TIPS AND TRICKS

ARITHMETIC

CHAPTER 1

1.1. Input and output of numbers

When working with numbers one often wants to input and output them via the screen. The following programs show how th is can be done with hexadecimal as well as decimal numbers.

1.1.1. Hexadecimal input

This program allows you to enter hexadecimal numbers using the keyboard. The number entered is displayed on the screen. The input stops if a character different from the hexadecimal numbers (0.. F) is entered.

The program first deletes memory locations EXPR and EXPR+1. This ensures a result equal to zero, even if an invalid number is entered. Next, the program reads a character and checks whether or not it is a hexadecimal number. If it is, then the upper bits of the number in the accumulator are erased and the lower bits are shifted up. Now, these four bits can be shifted to EXPR from the right. The preceeding number in EXPR is shifted to the left by doing so.

If you enter a number with more- than four digits, only the last four digits are used.

Example : ABCDEF => CDEF

HEXINPUT ROUTINE

```
                    EXPR        EQU $80.1
                    SCROUT      EQU $F6A4
                    GETCHR      EQU $F6DD
                                ORG $A800

A800: A2 00         HEXIN       LDX   #0
A802: 86 80                     STX   EXPR
A804: 86 81                     STX   EXPR+1
A806: 20 2C A8      HEXINI      JSR   NEXTCH
A809: C9 30                     CMP   '0
A80B: 94 1E                     BCC   HEXRTS
A80D: C9 3A                     CMP   '9+1
A80F: 90 0A                     BCC   HEXIN2
A811: C9 41                     CMP   'A
A813: 90 16                     BCC   HEXRTS
A815: C9 47                     CMP   'F+1
A817: 80 12                     BCS   HEXRTS
A819: E9 36                     SBC   'A-10-1
A81B: 0A            HEXIN2      ASL
A81C: 0A                        ASL
A81D: 0A                        ASL
A81E: 0A                        ASL
A81F: A2 04                     LDX   #4
A821: 0A            HEXIN3      ASL
A822: 26 80                     ROL   EXPR
```

```
A824: 26 81                 ROL   EXPR+1
A826: CA                    DEX
A827: DO F8                 BNE   HEXIN3
A829: FO DB                 BEQ   HEXINI ALWAYS !!
A82B: 60          HEXRTS    RTS

A82C: 20 DD F6    NEXTCH    JSR   GETCHR
A82F: 20 A4 F6              JSR   SCROUT SHOW CHARAC'.
A832: 60                    RTS

PHYSICAL ENDADDRESS: $A833
*** NO WARNINGS

EXPR  $80
GETCHR      $F6DD
HEXIN1      $A806
HEXIN3      $A821
NEXTCH      $A82C

SCROUT      $F6A4
HEXIN       $A800 UNUSED
HEXIN2      $A81b
HEXRTS      $A82B
```

1.1.2. Hexadecimal output

The next program explains the output process of the calculated numerals.

You will recognize, that the portion of the program which controls the output is a subroutine. This subroutine only displays the contents of the accumulator. This means that you first have to load the accumulator with, for example, the contents of EXPR+1, then jump into the subroutine where first the MSB (EXPR+1 in our case) and then the LSB (EXPR) will be printed. Subroutine PRBYTE independently prints the most significant bytes of the accumulator first and the least significant bytes second.

HEXOUT PRINTS 1 BYTE

```
                  EXPR        EPZ   $80.1
                  SCROUTE     EQU   $F6A4
                              ORG   $A800
A800: A5 81       PRWORD      LDA   EXPR+1
A802: 20 0B A8                JSR   PRBYTE
A805: A5 80                   LDA   EXPR
A807: 20 A8                   JSR   PRBYTE
A80A: 60                      RTS

* THE VERY PRBYTE ROUTINE

A80B: 48          PRBYTE      PHA
A80C: 4A                      LSR
A80D: 4A                      LSR
A80E: 4A                      LSR
```

```
A80F: 4A                    LSR
A810: 20 16 A8              JSR   HEXOUT
A813: 68                    PLA
A814: 29 0E                 AND   #$00001111
A816: C9 0A       HEXOUT    CMP   #10
A818: B0 04                 BCS   ALFA
A81A: 09 30                 ORA   '0
A81C: D0 02                 BNE   HXOUT
A81E: 69 36       ALFA      ADC   'A-10-1
A820: 4C A4 F6    HXOUT     JMP   SCROUT

PHYSICAL ENDADDRESS:$A823

*** NO WARNINGS

EXPR        $80
PRWORD      $A800   UNUSED
HEXOUT      $A816
HXOUT       $A820

SCROUT      $F6A4
PRBYTE      SA80P
ALFA        $A81E
```

1.1.3. Decimal input

When you calculate with numbers you probably prefer decimals over hexadecimals. The following program can be used to read decimal numbers and convert them into binary numbers readable by computers.

The program first checks, to see if the input is a decimal number (0..9) or if the input has been. terminated by another character. EXPR and EXPR+1 are erased. If a digit is accepted then the upper bits are erased. Next the contents of EXPR and EXPR+1 are multiplied by 10 and the new number is added. In the end the MSB is in location EXPR+1 and the LSB is in location EXPR.

Numbers greater than 65535 are displayed in modulo 65536 (the rest which remains after deduction of 65535).

DECIMAL TO 1 WORD CONVERSION

```
                  EXPR        EQU   $80.1
                  SCROUT      EQU   $F6A4
                  GETCHR      EQU   $F6DD
                              ORG   $A800

A800: A2 00       DECIN       LDX   #0
A802: 86 80                   STX   EXPR
A804: 86 81                   STX   EXPR+1
A806: 20 26 A8    DEC1        JSRN  EXTCH
```

```
A809: C9 30                 CMP   '0
A80B: 90 18                 BCC   DECEND
A80D: C9 3A                 CMP   '9+1
A80F: B0 14                 BCS   DECEND
A811: 29 0E                 AND   #$00001111
A813: A2 11                 LDX   #17
A815: D0 05                 BNE   DEC3        ALWAYS TAKEN ! !


A817: 90 02       DEC2      BCC   *+4
A819: 69 09                 ADC   #10-1
A81B: 4A                    LSR
A81C: 66 81       DEC3      ROR   EXPR+1
A81E: 66 80                 ROR   EXPR
A820: CA                    DEX
A821: DO F4                 BNE   DEC2
A823: F0 E1                 BEQ   DEC1        ALWAYS ! !
A825: 60          DECEND    RTS
A826: 20 DD F6    NEXTCH    JSR   GETCHR
A829: 20 A4 F6              JSR   SCROUT
A82C: 60                    RTS


PHYSICAL ENDADDRESS: $A82D
*** NO WARNINGS


EXPR        $80
GETCHR      $F6DD
DEC1        $A806
DEC3        $A81C
NEXTCH      $A826
SCROUT      $F6A4
DECIN       $A800 UNUSED
DEC2        $A817
DECEND      $A825
```

1.1.4. Decimal output

The next program allows you to display decimal numbers.

The program works as follows:

The X-register is loaded with the ASCII equivalent of the digit 0. This number is then incremented to the highest potency of 10 (10000) and is displayed on the screen.

The same procedure is repeated for 1000, 100, and 10. The remaining is converted into an ASCII number, using an OR-command, and is displayed.

You might want to change the output routine so that it avoids leading zeroes.

2 BYTE BINARY NUMBER TO 5 DIGITS DECIMAL CONVERSION WITH LEADING ZEROES

```
                    DECL        EQU   $80
                    DECH        EQU   $81
                    TEMP        EQU   $82

                    SCROUT      EQU   $F6A4
                                ORG   $A800

A800: A0 07         DECOUT      LDY   #7
A802: A2 30         DECOUT1     LDX   '0
A804: 38            DECOUT2     SEC
A805: A5 80                     LDA   DECL
A807: F9 2E A8                  SBC   DECTAB-1,Y
A80A: 48                        PHA
A80B: 88                        DEY
A80C: A5 81                     LDA   DECH
A80E: F9 30 A8                  SBC   DECTAB+1,Y
A811: 90 09                     BCC   DECOUT3
A813: 85 81                     STA   DECH
A815: 68                        PLA
A816: 85 80                     STA   DECL
A818: E8                        INX
A819: C8                        INY
A81A: DO E8                     BNE   DECOUT2
A81C: 68            DECOUT3     PLA
A81D: 8A                        TXA
A81E: 84 82                     STY   TEMP
A820: 20 A4 F6                  JSR   SCROUT
A823: A4 82                     LDY   TEMP
A825: 88                        DEY
A826: 10 DA                     BPL   DECOUT1
A828: A5 80                     LDA   DECL
A82A: 0930                      ORA   '0
A82C: 4C A4 F6                  JMP   SCROUT

A82F: 0A 00         DECTAB      DFW   10
A831: 64 00                     DFW   100
A833: E8 03                     DFW   1000
A835: 10 27                     DFW   10000

PHYSICAL ENDADDRESS:    $A837
*** NO WARNINGS

DECL        $80
TEMP        $82
DECOUT      $A800    UNUSED
DECOUT2     $A804
DECTAB      $A82F
DECH        $81
SCROUT      $F6A4
DECOUTI     $A802
DECOUT3     $A81C
```

1.2. 16-bit arithmetic without sign

1.2.1. 16-bit addition

The 16-bit addition is well known, but it is shown here one more time, together with the subtraction.

16 BIT ADDITION UNSIGNED INTEGER
EXPR : = EXPR1 + EXPR2

```
                EXPR1       EPZ   $80.1
                EXPR2       EPZ   $82.3

                            ORG   $A800

A800: 18        ADD         CLC
A801: A5 80                 LDA   EXPR1
A803: 65 82                 ADC   EXPR2
A805: 85 80                 STA   EXPR1
A807: A5 81                 LDA   EXPR1+1
A809: 65 83                 ADC   EXPR2+1
A80B: 85 81                 STA   EXPR1+1
A80D: 60                    RTS

PHYSICAL ENDADDRESS: $A80E

*** NO WARNINGS

EXPR1       $80
EXPR2       $82
ADD         $A800       UNUSED
```

1.2.2. 16-bit subtraction

16 BIT SUBTRACTION UNSIGNED INTEGER
EXPR : = EXPR1 - EXPR2

```
                EXPR1       EPZ   $80.1
                EXPR2       EPZ   $82.3

                            ORG   $A800

A800: 38        SUB         SEC
A801: A5 80                 LDA   EXPR1
A803: E5 82                 SBC   EXPR2
```

```
A805: 85 80                  STA   EXPR1
A807: A5 81                  LDA   EXPR1+1
A809: E5 83                  SBC   EXPR2+1
A80B: 85 81                  STA   EXPR1+1
A80D: 60                     RTS

PHYSICAL ENDADDRESS: $A80E

*** NO WARNINGS

EXPR1       $80
EXPR2       $82
SUB         $A800       UNUSED
```

1.2.3. 16-bit multiplication

The multiplication is much more complicated than addition or subtraction. Multiplication in the binary number system is actually the same as in the decimal system. LPt's have a look at how we multiply using the decimal system. For example, how do we calculate 5678*203?

```
   5678
    203 *
  17034
 00000
11356
1152634
```

With each digit the previous number is shifted to the right. If the digit is different from zero the new interim results are added. In the binary system it works the same way. For example:

```
    1011
    1101 *
    1011
   0000
  1011
 1011
10001111
```

.

As you can see it is simpler in the binary system than in the decimal system. Since the highest possible number for each digit is 1 the highest interim results is equal to the multiplicand.

The following program in principle does the same as the procedure described above, except that the interim result is shifted to the right and the multiplicand is added, if required. The results are the same.

Six memory locations are required. Two of these (SCRATCH and SCRATCH+1) are used only part of the time, while the other four locations keep the two numbers to be multiplied (EXPR1 and EXPR1+1, EXPR2 and EXPR2+1). After the calculations the result is in locations EXPR1 (LSB) and EXPR1+1 (MSB).

16 BIT MULTIPLICATION UNSIGNED INTEGER
EXPR := EXPR * EXPR2

```
                    EXPR1       EPZ   $80.1
                    EXPR2       EPZ   $82.3
                    SCRATCH     EPZ   $84.5

                                ORG   $A800

A800: A2 00         MUL         LDX   #0
A802: 86 84                     STX   SCRATCH
A804: 86 85                     STX   SCRATCH+1
A806: A0 10                     LDY   #16
A808: D0 0D                     BNE   MUL2 ALWAYS !!
A80A: 18            MUL1        CLC
A80B: A5 84                     LDA   SCRATCH
A80D: 65 82                     ADC   EXPR2
A80F: 85 84                     STA   SCRATCH
A811: A5 85                     LDA   SCRATCH+1
A813: 65 83                     ADC   EXPR2+1
A815: 85 85                     STA   SCRATCH+1
A817: 46 85         MUL2        LSR   SCRATCH+1
A819: 66 84                     ROR   SCRATCH
A81B: 66 81                     ROR   EXPR1+1
A81D: 66 80                     ROR   EXPR1
A81F: 88                        DEY
A820: 30 04                     BMI   MULRTS
A822: 90 F3                     BCC   MUL2
A824: B0 E4                     BCS   MUL1
A826: 60            MULRTS      RTS

PHYSICAL ENDADDRESS: $A827

*** NO WARNINGS

EXPR1       $80
EXPR2       $82
SCRATCH     $84
MUL         $A800       UNUSED
MUL1        $A80A
MUL2        $A817
MULRTS      $A826
```

1.2.4. 16-bit division

The division of two numbers actually is just the opposit of the multiplication. Therefore, you can see in the program below, that the divisor is subtracted and the dividend is shifted to the left rather than to the right. The memory locations used are the same as with the multiplication, except that locations SCRATCH and SCRATCH+1 are named REMAIN and REMAIN+1. This means the remainder of the division is stored in those locations.

16 BIT DIVISION UNSIGNED INTEGER
EXPR1 : = EXPR1 OVER EXPR2
REMAIN : = EXPR1 MOD EXPR2

```
                    EXPR1       EPZ   $80.1
                    EXPR2       EPZ   $82.3
                    REMAIN      EPZ   $84.5

                                ORG   $A800

A800: A2 00         DIV         LDX   #0
A802: 86 84                     STX   REMAIN
A804: 86 85                     STX   REMAIN+1
A806: A0 10                     LDY   #16
A808: 06 80         DIV1        ASL   EXPR1
A80A: 26 81                     ROL   EXPR1+1
A80C: 26 84                     ROL   REMAIN
A80E: 26 85                     ROL   REMAIN+1
A810: 38                        SEC
A811: A5 84                     LDA   REMAIN
A813: E5 82                     SBC   EXPR2
A815: AA                        TAX
A816: A5 85                     LDA   REMAIN+1
A818: E5 83                     SBC   EXPR2+1
A81A: 90 06                     BCC   DIV2
A81C: 86 84                     STX   REMAIN
A81E: 85 85                     STA   REMAIN+1
A820: E6 80                     INC   EXPR1
A822: 88            DIV2        DEY
A823: D0 E3                     BNE   DIV1
A825: 60                        RTS

PHYSICAL ENDADDRESS: $A826

*** NO WARNINGS

EXPR1       $80
EXPR2       $82
REMAIN      $84
DIV         $A800       UNUSED
DIV1        $A808
DIV2        $A822
```

STRINGOUTPUT

CHAPTER 2

2.1. Output of text

With most programs it is necessary to display text (menues etc.).

