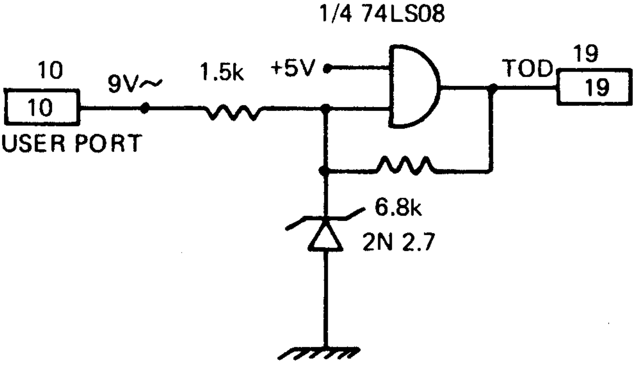


Figure 3-14: Generating a 60 Hz Square Wave.

## 3.6 Connection of the Digital Analog Converter ZN428E.

The Digital Analog Converter ZN428E (Ferranti Inc) is an 8-bit monolytic DA converter. The digital inputs are latched so it can be directly connected to a data bus. With ENABLE=L, the input is converted into a voltage. With ENABLE=H, the digital code is stored and changes of the input lines are ignored. Figure 3-15 shows the connection of the ZN428E to a CIA 6526. As there are only 8 lines used, it could be also connected to the USER-Port. The output of the converter is connected to an operational amplifier. This is used as a non-inverting amplifier.

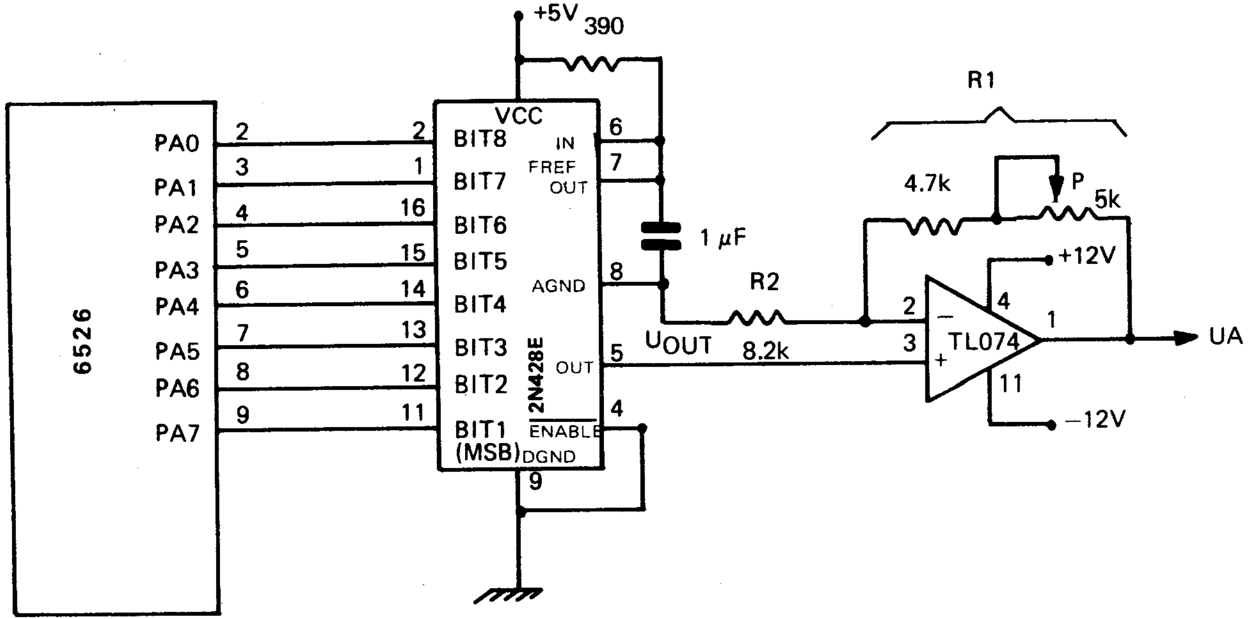


Figure 3-15: Control of the DAC ZN428E.

The output voltage is

$$Ua=(1+R1/R2)*Uout$$

For full-scale (all input lines are high), the output voltage is

$$Ua=(1+R1/R2)*Uref$$

The internal reference voltage is 2.5 volts. The non-inverting amplifier has a gain of 2. This can be adjusted with potentiometer P. For a full-scale of 5 volts, the output voltage must be 4. 82 volts with all input lines high. The operational amplifier TL074 can be replaced by a 741 operational amplifier. The program in Figure 3-16 can be used to adjust the output voltage. The 6526 registers have the addresses DE00 to DE0F in this program.

BASIC:

```
100 PA=56832
110 POKE PA+2,255
130 INPUT"I=";I
135 IF I<0 THEN END
140 POKE PA,I
150 GOTO 130
```

ASSEMBLER: not implemented.

FORTH:

```
SCR # 25
  0 ( ADW 428                     24.11.EF)
  1 HEX
  2 FF DE02 C!
  3 : OUT ( N) DE00 C! ;
  4 DECIMAL
  5
```

Figure 3-16: Program to adjust the DA Converter.

The Digital Analog Converter is used to generate complex wave forms. The next program generates a sawtooth waveform.

BASIC:

```
100 PA=56832
110 POKE PA+2,255
120 FOR I=0 TO 255
130 POKE PA,I
140 NEXT I
150 GOTO 120
```

ASSEMBLER:

```
                    PORTA     EQU $DE00
                    DDRA      EQU $DE02

                              ORG $C000
                    *SAWTOOTH
C000: A9FF                    LDA #$FF
C002: 8D02DE                  STA DDRA
C005: A200                    LDX #00
C007: 8E00DE M                STX PORTA
C00A: E8                      INX
C00B: D0FA                    BNE M
C00D: F0F8                    BEQ M

PHYSICAL ENDADDRESS: $C00F
```

FORTH:

```
SCR # 24
  0 [ DAW 428                  24.11.EF]
  1
  2 HEX
  3 FF DE02 C!
  4 : SZ BEGIN 100 0 DO I DE00 C!
  5   LOOP ?TERMINAL UNTIL ;
```

Figure 3-17: Sawtooth.

The following program generates a triangle waveform.

BASIC:

```
100 PA=56832
110 POKE PA+2,255
120 FOR I=0 TO 255
130 POKE PA,I
140 NEXT I
150 FOR I=255 TO 0 STEP -1
160 POKE PA,I
170 NEXT I
180 GOTO 110
```

ASSEMBLER:

```
                  PORTA     EQU $DE00
                  DDRA      EQU $DE02

                            ORG $C000
                  *TRIANGLE
C000: A9FF                  LDA #$FF
C002: 8D02DE                STA DDRA
C005: A200                  LDX #00
C007: 8E00DE M              STX PORTA
C00A: E8                    INX
C00B: D0FA                  BNE M
C00D: A2FF                  LDX #$FF
C00F: 8E00DE M1             STX PORTA
C012: CA                    DEX
C013: D0FA                  BNE M1
C015: F0F0                  BEQ M

PHYSICAL ENDADDRESS: $C017
```

FORTH:

```

  : DR BEGIN 100 0 DO I DE00 C!
    LOOP 0 100 DO I DE00 C! -1
    +LOOP ?TERMINAL UNTIL ;

```

Figure 3-18: Triangle Waveform.

The following program generates binary noise.

BASIC:

```
100 PA=56832
110 POKE PA+2,255
130 I=INT(RND(0)*256)
140 POKE PA,I
150 GOTO 130
```

ASSEMBLER:

```
                   PORTA     EQU $DE00
                   DDRA      EQU $DE02
                   AUX       EPZ $F8

                             ORG $C000
                   *BINARY NOISE
C000: A9FF                    LDA #$FF
C002: 8D02DE                  STA DDRA
C005: 200EC0 M                JSR RANDOM
C008: 8D00DE                  STA PORTA
C00B: 18                      CLC
C00C: 90F7                    BCC M

C00E: 38       RANDOM         SEC
C00F: 85F9                    STA AUX+1
C011: 65FC                    ADC AUX+4
C013: 65FD                    ADC AUX+5
C015: 85F8                    STA AUX
C017: A204                    LDX #$04
C019: B5F8     R              LDA AUX,X
C01B: 95F9                    STA AUX+1,X
C01D: CA                      DEX
C01E: 10F9                    BPL R
C020: 60                      RTS
PHYSICAL ENDADDRESS: $C021
```

FORTH:

```
  6 0 VARIABLE RND HERE RND !
  7 : RANDOM RND @ 31241 * 6972 +
  8   DUP RND ! ;
  9 : RNDNR ( N-N') RANDOM U* SWAP
 10   DROP ; ( 0 <=N' <N )
 11
 12 HEX FF DE02 C!
 13 : RS BEGIN 100 RNDNR DE00 C!
 14   ?TERMINAL UNTIL ;
 15 DECIMAL ;S
 OK
```

Figure 3-19: Binary Noise.

For this program, BASIC uses the RND function. In Assembler and in FORTH, special generators for random numbers are programmed. More complex waveforms can be generated in storing the digital values in a table. From here, they are picked out and converted into a voltage. The timers may be used to get samples at distinct times. In this case, SHANNON's theorem has to be regarded. If ta is the sampling rate, 2/ta is the highest frequency which can be generated. If ta is 0.1 ms, the output voltage may contain frequencies up to 5kHz. Higher frequencies have to be cut off by a low-pass filter.

## 3.7 Using a Digital-Analog Converter for Analog-Digital Conversion.

A Digital-Analog Converter can be used for an analog to digital conversion. As done earlier with the hardware solution, the successive approximation technique is used by beeing included in an actual program. The comparator in Figure 3-20 is added to the circuit in Figure 3-15. The unknown input voltage is connected to the negative input of the comparator and the output voltage of the DA converter is connected to the positive input.

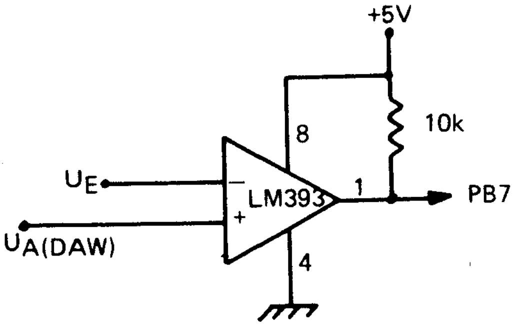


Figure 3-20: Addition of a Comparator to the DA Converter.

BASIC:not implemented.

ASSEMBLER:

```
                           PORTA     EQU  $DE00
                           PORTB     EQU  $DE01
                           DDRA      EQU  $DE02
                           BSOUT     EQU  $FFD2
                           AUX       EPZ  $F8

                                     ORG  $C000

       C000:  2046C0                 JSR  MAIN
       C003:  00                     BRK

       C004:  85F8         PRTBYT    STA  AUX
       C006:  4A                     LSR
       C007:  4A                     LSR
       C008:  4A                     LSR
       C009:  4A                     LSR
       C00A:  2015C0                 JSR  PRT
       C00D:  A5F8                   LDA  AUX
       C00F:  2015C0                 JSR  PRT
       C012:  A5F8                   LDA  AUX
       C014:  60                     RTS

       C015:  290F         PRT       AND  #$0F
       C017:  C90A                   CMP  #$0A
```

```
C019: 18                     CLC
C01A: 3002                   BMI P
C01C: 6907                   ADC #$07
C01E: 6930   P               ADC #$30
C020: 4CD2FF                 JMP BSOUT

C023: A9FF   INIT            LDA #$FF
C025: 8D02DE                 STA DDRA
C028: 60                     RTS

C029: A980   CONVERT         LDA #$80
C02B: 85F8                   STA AUX
C02D: A97F                   LDA #$7F
C02F: 8D00DE CO              STA PORTA
C032: EA                     NOP
C033: EA                     NOP
C034: AC01DE                 LDY PORTB
C037: 3002                   BMI C1
C039: 05F8                   ORA AUX
C03B: 46F8   C1              LSR AUX
C03D: B004                   BCS FIN
C03F: 45F8                   EOR AUX
C041: 90EC                   BCC CO
C043: 4C04C0 FIN             JMP PRTBYT

C046: 2023C0 MAIN            JSR INIT
C049: 2029C0                 JSR CONVERT
C04C: 00                     BRK

PHYSICAL ENDADDRESS: $C04D
```

FORTH:

```
SCR # 26
  0 ( ADW WITH DAW               29.11.EF)
  1 HEX
  2 DE00 CONSTANT PORTA
  3 DE02 CONSTANT DDRA
  4 DE01 CONSTANT PORTB
  5 : INIT FF DDRA C! ;
  6 CODE CONV 80 # LDA, N STA,
  7   7F # LDA,
BEGIN, DROP PORTA STA, NOP, NOP,
  PORTB LDY, 0< NOT IF, N ORA,
  THEN, N LSR, CS NOT IF, N EOR,
  ROT JMP, THEN,
  DEX, DEX, BOT STA, 0 # LDA,
  BOT 1+ STA, NEXT JMP, END-CODE

DECIMAL ;S
OK
```

Figure 3-21:Program Analog Digital Conversion.

In this program the following technique is used. The bit pattern %10000000 is stored in memory location AUX. The value %01111111 is stored in Port A. The output of the DA converter is Ufs/2. If the input voltage Ue is less than the output voltage Ua of the DAC the output of the comparator is high. PB7 is 1 and read into the Y-register. The BMI C1 command is executed. The next command shifts the content of AUX one time to the right. If this shift sets the Carry Bit, the conversion is completed. The EOR AUX command clears the next bit in the accumulator. After the first loop, the content of the accumulator is

%00111111

and the content of AUX is

%01000000

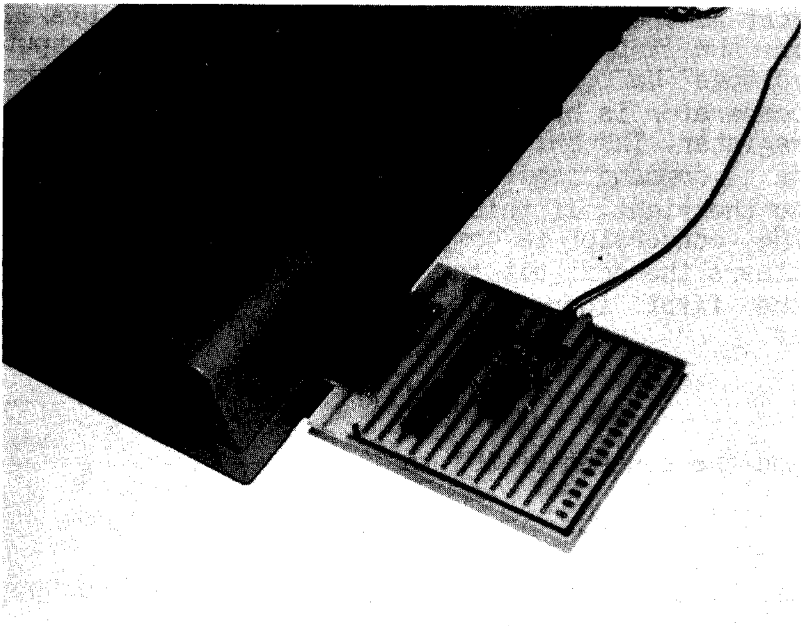
if the input voltage is less than the output voltage of the DAC.

If the input voltage is larger than the output voltage of the DAC, the output of the comparator is low the BMI C1 command is not executed. The next instruction is the ORA AUX command. This sets the corresponding bit in the accumulator to 1. After the first loop, the content of the accumulator is

%10111111

With PB7=0 the one remains in the accumulator, while PB7=1 sets it to zero. The conversion time depends on the speed of the program. To speed up the program, PB7 must be used to sense the comparator. This makes it possible to branch with the BMI instruction, otherwise an AND instruction must be used for masking the bit.

The FORTH word CONV is equal to the subroutine CONVERT in the Assembler program. The result of the conversion is placed on the stack.

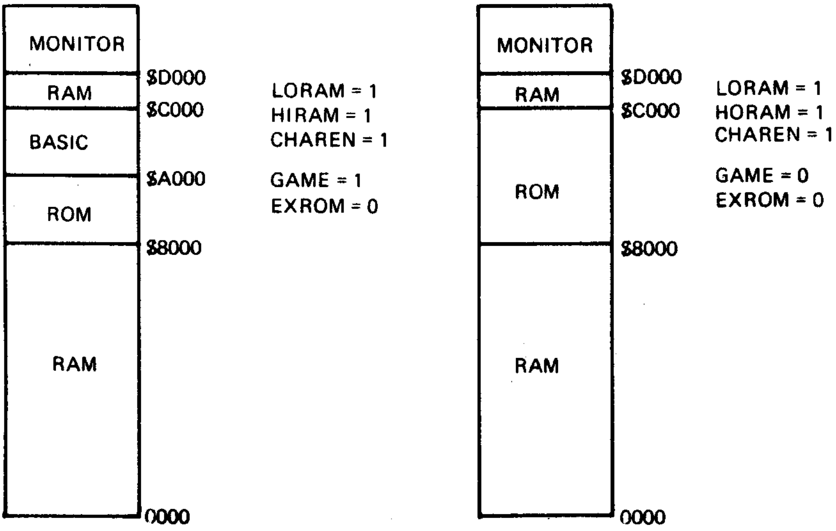


AD Converter plugged into the C64.

# 4 Using the ROM Area for Expansion

## 4.0 Using the ROM Area for Expansion.

The C64 is fully equipped with a memory. All 64k are occupied by RAM. Parts of the memory maybe switched off with some external signals. Two signals at the Expansion Bus, EXROM and GAME, are provided for this. Figure 4-1 shows two possible memory configurations.



After the computer is switched on, RAM is located at address $A000. With EXROM=L and GAME=H, the higest RAM address is $7FFF. The BASIC interpreter is still resident. If GAME is set low,the BASIC interpreter will be turned off. For the next experiments the line EXROM is low and GAME is high. The ROM space $8000 to $9FFF can be used for expansion.

## 4.1 Connecting the EPROM 2732.

The EPROM 2732 is a 4*8 bit memory. The content of the EPROM is burned in with an EPROM Burner and erased with an ultraviolet light. The pin layout is shown in Figure 4-2.

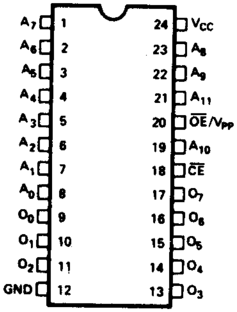


Figure 4-2: Pin Layout of the EPROM 2732.

The OE/Vpp line is low for reading the content of the EPROM. It is selected by the CS line. The signal for this line must be decoded using the address lines. An example is shown in Figure 4-3. For a better understanding see the the truth table of the decoder 74LS138 shown in Figure 4-4.

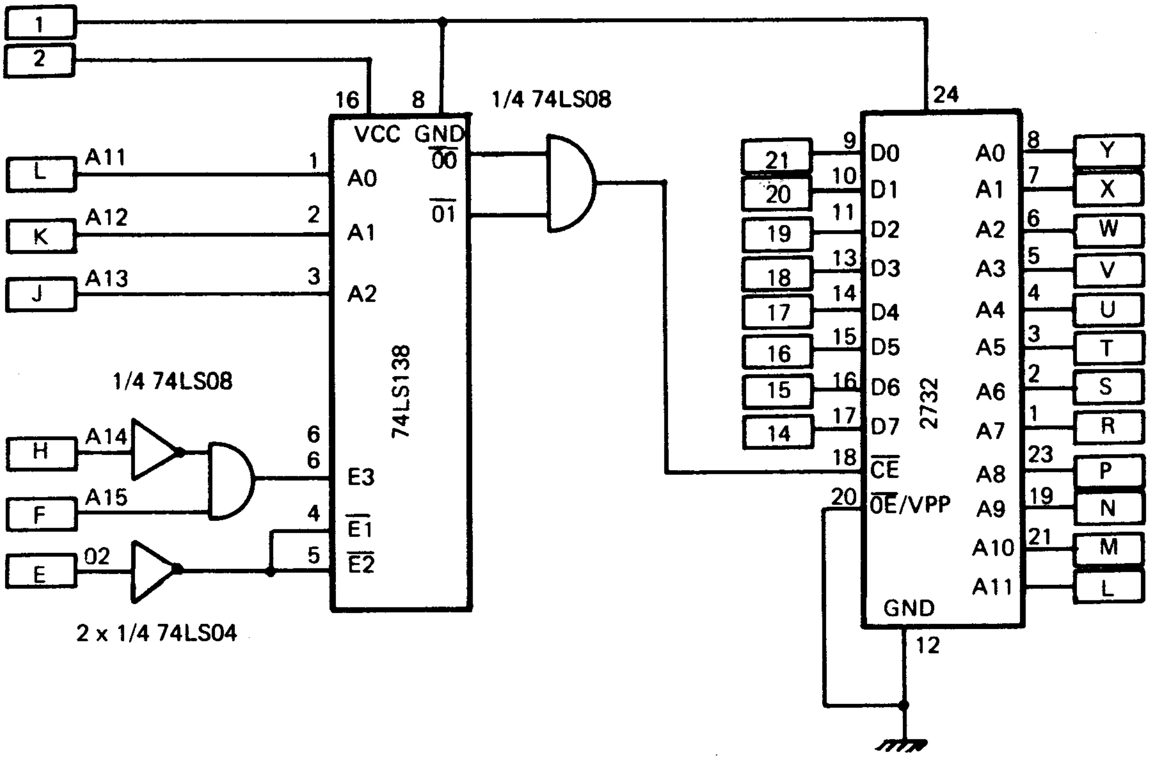
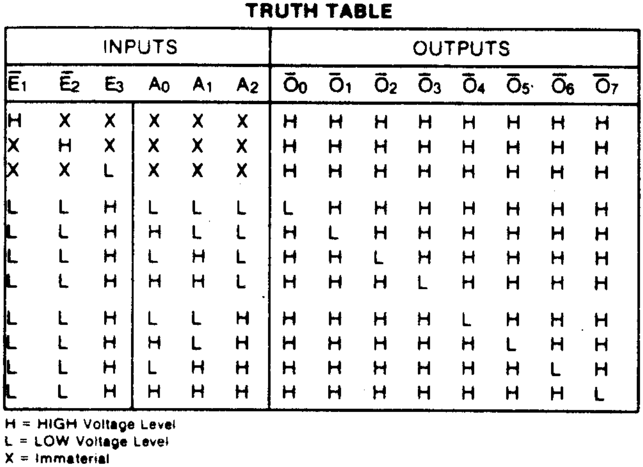


Figure 4-3: Schematic for Connecting the EPROM 2732.



**TRUTH TABLE**

| INPUTS | | | | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{E}_1$ | $\overline{E}_2$ | $E_3$ | $A_0$ | $A_1$ | $A_2$ | $\overline{O}_0$ | $\overline{O}_1$ | $\overline{O}_2$ | $\overline{O}_3$ | $\overline{O}_4$ | $\overline{O}_5$ | $\overline{O}_6$ | $\overline{O}_7$ |
| H | X | X | X | X | X | H | H | H | H | H | H | H | H |
| X | H | X | X | X | X | H | H | H | H | H | H | H | H |
| X | X | L | X | X | X | H | H | H | H | H | H | H | H |
| L | L | H | L | L | L | L | H | H | H | H | H | H | H |
| L | L | H | H | L | L | H | L | H | H | H | H | H | H |
| L | L | H | L | H | L | H | H | L | H | H | H | H | H |
| L | L | H | H | H | L | H | H | H | L | H | H | H | H |
| L | L | H | L | L | H | H | H | H | H | L | H | H | H |
| L | L | H | H | L | H | H | H | H | H | H | L | H | H |
| L | L | H | L | H | H | H | H | H | H | H | H | L | H |
| L | L | H | H | H | H | H | H | H | H | H | H | H | L |

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

Figure 4-4: Truth Table of the Decoder 74LS138.

The address line A14 is inverted and anded with A15. With A15=H and A14=L, the input E3 is high. The input lines A0, A1 and A2 are connected with the address lines A11, A12 and A13. The processor clock 02 is inverted and connected with the E1 and E2 inputs. With this decoding the output 00 becomes low if an address between

| A15 | A14 | A13 | A12 | A11 |
|---|---|---|---|---|
| H | L | L | L | L |

and

| A15 | A14 | A13 | A12 | A11 |
|---|---|---|---|---|
| H | L | L | L | H |

is addressed and Ø2 is positive. This is the address range $8000 to $87FF. Since the 4k EPROM needs the address range $8800 to $8FFF, two output 00 and 01 must be anded with an AND gate. A second EPROM can be used with the lines 02 and 03.

## 4.2 Decoding for more I/O Devices.

In Figure 4-5 the decoding is continued for the connection of more I/O devices. This is the same decoding used for the APPLE II slots. The APPLE II uses the address range $C000 to $CFFF. This range cannot be used with a C64. The address range for a C64 is $8000 to $8FFF. The output 01/1 is equal to I/O STROBE, the outputs 01/2 to 07/2 to DEVICE SELECT and 00/3 to 07/3 to I/O SELECT of the APPLE II.

Figure 4-6 shows an address table of the decoded addresses (see next Page).

Figure 4-7 shows a circuit board which was developed for 6502 (6510) computers. It contains the decoding from Figure 4-5 and four slots for expansion. The pin layout of the slots is the same as used in the APPLE II computer. Not all lines of the APPLE II slots are available at this expansion slot. The pin layout is shown in Figure 4-8.

All Address and data lines of the computer are connected. The signals DEVICE SELECT, I/O SELECT and I/O STROBE are provided by the decoding. For the supply voltages, an external power supply is used. The following addresses are available:

| Slot | I/O SELECT | DEVICE SEL | I/O STROBE |
|---|---|---|---|
| 1 | 8100–81FF | 8090–809F | 8800–8FFF |
| 2 | 8200–82FF | 80A0–80AF | 8800–8FFF |
| 3 | 8300–83FF | 80B0–80BF | 8800–8FFF |
| 4 | 8400–84FF | 80C0–80CF | 8800–8FFF |

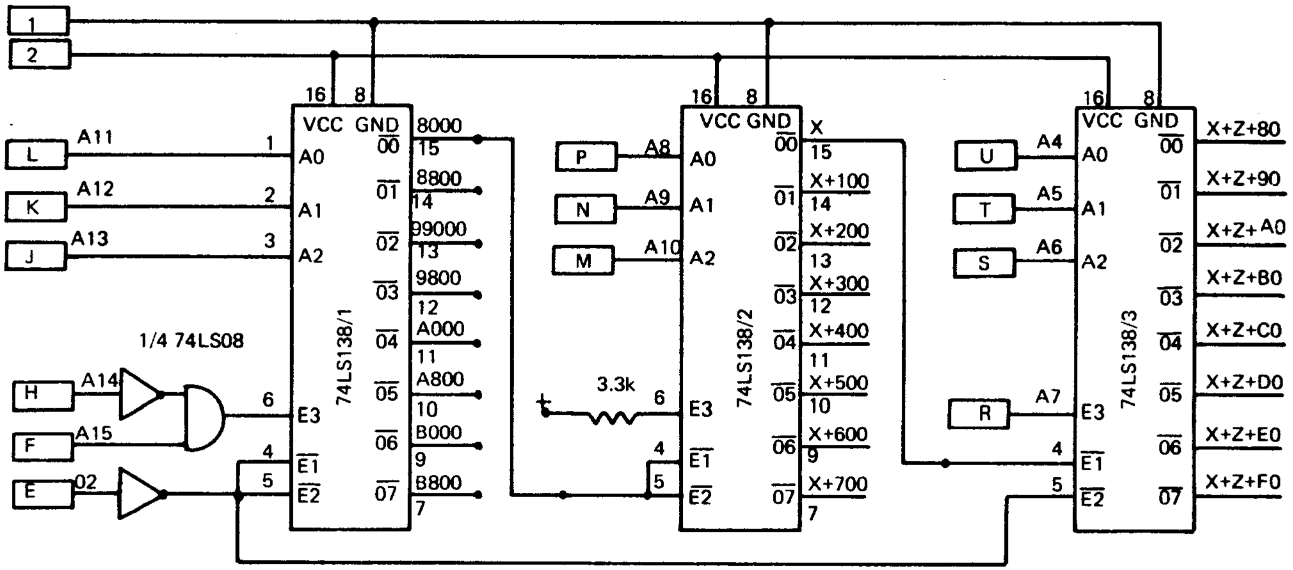


Figure 4-5: Decoding for more I/O Devices.

| A15 | A14 | A13 | A12 | A11 | X | A10 | A9 | A8 | Z | A7 | A6 | A5 | A4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | L | L | L | L | 8000 | L | L | L | X | H | L | L | L | X+Z+80 |
| H | L | L | L | H | 8800 | L | L | H | X+100 | H | L | L | H | X+Z+90 |
| H | L | L | H | L | 9000 | L | H | L | X+200 | H | L | H | L | X+Z+A0 |
| H | L | L | H | H | 9800 | L | H | H | X+300 | H | L | H | H | X+Z+B0 |
| H | L | H | L | L | A000 | H | L | L | X+400 | H | H | L | L | X+Z+C0 |
| H | L | H | L | H | A800 | H | L | H | X+500 | H | H | L | H | X+Z+D0 |
| H | L | H | H | L | B800 | H | H | L | X+600 | H | H | H | L | X+Z+50 |
| H | L | H | H | H | B800 | H | H | H | X+700 | H | H | H | H | X+Z+F0 |

Figure 4-6: Address Table.

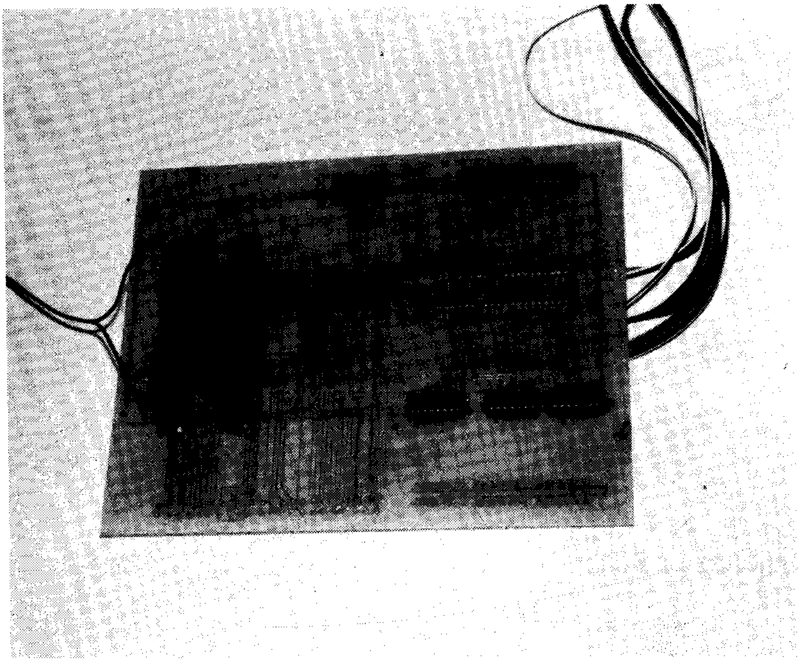


Figure 4-7: Expansion Board.

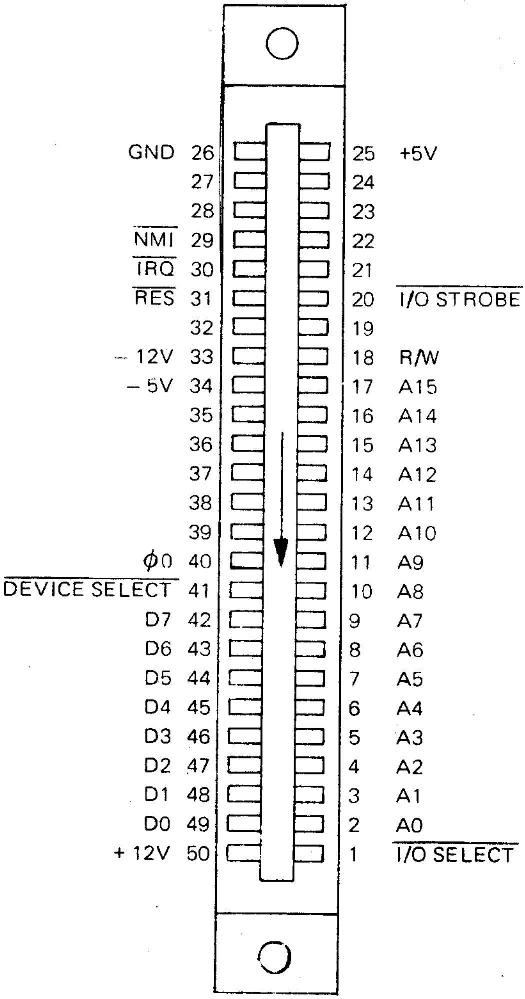


Figure 4-8: Pin Layout of the Slots of the Expansion Board.

## 4.3 The 6526 I/O Board.

Figure 4-9 shows a board which was developed as an 6522 extension for the APPLE II. The CIA 6526 is used with the C64. This board was heavily used with the APPLE II for connecting AD and DA converters and parallel and serial printers to the APPLE II. On the right side there is the CIA and 1/4 of RAM. The left side is left open for additional circuits. The schematic of the board is shown in Figure 4-10. The connection of the C64 with the expansion board and the I/O board is shown in Figure 4-11.

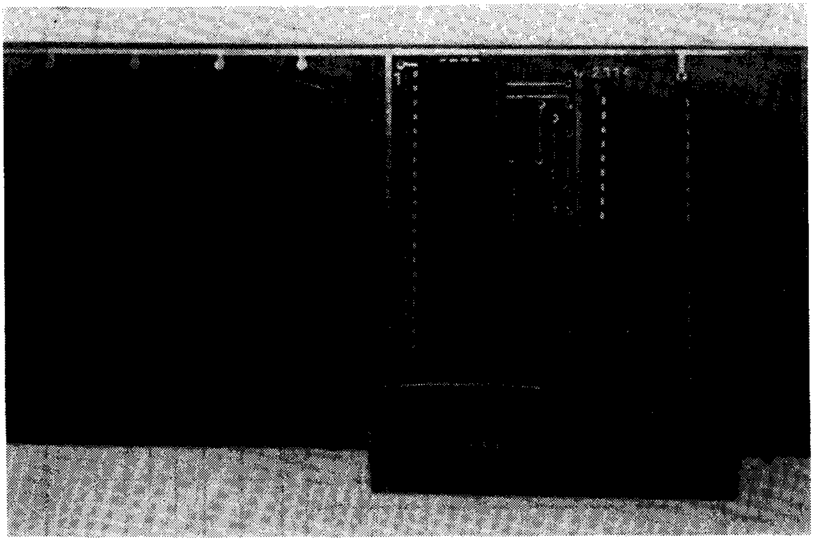


Figure 4-9: I/O Card 6526.

The lines between the computer and the Expansion Board shouldn't be very long. On the right side of the board, which plugs into the C64, there is a DIP switch to select the ROM area for I/O.

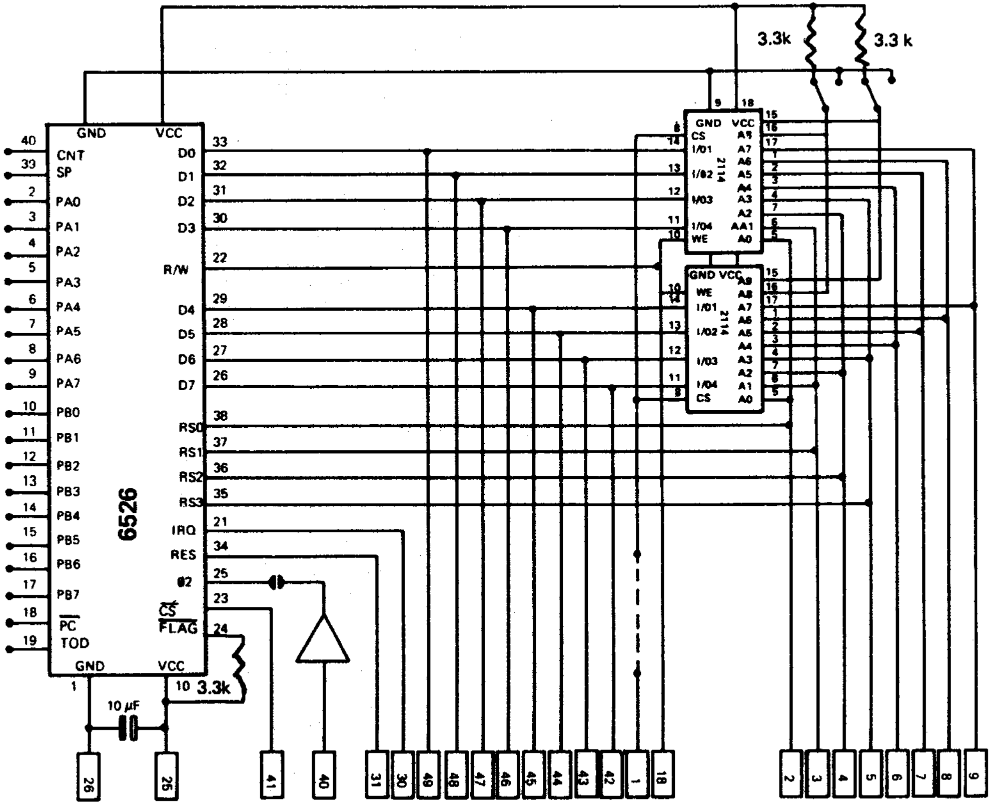


Figure 4-10: Schematic of the 6526 Board.