The following program allows you to display strings of any length at any location you desire. The output command can be located at any place within your program.

How does that program work ?

As you know the 6502 microprocessor uses its stack to store the return address if a JSR-command is to be executed. The number that is stored on the stack actually is the return-address minus one. The trick used in this program is, that the string to be printed is stored immediately after the JSR-command and the last character of the string is incremented by 128. The subroutine calculates the start address of tie string, using the number on the stack, and reads the string byte by byte, until it finds the byte which has been incremented by 128. The address of this byte now is stored on the stack and an RTScommand is executed. 8y doing so, the string is jumped and the command after it is executed.

STRINGOUTPUT FOR VARIOUS LENGTH

```
                    AUX         EPZ   $80
                    SCROUT      EQU   $F6A4

                                ORG   $A800

                    *     EXAMPLE
A800: 20 16 A8      EXAMPLE     JSR PRINT
A803: 54 48 49                  ASC \THIS IS AN EXAMPLE
A806: 53 20 49
A809: 53 20 41
A80C: 4E 20 45
A80F: 58 41 4D
A812: 50 4C C5
A815: 60                        RTS

                    *           THE VERY PRINTROUTINE

A816: 68            PRINT       PLA
A817: 85 80                     STA   AUX
A819: 68                        PLA
A81A: 85 81                     STA   AUX+1
A81C: A2 00                     LDX   #0
A81E: E6 80         PRINT1      INC   AUX
A820: D0 02                     BNE   *+4
A822: E6 81                     INC   AUX+1
A824: A1 80                     LDA   (AUX,X)
```

```
A826: 29 7E                 AND   #$7F
A828: 20 A4 F6              JSR   SCROUT
A82B: A2 00                 LDX   #0
A82D: A1 80                 LDA   (AUX,X)
A82F: 10 ED                 BPL   PRINTI
A831: A5 81                 LDA   AUX+1
A833: 48                    PHA
A834: A5 80                 LDA   AUX
A836: 48                    PHA
A837: 60                    RTS

PHYSICAL ADDRESS: $A838

*** NO WARNINGS

AUX         S80
SCROUT      $F6A4
EXAMPLE     $A800       UNUSED
PRINT       $A816
PRINT1      $A81E
```

INTRODUCTION TO CIO

CHAPTER 3

The CIO can handle up to 8 devices/files at the same time. This happens via so called Input Output Control Blocks (IOCB). This means that there are 8 IOCB's starting from $0340. Each of the IOCB's is 16 bytes long.

BLOCK #	ADDRESS
IOCB #0	$0340
IOCB #1	$0350
IOCB #2	$0360
IOCB #3	$0370
IOCB #4	$0380
IOCB #5	$0390
IOCB #6	$03A0
IOCB #7	$03B0

A single IOCB has the following internal scheme:

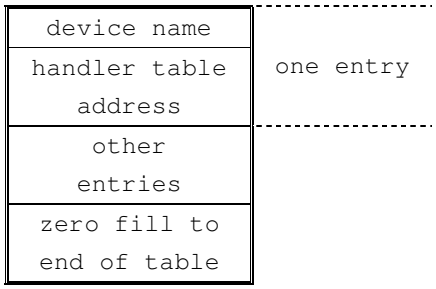
NAME	ADDRESS
ICHID	HANDLER ID
ICDNO	DEVICE NUMBER
ICCMD	COMMAND
ICSTA	STATUS
ICBAL ICBAH	BUFFERADR
ICPTL ICPTH	PUTADR
ICBLL ICBLH	BUFFERLEN
ICAX1	AUX1
ICAX2	AUX2
ICAX3	Remaining 4 bytes
ICAX4	
ICAX5	
ICAX6	

There are just a few locations which are important to the user:

- The commandbyte which contains the command to be executed by the CIO.
- The bufferaddress which contains the address of the actual databuffer.
  The bufferlength which contains the number of bytes to be transferred (rounded up to a variety of 128 bytes for the cassette device)
- And there are two auxiliaries which contain device-dependent information.

There are also locations which will be altered by CIO such as:

- The handler-ID is an offset to the devicetable. This table contains all device names and pointers to the device specific handlertable.



A handlertable looks like:

OPEN-1
CLOSE-1
GETBYTE-1
PUTBYTE-1
GETSTATUS-1
SPECIAL-1
JMP INIT & 00

The CIO is thus quite clear to the user. It is easy to add new devices by adding just 3 bytes to the devicetable and to make a specific handlertable for this device. You can also change the handlerpointer of an existing device and let point it to a new handler. Later we will describe how to add or change devices.

- The devicenumber shows us which subdevice is meant. (e.g. Disknumber or RS232 Channel).
- After calling CIO the status will be altered. A 1 means a successfull operation while a value greater than 128 means an error has occurred.
- PUTADR is used internally by the CIO
- If there have been less bytes transferred than desired, because of an EOL or an error, BUFLEN will contain the actual number of transferred bytes.

The standard CIO commands:

- OPEN opens a file.

Before execution the following IOCB locations have to be set:

COMMAND = $03

BUFFADR points to, device/filename specification (like C: or D: TEST. SRC) terminated by an EOL ($98)

AUX1 = OPEN-directionbits (read or write) plus devicedependent information.

AUX2 = devicedependent information.

After execution:

HANDLER ID = Index to the devicetable.

DEVICE NUMBER = number taken from device/f filename specification
STATUS = result of OPEN-Operation.
- CLOSE closes an open IOCB
   Before execution the following IOCB location has to be set:
COMMAND = $0C
   After execution: HANDLER ID = $FF
STATUS = result of CLOSE-operation
- GET CHARACTERS read byte aligned. EOL has no termination feature.
   Before execution the following IOCB locations have to be set:
COMMAND = $07
BUFFERADR = points to databuffer.
BUFFERLEN = contains number of characters to be read. If BUFFERLEN is equal to zero the 6502 A-register contains the data.
   After execution:
STATUS = result of GET CHARACTER-operation
BUFFERLEN = number of bytes read to the buffer. The value will always be equal before execution, only if EOF or an error occurred.
- PUT CHARACTERS write byte aligned
   Before execution the following IOCB locations have to be set:
COMMAND = $0B
BUFFERADR = points to the datab~ffer
BUFFERLEN = number of bytes to be put, if equal to zero the 6502 A-register has to contain the data.
   After execution:
STATUS = result of PUT CHARACTER-operation
GET RECORD characters are read to the databuffer until the buffer is full, or an EOL is read from the device/file.
Before execution the following IOCB locations have to be set:
COMMAND = $05
BUFFERADR = points to the databuffer.
BUFFERLEN = maximum of bytes to be read (Including EOL character)
After execution:
STATUS = result of the GET RECORDoperation
BUFFERLEN = number of bytes read to buf fer this may less then the maximum length.
- PUT RECORD characters are written to the device/file from the databuffer until the buffer is empty or an EOL is written. If the buffer is empty CIO will automatically send an EOL to the device/file.
Before execution the following IOCB locations have to be set:
COMMAND = $09
HUFFERADR = points to databuffer.
BUFFERLEN = maximum number of bytes in databuffer.
After execution:
STATUS = result of PUT RECORD-operation.

  In addition to the main-commands, there is also a GET STATUS ($0D) command, which obtains the status from the device/filecontroller and places these four bytes from location $02EA (DVSTAT). Commands greater than $0D are so called SPECIALS and devicehandler-dependent.
GET STATUS and SPECIALS have an implied OPEN-option. Thus the file will be automatically opened and closed if it wasn't already opened.

   How to link the CIO with machine language?

First we have to modify the IOCB before calling CIO.

The offset to the IOCB (IOCB# times 16) has to be in the X-register. The STATUS will be loaded in the Y-register of ter returning from CIO. It is not necessary to explicitly check the Y-register (Comparing with 128) because loading the status into the Y-register was the last instruction before leaving CIO with an RTS. We simply jump on the signflag (BMI or BPL). The sign flag is set if an error occurred. In the next section we will discuss it in more detail with an example.

How to read or write data in machine language?

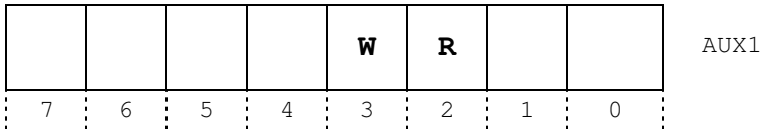
To describe the writing of data to a device/file we will take the cassettedevice as an example. We can also use any other device because CIO is very clear-cut (see introduction).

Before discussing the program, some conventions must be discussed.

The user has to put the address of his databuffer into the locations BUFFER ($80.1) and the bufferlength into the locations BUFLEN ($82.3). Then the program should be called as a subroutine. The description of this subroutine follows.

First we have to open the cassette, so we load the IOCB-offset in the X-register, store the OPEN-command in ICCMD, and let the BUFADR (ICBAL and ICBAH) point to the device/filename specification. We have to store the write-direction in ICAX1 and the tape-recordlength (128) in ICAX2, just call CIO ($E456). The Signflag indicates if an error occurred.

After a correct opening of the file for writing data, bit 3 is set because AUX1 contains a $08 (bit 2 is readbit).



ICCMD will be changed into the PUT CHARACTERS-command ($0B), BUFFADR points to the User-Databuffer (contents of BUFFER) and BUFLEN (ICBLL and ICBLH) will contain the number of bytes to write (the user stores this value BUFLEN ($82. 3)). Next CIO will be called, and after successfull operation, the file will be closed (ICCMD=$0C).

If, during any of these three CIO-calls, an error occurs, the file will be closed and both the ACCUMULATOR and Y-register will contain the STATUS (errorcode).

By changing the string at CFILE in for instance “D:TEST.TST” the program will write the buffer to the specified diskfile.

The second listing shows you a program that reads from a device, only two bytes are different, so the program is selfexplaining.

WRITE BUFFER TO CASSETTE

```
                BUFFER      EPZ   $80.1
                BUFLEN      EPZ   $82.3 - BUFLEN ROUNDED
                                        UP TO 128 BYTES
                ICCMD       EQU   $0342
                ICBAL       EQU   $0344
                ICBAH       EQU   $0345
                ICBLL       EQU   $0348
                ICBLH       EQU   $0349
                ICAX1       EQU   $034A
                ICAX2       EQU   $034B

                OPEN        EQU   3
                PUTCHR      EQU   11
                CLOSE       EQU   2

                WMODE       EQU   8
                RECL        EQU   128

                CIO         EQU   $E456

                EOL         EQU   $9B

                IOCBNUM     EQU   1

                            ORG   $A800

                *           OPEN FILE

A800: A2 10                 LDX   #IOCBNUM*16
A802: A9 03                 LDA   #OPEN
A804: 9D 42 03              STA   ICCMD,X
A807: A9 08                 LDA   #WMODE
A809: 9D 4A 03              STA   ICAXI,X
A80C: A9 80                 LDA   #RECL
A80E: 9D 4B 03              STA   ICAX2,X
A811: A9 56                 LDA   #CFILE:L
A813: 9D 44 03              STA   ICBAL,X
A816: A9 A8                 LDA   #CFILE:H
A818: 9D 45 03              STA   ICBAH,X
A81B: 20 56 E4              JSR   CIO
A81E: 30 29                 BMI   CERR

                * PUT BUFFER IN RECORDS TO CASSETTE

A820: A9 0B                 LDA   #PUTCHR
A822: 9D 42 03              STA   ICCMD,X
A825: A5 80                 LDA   BUFFER
A827: 9D 44 03              STA   ICBAL,X
A82A: A5 81                 LDA   BUFFER+1
A82C: 9D 45 03              STA   ICBAH,X
A82F: A5 82                 LDA   BUFLEN
```

```
A831: 9D 48 03              STA   ICBLL,X
A834: A5 83                 LDA   BUFLEN+1
A836: 9D 49 03              STA   ICBLH, X
A839: 20 56 E4              JSR   CIO
A83C: 30 08                 BMI   CERR

                *           CLOSE CASSETTE FILE

A83E: A9 0C                 LDA   #CLOSE
A840: 9D 42 03              STA   ICCMD,X
A843: 20 56 E4              JSR   CIO
A846: 30 01                 BMI   CERR

                *           RETURN TO SUPERVISOR

A848: 60                    RTS

                * RETURN WITH ERRORCODE IN ACCUMULATOR

A849: 98          CERR      TYA
A84A: 48                    PHA
A84B: A9 0C                 LDA   #CLOSE
A84D: 9D 42 03              STA   ICCMD,X
A850: 20 56 E4              JSR   CIO
A853: 68                    PLA
A854: A8                    TAY
A855: 60                    RTS
A856: 43 3A       CFILE     ASC   "C:"
A858: 9B          DFB       EOL

PHYSICAL ENDADDRESS: $A859

*** NO WARNINGS

BUFFER      $80
BUFLEN      $82
ICCMD       $0342
ICBAL       $0344
ICBAH       $0345
ICBLL       $0348
ICBLH       $0349
ICAX1       $034A
ICAX2       $034B
OPEN        $03
PUTCHR      $0B
CLOSE       $0C
WMODE       $08
RECL        $80
CIO         $E456
EOL         $9B
IOCBNUM     $01
CERR        $A849
CFILE       $A856
```