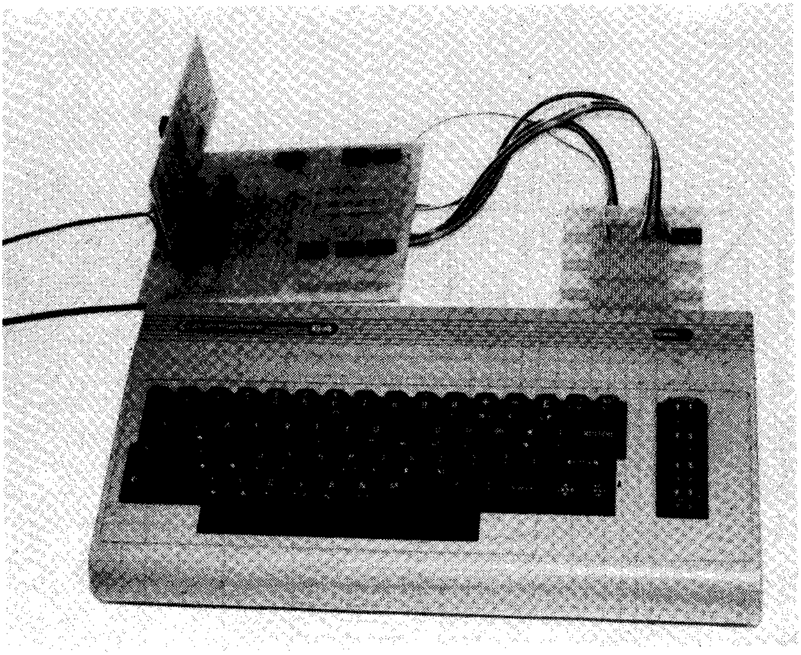


Figure 4-11: Connection of the C64 with the Expansion Board and the I/O Board.

## 4.4 Connecting the 12 Bit ADC 1210.

This is an applicaton of the 6526 I/O card. In some cases, the resolution of an 8 bit AD Converter is too small. A 12 bit ADC has much better performance. If it was used for the temperature measurement the resolution would have been 0.05 C. The schematic is shown in Figure 4-12. The output lines of the converter are connected to Port A and B of the 6526. The line PB4 is used to start a conversion. Line PB5 is used to sense the end of the conversion. The 1210 uses the method of succesive approximation for converting a voltage into a number, therefore it needs an external clock. The clock of the processor can be used. It is divided by a frequency divider 4024. In older data sheets, the maximum clock

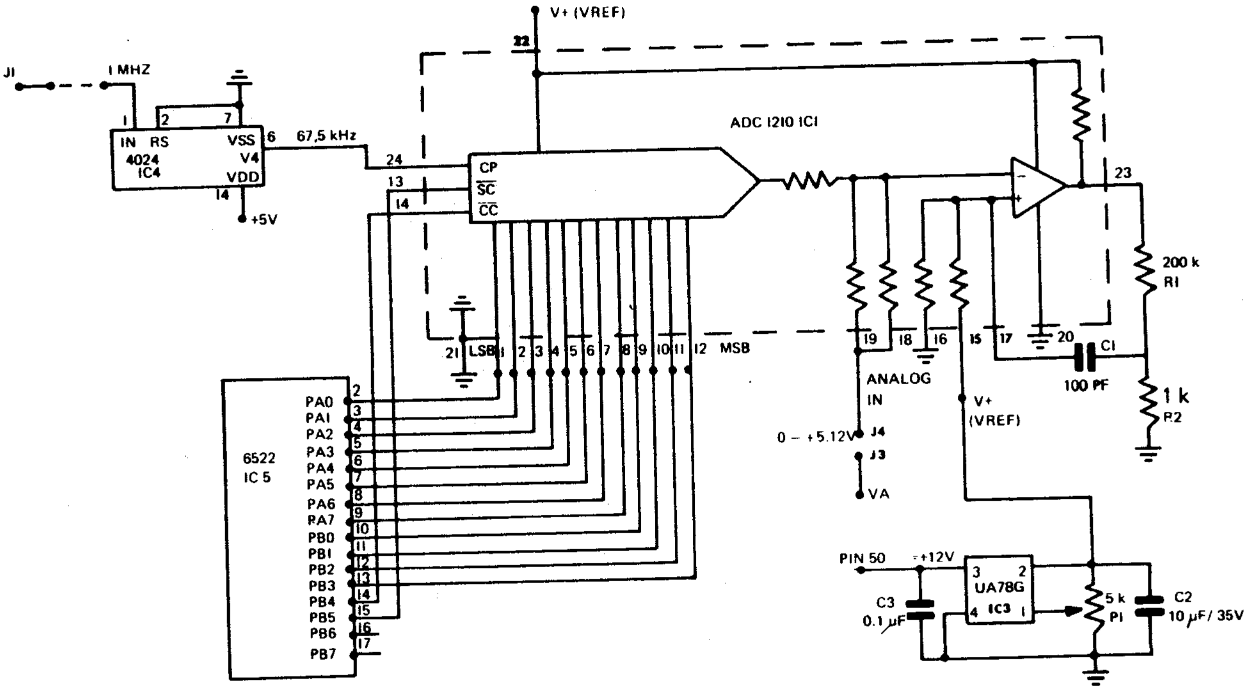


Figure 4–12: Schematic of the ADC 1210.

frequency for the 1210 is mentioned with 65kHZ, in newer data sheets with 250kHz. The positive reference voltage determines the input voltage range. The adjustable voltage reference is set to 5.12 volts. A voltage between 0 and 5. 12 volts can be measured with this. The program is shown in Figure 4-13. The internal timers are programmed to make a measurement every second. The Assembler program can be stored in the on board RAM. The converter is assembled on the free space of the I/O board.

BASIC:

```
100 DIM MA(500)
110 MS=49156:MZ=49165
120 MW=49196:NM=49232
130 MB=49406
200 PRINT""
210 INPUT"MEASUREMENT STARTS WITH RETURN";A$
220 I=0
230 SYS MS:SYS MZ: SYS MW
240 GOSUB 300
250 SYS NM:SYS MW:GOSUB 300:GOTO 250
300 M=PEEK(MB+1)*256+PEEK(MB)
310 PRINT M
320 MA(I)=M:I=I+1:RETURN
```

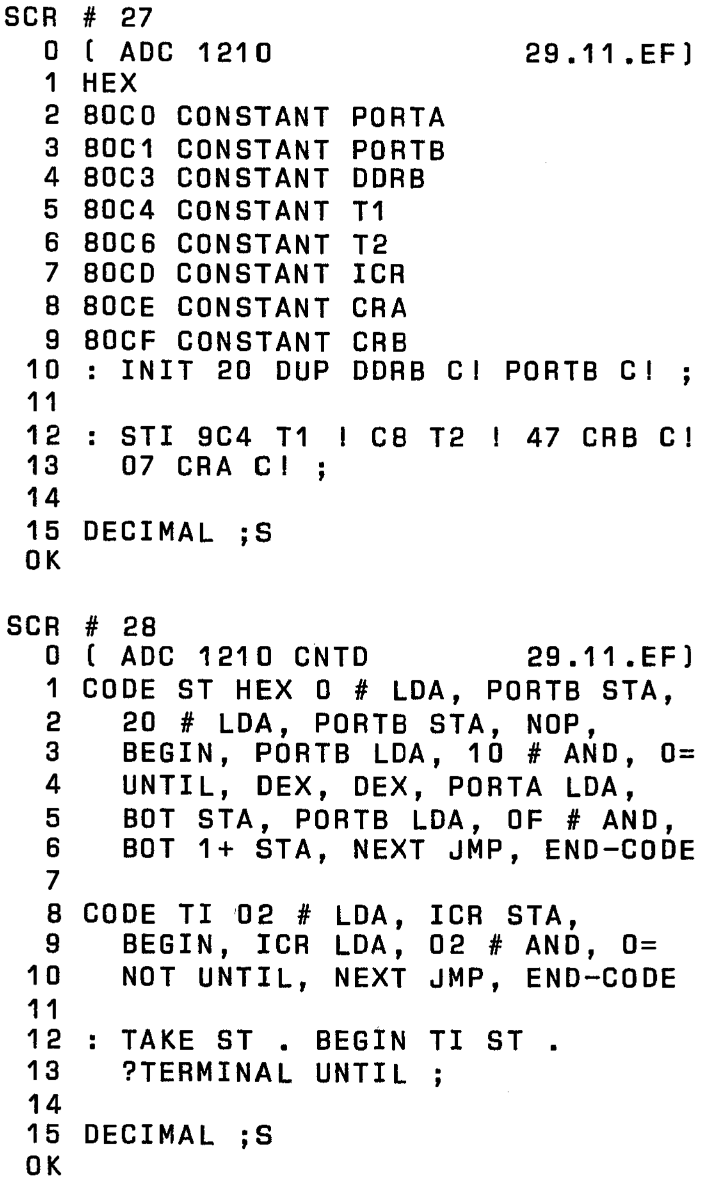
ASSEMBLER:

```
          PORTA     EQU $80C0
          PORTB     EQU $80C1
          DDRB      EQU $80C3
          T1        EQU $80C4
          T2        EQU $80C6
          ICR       EQU $80CD
          CRA       EQU $80CE
          CRB       EQU $80CF

          AUXA      EQU $84FE
```

```
                                  ORG $C000
C000: 205DC0                      JSR MAIN
C003: 00                          BRK


C004: A920      INIT              LDA #$20
C006: 8DC380                      STA DDRB
C009: 8DC180                      STA PORTB
C00C: 60                          RTS

C00D: A9C4      START             LDA #$C4
C00F: 8DC480                      STA T1
C012: A909                        LDA #$09
C014: 8DC580                      STA T1+1
C017: A9C8                        LDA #$C8
C019: 8DC680                      STA T2
C01C: A900                        LDA #$00
C01E: 8DC780                      STA T2+1
C021: A947                        LDA #$47
C023: 8DCF80                      STA CRB
C026: A907                        LDA #$07
C028: 8DCE80                      STA CRA
C02B: 60                          RTS

C02C: A900      MW                LDA #$0
C02E: 8DC180                      STA PORTB
C031: A920                        LDA #$20
C033: 8DC180                      STA PORTB
C036: ADC180    M1                LDA PORTB
C039: 2910                        AND #%00010000
C03B: D0F9                        BNE M1
C03D: ADC180                      LDA PORTB
C040: 290F                        AND #$0F
C042: 490F                        EOR #$0F
C044: 8DFF84                      STA AUXA+1
C047: ADC080                      LDA PORTA
C04A: 49FF                        EOR #$FF
C04C: 8DFE84                      STA AUXA
C04F: 60                          RTS

C050: A902      NMW               LDA #$02
C052: 8DCD80                      STA ICR
C055: ADCD80    M2                LDA ICR
```

```
C058: 2902                 AND #%00000010
C05A: F0F9                 BEQ M2
C05C: 60                   RTS

C05D: 2004C0 MAIN          JSR INIT
C060: 200DC0               JSR START
C063: 202CC0 MA            JSR MW
C066: 208EC0               JSR OUTA
C069: 2050C0               JSR NMW
C06C: 18                   CLC
C06D: 90F4                 BCC MA

             BSOUT         EQU $FFD2
             AUX           EPZ $F8


C06F: 85F8   PRTBYT        STA AUX
C071: 4A                   LSR
C072: 4A                   LSR
C073: 4A                   LSR
C074: 4A                   LSR
C075: 2080C0               JSR PRT
C078: A5F8                 LDA AUX
C07A: 2080C0               JSR PRT
C07D: A5F8                 LDA AUX
C07F: 60                   RTS

C080: 290F   PRT           AND #$0F
C082: C90A                 CMP #$0A
C084: 18                   CLC
C085: 3002                 BMI P
C087: 6907                 ADC #$07
C089: 6930   P             ADC #$30
C08B: 4CD2FF               JMP BSOUT

C08E: ADFF84 OUTA          LDA AUXA+1
C091: 206FC0               JSR PRTBYT
C094: ADFE84               LDA AUXA
C097: 206FC0               JSR PRTBYT
C09A: A90D                 LDA #$0D
C09C: 4CD2FF               JMP BSOUT
PHYSICAL ENDADDRESS: $C09F
```

FORTH:



```
SCR # 27
  0 ( ADC 1210                    29.11.EF)
  1 HEX
  2 80C0 CONSTANT PORTA
  3 80C1 CONSTANT PORTB
  4 80C3 CONSTANT DDRB
  5 80C4 CONSTANT T1
  6 80C6 CONSTANT T2
  7 80CD CONSTANT ICR
  8 80CE CONSTANT CRA
  9 80CF CONSTANT CRB
 10 : INIT 20 DUP DDRB C! PORTB C! ;
 11
 12 : STI 9C4 T1 ! C8 T2 ! 47 CRB C!
 13   07 CRA C! ;
 14
 15 DECIMAL ;S
 OK

SCR # 28
  0 ( ADC 1210 CNTD               29.11.EF)
  1 CODE ST HEX 0 # LDA, PORTB STA,
  2   20 # LDA, PORTB STA, NOP,
  3   BEGIN, PORTB LDA, 10 # AND, 0=
  4   UNTIL, DEX, DEX, PORTA LDA,
  5   BOT STA, PORTB LDA, 0F # AND,
  6   BOT 1+ STA, NEXT JMP, END-CODE
  7
  8 CODE TI 02 # LDA, ICR STA,
  9   BEGIN, ICR LDA, 02 # AND, 0=
 10   NOT UNTIL, NEXT JMP, END-CODE
 11
 12 : TAKE ST . BEGIN TI ST .
 13   ?TERMINAL UNTIL ;
 14
 15 DECIMAL ;S
 OK
```

Figure 4-13: Program to Control the ADC 1210.

BASIC and FORTH use subroutines written in Assembler. The subroutine INIT initializes the ports. The subroutine START starts the timers. Timer A is programmed to have a zero crossing every 2.5 ms. Timer B counts the zero crossings of Timer A and divides it to have a zero crossing every second. This signal is taken to make a sample of the input voltage and to store the result in $COFE and $COFF. The output code of the 1210 is a complementary code. With a full scale input, the output is 000. With an input voltage of zero, the output code is $FFF. The data is converted with the EOR $FF command. In the subroutine NMW, the program waits for the next timeout of Timer B. Bit 2 of the interrupt control register ICR is cleared. This bit is set if a timeout of Timer B occurs. This bit is tested by the program.

In BASIC, the subroutines are called by the SYS command. The data is printed on the screen and stored in an array MW. The Assembler program is started at location $C000. The data is only displayaed on the screen.

In FORTH, the word INIT initializes the ports. STI starts the timers. The word ST takes one sample. TI waits for the next timeout. The word TAKE starts the measurement and displays the result of the conversion on the screen.
A measurement may be missing with this program. It may be lost during an interrupt of the processor. To avoid this, the interrupt of the Processor dhould be disabled.

Final remark:
We could use only a few examples of how a computer may be used for measurement. Examples such as measuring velocity or displacement were not illustrated, but are solvable using the examples in this book. In the data processing of

measurements, there is no set solution for each individual task. Each task will have its own solution.

# A Basics of Operational Amplifiers

APPENDIX A.

Basics of Operational Amplifiers.

The term "Operational Amplifier" originates in the Analog Computing Technique. Analog voltages are used for computing instead of numbers in this technique. An operational amplifier is a DC amplifier with a very high open loop gain. Using resistors or capacitors as external components, an operational amplifier can be used as 1) a summing amplifier, 2) as a voltage inverter, as 3) an integrator or 4) as a differentiator.

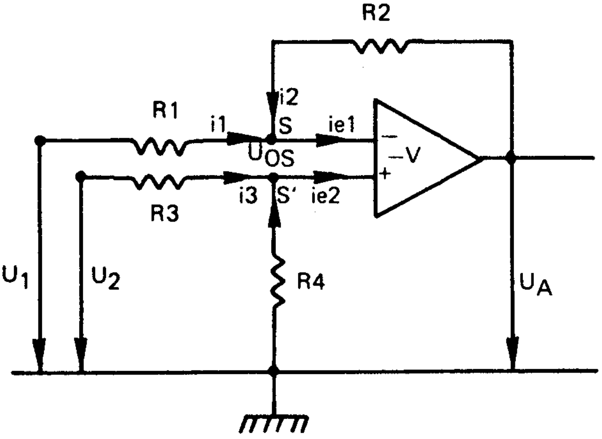


Figure A-1: Basic Circuit of a Differential Amplifier.

Thirty years ago operational amplifiers were built with vacuum tubes. The power consumption was very high. The integrated circuit operational amplifiers of today are much smaller and need less power. The output voltage swing is usually +- 15 volts. The old operational amplifiers had only one input. The new integrated amplifiers have two differential inputs. Figure A-1 shows the basic circuit of a differential amplifier. This circuit is now analyzed.

Some assumptions have to be made.
If U1 and U2 are zero, the output voltage Ua must be zero. Then it is

Ua=-V*Uos.

The open loop gain V is very high (10^5 to 10^6), therfore Uos must be zero. Both points S and S' are virtually connected. They behave in the same manner. S is called the summing node of the amplifier.

In S and S' the currents I1, I2 and Ie1 and I3, I4 and Ie2 respectively are added. When the ideal input currents of zero are achieved in an operational amplifier for Ie1 and Ie2 the following is produced:

I1+I2=0 and
I3+I4=0 .

For the network R4,R3 and U2 the equations are:

$$i_4 R - i\ R + U\ = 0$$
$$i_4 \cdot (R_4 + R_3) + U_2 = 0 \qquad (1)$$

and solved for I4:

$$i_4 = -\frac{U_2}{R_4 + R_3} \qquad (2)$$

For the network Ua,R2,R1 and U1 the equations are:

$$-U_A + i_2 \cdot R_2 - i_1 \cdot R_1 + U_1 = 0$$

$$i_2 (R_1 + R_2) + U_1 - U_A = 0 \qquad (3)$$

and solved for I2:

$$i_2 = \frac{U_A - U_1}{R_1 + R_2} \qquad (4)$$

There is a third equation for the network Ua,R2 and R4, which combines I2 and I4. The voltage Uos is assumed to be zero.

$$-U_A + i_2 R_2 - i_4 R_4 = 0 \qquad (5)$$

In this equation the equations (2) and (4) are inserted, and solved for Ua.

$$-U_A + \frac{U_A - U_1}{R_1 + R_2} R_2 + \frac{U_2}{R_3 + R_4} R_4 = 0$$

$$U_A \cdot (-1 + \frac{R_2}{R_1 + R_2}) - U_1 \cdot \frac{R_2}{R_1 + R_2} + U_2 \cdot \frac{R_2}{R_3 + R_4} = 0$$

$$U_A = -U_1 \cdot \frac{R_2}{R_1} + U_2 \cdot \frac{R_4}{R_3 + R_4} \cdot \frac{R_1 + R_2}{R_1}$$

$$U_A = -U_1 \cdot \frac{R_2}{R_1} + U_2 \cdot \frac{R_4}{R_1} \cdot \frac{R_1 + R_2}{R_3 + R_4} \qquad (6)$$

If in the last equation (6) R1 equals R3 and R2 equals R4 the output voltage Ua of an differential amplifier is:

$$U_A = \frac{R_2}{R_1}(U_2 - U_1) \qquad (7)$$

Figure A-2 shows the pin layout of the 741, a very common and cheap integrated operational amplifier.

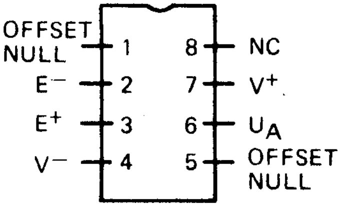


Figure A-2: Pin Layout of the 741.

Figure A-3 shows an inverting amplifier with a gain of ten. The voltage U2 in equation (7) is zero.

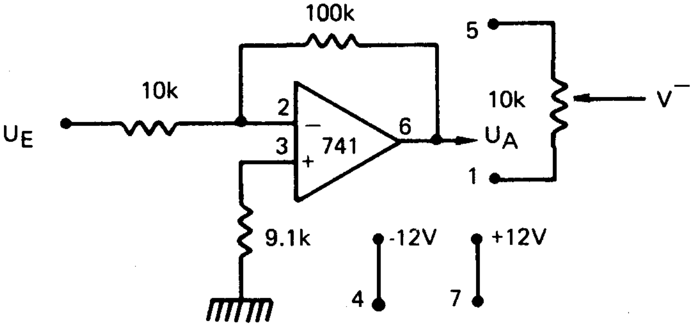


Figure A-3: Inverting Amplifier.

The positive input is connected to ground with a 9. 1k resistor. This minimizes the Output Offset voltage. Since the 741 is a real and not ideal amplifier there may be an output voltage even if the input voltage is zero. For a real amplifier, the input currents Ie1 and Ie2 are not zero. To adjust the offset voltage to zero a 10k potentiometer between pins 5 and 1 and the negative power supply voltage be used.

Figure A-4 shows a non inverting amplifier. The output voltage is:

$$U_A = U_E \cdot \left(1 + \frac{R_2}{R_1}\right) \tag{8}$$

The gain of the amplifier shown below is two.

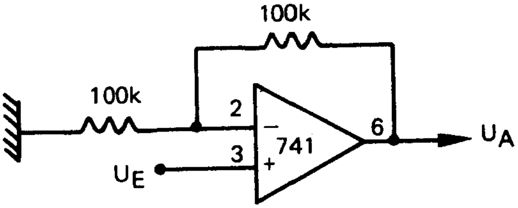


Figure A-4: Non Inverting Amplifier.

The next circuit is a voltage follower. The resistors R1 and R2 are zero. The input impedance is high, the output impedance is low. The gain is one.

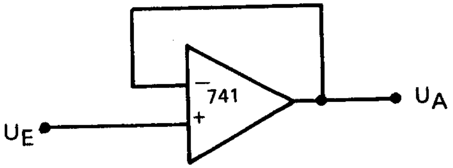


Figure A-5: Voltage follower or Unity Gain Amplifier.

Figure A-6 shows the practical circuit of a differential amplifier with a gain of 100.

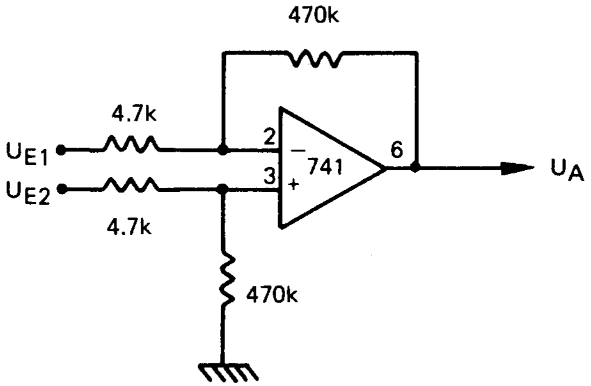


Figure A-6: Differential Amplifier.

This circuit has one disadvantage. The input impedances of the two inputs are small and differ from each other. Figure A-7 shows a better solution.

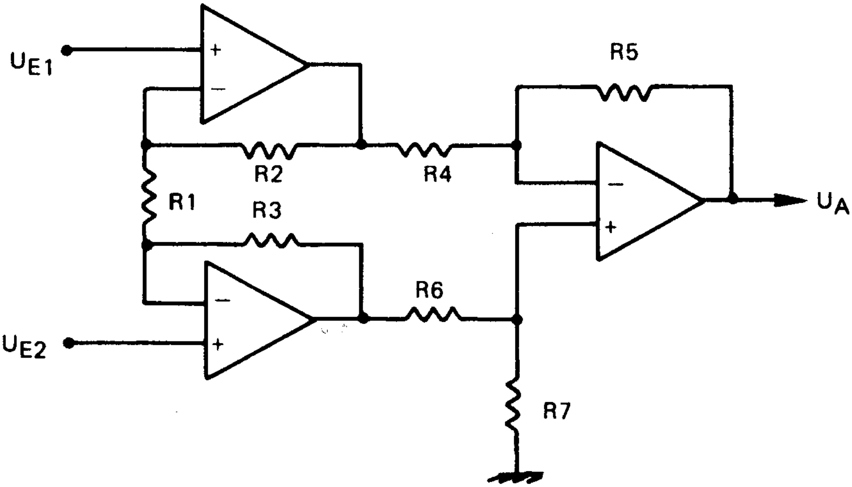


Figure A-7: Instrumentation Amplifier.

The first stage is a non inverting amplifier. With R2=R3 the gain of the first stage is:

$$U_A = (U_{E2} - U_{E1})(1 + \frac{2R_2}{R_1}) \qquad (9)$$

The second stage is a differential amplifier. The gain of this stage is given by the resistors R4 to R7. If they are equal, the gain of the amplifier is given by equation (9). For a practical circuit the Quad OPamps LM324, RC4138 or TL064C can be used. The next figure shows a summing amplifier.

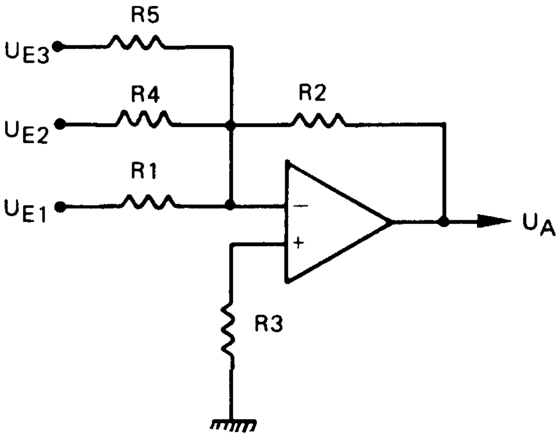


Figure A-8: Summing Amplifier.

The output voltage is:

$$U_A = -(U_{E1} \frac{R_2}{R_1} + U_{E2} \frac{R_2}{R_4} + U_{E3} \frac{R_2}{R_5}) \qquad (10)$$

A summing amplifier is often used to add a constant voltage to an alternating voltage.

An example: The input voltage range of an analog to digital converter is 0 volt to 10 volts. The output of a transducer is -5 to +5 volts. A summing amplifier is used to add 5 volts to the output of the transducer.

The amplifier LM3900 used in Chapter 2 is not an operational amplifier as described above. It is called a NORTON amplifier. The output voltage depends on the difference of the input currents at the positive and negative input. Therefore it behaves different from a normal operational amplifier. This amplifier can be used with a single power supply voltage instead of +- 15 volts, which are used by the normal operational amplifiers.

# B Basics of AD and DA Converters

APPENDIX B.

Basics of AD and DA Converters.

1. Digital to Analog Conversion.

Figure B-1 shows the basic circuit of a 3-bit digital to analog converter.

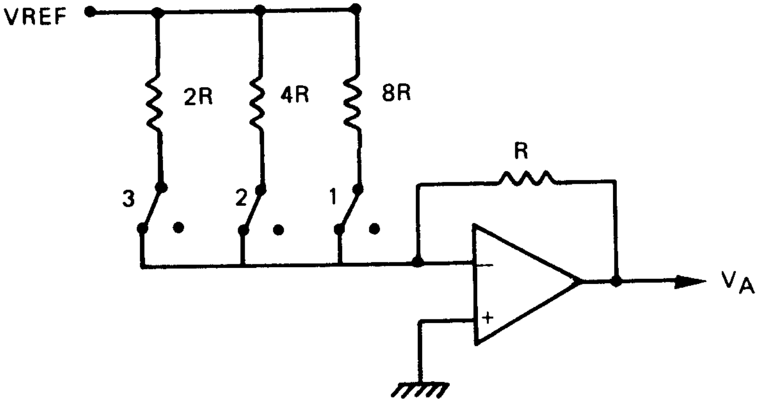


Figure B-1: 3-Bit Digital to Analog Converter.

The feedback resisitor of an operational amplifier is R. The resistors 2R, 4R and 8R are connected via switches to the summing node. If switch S3 is closed and all other switches are open, the gain of the amplifier is 1/2. The output voltage is Uref/2. The input code is %100.

If all switches are closed, the input resistor of the circuit is 8/7*R and the output voltage is 7/8*Uref. Figure B-2 shows the transfer function for the 3-bit DAC.

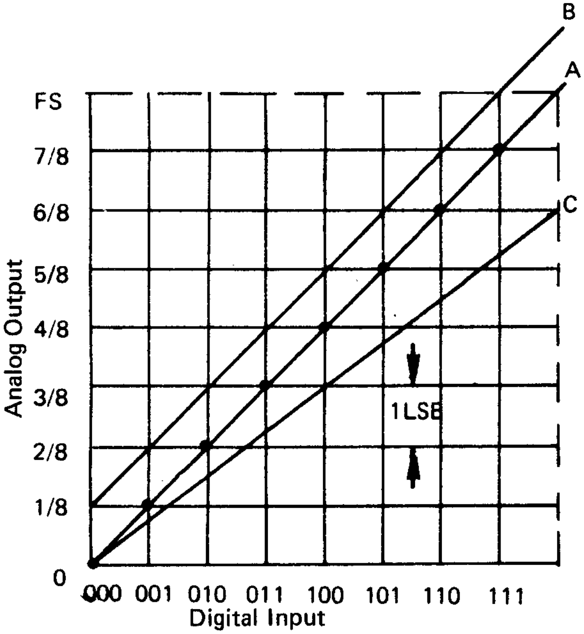


Figure B-2: Transfer Function of the 3-Bit DAC.

Line A is the ideal transfer function. The output voltage jumps 1 LSB (Least Significant Bit). This is the resolution of a DAC. For a full scale of 10 volts the resolution is:

|  |  |
|---|---|
| 8-bit DAC | 1LSB=39.1 mV |
| 10-bit DAC | 1LSB= 9.77 mV |
| 12-bit DAC | 1LSB= 2.44 mV |

The following errors may occur:

Offset Error.

Line B shows an offset error. The output voltage for the code %000 is 1/8 Uref. All points of line B have the same displacement.

Gain Error.

Line C shows a gain error. The slope of line C differs from the slope of line A. All points of line C differ from line A by the same percentage.

Both of these errrors can be corrected using external components.

Linearity Error.

Figure B-3 shows a linearity error in the lower left part. The output voltage for the code %010 is 1/2 LSB to high. This changes the line from a straight line to a curved line. In the upper right half of Figure B-3 an error in differential linearity is shown. The differential linearity is measured between two points. If the voltage difference is 1LSB, the error in differential linearity is zero.

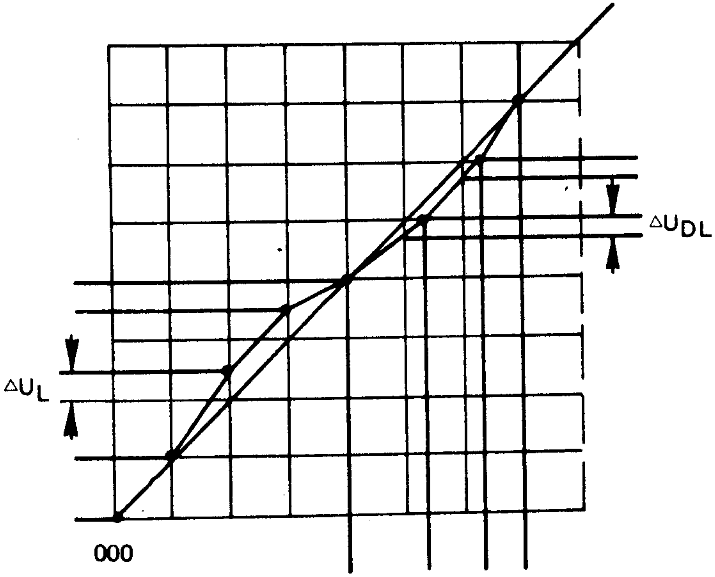


Figure B-3: Error in Linearity.

A large error in differential linearity is shown in Figure B-4. The output voltage for the code %100 is less than the output voltage for the code %011. Using such a ADC for analog to digital conversion leads to missing codes. The result is that a certain code will never occur in an analog to digital conversion.

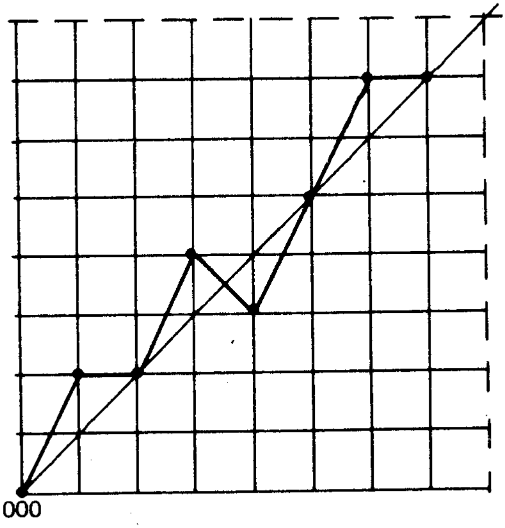


Figure B-4: Extreme Large Error in Linearity.

The conversion time is mainly determined by the settling time of the operaional amplifier. Figure B-5 shows one method to determine the settling time. It is the difference in time from the beginning of the conversion until the voltage stays within an error band.

Assume the error band to be 1/2 LSB. An 8-bit DAC has to settle within 20mV and a 12-bit DAC within 1.25mV, therefore a 12-bit DAC may be slower than an 8-bit DAC because of its stronger specification.

The settling time may be different for a zero to full scale swing or a full scale to zero transition.

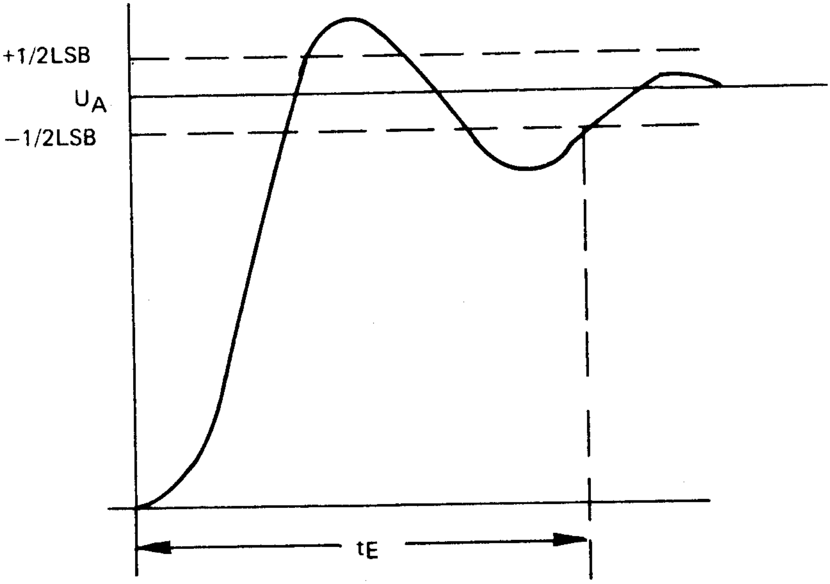


Figure B-5: Settling Time.

2. Analog to Digital Conversion.

Figure B-6 shows the transfer function for an ideal Analog to Digital Converter. The code of the output is generated by an input voltage range. This range is the codewidth of the converter. In an ideal converter, the codewidth is equal over the whole input range, except on both ends.

The following errors may occur:

Offset error.

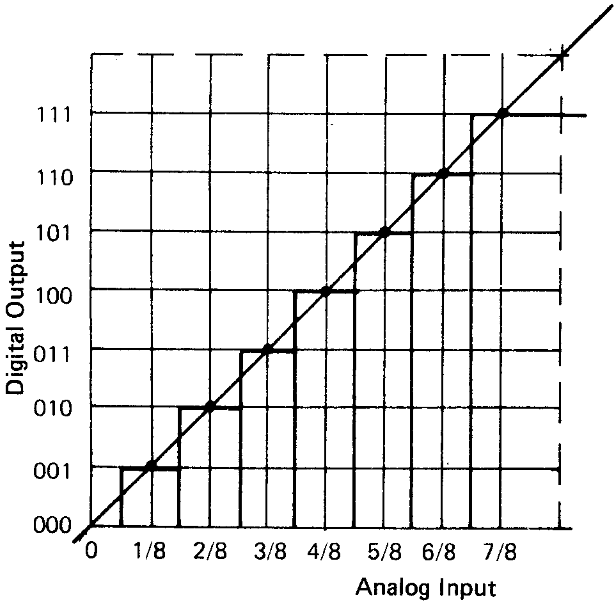


Figure B-6: Ideal Transfer Function of an Analog to Digital Converter.

With an ideal converter the LSB must toggle between 0 and 1 if an input voltage of a half codewidth is applied to the input. If not, an offset error exists.

Gain Error.

That is the same definition as with a Digital to Analog Converter.

Linearity Error.

This is the displacement of the codewidth from a straight line. In Figure B-6, the code midpoints are shifted to the right or to the left.

Error in Differential Linearity.

The error in differential linearity is the difference of the codewidth from 1LSB. If the codewidth is zero, a missing code will the result. This is the only error which can not be corrected. All other errors can be corrected by external components. An error in linearity can be corrected by software.

An other error may occur if the input voltage changes during conversion time. A sinusoidal oscillation of the form

$$u = U \sin(2\pi f t)$$

has the declination

$$u' = 2\pi f U \cos(2\pi f t)$$

The maximum declination is

$$u'_{max} = 2\pi f U$$

The input voltage must not change more than 1/2 LSB during conversion time. The maximum frequency of a sinusoidal oscillation with an amplitude of 10 Vss which can be resolved by an 8-bit converter and a conversion time of 15 us is 42 Hz. This changes to 1.1 Hz for a 12-bit converter with a conversion time of 35 us. For higher frequencies, a Sample and Hold amplifier must be used. One example is shown in Figure B-7.

Between two unity gain amplifiers with high input impedance and low output impedance, a switch and a capacitor connected to ground is mounted. If the switch S is closed, the voltage across the capacitor is the same as the input voltage Ue.