READ BUFFER FROM CASSETTE

```
                BUFFER      EPZ   $80.1
                BUFLEN      EPZ   $82.3   BUFLEN ROUNDED
                                          UP TO 128 BYTES
                ICCMD       EQU   $0342
                ICBAL       EQU   $0344
                ICBAH       EQU   $0345
                ICBLL       EQU   $0348
                ICBLH       EQU   $0349
                ICAX1       EQU   $034A
                ICAX2       EQU   $034B

                OPEN        EQU   3
                GETCHR      EQU   7
                CLOSE       EQU   12

                RMODE       EQU   4
                RECL        EQU   128

                CIO         EQU   $E456

                EOL         EQU   $9B

                IOCBNUM     EQU   1

                            ORG   $A800

                *           OPEN FILE

A800: A2 10                 LDX   #IOCBNUM*16
A802: A9 03                 LDA   #OPEN
A804: 9D 42 03              STA   ICCMD,X
A807: A9 04                 LDA   #RMODE
A809: 9D 4A 03              STA   ICAXI,X
A80C: A9 80                 LDA   #RECL
A80E: 9D 4B 03              STA   ICAX2,X
A811: A9 56                 LDA   #CFILE:L
A813: 9D 44 03              STA   ICBAL,X
A816: A9 A8                 LDA   #CFILE:H
A818: 9D 45 03              STA   ICBAH,X
A81B: 20 56 E4              JSR   CIO
A81E: 30 29                 BMI   CERR

                *           GET BUFFER IN RECORDS FROM CASSETTE

A820: A9 07                 LDA   #GETCHR
A822: 9D 42 03              STA   ICCMD,X
A825: A5 80                 LDA   BUFFER
A827: 9D 44 03              STA   ICBAL,X
A82A: A5 81                 LDA   BUFFER+1
A82C: 9D 45 03              STA   ICBAH,X
A82F: A5 82                 LDA   BUFLEN
```

```
A831: 9D 48 03              STA   ICBLL, X
A834: A5 83                 LDA   BUFLEN+1
A836: 9D 49 03              STA   ICBLH,X
A839: 20 56 E4              JSR   CIO
A83C: 30 0B                 BMI   CERR

                *           CLOSE CASSETTE FILE

A83E: A9 0C                 LDA   #CLOSE
A840: 9D 42 03              STA   ICCMD,X
A843: 20 56 E4              JSR   CIO
A846: 30 01                 BMI   CERR

                *           RETURN TO SUPERVISOR

A848: 60                    RTS

                *           RETURN WITH ERRORCODE ACCUMULATOR

A849: 98          CERR      TYA
A84A: 48                    PHA
A84B: A9 0C                 LDA   #CLOSE
A84D: 9D 42 03              STA   ICCMD,X
A850: 20 56 E4              JSR   CIO
A853: 68                    PLA
A854: A8                    TAY
A855: 60                    RTS
A856: 43 3A       CFILE     ASC   "C:"
A858: 9B                    DFB   EOL

PHYSICAL ENDADDRESS: $A859

*** NO WARNINGS

BUFFER      $80
BUFLEN      $82
ICCMD       $0342
ICBAL       $0344
ICBAH       $0345
ICBLL       $0348
ICBLH       $0349
ICAX1       $034A
ICAX2       $034B
OPEN        $03
GETCHR      $07
CLOSE       $0C
RMODE       $04
RECL        $80
CIO         $E456
EOL         $9B
IOCBNUM     $01
CERR        $A849
FILE        $A856
```

INTRODUCTION TO THE DISK CONTROLLER

CHAPTER 4

We already know how to handle any device/file via CIO, including handle a diskfile. Included on a disk is also a sector-IO which allows you to address a single sector for a read or write handling. Sector-IO doesn't need any file on the disk. The disk has only to be formatted.

A floppy disk with the ATARI drive has 720 sectors and each of them is fully addressable.

How does the sector-IO function?

The disk controller has a simplistic design containing a single IOCB like Data Control Block (DCB). This DCB is described in the following scheme.

DCBSBI	Serial bus ID
DCBDRV	Disk drive #
DCBCMD	Command
DCBSTA	IO Status
DCBUF LO DCBUF HI	Buffer IO address
DCBTO LO DCBTO HI	Timeout counter
DCBCNT LO DCBCNT HI	IO Buffer length
DCBSEC LO DCBSEC HI	IO Sector number

- Instead of a handler-ID there is a BUS ID (DCBSBI) to address a particular diskdrive via the Serial-Bus of the ATARI.
- Also a logical drivenumber (DCBDRV)
- A commandbyte (DCBCMD), which is similar to an IOCB, and 5 commands for sector-IO, which will be described later.
- The statusbyte for error detection after, and data-direction previous to execution of the command ($80 is write, $40 is read).
- The DCBBUF locations (L and H) to point to the databuffer.
- DCBTO (L and H) is a special word containing the maximum time for executing a command, so called timeout.
- DCBCNT (L and H) is a device specific word which contains the sector length (128 for the 810-drive or 256 for the double density drives).
- DCBSEC (L & H) contains the sector number to do IO on.

Prior to executing any DCB-command, the following DCB-entries must be set.
DCBSBI has to contain the bus-ID of the drive:
DRIVE 1 = $31 = '1
DRIVE 2 = $32 = '2
DRIVE 3 = $33 = '3
URIVE 4 = $34 = '4

DCBDRV has to contain the logical drive number (1..4).
DCBTO the timeout (normally 15; lowbyte=$0F highbyte=$00).

- READ SECTOR reads one sector specified by the user
  DCBCMD = $52 = 'R
  DCBBUF = points to databuffer
  DCBCNT = contains sector length
  DCBSEC = number of sector to read

  After execution:
  DCBSTAT = result of HEAD SECTUR-operation

- PUT SECTOR writes one sector specified by the user without verify.
  DCBCMD = $50 = 'P
  DCBBUE' = points to databufter
  DCBSEC = number of sector to write

  After execution:
  DCBSTAT = result of PUT SECTOR-operation

- WRITE SECTOR writes one sector specified by the user with automatic verify.
  DCBCMD = $57 = 'W
  Further like PUT SECTOR.

- STATUS REQUEST obtains the status from the specified drive.
  DCBCMD = $53 = 'S

  After execution:
  DCBSTAT = result of STATUS REQUEST operation
  The drive outputs four bytes end the controller puts them to $02EA (DVSTAT).
- FORMAT formats the specified disk.
  DCBCMD = $21 = '!

DCBTO = has to be larger than 15 due to more time taken by the FORMAT-command. You can ignore the error, but this will be risky.

After execution:
DCBSTAT = result of the FORMAT-operation.

How is the disk controller invoked?

Because the disk controller is resident, this is a simple process. You don't have to load DOS, nor anything similar. You just have to call the SIO (Serial IO $E459) instead of the CIO. Therefore, you can see that it is quite easy to link the Diskcontroller with machine language.

How to write a sector to disk

The first program writes a specified sector from a buffer to diskdrive#1. There are a few conventions to call this program as subroutine. The user has to put the buffer address into the pointer locations labelled BUFFER and the sector number into the locations labelled SECTR. The program also needs a RETRY-location, to serve as a counter so the program is able to determine how of ten it will retry the IO.

The next paragraph describes the subroutine.

At first we built the DCB, special we move a $80 (BIT 3 the write bit is set) to DCBSTA and we retry the IO 4 times. SIO does, as well as CIO, load the STATUS into the Y-register so you only have to check the signflag again. After an error occurence we decrement the retry value and set DCBSTA again, then try again.

By using this program, you only have to look at the signflag after returning for error detection (signflag TRUE means error, otherwise success).

The second program reads a sector instead of writing it. The only two bytes which are different are the DCBCMD and the DCBSTA ($90 for read) .

WRITE A SECTOR TO DISK

```
SECTR       EQU   $80.1
BUFFER      EQU   $82.3
RETRY       EQU   $84

DCBSBI      EQU   $0300
DCBDRV      EQU   $0301
DCBCMD      EQU   $0302
DCBSTA      EQU   $0303
DCBBUF      EQU   $0304
DCBTO       EQU   $0306
DCBCNT      EQU   $0308
DCBSEC      EQU   $030A

SIO         EQU   $E459

            ORG   $A80 0
```

```
A800: A5 82       WRITSECT    LDA   BUFFER
A802: 8D 04 03                STA   DCBSUF
A805: A5 83                   LDA   BUFFER+1
A807: 8D 05 03                STA   DCBBUF+1
A80A: A5 80                   LDA   SECTR
A80C: 8D 0A 03                STA   DCBSEC
A80F: A5 81                   LDA   SECTR+1
A811: 8D 0B 03                STA   DCBS EC+1
A814: A9 57                   LDA   'W          REPLACE "W" BY A "P" IF
A816: 8D 02 03                STA   DCBCMD       YOU WANT IT FAST
A819: A9 80                   LDA   #$80
A81B: 8D 03 03                STA   DCBSTA
A81E: A9 31                   LDA   '1
A820: 8D 00 03                STA   DCBSBI
A823: A9 01                   LDA   #1
A825: 8D 01 03                STA   DCBDRV
A828: A9 0F                   LDA   #15
A82A: 8D 06 03                STA   DCBTO
A82D: A9 04                   LDA   #4
A82F: 85 84                   STA   RETRY
A831: A9 80                   LDA   # 12 8
A833: 8D 08 03                STA   DCBCNT
A836: A9 00                   LDA   #0
A838: 8D 09 03                STA   DCBCNT+1
A83B: 20 59 E4    JMPSIO      JSR   SIO
A83E: 10 0C                   BPL   WRITEND
A840: C6 84                   DEC   RETRY
A842: 30 08                   BMI   WRITEND
A844: A2 80                   LDX   #$80
A846: 8E 03 03                STX   DCBSTA
A849: 4C 3B A8                JMP   JMPSIO
A84C: AC 03 03    WRITEND     LDY   DCBSTA
A84F: 60                      RTS

PHYSICAL ENDADDRESS: $A850

*** NO WARNINGS

SECTR       $80                     BUFFER      $82
RETRY       $84                     DCBSBI      $0300
DCBDRV      $0301                   DCBCMD      $0302
DCBSTA      $0303                   DCBBUF      $0304
DCBTO       $0306                   DCBCNT      $0308
DCBSEC      $030A                   SIO         $E459
WRITSECT    $A800       UNUSED      JMPSIO      $A83B
WRITEND     $A84C
```

READ A SECTOR FROM DISK

```
                    SECTR       EQU   $80.1
                    BUFFER      EQU   $82.3
                    RETRY       EQU   $84

                    DCBSBI      EQU   $0300
                    DCBDRV      EQU   $0301
                    DCBCMD      EQU   $0302
                    DCBSTA      EQU   $0303
                    DCBBUF      EQU   $0304
                    DCBTO       EQU   $0306
                    DCBCNT      EQU   $0308
                    DCBSEC      EQU   $030A

                    SIO         EQU   $E459

                                ORG   $A800

A800: A5 82         READSECT    LDA   BUFFER
A802: 8D 04 03                  STA   DCBSUF
A805: A5 83                     LDA   BUFFER+1
A807: 8D 05 03                  STA   DCBBUF+1
A80A: A5 80                     LDA   SECTR
A80C: 8D 0A 03                  STA   DCBSEC
A80F: A5 81                     LDA   SECTR+1
A811: 8D 0B 03                  STA   DCBS EC+1
A814: A9 52                     LDA   'R
A816: 8D 02 03                  STA   DCBCMD
A819: A9 40                     LDA   #40
A81B: 8D 03 03                  STA   DCBSTA
A81E: A9 31                     LDA   '1
A820: 8D 00 03                  STA   DCBSBI
A823: A9 01                     LDA   #1
A825: 8D 01 03                  STA   DCBDRV
A828: A9 0F                     LDA   #15
A82A: 8D 06 03                  STA   DCBTO
A82D: A9 04                     LDA   #4
A82F: 85 84                     STA   RETRY
A831: A9 80                     LDA   #128
A833: 8D 08 03                  STA   DCBCNT
A836: A9 00                     LDA   #0
A838: 8D 09 03                  STA   DCBCNT+1
A83B: 20 59 E4      JMPSIO      JSR   SIO
A83E: 10 0C                     BPL   READEND
A840: C6 84                     DEC   RETRY
A842: 30 08                     BMI   READEND
```

```
A844: A2 80                 LDX   #$80
A846: 8E 03 03              STX   DCBSTA
A849: 4C 3B A8              JMP   JMPSIO
A84C: AC 03 03    READEND   LDY   DCBSTA
A84F: 60                    RTS

PHYSICAL ENDADDRESS: $A850

*** NO WARNINGS

SECTR       $80
BUFFER      $82
RETRY       $84
DCBSBI      $0300
DCBDRV      $0301
DCBCMD      $0302
DCBSTA      $0303
DCBBUF      $0304
DCBTO       $0306
DCBCNT      $0308
DCBSEC      $030A
SIO         $E459
READSECT    $A800       UNUSED
JMPSIO      $A83B
READEND     $A84C
```

HOW TO MAKE A BOOTABLE PROGRAM

CHAPTER 5

What is a bootable program ?

A bootable program is a program which will be automatically loaded at powering up the ATARI, and directly after loading be executed.

A bootable program needs a header with specific information about the program, such as the length and the start address. The header of a bootable program looks like the following scheme:

# Byte	Destination
1	unused (0)
2	# of 128bytes sectors
3	Store
4	Address
5	Initialization
6	Address
7	boot
:	continuation
:	code

- The first byte is unused, and should equa1 zero.
- The second byte contains the length of the program, in records (128 bytes length), (rounded up).
- The next word contains the store address of the program.
- The last word contains the initialization-address of the program. This vector will be transferred tb the CASINI-vector ($02.3).

After these 6 bytes there has to be the boot continuation code. This is a short program, the OS will jump to directly after loading. This program can continue the boot process (multistage boot) or stop the cassette by the following sequence

```
LDA   #$3C
STA   PACTL ; $D302
```

The program then allows the DUSVEC ($OA. e) to point to the start address of the program. It is also possible, to store in MEMLO ($02E7. 8), the first unused memory address. The continuation code must return to the OS with C=0 (Carry clear). Now OS jumps via DOSVEC to the application program.

So far we know what a bootable cassette looks like, but how do we create such a bootable tape?

If there is a program, we only have to put the header in front of it (including the continuation code) and to save it as nornml data on the tape. We can use the later described program to write the contents of a buffer on the tape or the boot generator.

If the program is saved, we can put the tape in the recorder, press the yellow START-key, power on the ATARI and press RETURN. Now the program on the tape will be booted.

The next listing shows us the general outline of a bootable program.

GENERAL OUTLINE OF AN BOOTABLE PROGRAM

```
PROGRAM START

            ORG   $A800       (OR AN OTHER)

*           THE BOOTHEADER

            PST   DFB 0       SHOULD BE 0
            DFW   PND-PST+127/128 # OF RECORDS
            DFW   PST         STORE ADDRESS
            DFW   INIT        INITALIZATION ADDRESS

*           THE BOOT CONTINUATION CODE

            LDA   #$3C
            STA   PACTL       STOP CASSETTE MOTOR
            LDA   #PND:L
            STA   MEMLO
            LDA   #PND:H
            STA   MEMLO+1     SET MEMLO TO END OP PROGRAM
            LDA   #RESTART:L
            STA   DOSVEC
            LDA   #RESTART:H
            STA   DOSVEC+1    SET RESTART VECTOR IN OOSVECTOR
            CLC
            RTS               RETURN WITH C=0 (SUCCESSFULL BOOT)

*           INITIALIZATION ADDRESS

INIT        RTS               RTS IS THE MINIMUM PROGRAM

*           THE MAIN PROGRAM

RESTART     EQU *

THE MAIN PROGRAM ENDS   HERE

PND         EQU *             NEXT FREE LOCATION
```

How to make a bootable disk?

Making a bootable disk is in fact the same as for the cassette. The only exceptions are as follows.