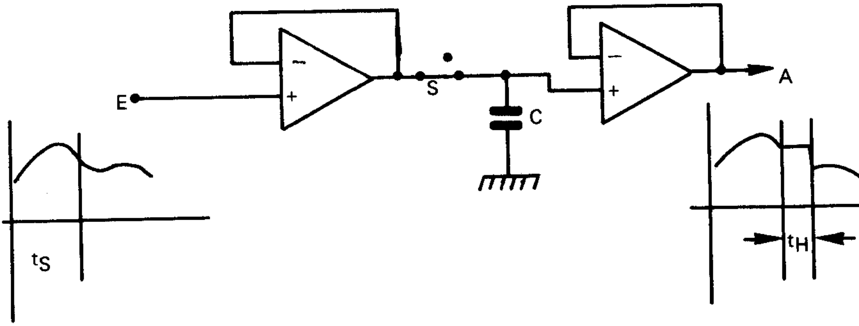


Figure B-7: Sample amd Hold Amplifier.

If the switch is opened, the output voltage Ua stays at the same level. This voltage is then converted to a digital code.

For a practical circuit, the integrated S&H amplifiers AD582 or AD585 can be used.

There is another problem in converting sinusoidal waveforms into digital code. SHANNON's theorem proves that a wave with the maximum frequency f has to be sampled with the frequency 2*f to reconstruct the original waveform. If the sample rate is less, so-called "aliasing" frequencies may be obtained. This is shown in Figure B-8.

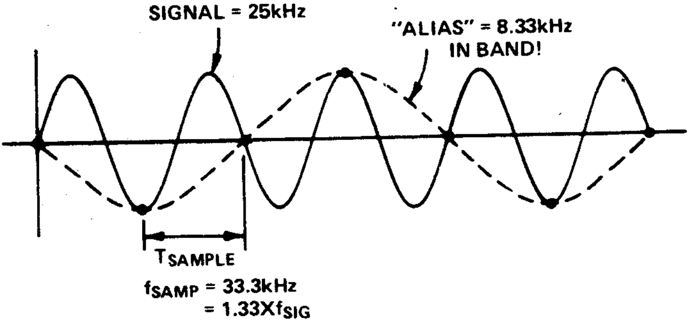


Figure B-8: Alias Frequencies.

The waveform with a frequency of 25kHZ is sampled with 33. 3kHz. The result is an alias frequency of 8.33kHz.The original signal does not contain such a component.

# NOTES

# C RS232 Interface

APPENDIX C.

RS232 Interface.

This is another application of the 6526 I/O card.

The C64, like most newer home computers, provides an RS232 interface. This is a serial interface to connect seperate computers, printers, modems and other external devices together. For exchange of data, the following voltage levels were defined. A One is represented by an level less than -3 volts, a zero is represented by a voltage level larger than +3 volts. TTL levels are usually used today. A One is represented by a voltage between 2. 5 and 5 volts, a zero by a voltage less 0.8 volts. But even this may not be true. Within a CMOS computer a voltage of 10 volts was found for the One.

The control lines are also used in different ways. This makes it very "easy" to connect two so-called RS232 compatible devices together.

Figure C-1 shows the pin layout of the signals of an RS232 interface.

| Pin # | | Description | |
|---|---|---|---|
| 1 | | Ground | |
| 2 | TxD | Transmit Data | (output) |
| 3 | RxD | Received Data | (input) |
| 4 | RTS | Request to Send | (output) |
| 5 | CTS | Clear to Send | (input) |
| 6 | DSR | Data Set Ready | (input) |
| 7 | | Signal Ground | |
| 8 | | Received Line Signal Detect | (input) |
| 20 | DTR | Data Terminal Ready | (output) |

Figure C-1: Pin Layout of an RS232 Interface.

Line 2 (TxD) is used to transmit the data. This data is sent out bit by bit. Line 3 (RxD) receives data. Lines 4 (RTS) and 5 (CTS) are used to control the receiving of data. Line 5 can sense line 4 of the sending device if it is ready to send. Line 6 of the receiving device senses if the transmitting device is clear to send. Line 20 sends a data terminal ready signal to the transmitter. The following baud rates can be used for exchanging data: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600 and 19200. Most printer and modems receive data with 300 baud.

For higher speeds, 1200 baud are used. Between terminals and mainframe computers, the data is exchanged with a 19200 baud rate. Figure C-2 shows the sequence for one character.

In Figure C-2 TTL level is assumed. The signal starts with the startbit. Seven data bits follow. The first bit is the LSB of the character to be sent. The character ends with two stop bits. This is only an example. For high transfer rates often only one stop bit is used. A parity bit can be inserted, which tests for even or odd parity. The parity is even if the number of ones is even, otherwise the parity is odd. On some computers the format of the signal can be programmed.

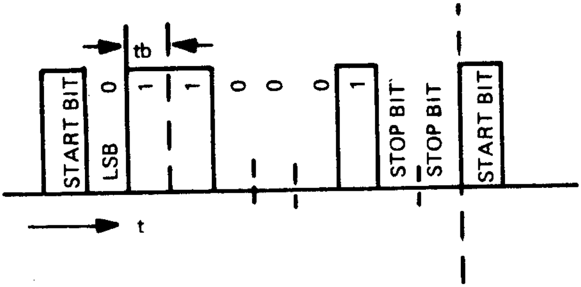


Figure C-2: Sequence of Bits for one Character.

Figure C-3 shows the schematic of an RS232 interface using the CIA 6526. Instead of one external device, up to four RS232 devices can be connected. For the first channel, the data is sent out via line PA0. The line PA1 is used as DSR line. An interrupt technique is used for the incomming data. All datalines are connected together by the Wired Or function and connected to the FLAG input and also to the I/O lines PB0 to PB3. The startbit on one of the four lines creates an interrupt. The interrupt request routine senses which lines has caused the interrupt and takes the incoming data stream.

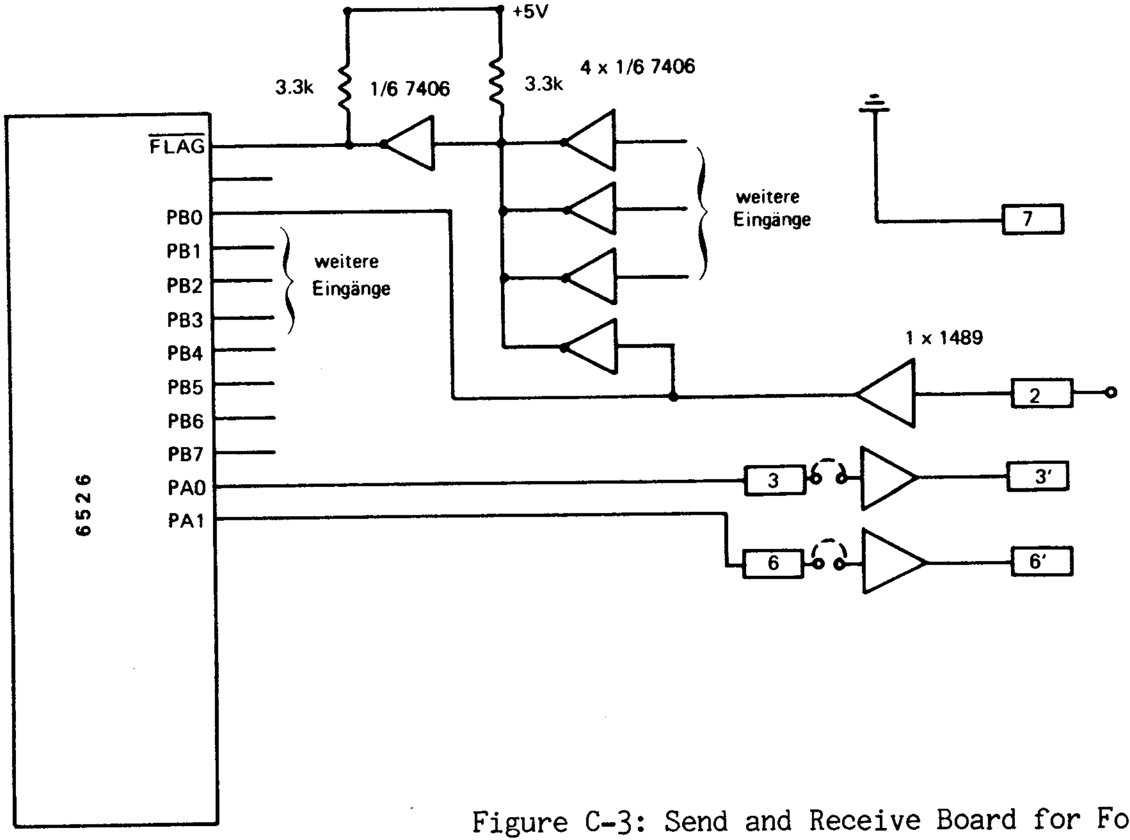


Figure C-3: Send and Receive Board for Four RS232 Devices.

The last Figure C-4 shows the connection of a printer. Only the lines 3 and 7 (Ground) are connected to the printer. If the printer is on, line 20 is one. Lines 6 and 8 are high. The printer can receive data.

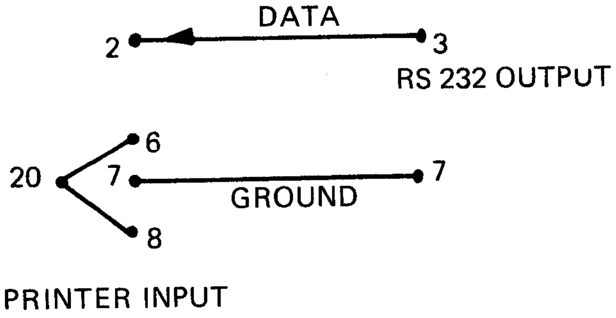


Figure C-4: Connection of a Printer to the RS232 Interface.

# NOTES

# D

# RS 232 KIT for the Commodore-64

The Commodore 64 is a very inexpensive computer, if you compare what you get for your money. But when you want to connect a printer to your C-64 the situation becomes different. In many cases, you only can connect a printer to the serial IEEE port of the C-64. You are limited to a few printers on the market.

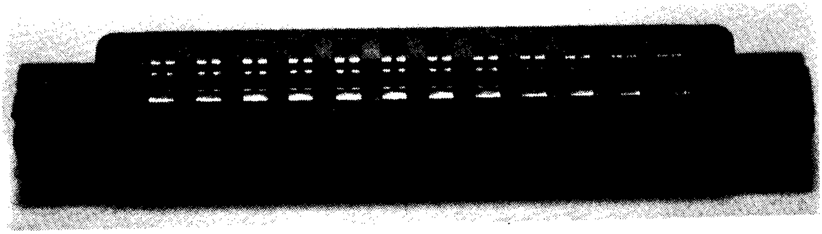
In this construction article we will show you how to connect a serial printer or an inexpensive typewriter with an RS232 port to your C-64. You don't have to buy an expensive cable. You only have to know how to use a soldering iron and how to solder a few components and two or three wires. Your C-64 is completely equipped with the hardware and the software to drive a serial printer, however the manual does not tell you how to connect it and use it.

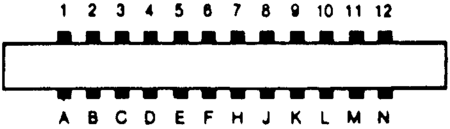
The RS232 interface, which is implemented through the user port, is not a real RS232 with the +-12V levels.

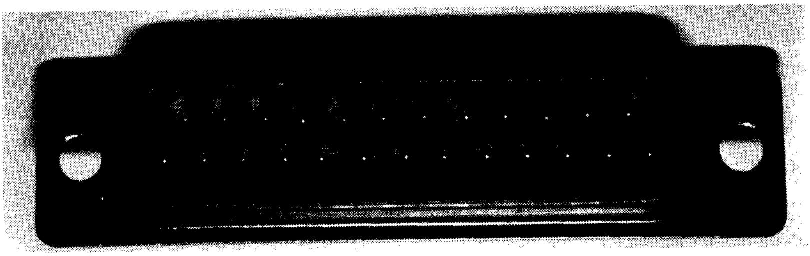
There is only a TTL-level as a transmitted data line available. We used those TTL-level RS232 interfaces with a variety of printers like the DECWRITER, QUME Sprint9, the BROTHER HR15, and the NEC Spinwriter. On all these printers, and even with the Smartmodem from Hayes, the TTL-level RS232 worked fine. The RS232 interface software is built in and the transmission specifications can be set up via certain register settings.

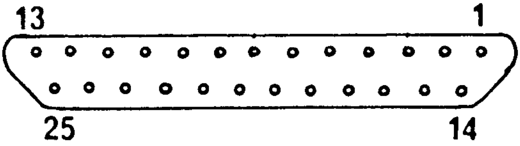
To hook up your RS232 printer all you need is the following:

1. A user port connector 24 pin from TRW CINCH 251-12-50-170/50-24sn-98124 available from your local computerstore or from a distributor that specializes in connectors.





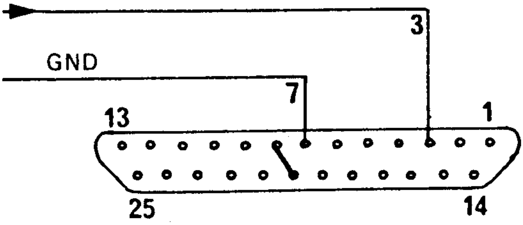




3. Two or three wires

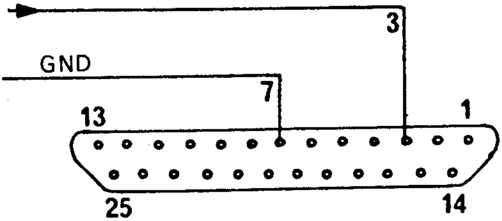
It turned out that a 3 line interface, without any handshaking, was the easiest to connect to our C-64 with a Qume Sprint9 letter quality printer. To wire a 3 line interface you only need to connect:

Receive data on Qume Sprint 9

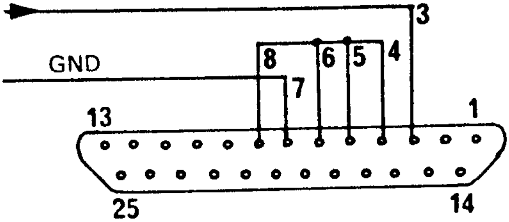


If you want to connect a RS232 printer, you only need to wire the two lines GND and transmitted data.

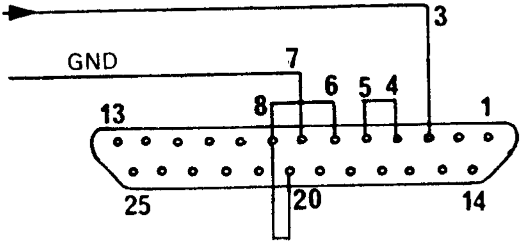
Receive data on DEC-writer



Receive data on Brother HR-15

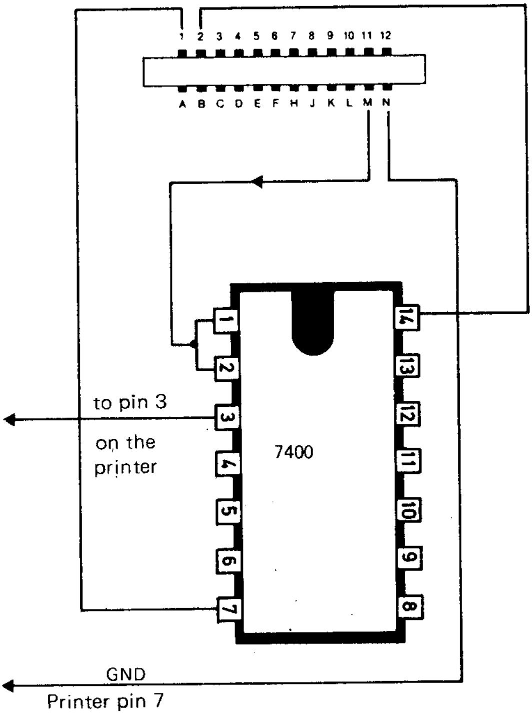


Receive DATA on NEC Spinwriter



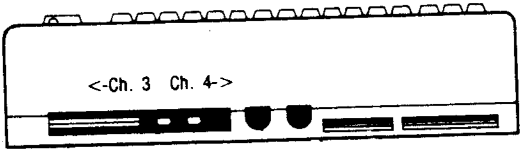
We found out that the transmit data line, comming out of pin M, must be inverted before feeding into the printer. You can connect a 7400 NAND gate directly to the user port connector using Pin 1 and Pin 2 as power supply lines.

If you need more than 4 lines to be inverted, use a 7404 inverter chip containing 6 inverters. A lot of printers with RS232 interface need some handshaking. Therefore you have to wire the printer input connector according to the following schematics.



Comments: Because we operate the C-64 in the 3-wire-mode, no handshaking provided. We must set the jumpers on our printer so that it does not need any handshaking signals. We must also study the manuals carefully and find out which pin must be GND and which must be high (+5V) to receive only data via the lines "Receive Data" and "GND". It usually is pin 3 and 7 at the 25 pin connector on your serial printer or typewriter. Sometimes the line "Receive Data" has to be inverted.

The user port is located on the backside of the C-64, on the left side as seen from the keyboard:



One of the two CIA6526 (Complex Interface Adapters) are used for the RS232 interface. Port lines PB0-PB7 plus one portline from port A (PA2) and one flag pin are used for the RS232 interface.

The pinout of the C-64 user port looks as follows:

| | | |
|---|---|---|
| PB0 | Receive Data Pin C | (Input) |
| PB1 | Request to send | (Output) |
| PB2 | Terminal Ready Pin E | (Output) |
| PB3 | Incoming Call Pin F | (for modem only) |
| PB4 | Input Signal Pin M | (Input) |
| PB5 | NC Pin J | |
| PB6 | Clear to Send Pin K | (Input) |
| PB7 | Data Set Ready Pin L | (Input) |
| FLAG2 | Receive Data Flag,Pin B | (Input) |
| PA2 | Transmit Data, Pin M | (Output) |
| GND | Pin A | |
| GND | Pin N | |

Construction of the cable and inverter.

To construct a cable for RS232 +5V operation we need the following parts:

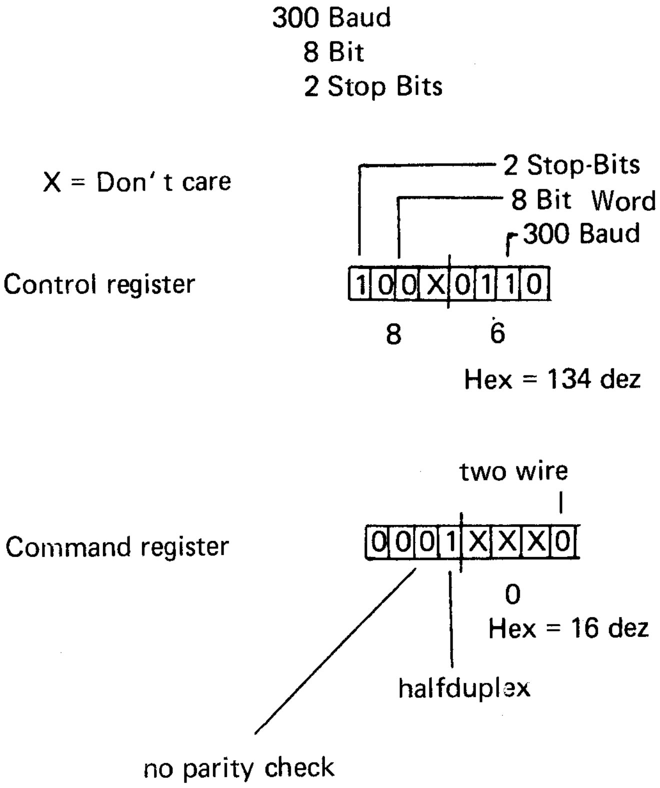
1 User port connector
TRW CINCH 251-12-50-170/24su-98124 or
similar

1 RS232 25 pin connector (male)
1 7400 or 7404 TTL IC
4-7 feet of wires.

How to program the RS232 interface? The built in RS232 interface can be programmed by an OPEN command. In our case we will set the following conditions:

| | |
|---|---|
| Baud rate: | 300 baud |
| Data bits: | 8 data bits |
| Stop bits: | 2 stop bits |

The control register now looks as follows:



This comes up to a content of decimal 134 or hex 86. We must set the Command Register for halfduplex and two resp. three wire operations.

After you have wired everything correctly and connected the C-64 to your serial printer you can test the cable and the connection with the following program:

```
10 OPEN 1,2,0,CHR$(134)+CHR$(16)
20 PRINT #1,"U; : GOTO20
```

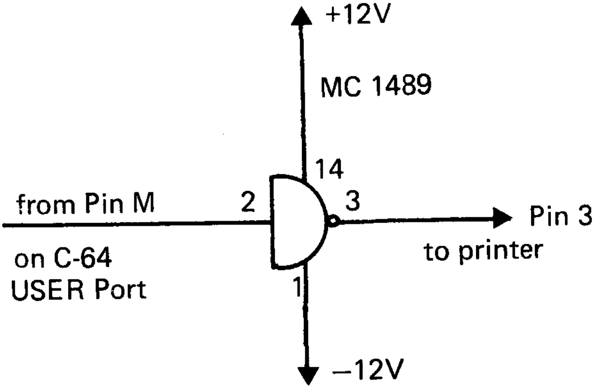
After typing RUN, the printer should start printing. If you want to list a BASIC program to a a printer you have to type in the following:

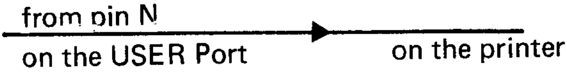
In the direct mode:

```
OPEN 1,2,0,CHR$(134)+CHR$(16)
CMD 1
LIST
```

How to convert your +5V RS232 into a real RS232?

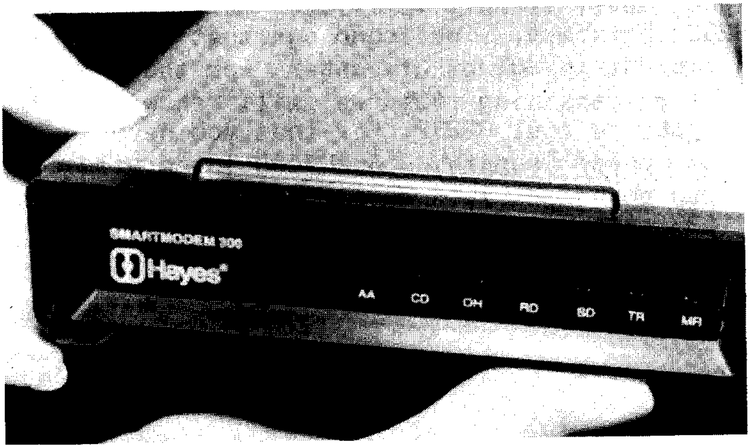
As mentioned earlier our RS232 interface described above is not a real RS232 interface because the signal level is only +5V (TTL). Working with many RS232 printers, we found out that a +5V level is sufficient in 90% of all cases. For those who are in the remaining 10% we will show you how to implement a real RS232. For that you need an extra +-12V power supply. A MC1489 integrated circuit has to be added into our project. Because the MC1489 inverts the signal itself, we don't need the inverter circuit anymore. The schematic now looks like this:





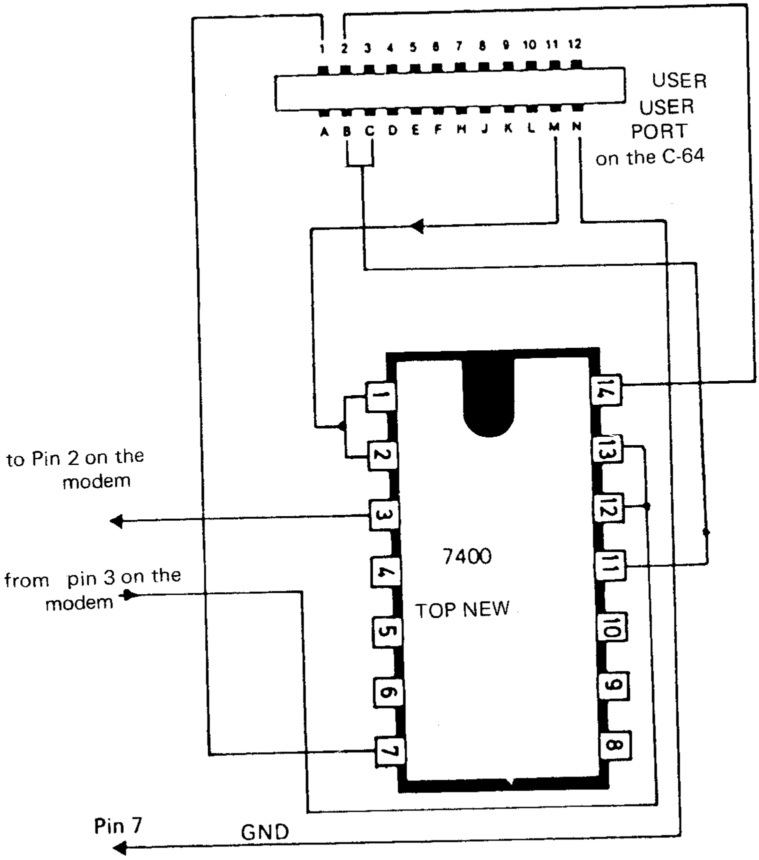
How to connect a modem to your Commodore-64 ?

BLIZTEXT allows you to go from the editor directly into a terminal mode. This gives BLIZTEXT an outstanding feature never seen before on a wordprocessor : You can type your text, format it, save it on disk or cassette or even send it via the Smartmodem into a network or to another computer. You also can download incoming text from a modem into your C -64 and save it for later on cassette or disk.

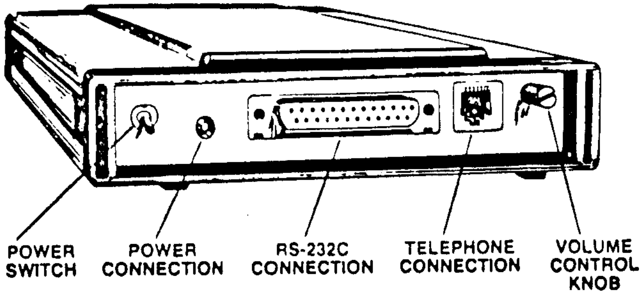


To connect the Smartmodem 300 to your C -64 you need the following:

1. 25 pin RS232 connector (male)
2. 24 pin user port connector for your C-64 user port
3. Three wires approx. 5 feet long



Do not change the factory setting for the configuration switches. You can connect a Smartmodem to your C-64 using the schematic shown above. After wiring the cable and hooking up the phone and power supply, switch on the smartmodem. Jump into the terminal mode from the editor using the command line command T.

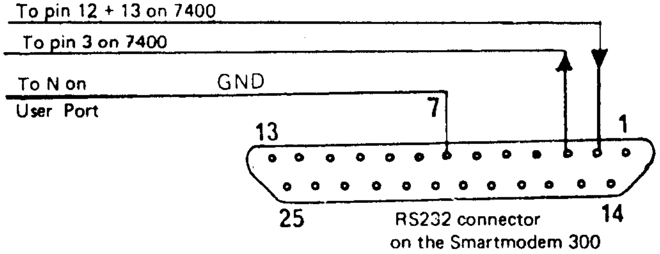


Then type once or twice RETURN, because the first character is always wrong. Type in for instance:

```
ATFO
OK
ATD5033434352
```

which dials a phone number in Oregon with a smart modem hooked up. You should use the telephone number from your network here. When you get double characters stop and switch to full duplex and enter ATF1 at the beginning. For more details and on how to program the Smartmodem, please refer to your Smartmodem 300 Owner's Manual.

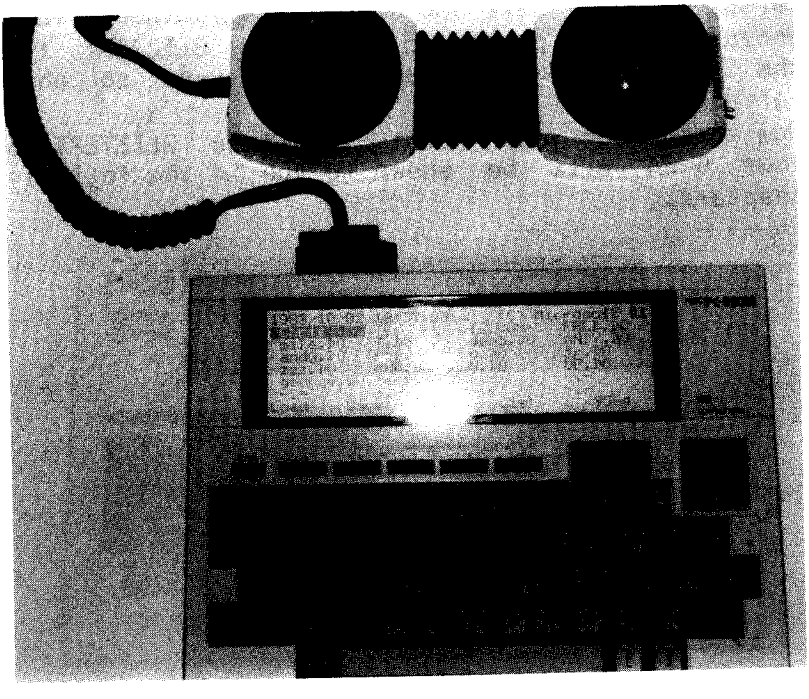
How to wire the connector on the Smartmoden:



We found that the TTL-Level 5V connection to the Smartmodem worked fine. If you use an accustically coupled modem, you may run into problems with the 5V level. Then you have to build the +-12V RS232 cable as described before.

# E Some Interesting Applications for the use of BLITZTEXT

Because of its unique feature which allows you to send and rceive text with BLIZTEXT on your Commodore-64 there now are many useful application hints. The ones discussed below will help you get the most out of your Commodore-64. Introduction of the real portable computers like the TRS-80 Model 100, the NEC 800, the CASIO P200 or the NEC portable has opened a variety of new applications in which BLIZTEXT may be used.

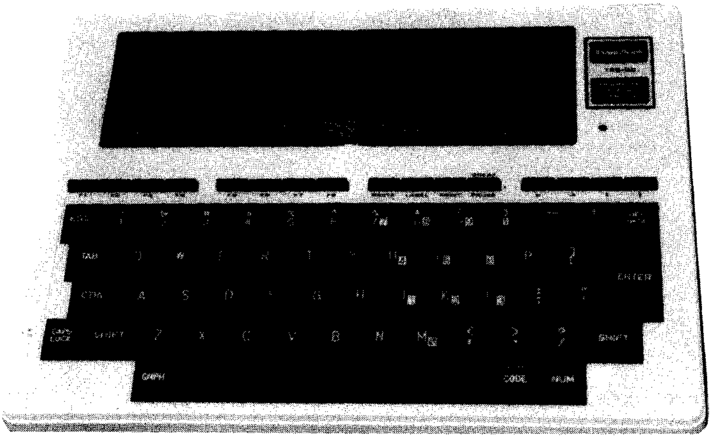


## External text aquisition using a lap computer and BLIZTEXT

The TRS-80 Model 100 is one of the first truly portable computers. It has a built-in simple textprocessing program, which allows you to input and modify text and store this text as a DO file in memory. The information is retained even if the power is switched off.

### The basic concept

You can use your Model 100 or a similar lap computer with an RS232 interface to type your text in while you are on a business trip. The Model 100 may then be connected into BLIZTEXT (with your Commodore-64). The text can be transferred from the Model-100 into the wordprocessor. There you can modify, insert and format the text and store it on cassette or disk or send it to the printer. You also can upload parts of the text and accumulate it in BLIZTEXT, because you can place the text in BLIZTEXT from the current cursor position on and so chain various parts of text.
How to connect your Model 100 into BLIZTEXT on your C-64 will be shown to you in the following chapters.



The UPLOAD function of the Model 100 in the telecommunication mode must be used to send text to BLIZTEXT.

UPLOADING from Model 100 into BLIZTEXT
Type in the text into the Model 100. Check the available memory and make sure that we have enough space for our textfile. If enough space is not available some current DO files must be killed to make room for our new file.

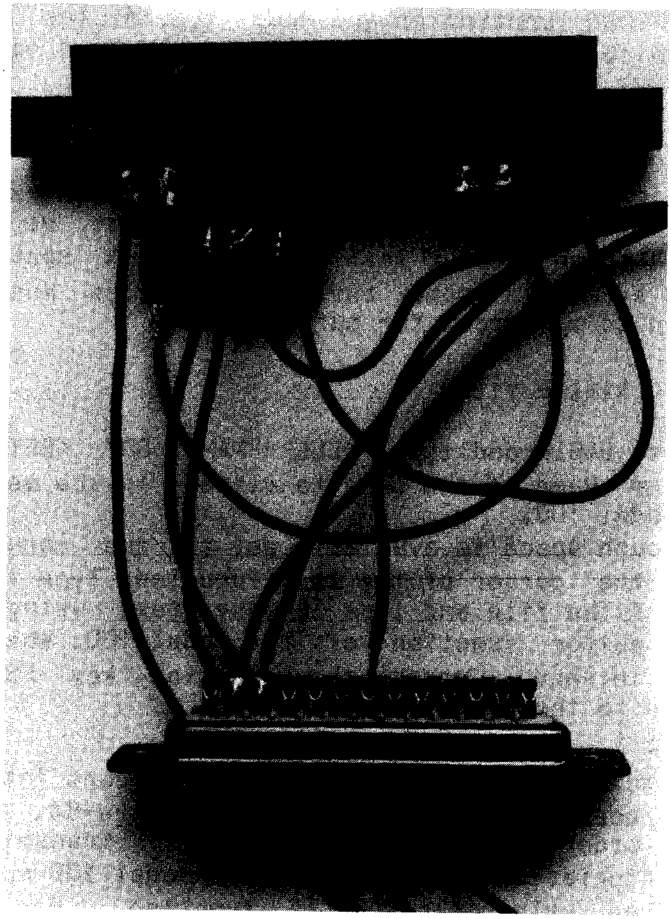
How to kill a file?

Go into BASIC and type KILL "NAME. DO" <RETURN>.
NAME = Name of the DO file already in the menu of the Model 100.
If enough space is available go to the menu and move the cursor of the text function. Type in the name of the file and type in your text using the text editor function of the Model 100. When you are finished, press the function key F8 and return to the menu.

The text can be saved and later sent into the wordprocessor BLIZTEXT on your Commodore-64.
In order for the right transmission characteristics to be set, the Model 100 must be prepared using the "STAT" function. For more information, refer to your Model 100 manual describing the setting procedures using the STAT function.

The Model 100 can now send text via the RS232 interface, so that BLIZTEXT can receive it properly. Proper connections must be made between the two computers.

You need the following parts:
1 connector TRW CINCH 251-12-50-170-50-24su-9824 1
1 RS232 25 pin standard connector 1 SN 7400 (4 NAND-gates TTL-chip) 3 wires approx. 3 feet long
Preparation of the C-64 and BLIZTEXT



Now that we have prepared the Model 100 and wired the connections between the C-64 and the Model 100 we can boot up the Commodore-64 and start BLIZTEXT. After the program has been started, clear the text buffer using the command K in the command line and enter the terminal modus as follows: <CTRL>-<AA> to <CTRL>-<A> <CTRL>-<A>. BLIZTEXT now shows on top of the screen that you are in the terminal facilities. Depress the F1 key. Depress the F1 key only once. It works as a flip-flop and if depressed a second time, it will disconnect the BLIZTEXT terminal facilities from the transmission line. Thus you also can use the F1 key to receive only parts of text. The C-64 is now ready to receive text. The text must be now uploaded from the Model 100 into the C-64. The "Status" of the Model 100 has been set. Go into the main menu, select the TELECOM mode and depress the function F4 key. Then depress the F3 key for uploading text. The Model 100 will ask you for a filename (i.e. which file to upload). Type in the name of the DO file and the width of the text. This is the number of characters which will be placed after a carriage return by the Model 100. We recommend a width of 39 because this matches with the 40 characters per line on the C-64. BLIZTEXT should now receive the text and displays it on the screen. When the transmission is finished, type F1 (function key) on the Commodore-64, and then <SHIFT>-<F1> to return to BLIZTEXT. The text from the Model 100 is now in the wordprocessor and can be modified, formatted or even stored on disk or on cassette using the BLIZTEXT wordprocessor.

The transmission in the other direction (from BLIZTEXT into the Model 100) can be done the same way. The Model-100 must be prepared for downloading instead for UPLOADING.

Downloading text from BLIZTEXT into the Model-100

Text can be downloaded from the Commodore-64 into the Model 100. An example of this application would be if a businessman wants to take a textfile from BLIZTEXT on his business trip and print this out on printer at the customers office.