The program (including the header) must be stored up from sector one. The boot continuation code doesn't need to switch off anything such as the cassette motor.

How to create a bootable disk?

This is only a bit more complicated than the cassette version. We need our writesector program we described earlier. Then we have to write, sector by sector, to disk. You can also make a bootable cassette first and then copy it directly to disk with the later discussed program.

HOW TO MAKE A BOOTABLE CARTRIDGE

CHAPTER 6

Preparing the program.

Most of the games and some other programs written in machine language are stored in a cartridge. Booting a program, the OS recognizes the cartridge and starts the program.

What do you have to do when you want to make a bootable cartridge of your own program ?

As an example we will make a cartridge with a program for testing the memory. The bit pattern

```
10101010 = $AA
01010101 = $55
00000000 = $00
11111111 = $FF
```

is written in every memory location starting above the hardware stack at address $200. First the content is saved, then the bit pattern is written into and read from the memory location. If there is any difference in writing and reading the program prints an error message : ERROR IN <~R> . Then the program waits in an endless loop. If the error message is ERROR IN A000, the RAM is ok because $A000 is the first address of the ROM in the left cartridge.

The address range for the left cartridge ranges from $A000 to $BFFF and $8000 to $9FFF for the right cartridge. As starting address for our memory test program we choose $BF00. This is the last page of the left cartridge. The software for the EPROM burner is also stored in a cartridge. Therefore the object code generated by the assembler is stored at $9000.

Like a bootable program the cartridge has a header. The following scheme shows the outline of this cartridge header.

CARTRIDGE START ADDRESS	$BFFA or $9FFA
00	-
OPTION BYTE	-
CARTRIDGE INIT ADDRESS	$BFFF or $9FFF

The header for the right cartridge starts at $9FFA, for the left cartridge (the more important for us) at $BFFA.

- The first two bytes contain the start address of the cartridge.
- The third byte is the cartridge-ID. It shows the OS that a cartridge has been inserted. It must be 00.
- The fourth byte is the option-byte. This byte has the following options:

  BIT 0 = 0 don't allow diskboot 1 allow diskboot

  BIT 2 = 0 only initialize the cartridge 1 initialize and start the cartridge

BIT 7 = 0 Cartridge is not a diagnostic cartridge
1 Cartridge is a diagnostic cartridge
before OS is initialized the cartridge takes control

- The last two bytes contain the cartridge initialization address.

The initialization address is the starting address of a program part which is executed in advance of the main program. If there is no such a program this address must be the address of an RTS instruction. In our example the low byte of the starting address $BF00 is stored in location $BFFA, the high byte in location $BFFB.

The option byte in location $BFFD is 04.

The program in the cartridge is initialized and started, but there is no disk boot. The initializing address is $BF63, an RTS instruction within the program.

After assembling and storing the object code the burning of an EPROM can start.

GENERAL OUTLINE OF A CARTRIDGE

```
*           THE CARTRIDGE START (LEFT CARTRIDGE)

            ORG   $A000       $8000 FOR RIGHT CARTRIDGE
*           THE INITIALIZATION ADDRESS

INIT        RTS

*           THE MAIN PROGRAM

RESTART     EQU   *

*           THE CARTRIDGE HEADER

            ORG   $BFFA       $9FFA FOR RIGHT CARTRIDGE

            DFW   RESTART
            DFB   0           THE CARTRIDGE ID SHOULD BE ZERO
            DFB   OPTIONS     THE OPTION BYTE
            DFW   INIT        THE CARTRIDGE INITIALIZATION ADDRESS
```

Sample program for a cartridge:

MEMORY TEST

```
                  AUXE        EPZ   $FE
                  TEST        EPZ   $F0
                  OUTCH       EQU   $F6A4
                              ORG   $BF00,$9000

BF00: A9 7D       START       LDA   #$7D
BF02: 20 A4 F6                JSR   OUTCH
BF05: 20 64 BF                JSR   MESS
BF08: 4D 45 4D                ASC   \MEMORY TEST\
BF0B: 4F 52 59
BF0E: 205445
BF11: 53 D4
BF13: A0 00                   LDY   #00
BF15: 84 F0                   STY   TEST
BF17: A9 02                   LDA   X02
BF19: 85 F1                   STA   TEST+1
BF1B: B1 F0       TEST1       LDA   (TEST),Y
BF1D: 85 F2                   STA   TEST+2
BF1F: A9 AA                   LDA   #$AA
BF21: 20 59 BF                JSR   TST
BF24: A9 55                   LDA   #$55
BF26: 20 59 BF                JSR   TST
BF29: A9 00                   LDA   #00
BF2B: 20 59 BF                JSR   TST
BF2E: A9 FF                   LDA   #$FF
BF30: 20 59 BF                JSR   TST
BF33: A5 F2                   LDA   TEST+2
BF35: 91 F0                   STA   (TEST),Y
BF37: E6 F0                   INC   TEST
BF39: D0 E0                   BNE   TESTl
BF3B: E6 F1                   INC   TEST+1
BF3D: 18                      CLC
BF3E: 90 DB                   BCC   TEST1
BF40: 20 64 BF    FIN         JSR   MESS
BF43: 45 52 52                ASC   \ERROR IN \
BF46: 4F 52 20
aF49: 49 4E A0
BF4C: A5 F1                   LDA   TEST+1
BF4E: 20 86 BF                JSR   PRTBYT
BF51: A5F0                    LDA   TEST
RF53: 20 86 BF                JSR   PRTBYT
BF56: 4C 56 BF    FINI        JMP   FINI
```

```
BF59: 85 F3       TST         STA   TEST+3
BF5B: 91 F0                   STA   (TEST),Y
BF5D: B1 F0                   LDA   (TEST),Y
BF5F: C5 F3                   CMP   TEST+3
BF61: D0 D0                   BNE   FIN
BF63: 60          FRTS        RTS
BF64: 68          MESS        PLA
BF65: 85 FE       STA         AUXE
BF67: 68                      PLA
BF68: 85 FF                   STA   AUXE+1
BF6A: A2 00                   LDX   #0
BF6C: E6 FE       MS1         INC   AUXE
BF6E: D0 02                   BNE   *+4
BF70: D6 FF                   INC   AUXE+1
BF72: A1 FE                   LDA   (AUXE,X)
BF74: 29 7F                   AND   #$7F
BF76: 20 A4 F6                JSR   OUTCH
BF79: A2 00                   LDX   #0
BF7B: Al FE                   LDA   (AUXE,X)
BF7D: l0 ED                   BPL   MS1
BF7F: A5 FF                   LDA   AUXE+1
BF81: 48                      PHA
BF82: A5 FE                   LDA   AUXE
BF84: 48                      PHA
BF85: 60                      RTS
BF86: 48          PPRTBYT     PHA
BF87: 4A                      LSR
BF88: 4A                      LSR
BF89: 4A                      LSR
BF8A: 4A                      LSR
BF8B: 20 91 BF                JSR   HEX21
BF8E: 68                      PLA
BF8F: 29 0F                   AND   #$OF
BF91: C9 0A       HEX21       CMP   #9+1
BF93: B0 04                   BCS   BUCHST
BF95: 09 30                   ORA   '0
BF97: D0 03                   BNE   HEXOUT
BF99: 18          BUCHSST     CLC
BF9A: 69 37                   ADC:  'A-10
BF9C: 4C A4 F6    HEXOUT      JMP   OUTCH

                              ORG   $BFFA, $90FA
BFFA: 00 BF                   DFW   START
BFFC: 00                      DFB   00
BFFD: 04                      DFB   04
BFFE: 63 BF                   DFW   DRTS

PHYSICAL ENDADDRESS: $9100
*** NO WARNINGS
```

EPROMBURNER FOR THE ATARI 800 / 400

With this epromburner you can burn your EPROMS. It is possible to burn four different types. The four types are the 2532(4k), the 2732(k),the 2516(2k) and the 2716 (2k). The burner uses the game ports 1, 2 and 3.

1) THE HARDWARE

The circuit of the epromburner is shown in FIG.1.The data for the burner is exchanged via game port 1 and 2. The control signals are provided by game port 3. The addresses are decoded by two 7 bit counters 4024. The physical addresses for the EPROMS are always in the range of 0000 to 07FF for 2k and 0000 to 0FFF for 4k. This counter is reset by a signal, decoded from PB0 and PB1 via the 74LS139. PB2 is used to decide if a 2532, or a 2716 has to be burned.

Not all signals for the different types of EPROMS are switched by software. A three pole, double throw switch is used to switch between the different types. The software tells you when you have to set the switch into the correct position. For burning, you need a burn voltage of 25 Vo1ts. This voltage is converted from the 5 Volts of the game port to 28 Volts by the DCDC converter DCP528. This voltage is limited to 25 Volts by two Zener diodes in serie (ZN24 and ZN1). Three universal NPN transistors are used to switch between low level voltages and the high level of the burning voltage.

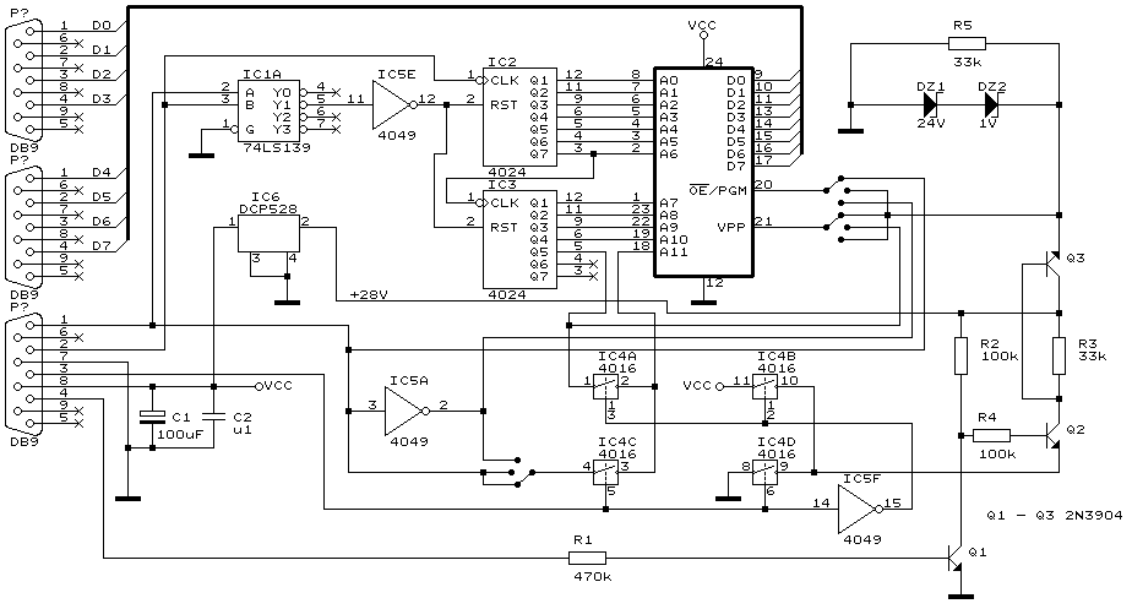


Fig.1 Eprom burner schematic

3) THE SOFTWARE

The software for the burner is completely written in machine code. It comes on a bootable diskette. To load the program, insert the disk and REMOVE ALL CARTRIDGES. Turn on the disk drive and the ATARI. After a short moment, you will see the first menu:

```
WHICH EPROM DO YOU WANT TO BURN?


              A)   2532
              B)   2732
              C)   2716, 2516

              WHAT:
```

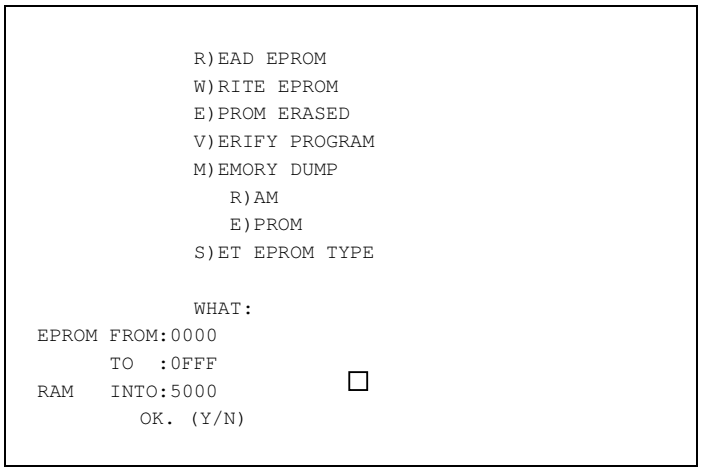
You are asked what type of EPROM you want to burn. After typing the apriopriate character, you get the message to set the switch to the correct position and insert the EPROM. This is shown in the following example:

```
WHICH EPROM DO YOU WANT TO BURN?


              D)   2532
              E)   2732
              F)   2716, 2516

              WHAT:
SET SWITCH TO POSITION 2532
NOW INSERT EPROM
 PRESS SPACE BAR
```

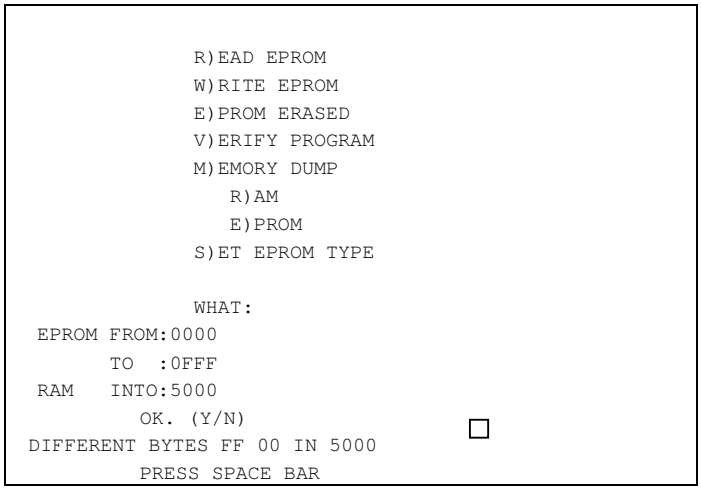
Then, pressing the space bar, you see the main menue:

```
              R)EAD EPROM
              W)RITE EPROM
              E)PROM ERASED
              V)ERIFY PROGRAM
              M)EMORY DUMP
                 R)AM
                 E)PROM
              S)ET EPROM TYPE

              WHAT:
```

First we want to R)EAD an EPROM. Type R and then the addresses FROM and TO. The physical addresses of the EPROM are always in range between 0000 and 0FFF. You can read the whole EPROM or only a part of it. Next you to type the address INTO which the content of the EPROM is read. All addresses which are not used by the system or the burner software (A800 to AFFF) are accessible. By typing Y after the question OK (Y/N), the program is loaded. There is a very important key, X key. This key cancels tlhe input and leads back to the main menue. An example of reading an EPROM is shown in the next figure:



To verify that the content of the RAM is identical the content of the EPROM, type V. After specifing addresses for the EPROM and the RAM and typing Y, the contents are compared. If there are any differences you get an error message, such as the following:



You may then make a memory dump. Type M for M)EMORY, either R for R)AM or E for E)PROM, and the address range. There is a slight difference in memory dumps. With the memory dump of RAM, , the bytes are printed, if a is possible, as ASCII characters.

Burning an EPROM begins by testing as to whether or not the EPROM is erased in the address range you want to burn. Type E and the address range. You will get the message EPROM ERASED when the assigned address range has been erased, or the message EPROM NOT ERASED IN CELL NNN.

For writing the EPROM, type W, the address range in RAM, and the starting address in EPROM. After hitting Y, you have to wait two minutes for burning 2k and four minutes for burning 4k. Don't get angry, the program will stop. After burning one cell the program does an automatic verify. If there is a difference you receive the error message EPROM NOT PROGRAMMED IN CELL NNN and the burning stops. Otherwise if all goes well the message EPROM PROGRAMMED is printed.

For changig the type of EPROM you want to burn, type S. The first menue is shown and you can begin a new burning procedure.

PARTS LIST.

IC1	74LS139
IC2,IC3	4024
IC4	4016
IC5	4049
T1,T2,T3	UNIVERSAL NPN TRANSISTOR 30V, 0.3W (2N3390 - 2N3399)
R1	470K RESISTOR
R2,R3	10K RESISTOR
R4,R5	33K RESISTOR
Z1	1V ZENER DIODE
Z2	24V ZENER DIODE
M1	DCP528 DCDC CONVERTER ELPAC POWER SYSTEMS
C1,C2	100nF CAPACITOR
C3	10uF TANTAL CAPACITOR
S1	3P2T SWITCH
1	24PIN TEXTOOL SOCKET
3	14PIN IC SOCKET
2	16PIN IC SOCKET
3	FEMALE PLUGS, ATARI GAME CONNECTORS

5) STEP BY STEP ASSEMBLING.

1. Insert and solder sockets.
* Component side shows the text EPROMBURNER.
2. Insert and solder resistors.
3. Insert and solder Zener diodes.
* The anodes are closest to the transistors.
4. Insert and solder transistors.
5. Insert and solder capacitors.
* The + pole of the tantal is marked.
6. Mount the DCDC converter module.
7. Turn the board to the soldering side.
8. Insert from this side the TEXTOOLL socket.
* The knob should be in the upper right corner.
* Solder the socket.
9. Make the connections on the switch. (FIG.5)
* Connect switch and board via a 7 lead flatband cable.
10. Connect the plugs to the board. (FIG.5)
11. Insert the integrated circuits. (FIG.2)
12. Turn off the ATARI. Insert the plugs.

* Insert the diskette and turn on the ATARI.

HEXDUMP of the EPROM BURNER software

```
A800  2076A9204CA82078      v) L( x
A808  A8006885EE6885EF     (@hEnhEo
A810  A200E6EED002E6EF     "@fnPBfo
A818  A1EE297F20A4F6A2     !n)  $v"
A820  00A1EE10EDA5EF48     @!nPm%oH
A828  A5EE4860A5FD2940     %nH'% )@
A830  F006A5FE0901D004     pF% IAPD
A838  A5FE290E8D01D348     % )NMASH
A840  68AD00D348A5FE8D     h-@SH% M
A848  01D36860A90085F0     ASh')@Ep
A850  85F185F8A9308D03     EqEx)0NC
A858  D3A90F8D01D385F5     S)OMASEu
A860  A9348D03D3A9FF85     )4MCS) E
A868  F4A9B085F9A9028D     t)0Ey)BM
A870  01D360A99B4CA4F6     AS')[L$v
A878  A97D20A4F6A90585     )  $v)EE
A880  54A90A8555A90085     T)JEU)@E
A888  56200AA852294541     V J(R)EA
A890  44204550524FCD20     D EPROM
A898  73A8A90A8555200A     s()JEU J
A8A0  A857295249544520     (W)RITE
A8A8  4550524FCD2073A8     EPROM s(
A8B0  A90A8555200AA845     )JEU J(E
A8B8  2950524F4D204552     )PROM ER
A8C0  415345C42073A8A9     ASED s()
A8C8  0A8555200AA85629     JEU J(V)
A8D0  4552494659205052     ERIFY PR
A8D8  4F475241CD2073A8     OGRAM s(
A8E0  A90A8555200AA84D     )JEU J(M
A8E8  29454D4F52592044     )EMORY D
A8F0  554DD02073A8A90D     UMP s()M
A8F8  8555200AA8522941     EU J(R)A
A900  CD2073A8A90D8555     M s()MEU
A908  200AA8452950524F      J(E)PRO
A910  CD2073A8A90A8555     M s()JEU
A918  200AA85329455420      J(S)ET
A920  4550524F4D205459     EPROM TY
A928  50C52073A82073A8     PE s( s(
A930  A90A8555200AA857     )JEU J(W
A938  484154BA20F0AE48     HAT: p.H
A940  20A4F668C952D003      $vhIRPC
```

```
A948  4C30ACC957D0034C     L0,IWPCL
A950  10ADC945D0034C8B     P-IEPCLK
A958  ACC956D0034C2DAF     ,IVPCL-/
A960  C953D0034C76A9C9     ISPCLv)I
A968  4DD0034CFBADA9FD     MPCL{-)
A970  20A4F66C0A00A97D      Sv1J@)
A978  20~A4F62073A8200A     $v s( J
A980  A857484943482045     (WHICH E
A988  50524F4D20444F20     PROM DO
A990  594F552057414E54     YOU WANT
A998  20544F204255524E      TO BURN
A9A0  20BFA9088554A90A      ?)HET)J
A9A8  8555200AA8412920     EU J(A)
A9B0  323533B22073A8A9     2532 s()
A9B8  0A8555200AA84229     JEU J(B)
A9C0  20323733B22073A8      2732 s(
A9C8  A90A8555200AA843     )JEU J(C
A9D0  2920323731362032     ) 2716,2
A9D8  3531B62073A82073     516 s( s
A9E0  A8A90A8555200AA8     ()JEU J(
A9E8  57484154BA20F0AE     WHAT: p.
A9F0  4820A4F66885FCC9     H $vhE|I
A9F8  41D006A90085FDF0     APF)@E p
AA00  12C942D006A98085     RIBPF)@E
AA08  FD3008C943D078A9      0HICPx)
AA10  C085FD2073A82073     @E  s( s
AA18  A8200AA853455420     ( J(SET
AA20  5357495443482054     SWITCH T
AA28  4F20504F53495449     O POSITI
AA30  4F4EA0A5FCC941D0     ON %|IAP
AA38  0A200AA8323533B2     J J(2532
AA40  18901EC942D00A20     XP^IBPJ
AA48  0AA8323733B21890     J(2732XP
AA50  10C943D032200AA8     PICP2 J(
AA58  3237313620323531     2716,251
AA60  B62073A82073A8A9     6 s( s()
AA68  0A8555200AA84E4F     JEU J(NO
AA70  5720494E53455254     W INSERT
AA78  204550524FCD20D7      EPROM W
AA80  AB208FAA4C03A8A9     + O*LC()
AA88  FD20A4F64CEDA920       $vLm)
AA90  73A8A90A8555200As    s()JEU J
AA98  A850524553532053     (PRESS S
AAA0  50414345204241D2     PACE BAR
AAA8  20F0AE602073A8A9      p.' s()
AAB0  0A8555200Ą.A84F4B    JEU J(OK
AAB8  2028592F4EA920F0      (Y/N) p
AAC0  AE4820A4F668C94E     .H $vhIN
```

```
AAC8  F003A90060A90160     pC)@')A'
AAD0  484A4A4A4A20DBAA     HJJJJ [*
AAD8  68290FC90AB00409     h)OIJ0DI
AAE0  30D0031869374CA4     0PCXi7L$
AAE8  F6A90085F285F385     v)@ErEsE
AAF0  FEA90485FC20F0AE      )DE| p.
AAF8  48C99BF00320A4F6     HI[pC $v
AB00  68C9303025C94710     hI00%IGP
AB08  21C93A3007C94130     !I:0GIA0
AB10  191869090A0A0A0A     YXiIJJJJ
AB18  A0042A26F226F388      D*&r&sH
AB20  D0F8A98085FEC6FC     Px)@E F|
AB28  D0CB60A9308D02D3     PK')0MBS
AB30  A9FF8D00D3A9348D     ) M@S)4M
AB38  02D360A9308D02D3     BS')0MBS
AB40  A9008D00D3A9348D     )@M@S)4M
AB48  02D3602073A820FD     BS’ s(
AB50  AEA90A8555200AA8     .)JEU J(
AB58  46524F4DBA20E9AA     FROM: i*
AB60  A5FE300DA5F120D0     % 0M%q P
AB68  AAA5F020D0AA4C79     *%p P*Ly
AB70  ABA5F285F0A5F385     +%rEp%sE
AB78  F12073A8A90A8555     q s()JEU
AB80  200AA8544F2020BA      J(TO  :
AB88  20E9AAA5FE300DA5      i*% 0M%
AB90  F520D0AAA5F420D0     a P*%t P
AB98  AA4CA4ABA5F285F4     *L$+%rEt
ABA0  ASF385F5A5FB302E     %sEu%{0.
ABA8  2073A82015AFA90A      s( U/)J
ABB0  8555200AA8494E54     EU J(INT
ABB8  4FBA20E9AAA5FE30     O: i*% 0
ABC0  0DA5F920D0AAA5F8     M%y P*%x
ABC8  20D0AA4CD6ABA5F2      P*LV+%r
ABD0  85F8A5F385F960A9     Ex%sEy')
ABD8  0185FEA90385FCA9     AE )CE|)
ABE0  0985FFA5FD1021A9     IE % P!)
ABE8  041865FE85FEA904     DXe E )D
ABF0  1865FC85FCA90418     Xe|E|)DX
ABF8  65FF85FFA5FD2940     e E % )@
AC00  F006A5FE290F85FE     pF% )OE
AC08  60A5F085F2A5F185      %pEr%qE
AC10  F3A5F2D002A5F3F0     s%rPB%sp
AC18  16A5FC8D01D3A5FE     V%|MAS%
AC20  8D01D3C6F2A5F2C9     MASFr%rI
AC28  FFD0E6C6F310E260      PfFsPb'
AC30  A98085FAA90085FB     )@Ez)@E{
AC38  203BAB204BAB20AC      ;+ K+ ,
AC40  AAD0F820D7AB2009     *Px W+ I
```

```
AC48  ACA000202CA891F8     , @ ,(Qx
AC50  A5F1C5F59004A5F0     %qEuPD%p
AC58  C5F4F019E6F0D002     EtpYfpPB
AC60  E6F1E6F8D002E6F9     fqfxPBfy
AC68  A5FC8D01D3A5FE8D     %|MAS% M
AC70  01D31890D42073A8     ASXPT s(
AC78  A90A8555200AA84C     )JEU J(L
AC80  4F414445C4208FAA     OADED O*
AC88  4C03A8A98085FB85     LC()@E{E
AC90  FA203BAB204BAB20     z ;+ K+
AC98  ACAAD0F820D7AB20     ,*Px W+
ACA0  09ACA000202CA8C9     I, @ ,(I
ACA8  FFD039A5F1C5F590      P9%qEuP
ACB0  04A5F0C5F4F013E6     D%pEtpSf
ACB8  F0D002E6F1A5FC8D     pPBfq%|M
ACC0  01D3A5FE8D01D318     AS% MASX
ACC8  90D82073A8A90A85     PX s()JE
ACD0  55200AA845524153     U J(ERAS
ACD8  45C4208FAAA90085     ED O*)@E
ACE0  FB4C03A82073A8A9     {LC( s()
ACE8  0A8555200AA84E4F     JEU J(NO
ACF0  5420455241534544     T ERASED
ACF8  20494EA0A5F12UD0      IN %q P
AD00  AAA5F020D0AA208F     *%p P* O
AD08  AAA90085FB4C03A8     *)@E{LC(
AD10  A90085FB85FA202B     )@E{Ez +
AD18  AB204BAB20ACAAD0     + K+ ,*P
AD20  F820D7ABA5F885F2     x W+%xEr
AD28  A5F985F32011ACA0     %yEs Q,
AD30  00B1F08D00D320A9     @lpM@S )
AD38  ADA5F1C5F59004A5     -%qEuPD%
AD40  F0C5F4F013E6F0D0     pEtpSfpP
AD48  02E6F1A5FC8D01D3     Bfq%|MAS
AD50  A5FE8D01D31890D7     % MASXPW
AD58  2073A8A90A855520      s()JEU
AD60  0AA850524F475241     J(PROGRA
AD68  4D4D45C4208FAA4C     MMED O*L
AD70  03A82073A8A90A85     C( s()JE
AD78  55200AA843454C4C     U J(CELL
AD80  A0A5F120D0AAASF0      %q P*%p
AD88  20D0AA200AA8204E      P* J( N
AD90  4F542050524F4752     OT PROGR
AD98  414D4D45C4208FAA     AMMED O*
ADA0  4C03A8A0FF88D0FD     LC(  HP
ADA8  60A5FF8D01D320A3     '% MAS #
ADB0  AD290E8D01D34820     -)NMASH
ADB8  DDAD6809018D01D3     ]-hIAMAS
ADC0  A5FE8D01D320A3AD     % MAS #-
```

```
ADC8  203BAB202CA8A000      ;+ ,( @
ADD0  D1F0F00568684C72     QppEhhLr
ADD8  AD202BAB60A9FF85     - ++') E
ADE0  F6A90B85F7A5F6D0     v)Kew%vP
ADE8  02A5F7F00DC6F6A5     B%wpMFv%
ADF0  F6C9FFD0F0C6F718     vI PpFwX
ADF8  90EB6020F0AE4820     Pk' p.H
AE00  A4F668C952D006A9     $vhIRPF)
AE08  0085FAF012C945D0     @EzpRIEP
AE10  06A98085FA3008A9     F)@EzOH)
AE18  FD20A4F64CFBAD20       $vL{-
AE20  3BABA98085FB204B     ;+)@£{ K
AE28  AB20ACAAD0F820D7     + ,*Px W
AE30  A82037AE4C03A8A5     + 7.LC(%
AE38  FA10032009ACA97D     zPC I,)
AE40  20A4F6A90085F620      $v)@Ev
AE48  73A8A90085F7A5F1     s()@Ew%q
AE50  85F320D0AAA5F085     Es P*%pE
AE58  F220D0AA20DBAEAS     r P* [.%
AE60  FA100620E0AE1890     zPF '.XP
AE68  04A000B1F020D0AA     D @1p P*
AE70  E6F7A5F7C908F00B     fw%wIHpK
AE78  20DBAEE6F0D002E6      [.fpPBf
AE80  F1D0DCA90085F720     qP\)@Ew
AE88  DBAEA5FA3021A000     [.%z0! @
AE90  B1F2C9209004C97A     1rI PDIz
AE98  9002A92E20A4F6E6     PB). $vf
AEA0  F7A5F7C908F008E6     w%wIHpHf
AEA8  F2D002E6F3D0DBAS     rPBfsP[%
AEB0  F1C5F59004A5F0C5     qEuPD%pE
AEB8  F49006208FAA4C03     tPF O*LC
AEC0  A8E6E0D002E6F1E6     (fpPBfqf
AEC8  F6A5F6C914F0034C     v%vITpCL
AED0  47AE208FAA20D7AB     G. O* W+
AED8  4C3EAEA9204CA4F6     L>.) L$v
AEE0  202CA848A5FC8D01      ,(H%|MA
AEE8  D3A5FE8D01D36860     S% MASh'
AEF0  20E2F6C958D00568      bvIXPEh
AEF8  684C03A860A90485     hLC(‘)DE
AF00  55A5FA1009200AA8     U%zPI J(
AF08  4550524FCD60200A     EPROM' J
AF10  A85241CD60A90485     (RAM')DE
AF18  55A5FA1007200AA8     U%zPG J(
AF20  5241CD60200AA845     RAM' J(E
AF28  50524FCD60A98085     PROM')@E
AF30  FAA90085FB203BAB     z)@E{ ;+
AF38  204BAB20ACAAD0FB      K+ ,*Px
AF40  20D7AB2009ACA000      W+ I, @
```

```
AF48  202CA848D1F8D03E      ,(HQxP>
AF50  68A5F1C5F59004A5     h%qEuPD%
AF58  F0C5F4F019E6F0D0     pEtpYfpP
AF60  02E6F1E6F8D002E6     BfqfxPBf
AF68  F9A5FC8D01D3A5FE     y%|MAS%
AF70  8D01D31890D02073     MASXPP s
AF78  A8A90A8555200AA8     () JEU (
AF80  5645524946494504     VERIFIED
AF88  208FAA4C03A82073      O*LC( s
AF90  A8200AA844494646     ( J(DIFF
AF98  4552454E54204259     ERENT BY
AFA0  544553A06820D0AA     TES h P*
AFA8  20DBAEA000B1F820     [. @lx
AFB0  D0AA200AA820494E     P* J( IN
AFB8  A0A5F920D0AAA5F8      %y P*%x
AFC0  20D0AA208FAA4C03      P* 0*LC
AFC8  A8000000000000000    (@@@@@@@
```