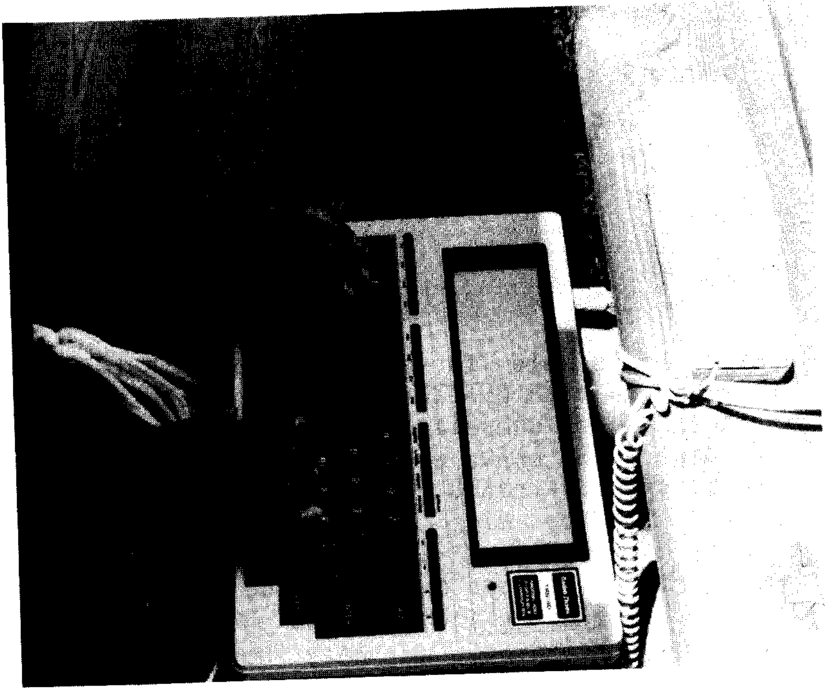
How to upload a textfile?

Prepare the C-64 in the same way as described for uploading. On the Model-100, select the terminal mode using the F4 function key. Then depress the F2 function key for downloading. You also have to input a filename,into which the text then will be stored.

# NOTES

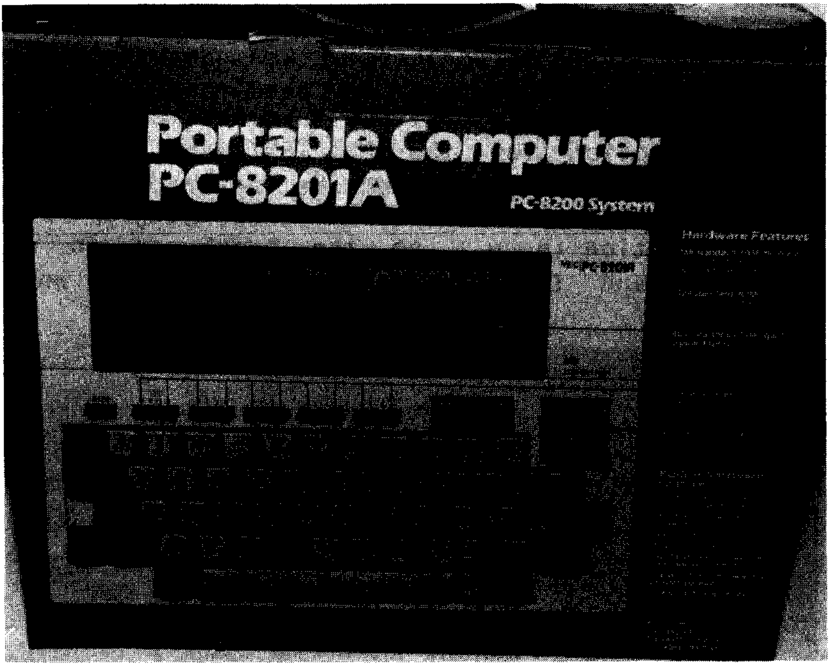
# F Transfer of Textfiles

The "BLIZTEXT" wordprocessor was developed by HOFACKER and allows you to send and receive informations with its built-in terminal mode.
The description for the BLIZTEXT word processor shows you how data (text) can be entered into a a portable tandy model 100 at a geographical remote location (e. g. at the beach or in an airplane) and later sent to the BLIZTEXT word processor.



A lot of interest has been generated in this type of word-processing because it is more convenient for some business (e.g. tourism) to work in this way.
In addition to a model 100, you also may use one of these other popular portable computers:

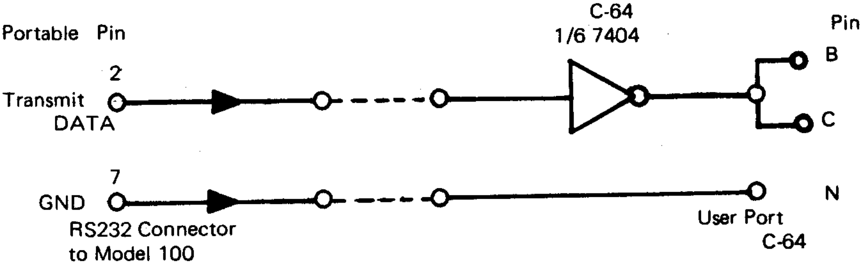
Casio EP 200
NEC pc 8201A
Olivetti M10
PC 1500 (Sharp, with RS232)



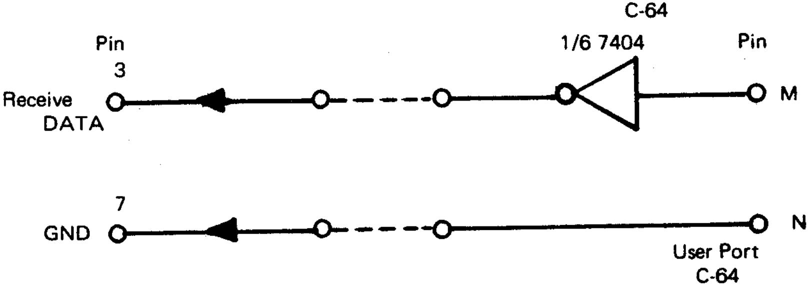
In order to be able to transfer data between a COMMODORE 64 with BLIZTEXT and a portable computer (or any other computer), you need an RS232 interface which works with TTL-level signals (5V), 300 Baud transfer rate, 7 bit wordlength, even parity, 1 stop bit, and two or three wires for transfer without handshaking.
What is the difference between a two and a three wire RS 232 connection? If we only want to send data in one direction, we only need two wires, SIGNAL and SIGNAL GROUND.

The following signals are needed from the RS232 interface:

1. For sending data from the portable computer to BLIZTEXT:



2. For sending data from BLIZTEXT to the portable computer:



When only two wires are used they must be connected to ground (GND) and RECEIVE DATA or SEND DATA.
If three wires are used, they must be connected to GND, SEND DATA, and RECEIVE DATA.
The following signals maybe used with the handshake mode:
1.) Data Terminal Ready
2.) Data Set Ready
3.) Request To Send
4.) Clear To Send
In our case you don't have to connect these. Text maybe entered into the model 100 (NEC and Olivetti are similar) in the following manner.
1.) Turn computer on and select text editing mode.

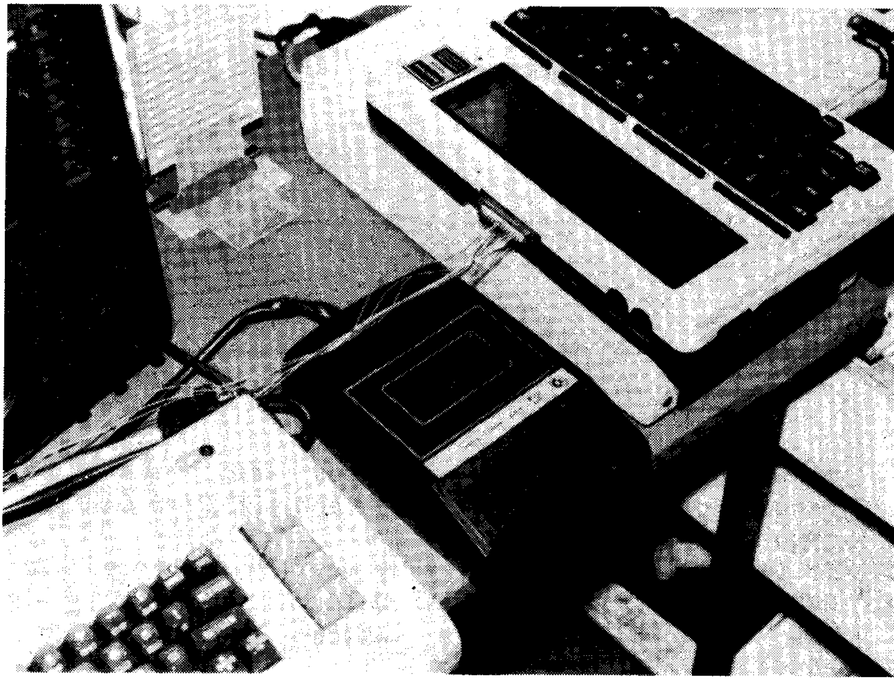
2. ] Check that enough memory is available. If not, go to BASIC and use the "KILL" command to DELETE all unneeded files.

3. ] After entering your text mode, enter the name of the text file. The name should be the file with the text to be saved. The Model 100 will add "DO" to that name automatically.

4.] Check the manual for further details.

5.] After text has been entered, press function key "F8" to terminate.

6. ] This brings you back to "MENU" and the text is stored for future use.



This text can now be sent to your COMMODORE 64 via a cable (see instructions above] or via a modem and the phone line to another location.

## 1. Transfer of text from Model 100 into BLIZTEXT.

Select RS232 mode using the STAT function:

```
                              3 8 N 1 E
                              : : : : :
                              : : : : :
                              : : : : :
                              : : : : :
3=  300 Baud .........          : : : :

8=  8 Bit Word ..........:      : : :

                                : : :
                                : : :
N=  no Parity Check ..........    : :
                                  : :
                                  : :
1=  1 Stopbit ................      :
                                    :
E=  Enable ...................
```

Select the terminal mode on the Commodore-64. This is done by placing the cursor at the beginning of the text with "HOME". Then go to the command line with "CTRL-A". Enter "TO" and "CTRL-A" twice to terminate the command line. You are now in the terminal mode of BLIZTEXT and a different cursor should appear on the screen.

Press RETURN and then the F1 key. Be sure that you depress the F1 key only once. This key works like a flip-flop. You would switch off if you would press it a second time. The function of this key is to allow you to store (keep) received information. If the switch is off, then the information received in the terminal mode is displayed on the screen but not stored. If the switch is in the on position, then the information received is displayed and stored, so that it can be edited, or printed, or saved later with the wordprocessor. This will allow you to save only parts of the information received while omitting out less important information. When you enter the terminal mode this key is in the off position.

Now your C-64 is ready to receive text. To prepare the TRS-80 model 100 for sending, we select the TELECOM mode and press function key F4. Next, press F3 for upload. The file to be sent has to be in the computer as a DO file. Enter the name of the file with the width of the output. This maybe any value up to 254. The Model 100 will start sending information and we should be able to see the text received by the C-64 on its screen.

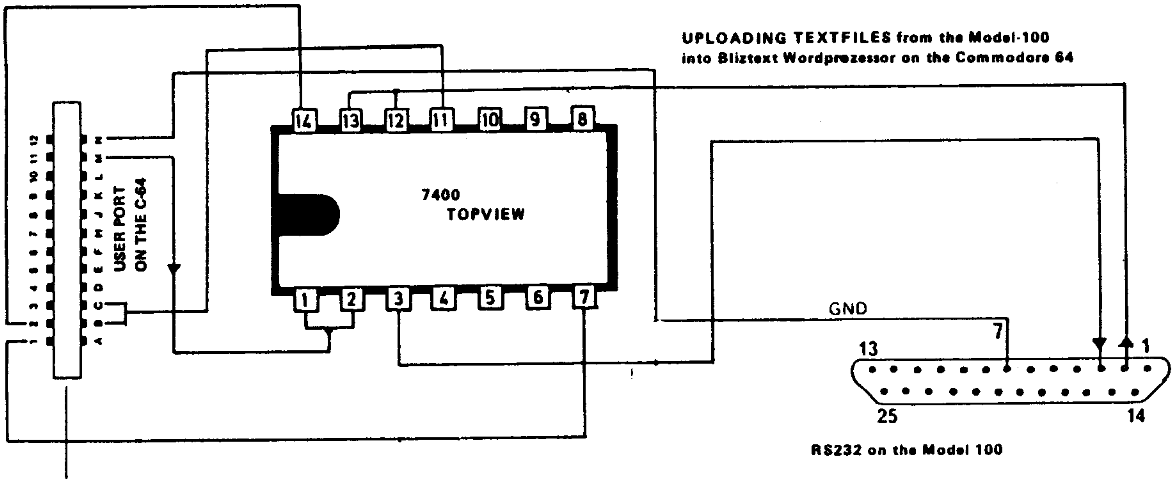
When the transfer is finished, press F8 on the Model 100 and DISCONNECT it. On the C-64, press F1 to terminate the storage and SHIFT-F1 (=F2) to return to the edit mode of BLIZTEXT. You should now see the text received.

We have tested the procedure described above and found it to work very well. We also have done a transfer from an ATARI 800 via the RS232 interface. A transfer via a modem and the telephone system is also possible with the terminal mode of BLIZTEXT.

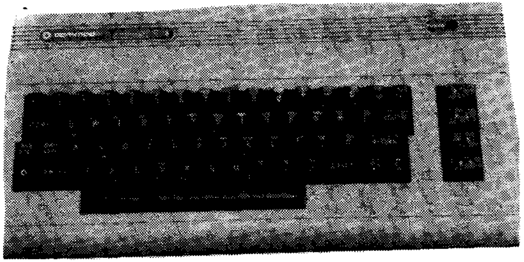
Since the terminal mode of BLIZTEXT allows you to upload and download, it is also possible to send information from the C-64 to the Model 100 or another computer.

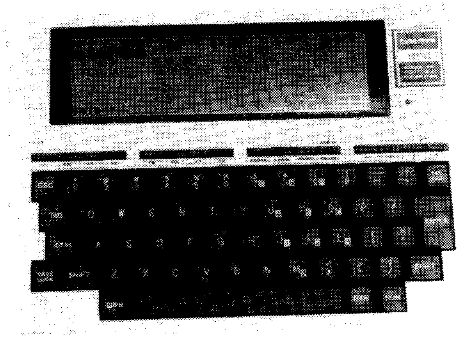
The following figure shows how to connect Model 100 and C-64.

**UPLOADING TEXTFILES from the Model-100**
**into Bliztext Wordprezessor on the Commodore 64**



USER PORT





## 2. Transfer of textfiles from C-64 to Model 100

If you have entered text into your C-64 with BLIZTEXT, we can send this text either formatted or unformatted to the Model 100.
The C-64 must be prepared first. Place the cursor at the position in the text that you want to send. If the entire text should be sent, press HOME. Next, go to the terminal mode by entering CTRL-A TO CTRL-A CTRL-A.(O=zero)
The Model 100 must be prepared to receive (download). Select the TELECOM mode from the menu and press F4 for terminal mode. Next, press F2 for download (send information into the Model 100). F4 allows you to switch between half and full duplex mode (we select half duplex). After F2 has been entered, enter the name under which we want the text received to be stored. If F2 is pressed again, the download will be terminated.
Once the Model 100 is ready to receive, press F3 on the C-64 to start the Model 100. After the download is finished, press F8 on the Model 100, after which you will get the display 'DISCONNECT ? '. Enter 'Y' here. Pressing F8 again brings us back to the menu, where we should be able to see the new file with the extension 'DO', which was added automatically.
Since the Model 100 contains a simple wordprocessor, we are able to edit the received text. To do so, select the textmode and enter the filename. The text is now available for editing. It may also be sent back to the C-64. If you want to print the text on a printer hooked up to the parallel interface, enter SHIFT-PRINT.

The following is an example of how these applications maybe used in the business world. A businessman could prepare his correspondence and price lists using his BLIZTEXT word-processor at home. This text could then be transferred to a portable Model 100 which he could take with him while calling on clients. This text could also be transferred to the client's printer. If the need arose, he could alter the text by utilizing the mini-wordprocessor built-in to the Model 100. If new input text is required from his home or business, it can be sent via the modem and the telephone.

# NOTES

# G An Overview of the Connectors for the C-64

Appendix G.

An Overview of the Connectors for the C64.

For the expansion and the experiments, connectors must be used. A newcomer can't imagine how difficult it is to get the right connector. The following is a short overview of the connectors which can be used with the C64.

The Connector of the Expansion Port.

For the expansion port, a 44 pin connector, shown in Figure E-1 is used. It is located at the rear right side.

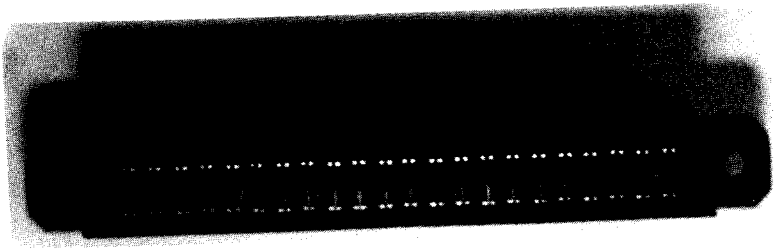


Figure E-1: 44 Pin Connector for the Expansion Port.

In this connector an expansion board can be plugged in. This board is shown in Figure E-2.

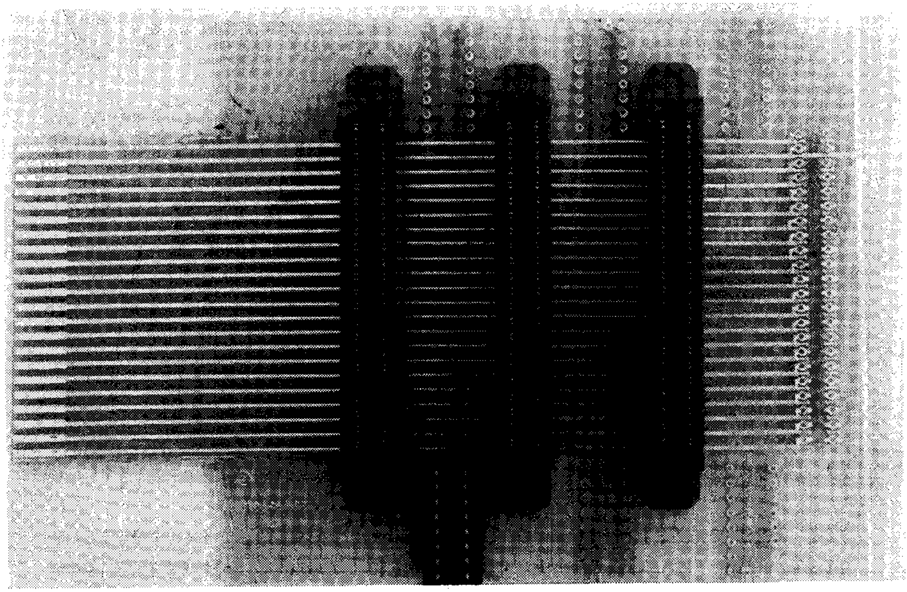


Figure E-2: Expansion Board.

This expansion board can also be used as a plug in, to connect the C64 with other devices. This is shown in Figure 4-11. The numbering of the expansion port in the "Commodore 64 Micro Handbook" contains an error. The correct numbering is shown below (as seen from the rear).

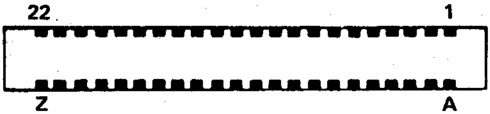


Figure E-3 shows another printed circuit board, which can be used with the expansion port.