This hexdump has to be keyed in starting at address A800. This means you need a 48K RAM ATARI and a machine language monitor (ATMONA-1, Editor/Assembler cartridge from ATARI or ATMAS-1). The program starts at address A800 hex.

Using the EPROM board Kit from HOFACKER

After you burned an EPROM you certainly want to plug it into your ATARI. for this you need a pc-board. You can buy those boards from various vendors (APEX, ELCOMP PUBLISHING).

The following description shows how to use the EPROM board from ELCOMP PUBLISHING, INC.

With this versatile ROM module you can use 2716, 2732 and 2532 type EPROMs.

To set the board for the specific EPROM, just solder their jumpers according to the list shown below. Without any soldering you can use the module for the 2532 right away.

If you use only one EPROM, inxrt it into the right socket.

If you use two EPROMs, put the one with the higher address into the right socket.

The modul must be plugged into the left slot of your ATARI computer with the parts directed to the back of the computer.

EPROM	2716	2732	2516	2532
1	S	O	S	S
2	O	S	O	O
3	S	S	S	O
4	O	O	O	S
5	O	S	O	O

S = means connected (short)
O = means open

HOW TO ADD OR CHANGE A DEVICE

CHAPTER 7

If you want to add your own device, you first have to write a handler/controller (interface). You have to submit the handler on the following design decisions.

- There has to be an OPEN routine, which opens the device/file and returns with the status of these operations stored in the Y-register of your 6502.
- You also need a CLOSE routine, which unlinks the device and returns the status as the OPEN-routine does.
- Further needed is a GETBYTE routine, which receives the data from your device and returns the data in the A-register and the status in the Y-register. If your device is a write only device (such as a printer) you have to return with errorcode 146 (not implemented function) in the Y-register.
- A PUTBYTE routine, sends a byte (which will be in the A-register) to your device, and returns, as the other routines do, the status. If your device is read only, then return the 146 errorcode.
- A GET STATUS routine stores the status of your device (max. 4 bytes) at DVSTAT ($02EA. D). If the GET STATUS function is not necessary, you have to leave the dummy routine . with 146 in your Y-register (error).
- A SPECIAL COMMAND routine is required, if you need more commands than previous. If not, return with Y=146.

OS will load the X-register with the IOCB number times 16 so you are able to get specific file information out of the user IOCB.

These 6 entries have to be placed in a so called handlertable. The vectors of these have to be one less than the real address, due to OS requirements.

OPEN vector - 1
CLOSE vector – 1
GETBYTE vector – 1
PUTBYTE vector – 1
GETSTAT vector – 1
SPECIAL vector - 1

Now you have to add the device to the device table. A device entry needs 3 bytes. The device name, which is usually character that indicates the device (first character of the full devicename) is first. Second, a vector that points to the devicehandler.

device name	
handler table address	

If you only want to change the handler of a device to your own handler, you only have to scan the devicetable (started from $031A) and let the vector point to your handler table.

If 'it is a totally new device, you have to add it, at the next free position of the device table (filled with zero).

The first listing shows you a handler for a new printer device. Calling INITHAN will install the new handler-table. Now you can connect a printer with centronics interface at gameport 3 & 4 (see connection scheme). After each SYSTEM RESET you have to initialize the device again. For program description see program listing.

The second listing is a listing of an inexpensive (write only) RS232 interface for your ATARI. Just call INITHAN and the new device will be added to the device table. It is now possible to use it like any other device. The RS232 output is on gameport 3 (see connection scheme). It is not our intention to describe detail the working of the RS232 interface. The comments in the program should help a bit though.

CENTRONICS PARALLEL INTERFACE

```
                    PRTENTRY    EQU$031A          STANDARD ENTRY BY SYSTEM

                    TRIG3       EQU   $D013
                    PACTL       EQU   $D303
                    PORTA       EQU   $D3C1

                    EOL         EQU   $9B
                    CR          EQU   $0D
                    LF          EQU   $0A

                                ORG   $0600,  $A800

               *                THE HANDLERTABLE

0600: 0F 06         HANDLTAB    DFW   OPEN-1
0602: 23 06                     DFW   CLOSE-1
0604: 26 06                     DFW   GETBYTE-1
0606: 29 06                     DFW   PUTBYTE-1
0608: 26 06                     DFW   STATUS-1
060A: 26 06                     DFW   SPECIAL-1

060C: 00 00 00                  DFB   0,0,0,0     FILL REST WITH ZER0
060F: 00
               *                THE OPEN ROUTINE

                    OPEN        EQU   *
0610: A930          INIT        LDA   #$30
0612: 8D 03 D3                  STA   PACTL
0615: A9 FF                     LDA   #$FF
0617: 8D 01 D3                  STA   PORTA
061A: A9 34                     LDA   #$34
061C: BD 03 D3                  STA   PACTL
061F: A9 80                     LDA   #$80
0621: 8D 01 D3                  STA   $D301
0624: A0 01         SUCCES      LDY   #1
0626: 60                        RTS

               *                THE CLOSE DUMMY ROUTINE
               *                ONLY RETURN SUCCESS IN Y (1)
```

```
                CLOSE       EQU SUCCES

0627: A0 92     NOTIMPL     LDY   #146
0629: 60                    RTS
          *                 THE POLLOWING COMMANDS ARE
          *                 NOT IMPLEMENTED SO GET ERROR
          *                 CODE 116

                GETBYTE     EQU   NOTIMPL
                STATUS      EQU   NOTIMPL
                SPECIAL     EQU   NOTIMPL

          *                 THE PUTBYTE ROUTINE 1

062A: C9 9B     PUTBYTE     CMP   #EOL
062C: D0 07                 BNE   NOEOL

          *                 IF EOL THEN CRLF TO PRINTER

062E: A9 0D                 LDA   #CR
0630: 20 3B 06              JSR   PARAOUT
0633: A9 0A                 LDA   #LF
0635: 20 3B 06  NOEOL       JSR   PARAOUT
0638: A0 01                 LDY   #1
063A: 60                    RTS

          *                 TBE PARALLEL OUT

0636: AC 13 D0  PARAOUT     LDY   TRIG3
063E: D0P B                 BNE   PARAOUT     WAIT IF BUSY
0640: A0 80                 LDY   #%10000000
0642: 09 80                 ORA   #%10000000
0644: 8D 01 D3              STA   PORTA   STROBE ON AND PUT DATA ON BUS
0647: 297F                  AND   #%01111111
0649: 8D 01 D3              STA   PORTA       STROBE OFF
061C: 8C 01 D3              STY   PORTA       CLEAR BUS
061P: 60    RTS

          *                 PUT NEW ADDRESS IN HANDLER VECTOR

0650: A9 00     INITHAN     LDA   #HANDLTAB:L
0652: 8D 1B 03              STA   PRTENTRY+1
0655: A90 6                 LDA   #HANDLTAB:H
0657: 8D1C03                STA   PRTENTRY+2        '
065A: 4C1006                JMP   OPEN

PHYSICAL ENDADDRESS: $A85D
*** NO WARNINGS

PRTENTRY    $031A                 TRIG3       $D013
PACTL       $D303                 PORTA       $D301
EOL         $9B                   CR          $0D
LP          $0A                   HANDLTAB    $0600
OPEN        $0610                 INIT        $0610       UNUSED
SUCCES      $0621                 CLOSE       $0624
NOTIMPL     $0627                 GETBYTE     $0627
STATUS      $0627                 SPECIAL     $0627
PCTBYTE     $062A                 NOEOL       $0635
```

PARAOUT $0638 INITHAN $0650 UNUSED

For more information about the parallel interface refer to Page 70.

RS 232 SERIAL INTERFACE

```
                    COUNT       EPZ   $1F
                    RSENTRY     EQU   $032C       NEXT FREE POSITION IN
                                                        DEVICE TABLE
                    PACTL       EQU   $D303
                    PORTA       EQU   $D301
                    NMIEN       EQU   $D40E
                    DMACTL      EQU   $D400
                    EOL         EQU   $9B
                    CR          EQU   $0D
                    LF          EQU   $0A
                    K           EQU   150   110 AND 300 BAUD
                    L           EQU   6     300 BAUD
                    *L          EQU   18    110 BAUD

                                ORG   $0600,$A800

0600: 0F 06         HANDLTAB    DFW   OPEN-1
0602: 29 06                     DFW   CLOSE-1
0604: 2C 06                     DFW   GETBYTE-1
0606: 2F 06                     DFW   PUTBYTE-1
0608: 2C 06                     DFW   STATUS-1
060A: 2C 06                     DFW   SPECIAL-1
060C: 00 00 00                  DFB   0,0,0,0 JUST FILL WITH ZERO
060F: 00
            *                   THE OPEN ROUTINE

                    OPEN        EQU   *
0610: A9 30         INIT        LDA   1530
0612: 8D 03 D3                  STA   PACTL
0615: A9 01                     LDA   #%00000001
0617: 8D 01 D3                  STA   PORTA
061A: A9 34                     LDA   1534
061C: 8D 03 D3                  STA   PACTL
061F: A9 00                     LDA   1500
0621: 8D 01 D3                  STA   PORTA
0624: 20 85 06                  JSR   BITWAIT
0627: 20 85 06                  JSR   BITWAIT
062A: A0 01         SUCCES      LDY   #1
062C: 60                        RTS

            *                   THE CLOSE ROUTINE IS A DUMMY
```

```
           *               BUT Y#1 (SUCCESSFULL CLOSE)

                 CLOSE     EQU   SUCCES
062D: A0 92      NOTIMPL   LDY   #146        RETURN WITH Y=116
062F: 60                   RTS

           *               THE FOLLOWING COMMANDS ARE NOT IMPLEMENTED

                 GETBYTE   EQU         NOTIMPL
                 STATUS    EQU         NOTIMPL
                 SPECIAL   EQU         NOTIMPL

           *               THE PUTBYTE COMMAND
           *               DATA IN ACCU
           *               STATUS IN Y (=1)

0630: 48         PUTBYTE   PHA
0631: C9 98                CMP   #EOL
0633: D0 07                BNE   NOEOL

           *               IF EOL GIVE CRLF TO DEVICE

0635: A9 0D                LDA   #CR
0637: 20 43 06             JSR   SEROUT
063A: A9 0A                LDA   #LF
063C: 20 43 06   NOEOL     JSR   SEROUT
063F: 68                   PLA
0640: A0 01                LDY #1
0642: 60                   RTS

           *               SERIALOUT FIRST REVERSE BYTE

0643: 49 FF      SEROUT    EOR   #%llllllll
0645: 8D A2 06             STA   BUFFER

           *               DISABLE INTERRUPTS

0648: 78                   SEI
0649: A9 00                LDA   #0
0648: 8D 0E D4             STA   NMIEN
064E: BD 00 D4             STA   DMACTL

           *               SEND STARTBIT

0651: A9 01                LDA   #%00000001
0653: BD 01 D3             STA   PORTA
0656: 20 85 06             JSR   BITWAIT

           *               SEND BYTE
```

```
0659: A0 08                 LDY   #8
065B: 84 1F                 STY   COUNT
065D: AD A2 06    SENDBYTE  LDA   BUFFER
0660: BD 01 D3              STA   PORTA
0663: 6A                    ROR
0664: BD A2 06              STA   BUFFER
0667: 20 85 06              JSR   BITWAIT
066A: C6 1F                 DEC   COUNT
066C: D0 EF                 BNE   SENDBYTE

           *                SEND TWO STOPBITS

066E: A9 00                 LDA   #%00000000
0670: BD 01 D3              STA   PORTA
0673: 20 85 06              JSR   BITWAIT
0676: 20 85 06              JSR   BITWAIT

           *                ENABLE INTERRUPTS

0679: A9 22                 LDA   #$22
067B: BD 00 D4              STA   DMACTL
067E: A9 FF                 LDA   #$FF
0680: BD 0E D4              STA   NMIEN
0683: 58                    CLI
0684: 60                    RTS

           *                THE   BITTIME ROUTINE FOR AN  EXACT BAUDRATE

0685: A2 96       BITWAIT   LDX   #K
0687: A0 06       LOOPR     LDY   #L
0689: 88          LOOPL     DEY
068A: D0 FD                 BNE   LOOPL
068C: CA                    DEX
068D: DO FB                 BNE   LOOPR
068F: 60                    RTS

           *                ROUTINE FOR INSTALLING THE RS232 HANDLER

0690: A9 52       INITHAN   LDA   'R          DEVICE NAME
0692: BD 2C 03              STA   RSENTRY
0695: A9 00                 LDA   #HANDLTAB:L
0697: 8D 2D 03              STA   RSENTRY+1
069A: A9 06                 LDA   #HANDLTAB:H
069C: 80 2E 03              STA   RSENTRY+2
069F: 4C 10 06              JMP   OPEN

                  BUFFER    EQU   *           ONE BYTE    BUFFER
```

```
PHYSICAL END ADDRESS: $A8A2

*** NO WARNINGS
COUNT       $1F                 RSENTRY     $032C
PACTL       $D303               PORTA       $D301
NMIEN       $D40E               DMACTL      $D400
EOL         $98                 CR          $OD
LF          $0A                 K           $96
L           $06                 HANDLTAB    $0600
OPEN        $0610               INIT        $0610       UNUSED
SUCCES      $062A               CLOSE       $062A
NOTIMPL     $062D               GETBYTE     $062D
STATUS      $062D               SPECIAL     $062D
PUTBYTE     $0630               NOEOL       $063C
SEHUUT      $0643               SENDBYTE    $065D
BITWAIT     $0685               LOOPK       $0687
LOOPL       $0689               INITHAN     $0690       UNUSED
BUFFER      S06A2
```

A BOOTABLE TAPE GENERATOR PROGRAM

CHAPTER 8

The following program allows you to generate a bootable program on tape. This generator must be in memory at the same time as the program.

After you have jumped to the generator, a dialogue will be started. First, the boot generator asks for the address where your program is stored (physical address). After you have entered start and end address (physical), you will be asked to enter the address where the program has to be stored during boot (logical address). The generator further asks for the restart address (where OS must jump to, to start your program).

There is no feature to define your own initialization address. This address will be generated automatically and points to a single RTS.

Also given is the boot continuation code, which will stop the cassette motor, and store the restart address into DOSVEC ($0A.B).

So, you just have to put a cassette in your recorder, start the generator, and the dialogue will be started.

The generator puts the boot information header in front of your program, so there have to be at least 31 free bytes in front of the start address (physical & logical).

The generator program will not be explained here, but after reading the previous chapters you should have the knowledge to understand it. There are also some helpfull comments in the program.

BOOT – GENERATOR

```
STOREADR    EPZ   $F0.1
ENDADR      EPZ   $F2.3
PROGLEN     EPZ   $F4.5
JMPADR      EPZ   $F6.7
EXPR        EPZ   $F8.9
LOGSTORE    EPZ   $FA.B
HEXCOUNT    EPZ   $FC

DOSVEC      EPZ   $0A

MEMLO       EPZ   $02E7

ICCOM       EQU   $0342
ICBAL       EQU   $0344
ICBAH       EQU   $0345
ICBLL       EQU   $0348
ICBLH       EQU   $0349
ICAX1       EQU   $034A
ICAX2       EQU   $034B
```

```
                    OPEN        EQU   $03
                    PUTCHR      EQU   $0B
                    CLOSE       EQU   $0C

                    OPNOT       EQU   8

                    SCROUT      EQU   $F6A4
                    GETCHR      EQU   $F6DD
                    BELL        EQU   $F90A
                    CIOV        EQU   $E456
                    PACTL       EQU   $D302

                    CLS         EQU   $7D
                    EOL         EQU   $9B
                    BST         EQU   $1E
                    CR          EQU   $0D
                    IOCBNUM     EQU   1

                                ORG   $A800

A800: A9 7D       START         LDA   #CLS
A802: 20 A4 F6                  JSR   SCROUT

            *                   PRINT MESSAGE

A805: 20 00 AA                  JSR   PRINT
A808: 0D 0D                     DFB   CR,CR
A80A: 42 4F 4F                  ASC   \BOOTGENERATOR FROM HOFACKER\
A80D: 54 47 45
A810: 4E 45 52
A813: 41 54 4F
A816: 52 20 46
A819: 52 4F 4D
A81C: 20 48 4F
A81F: 46 41 43
A822: 4B 45 D2

               *                GET   STOREADDRESS

A825: 20 00 AA                  JSR   PRINT
A828: 0D 0D                     DFB   CR,CR
A82A: 53 54 4F                  ASC   \STOREADDRESS :$\
A82D: 52 45 41
A830: 44 44 52
A833: 45 53 53
A836: 20 3A A4
A839: 20 28 AA                  JSR   HEXIN
A83C: 89 F0                     STY   STOREADR
A83E: 85 F1                     STA   STOREADR+1
```

```
           *               GET ENDADDRESS

A840: 20 00 AA             JSR   PRINT
A843: 0D 0D 0D             DFB   CR,CR,CR
A846: 45 4F 44             ASC   \ENDADDRESS :$\
A849: 41 44 44
A84C: 52 45 53
A84F: 53 20 20
A852: 20 3A A4
A855: 20 28 AA             JSR   HEXIN
A858: 84 F2                STY   ENDADR
A85A: 85 F3                STA   ENDADR+1

           *               GET LOGICAL STORE

A85C: 20 00 AA             JSR   PRINT
A85F: 0D 0D 0D             DFB   CR,CR,CR
A862: 4C 4F 47             ASC   \LOGICAL STOREADDRESS :$\
A865: 49 43 41
A868: 4C 20 53
A86B: 54 4F 52
A86E: 45 41 44
A871: 44 52 45
A874: 53 53 20
A877: 3A A4
A879: 20 28 AA             JSR   HEXIN
A87C: 84 FA                STY   LOGSTORE
A87E: 85 FB                STA   LOGSTORE+1

           *               GET JUMP

A880: 20 00 AA             JSR   PRINT
A883: 0D0D0D               DFB   CR,CR,CR .
A886: 4A 55 4D             ASC   \JUMPADDRESS :$\
A889: 50 41 44
A88C: 44 52 45
A88F: 53 53 20
A892: 20 20 20
A895: 3A A4
A897: 20 28 AA             JSR   HEXIN
A89A: 84 F6                STY   JMPADR
A89C: 85 F7                STA   JMPADR+1

           *               CALCULATE NEW STORE

A89E: A5 F0                LDA   STOREADR
A8A0: 38                   SEC
A8A1: E9 20                SBC   #(HEADEND-HEAD)+1
A8A3: 85 F0                STA   STOREADR
```

```
A8A5: B0 02                 BCS   *+4
A8A7: C6 F1                 DEC   STOREADR+1

           *                CALCULATE LOGICAL STORE

A8A9: A5FA                  LDA   LOGSTORE
A8AB: 38                    SEC
A8AC: E9 20                 SBC   #(HEADEND-HEAD)+1
A8AE: 85 FA                 STA   LOGSTORE
A8B0: B0 02                 BCS   *+4
A8B2: C6 FB                 DEC   LOGSTORE+1

           *                MOVE HEADER IN FRONT OF PROGRAM

A8B4: 20 F5 A9              JSR   MOVEHEAD

           *                CALCULATE LENGTHE OF PROGR.

A8B7: A5 F2                 LDA   ENDADR
A8B9: 38                    SEC
A8BA: E5 F0                 SBC   STOREADR
ABBC: 85 F4                 STA   PROGLEN
ABBE: A5 F3                 LDA   ENDADR+1
A8C0: E5 F1                 SBC   STOREADR+1
A8C2: 85 F5                 STA   PROGLEN+1
A8C4: B0 03                 BCS   *+5
A8C6: 4C DA A9              JMP   ADRERR

           *                ROUND UP TO 128 RECORDS

A8C9: A5 F4                 LDA PROGLEN
A8CB: 18                    CLC
A8CC: 69 7F                 ADC   #127
A8CE: 29 80                 AND   #128
A8D0: 85 F4                 STA   PROGLEN
A8D2: 90 02                 BCC   *+4
A8D4: E6 F5                 INC   PROGLEN+1

           *                CALCULATE NUMBER OF RECORDS

A8D6: 0A                    ASL
A8D7: A5 F5                 LDA   PROGLEN+1
A8D9: 2A                    ROL
A8DA: A0 01                 LDY   #RECN-HEAD
A8DC: 91 F0                 STA   (STOREADR),Y
A8DE: A0 02                 LDY   #PST-HEAD
A8E0: A5 FA                 LDA   LOGSTORE
A8E2: 91 F0                 STA   (STOREADR),Y
A8E4: A5 FB                 LDA   LOGSTORE+1
```

```
A8E6: C8                    INY
A8E7: 91 F0                 STA   (STOREADR),Y
A8E9: A0 04                 LDY   #PINITADR-HEAD ABEB: 18 CLC
A8EC: A5 FA                 LDA   LOGSTORE
A8EE: 69 1F                 ADC   #PINIT-HEAD
A8F0: 91 F0                 STA   (STOREADR),Y
A8F2: C8                    INY
A8F3: A5 FB                 LDA   LOGSTORE+1
A8F5: 69 00                 ADC   #0
A8F7: 91 F0                 STA   (STOREADR),Y
A8F9: A0 0C                 LDY   #PNDLO-HEAD
A8FB: A5 FA                 LDA   LOGSTORE
A8FD: 18                    CLC
A8FE: 65 F4                 ADC   PROGLEN
A900: 91 F0                 STA   (STOREADR),Y
A902: A0 11                 LDY   #PNDHI-HEAD
A904: A5 FB                 LDA   LOGSTORE+1
A906: 65 F5                 ADC   PROGLEN+1
A908: 91 F0                 STA   (STOREADR),Y
A90A: A0 16                 LDY   #JUMPADRL-HEAD
A90C: A5 F6                 LDA   JMPADR
A90E: 91 F0                 STA   (STOREADR) ,Y
A910: A0 1A                 LDY   #JUMPADRH-HEAD
A912: A5 F7                 LDA   JMPADR+1
A914: 91 F0                 STA   (STOREADR),Y

           *                BOOTTAPE GENERATION PART, GIVE INSTRUCTIONS

A916: 20 00 AA              JSR   PRINT
A919: 0D 0D                 DFB   CR,CR
A91B: 50 52 45              ASC   "PRESS PLAY & RECORD"
A91E: 53 53 20
A921: 50 4C 41
A924: 59 20 26
A927: 20 52 45
A92A: 43 4F 52
A92D: 44
A92E: 0D 0D                 DFB   CR,CR
A930: 41 46 54              ASC   \AFTER THE BEEPS 'RETURN'\
A933: 45 52 20
A936: 54 48 45
A939: 20 42 45
A93C: 45 50 53
A93F: 20 27 52
A942: 45 54 55
A945: 52 4E A7
```

```
           *                OPEN CASSETTE FOR WRITE

A948: A2 10       OPENIOCB    LDX   #IOCBNUM*16
A94A: A9 03                   LDA   #OPEN
A94C: 9D 42 03                STA   ICCOM,X
A94F: A9 08                   LDA   #OPNOT
A951: 9D 4A 03                STA   ICAXI,X
A954: A9 80                   LDA   #128 ,
A956: 9D 4B 03                STA   ICAX2,X
A959: A9 F2                   LDA   #CFILE:L
A95B: 9D 44 03                STA   ICBAL,X
A95E: A9 A9                   LDA   #CFILE:H
A960: 9D 45 03                STA   ICBAH,X
A963: 20 56 E4                JSR   CIOV
A966: 30 28                   BMI   CERR

           *                PUT PROGRAM ON TAPE

A968: A9 0B       PUTPROG     LDA   #PUTCHR
A96A: 9D 42 03                STA   ICCOM,X
A96D: A5 F0                   LDA   STOREADR
A96F: 9D 44 03                STA   ICBAL,X
A972: A5 F1                   LDA   STOREADR+1
A974: 9D 45 03                STA   ICBAH,X
A977: A5 F4                   LDA   PROGLEN
A979: 9D 48 03                STA   ICBLL,X
A97C: A5 F5                   LDA   PROGLEN+1
A97E: 9D 49 03                STA   ICBLH,X
A981: 20 56 E4                JSR   CIOV
A984: 30 0A                   BMI   CERR

           *                CLOSE IOCB

A986: A9 0C        CLOSIOCB   LDA   #CLOSE
A988: 9D 42 03                STA   ICCOM,X
A98B: 20 56 E4                JSR   CIOV
A98E: 10 24                   BPL   SUCCES

           *                IF ERROR OCCURS SHOW THE ERRORNUMBER

A990: 98          CERR        TYA
A991: 48                      PHA
A992: A2 10                   LDX   #IOCBNUM*16
A994: A9 0C                   LDA   #CLOSE
A996: 9D 42 03                STA   ICCOM,X
A999: 20 56 E4                JSR   CIOV
A99C: 20 00 AA                JSR   PRINT
A99F: 0D 0D                   DFB   CR,CR
A9A1: 45 52 52                ASC   \ERROR- \
```

```
A9A4: 4F 52 2D
A9A7: A0
A9A8: 68                    PLA
A9A9: AA                    TAX
A9AA: 20 88 AA              JSR   PUTINT
A9AD: 20 00 AA              JSR   PRINT
A9B0: 8D                    DFB   CR+128
A9B1: 4C A2 AA              JMP   WAIT

           *                IF NO ERROR OCCURS TELL IT THE USER

A9B4: 20 00 AA    SUCCES    JSR PRINT
A9B7: 0D0D                  DFB CR,CR
A9B9: 53 55 43              ASC "SUCCESFULL BOOTTAPE GENERATION"
A9BC: 43 45 53
A9BF: 46 55 1C
A9C2: 4C 20 42
A9C5: 4F 4F 54
A9C8: 54 41 50
A9CB: 45 20 47
A9CE: 45 4E 45
A9D1: 52 41 54
A9D4: 49 4F 4E
A9D7: 0D8D                  DFB CR,CR+128

           *                BRK-INSTRUCTION TO TERMINATE THE PROGRAM.
           *                MOSTLY A JUMP INTO THE MONITOR-PROGRAM FROM
           *                WHERE YOU STARTED THE PROGRAM. INSTEAD OF THE
           *                'BRK' YOU ALSO CAN USE THE 'RTS' THE RTS
           *                INSTRUCTION, IF THIS PROGRAM WAS CALLED AS A
           *                SUBROUTINE.

A9D9: 00                    BRK

           *                IF ERROR IN THE ADDRESSES TELL IT THE USER

A9DA: 20 00 AA    ADRERR    JSR   PRINT
A9DD: 0D0D                  DFB   CR,CR
A9DF: 41 44 44              ASC   \ADDRESSING ERROR\
A9E2: 52 45 53
A9E5: 53 49 4E
A9EB: 47 20 45
A9EB: 52 52 4F
A9EE: D2
A9EF: 4C A2 AA              JMP   WAIT

           *                THESE 2 CHARACTERS ARE NEEDED TO OPEN
           *                A CASSETTE IOCB.
```

```
A9F2: 43 3A       CFILE       ASC   "C:"
A9F4: 9B                      DFB   EOL

            *                 ROUTINE FOR MOVING THE HEADER
            *                 IN FRONT OF THE USER-PROGRAM

A9F5: A0 1F       MOVEHEAD    LDY   #HFADEND-HEAD
A9F7: B9 A8 AA    MOVELOOP    LDA   HEAD,Y
A9FA: 91 F0                   STA   (STOREADR),Y
A9FC: 88                      DEY
A9FD: 10F8                    BPL   MOVELOOP
A9FF: 60                      RTS

            *                 THIS ROUTINE PRINTS A CHARACTERS
            *                 WHICH ARE BE POINTED BY THE
            *                 STACKPOINTER (USING THE 'JSR'
            *                 TO CALL THIS ROUTINE) .
            *                 THE STRING HAS TO BE TERMINATED
            *                 BY A CHARACTER WHOSE SIGNBIT
            *                 IS ON.

AA00: 68          PRINT       PLA
AA01: 85 F8                   STA   EXPR
AA03: 68                      PLA
AA04: 85 F9                   STA   EXPR+1
AA06: A2 00                   LDX   #0
AA08: E6 F8       PRINTI      INC   EXPR
AA0A: D0 02                   BNE   *+4
AA0C: E6 F9                   INC   EXPR+1
AA0E: A1 F8                   LDA   (EXPR,X)
AA10: 29 7F                   AND   #%01111111
AA12: C9 0D                   CMP   #CR
AA14: D0 02                   BNE   NOCR
AA16: A9 9B                   LDA   #EOL
AA18: 20 A4 F6                NOCR  JSR   SCROUT
AA1B: A2 00                   LDX   #0
AA1D: A1 F8                   LDA   (EXPR,X)
AA1F: 10 E7                   BPL   PRINTI
AA21: A5 F9                   LDA   EXPR+1
AA23: 48                      PHA
AA24: A5 F8                   LDA   EXPR
AA26: 48                      PHA
AA27: 60                      RTS

            *                 HEX INPUT ROUTINE WAITS FOR CORRECT FOUR
            *                 DIGITS OR 'RETURN'

AA28: A9 00       HEXIN       LDA   #0
AA2A: 85 F8                   STA   EXPR
```

```
AA2C: 85 F9                 STA   EXPR+1
AA2E: A9 03                 LDA   #3
AA30: 85 FC                 STA   HEXCOUNT
AA32: 30 31       HEXINI    BMI   HEXRTS
AA34: 20 DD F6              JSR   GETCHR
AA37: 48                    PHA
AA38: 20 A4 F6              JSR   SCROUT
AA3B: 68                    PLA
AA3C: C9 9B                 CMP   #EOL
AA3E: F0 25                 BEQ   HEXRTS
AA40: C9 58                 CMP   'X
AA42: F0 96                 BEQ   ADRERR
AA44: C9 30                 CMP   '0
AA46: 90 22                 BCC   HEXERR
AA48: C9 3A                 CMP   '9+1
AA4A: B0 08                 BCS   ALFA
AA4C: 29 0F                 AND   #%00001111
AA4E: 20 75 AA              JSR   HEXROT
AA51: 4C 32 AA              JMP   HEXIN1
AA54: C9 41       ALFA      CMP   'A
AA56: 90 12                 BCC   HEXERR
AA58: C9 47                 CMP   'F+1
AA5A: B00E                  BCS   HEXERR
AA5C: 38                    SEC
AA5D: E9 37                 SBC   'A-10
AA5F: 20 75 AA              JSR   HEXROT
AA62: 4C 32 AA              JMP   HEXINI
AA65: A4 F8       HEXRTS    LDY   EXPR
AA67: A5 F9                 LDA   EXPR+1
AA69: 60                    RTS

            *               IF WRONG DIGIT RINGS THE BUZZER
            *               AND   PRINT BACKSTEP

AA6A: 20 0A F9    HEXERR    JSR   BELL
AA6D: A9 1E                 LDA   #BST
AA6F: 20 A4 F6              JSR   SCROUT
AA72: 4C 32 AA              JMP   HEXIN1
AA75: C6 FC       HEXROT    DEC   HEXCOUNT
AA77: 08                    PHP
AA78: A2 04                 LDX   #4
AA7A: 0A                    ASL
AA7B: 0A                    ASL
AA7C: 0A                    ASL
AA7D: 0A                    ASL
AA7E: 0A          HEXROTl   ASL
AA7F: 26 F8                 ROL   EXPR
AA81: 26 F9                 ROL   EXPR+1
AA83: CA                    DEX
```

```
AA84: D0 F8                 BNE   HEXROTI
AA86: 28                    PLP
AA87: 60                    RTS

           *                THE RECURSIVE PUTINT FOR PRINTING ONE BYTE
           *                IN DECIMAL FORM

AA88: 48          PUTINT    PHA
AA89: 8A                    TXA
AA8A: C9 0A                 CMP   #10
AA8C: 90 0D                 BCC   PUTDIG   -IF A<10 THEN STOP RECURSION
AA8E: A2 FF                 LDX   #-1

*** WARNING: OPERAND OVERFLOW

AA90: E9 0A       DIV       SBC   #10
AA92: E8                    INX
AA93: B0 FB                 BCS   DIV
AA95: 69 0A                 ADC   #10
AA97: 20 88                 JSR   PUTINT - THE RECURSION STEP
AA9A: 18 CLC
AA9B: 69 30       PUTDIG    ADC   '0
AA9D: 20 A4 F6              JSR   SCROUT
AAAO: 68                    PLA
AAA1: 60                    RTS

           *                WAIT FOR ANY KEY

AAA2: 20 DD F6    WAIT      JSR   GETCHR
AAA5: 4C 00 A8              JMP   START

           *                THE BARECODE FOR THE HEADER TO PUT IN FRONT
           *                OF PROGRAM'

           *                THE DUMMY HEADER DUMMY EQU 0

AAA8: 00          HEAD      DFR   0
AAA9: 00          RECN      DFB   DUMMY
AAAA: 00 00       PST       DFW   DUMMY
AAAC: 00 00       PINITADR  DFW   DUMMY

           *                THE BOOT CONTINUATION CODE

AAAE: A9 3C                 LDA   #$3C
AAB0: 8D 02 D3              STA   PACTL
AAB3: A9 00                 LDA   #DUMMY
                  PNDLO     EQU   *-1
AAB5: 8D E7 02              STA   MEMLO
AAB8: A9 00                 LDA   #DUMMY
                  PNDHI     EQU   *-1
```

```
AABA: 8D E8 02               STA   MEMLO+1
AABD: A9 00                  LDA   #DUMMY
                 JUMPADRL    EQU   *-1
AABF: 85 0A                  STA   DOSVEC
AAC1: A9 00                  LDA   #DUMMY
                 JUMPADRH    EQU   *-1
AAC3: 85 0B                  STA   DOSVEC+1

AAC5: 18                     CLC
AAC6: 60                     RTS

                 HEADEND     EQU   *
AAC7: 60         PINIT       RTS

PHYSICAL ENDADDRESS: $AAC    8

STOREADR    $F0                  ENDADR      $F2
PROGLEN     $F4                  JMPADR      $F6
EXPR        $F8                  HEXCOUNT    $FC
MEMLO       $02E7                ICBAL       $0344
ICBLL       $0348                ICAX1       $034A
OPEN        $03                  CLOSE       $OC
SCROUT      $F6A4                BELL        $F90A
PACTL       $D302                EOL         $9B
CR          $0D                  START       $A800
PUTPROG     $A968 UNUSED         CERR        $A990
ADRERR      $A9DA                MOVEHEAD    $A9F5
PRINT       $AA00                NOCR        $AA18
HEXINI      $AA32                HEXRTS      $AA65
HEXROT      $AA75                PUTINT      $AA88
PUTDIG      $AA9B                DUMMY       $00
RECN        $AAA9                PINITADR    $AAAC
PNDHI       $AAB9                JUMPADRH    $AAC2
PINIT       $AAC7                LOGSTORE    $FA
DOSVEC      $0A                  ICCOM       $0342
ICBAH       $0345                ICBLH       $0349
ICAX2       $034B                PUTCHR      $0B
OPNOT       $08                  GETCHR      $F6DD
CIOV        $E456                CLS         $7D
BST         $1E                  IOCBNUM     $01
OPENIOCB    $A948 UNUSED         CLOSIOCB    $A986 UNUSED
SUCCES      $A9B4                CFILE       $A9F2
MOVELOOP    $A9F7                PRINTI      $AA08
HEXIN       $AA28                ALFA        $AA54
HEXERR      $AA6A                HEXROT1     $AA7E
DIV         $AA90                WAIT        $AAA2
HEAD        $AAA8                PST         $AAAA
PNDLO       $AAB4                JUMPADRL    $AABE
HEADEND     $AAC7
```