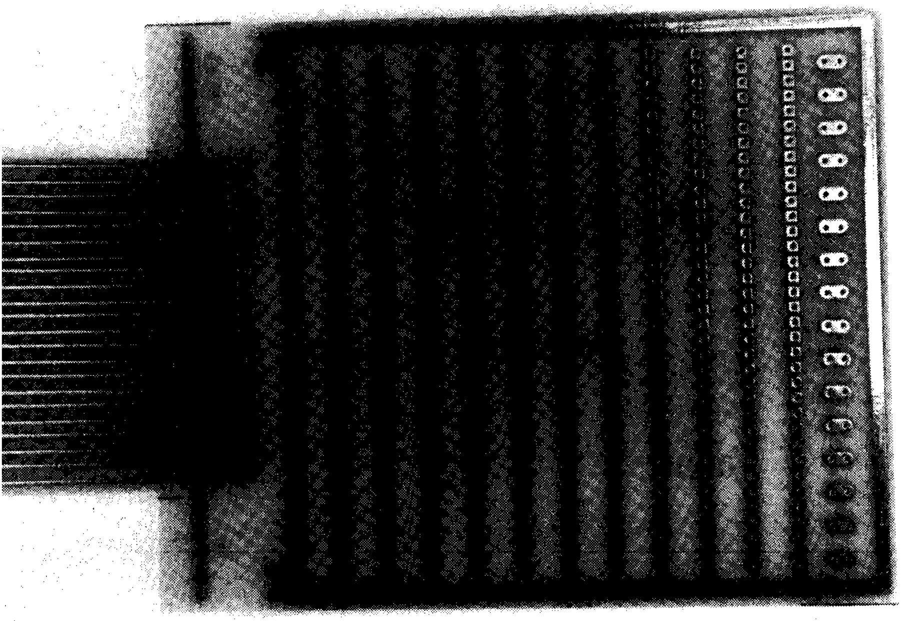
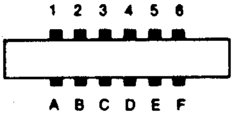
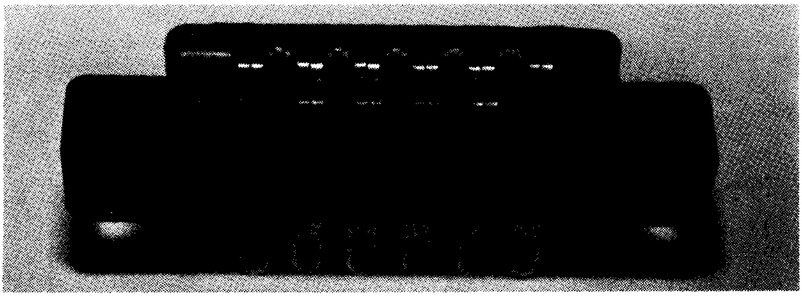


Figure E-3: Experimenter Board for the Expansion Port.

The Connector for the Cassette Port.

Figure E-4 shows the connector for the cassette port. At this port, pin 2 provides +5 volts, 0.5 amps. Pin 1 is ground (GND). This voltage can be used for experimenting. The pinlayout is shown below.





The connector for the USER-Port is shown in Figure E-4. It is a 22 pin female plug.

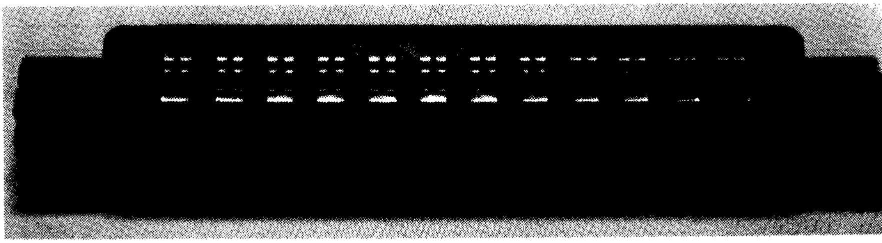
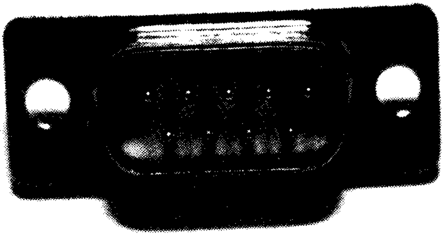


Figure E-4: Connector for the USER-Port.

The next Figure E-5 shows the female plug for the Joystick port. This plug can be used for connecting a light pen to the C64 or to use the built in A/D converter.



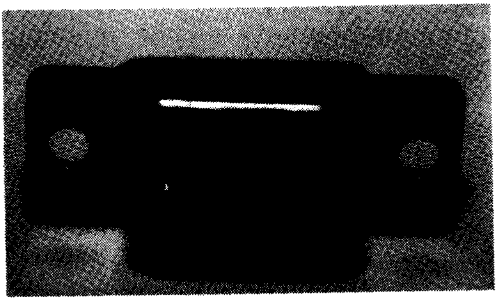
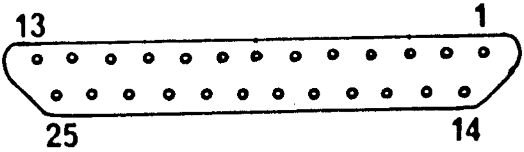


Figure E-5: Female and Male Plug for the Joystick Port.

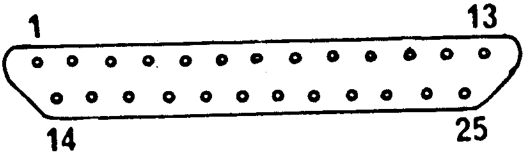
The Connectors for the RS232 Interface.

Figure E-6 shows the connector which is used for the RS232 interface. The outlet of the computer and the input at a printer are always female plugs. The numbering of the plug is shown below (as seen from the rear). For connecting the C64 to the RS232 input of a printer, a male plug shown in Figure E-7 must be used. The numbering (as seen from the rear) is shown below.



Male plug, seen from the rear

Female plag, seen from the rear



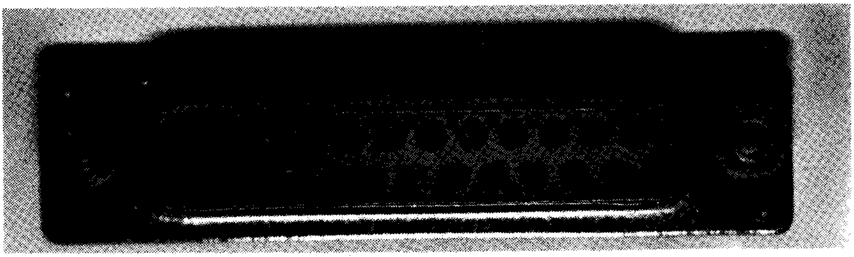


Figure E-6: Female Plug for the RS232 Interface.

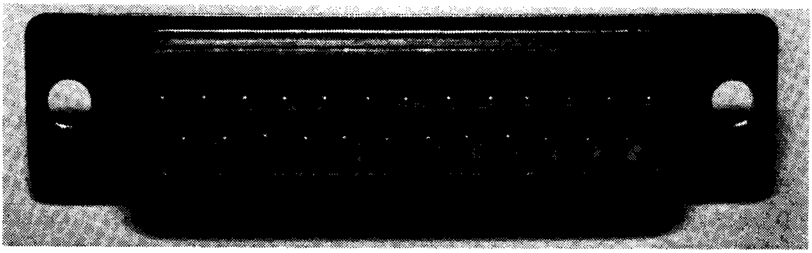
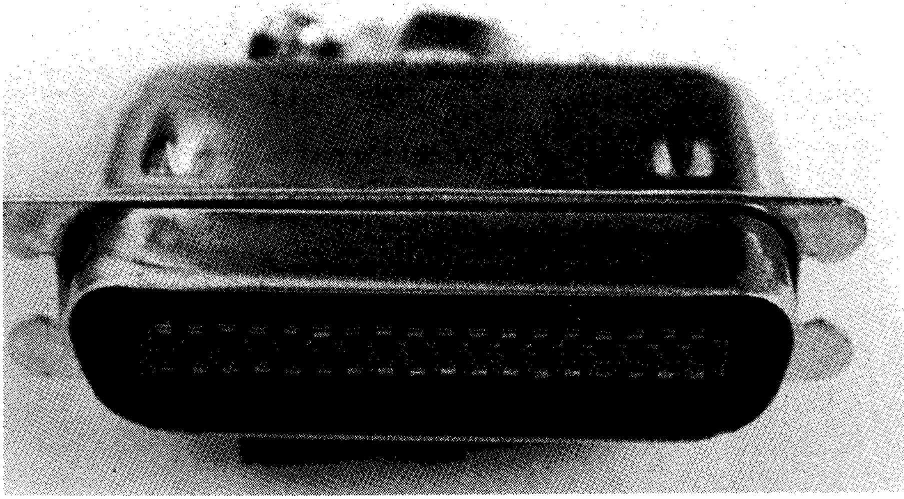


Figure E-7: Male Plug for the RS232 Interface.

The last connector shown in Figure E-8 is a connector used with the Centronics interface or the IEC Bus. This is a 36 pin connector.



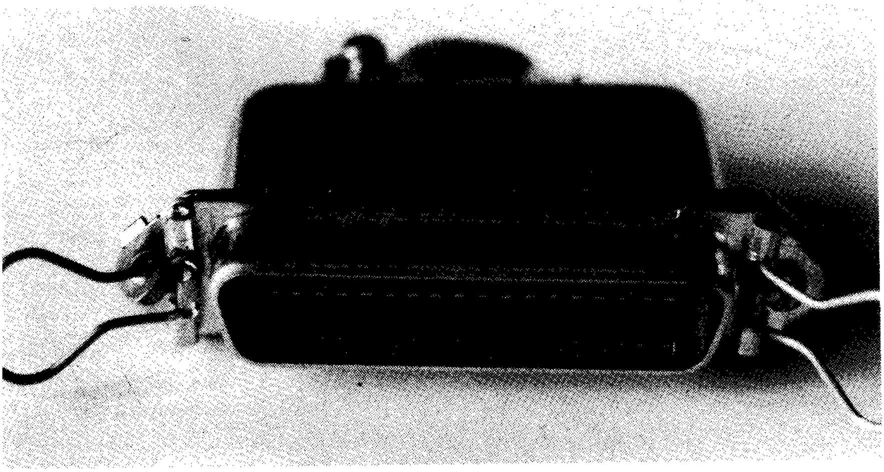


Figure E-8: 36 Pin Connector for the Centronics Interface.

# NOTES

# H Forth Word Set

FORTH-GLOSSARY

## STACK MANIPULATION

| | | |
|---|---|---|
| DUP | ( n-nn) | Duplicate the top of the stack. |
| DROP | ( n) | Throw away the top element of the stack. |
| SWAP | ( nn'-n'n) | Reverse the two top elements. |
| OVER | ( nn'-nn'n) | Copy second element on top of the stack. |
| ROT | ( nn1n2-n1n2n) | Rotate the top three elements counterclockwise. |
| >R | ( n) | Move top element to the return stack. |
| R> | ( -n) | Retrieve top element from the return stack. |
| R | ( -n) | Copy top element of return stack to parameter stack |

## ARITHMETIC AND LOGICAL

| | | |
|---|---|---|
| + | ( nn'-n1) | n1=n+n' |
| - | ( nn'-n1) | n1=n-n' |
| * | ( nn'-n1) | n1=n*n' |
| / | ( nn'-n1) | n1=n/n' |
| */ | ( n1n2n3-n) | n=n1*n2/n3 with double precision intermediate. |
| MOD | ( nn'-n1) | n1 remainder of n/n'. |
| /MOD | ( nn'-n1n2) | n1 remainder, n2 qoutient of n/n'. |
| */MOD | ( n1n2n3-nn') | n remainder, n' quotient of n1*n2/n3. |
| MINUS | ( n- -n) | Change sign. |
| MAX | ( nn'-n1) | n1=n if n>n' else n1=n'. |
| MIN | ( nn'-n1) | n1=n if n<n' else n1=n'. |
| ABS | ( n-n') | n' absolute of n. |
| D+ | ( dd'-d1) | d1=d+d' double precision addition. |
| DMINUS | ( d-d') | Change sign. |
| DABS | ( d-d') | d' absolute of d. |
| AND | ( nn'-n1) | Logical AND bitwise. |
| OR | ( nn'-n1) | Logical OR bitwise. |
| XOR | ( nn'-n1) | Logical XOR bitwise. |

## CONTROL STRUCTURES

| | | |
|---|---|---|
| DO...LOOP | | |
| DO | ( nn') | Loops from n' to n-1, loop increment is one. |
| DO...+LOOP | | |
| DO | ( nn') | Loops from n' to n-1. |
| +LOOP | ( n) | Loop increment is n (may be negative). |
| I | ( n) | Put loop index on the stack, same as R. |
| LEAVE | (  ) | Terminate loop at next LOOP or +LOOP. |
| IF <words> THEN (ENDIF) | | |

| | | |
|---|---|---|
| IF | ( f) | If f is not zero, <words> are executed. |
| IF <words1> ELSE <words2> THEN (ENDIF) | | If f is not zero, <words1> are executed, else <words2>. |
| IF | ( f) | |
| BEGIN <words> UNTIL (END) | | <words> are repeated until f is non zero. |
| UNTIL (END) | ( f) | |
| BEGIN <words1> WHILE <words2> REPEAT | | If f is zero, program continues after REPEAT, else unconditional branch back from REPEAT to BEGIN. |
| WHILE | ( f) | |

## MEMORY

| | | |
|---|---|---|
| @ | ( a-n) | Fetch content from address a and a+1. |
| C@ | ( a-b) | Fetch byte from address a. |
| ! | ( na) | Store n in address a and a+1. |
| C! | ( ba) | Store byte b in address a. |
| ? | ( a) | Print content of address a and a+1. |
| +! | ( na) | Add n to the content of address a and a+1. |
| CMOVE | ( aa'n) | Move n bytes from a to a'. a+n<a'<a. |
| FILL | ( anb) | Store n bytes b into memeory starting at address a. |
| ERASE | ( an) | Store n ASCII 0 into memeory starting at address a. |
| BLANKS | ( an) | Store n ASCII 32 into memory starting at address a. |
| , | ( n) | Store n on top of dictionary. Add two to HERE. |
| C, | ( b) | Store b on top of dictionary. Add one to HERE. |
| ALLOT | ( n) | Leave gap of n bytes on top of dictionary. |

## COMPARSION

| | | |
|---|---|---|
| < | ( nn'-f) | f=1, if n<n'. |
| > | ( nn'-f) | f=1, if n>n'. |
| = | ( nn'-f) | f=1, if n=n'. |
| 0< | ( n-f) | f=1, if n<0. |
| 0= | ( n-f) | f=1, if n=0. |

## NUMBER BASES

| | | |
|---|---|---|
| DECIMAL | ( ) | Set decimal base. |
| HEX | ( ) | Set hexadecimal base. |
| BASE | ( -a) | Variable, contains number base. |

## INPUT-OUTPUT

| | | |
|---|---|---|
| . | ( n) | Print n. |
| ." xxx" | (  ) | Print message xxx. Message ends with "  . |
| .R | ( nn') | Print n, rightjustified in field. Fieldwidth is n'. |
| D. | ( d) | Print double precision number d. |
| D.R | ( dn) | Print d rightjustified in field. Fieldwidth is n. |
| U. | ( n) | Print n as unsigned number u. |
| EMIT | ( c) | Print ASCII character c. |
| TYPE | ( an) | Print n bytes starting at address a. |
| KEY | ( -c) | Get character from keyboard. |
| ?TERMINAL | ( -f) | f=1, if BREAK key was pressed. |
| EXPECT | ( an) | Expects n bytes at address a. |
| WORD | ( c) | Read characters from the input buffer until delimiter c is found. |
| COUNT | ( a-a'n) | Change length byte at address a to a'=a+1 and length n. Ready for TYPE. |

## NUMBER FORMATTING

| | | |
|---|---|---|
| <# | (  ) | Start converting a number to a string. |
| # | (  ) | Convert one digit and add it to the string. |
| #S | (  ) | Convert remaining digits. |
| #> | ( an) | End of conversion. Puts starting address a and length n on the stack. Ready for TYPE. |
| HOLD | ( c) | Inserts the ASCII character into the string. |
| SIGN | ( n) | Inserts sign of n into the string. |
| NUMBER | ( a-d) | Convert a string at address a+1 to a double precision number. The length of the sring must be stored at address a. |

## DEFINING WORDS

| | | |
|---|---|---|
| : xxx | (  ) | Begin of a colon definition. |
| ; | (  ) | End of a colon definition. |
| VARIABLE <name> | ( n) | Create variable <name> with the initial value n. |
| <name> | ( a) | The address a of the variable is put on the stack. |
| CONSTANT <name> | ( n) | Create a constant <name> with  the value n. |
| <name> | ( n) | The value n of the constant is returned. |
| <BUILDS <wordsl> DOES> <words2> | | Used to create a new defining word. <wordsl> are executed during compile time. <words2> are executed during run time. |
| CREATE <name> | (  ) | Creates an entry <name> into the vocabulary. The codefield address of <name> is the parameter field address. |

VOCABULARIES

| | | |
|---|---|---|
| CONTEXT | ( a) | Returns address of CONTEXT vocabulary. This is searched first. |
| CURRENT | ( a) | Returns address of CURRENT vocabulary. New definitions are put here. |
| FORTH | ( ) | Main FORTH vocabulary. |
| VOCABULARY <name> | ( ) | Opens new vocabulary. Sets CURRENT to <name>. |
| DEFINITIONS | ( ) | Sets CONTEXT to CURRENT. |
| VLIST | ( ) | Print all words. |
| FORGET <name> | ( ) | Forqet all definitions back and including <name>. |
| ' <name> | ( a) | Get parameter field address of <name>. |

DISK

| | | |
|---|---|---|
| LIST | ( n) | List content of text screen n. |
| LOAD | ( n) | Compile text screen n into dictionary. |
| BLOCK | ( n-a) | Read block n. |
| EMPTY-BUFFERS | ( ) | Erase all disk buffers. |
| UPDATE | ( ) | Mark last buffer accessed. |
| FLUSH | ( ) | Save all updated disk buffers. |
| INDEX | ( nn') | List all first lines of text screens n to n'. |

SOME VARIABLES

| | | |
|---|---|---|
| DP | ( -a) | Dictionary pointer. Contains the first free memory location on top of the vocabulary. |
| HERE | ( -n) | Fetches DP. |
| PAD | ( -a) | Scratch buffer PAD. 68 bytes above HERE. |
| TIB | ( -a) | Terminal input buffer. |
| IN | ( -n) | Offset to terminal input buffer. |
| S0 | ( -a) | Contains the initial address of the parameter stack. |
| SP@ | ( -a) | Fetch content of S0. |
| BLK | ( -a) | Contains current block number. |
| SCR | ( -a) | Contains current screen number. |
| B/BUF | ( -n) | Constant, gives block size in bytes. |

**FORTH Word Set**

# NOTES