A DIRECT CASSETTE TO DISK COPY PROGRAM

CHAPTER 9

If you have a bootable program on cassette, and you want to have it on a bootable disk, the following program will help you.

This program is easy to understand if you have read the previous chapters. It allows you to copy direct from tape to disk, using a buffer.

When you start your program from your machine language monitor, you must put the cassette into the recorder and the formatted disk into the drive (#1). After the beep, press return, and the cassette will be read. After a successful read the program will be written on the disk. If, during one of these IO's an error occurs, the program stops and shows you the error code.

Now, power up the ATARI again and the disk will be booted. Sometimes the program doesn't work correctly. Just press SYSTEM RESET and most of the time the program will work.

The copy program will not be described, but it has helpful comments, and you possess the knowledge of the IO.

It is important that the buffer (BUFADR) is large enough for the program.

DIRECT CASSETTE TO DISK COPY PROGRAM

```
SECTR       EPZ   $80.1
DBUFFER     EPZ   $82.3
BUFFER      EPZ   $84.5
BUFLEN      EPZ   $86.7
RETRY       EPZ   $88
XSAVE       EPZ   $89

DCBSBI      EQU   $0300
DCBDRV      EQU   $0301
DCBCMD      EQU   $0302
DCBSTA      EQU   $0303
DCBBUF      EQU   $0304
DCBTO       EQU   $0306
DCBCNT      EQU   $0308
DCBSEC      EQU   $030A

ICCMD       EQU   $0342
ICBAL       EQU   $0344
ICBAH       EQU   $0345
ICBLL       EQU   $0348
ICBLH       EQU   $0349
ICAX1       EQU   $034A
ICAX2       EQU   $034B


OPEN        EQU   3
GETCHR      EQU   7
CLOSE       EQU   12
```

```
                RMODE       EQU   4
                RECL        EQU   128

                CIO         EQU   $E456
                SIO         EQU   $E459
                EOUTCH      EQU   $F6A4

                EOL         EQU   $9B
                EOF         EQU   $88
                IOCBNUM     EQU   1

                            ORG   $A800

           *                OPEN CASSETTE FOR READ

A800: 20 A7 A8  MAIN        JSR   OPENCASS
A803: 30 63                 BMI   IOERR

           *                INITIALIZE BUFFERLENGTH & BUFFER POINTER

A805: A9 56                 LDA   #BUFADR:L
A807: 85 84                 STA   BUFFER
A809: A9 A9                 LDA   #BCFADR:H
A80B: 85 85                 STA   BUFFER+1
ASOD: A9 80                 LDA   #128
A80F: 85 86                 STA   BUFLEN
A811: A9 00                 LDA   #0
A813: 85 87                 STA   BUFLEN+1

           *                READ RECORD BY RECORD TO BUFFER UNTILL EOF
           *                REACHED

A815: 20 C8 A8  READLOOP    JSR   READCASS
A818: 30 10                 BMI   QEOF

           *                IF NO ERROR OR EOF INCREASE THE BUFFERPOINTER

A81A: A5 84                 LDA   BUFFER
A81C: 18                    CLC
A81D: 69 80                 ADC   #128
A81F: 85 84                 STA   BUFFER
A821: A5 85                 LDA   BUFFER+1
A823: 69 00                 ADC   #0
A825: 85 85                 STA   BUFFER+1
A827: 4C 15 A8              JMP   READLOOP
           *                IF EOF REACHED THEN WRITE BUFFFR TO DISK
           *                ELSE ERROR

A82A: C0 88     QEOF        CPY   #EOF
A82C: D0 3A                 BNE   IOERR
A82E: 20 E9 A8              JSR   CLOSCASS
A831: 30 35                 BMI   IOERR

           *                INIT POINTERS FOR SECTOR WR1TE

A833: A9 01                 LDA   #1
A835: 85 80                 STA   SECTR
A837: A9 00                 LDA   #0
```

```
A839: 85 81                 STA   SECTOR+1
A83B: A9 56                 LDA   #BUFADR:L
A83D: 85 82                 STA   DBUFFER
A83F: A9 A9                 LDA   #BUFADR:H
A841: 85 83                 STA   DBUFFER+1

           *                WRITE SECTOR BY SECTOR BUFFER TO DISK

A843: 20 06 A9    WRITLOOP  JSR   WRITSECT
A846: 30 20                 BMI   IOERR

           *                IF BUFFER IS WRITTEN THEN STOP PROGRAM

A848: A5 82                 LDA   DBUFFER
A84A: C5 84                 CMP   BUFFER
A84C: A5 83                 LDA   DBUFFER+1
A84E: E5 85                 SBC   BUFFER+1
A850: B0 15                 BCS   READY

           *                INCREASE BUFFER AND SECTOR POINTERS

A852: A5 82                 LDA   DBUFFER
A854: 18                    CLC
A855: 69 80                 ADC   #128
A857: 85 82                 STA   DBUFFER
A859: A5 83                 LDA   DBUFFER+1
A85B: 69 00                 ADC   #0
AB5D: 85 83                 STA   DBUFFER+1

AB5F: E6 80                 INC   SECTR
A861: D0 02                 BNE   *+4
A863: E6 81                 INC   SECTR+1
A865: D0 DC                 BNE   WRITLOOP    JUMP ALWAYS!!!



      *                THE BREAK FOR RETURNING TO THE CALLING MONITOR

A867: 00          READY     BRK

A868: 98          IOERR     TYA
A869: 48                    PHA
A86A: A208                  LDX   #LENGTH
A86C: 86 89       ERRLOOP   STX   XSAVE
A86E: BD 84 A8              LDA   ERROR,X
A871: 20 A4 F6              JSR   EOUTCH
A874: A6 89                 LDX   XSAVE
A876: CA                    DEX
A877: 10F3                  BPL   ERRLOOP
A879: 68                    PLA
A87A: AA                    TAX
A87B: 20 8D A8              JSR   PUTINT
A87E: A9 9B                 LDA   #EOL
A880: 20 A4 F6              JSR   EOUTCH

      *                THE   BREAK FOR RETURNING TO THE CALLING MONITOR
```

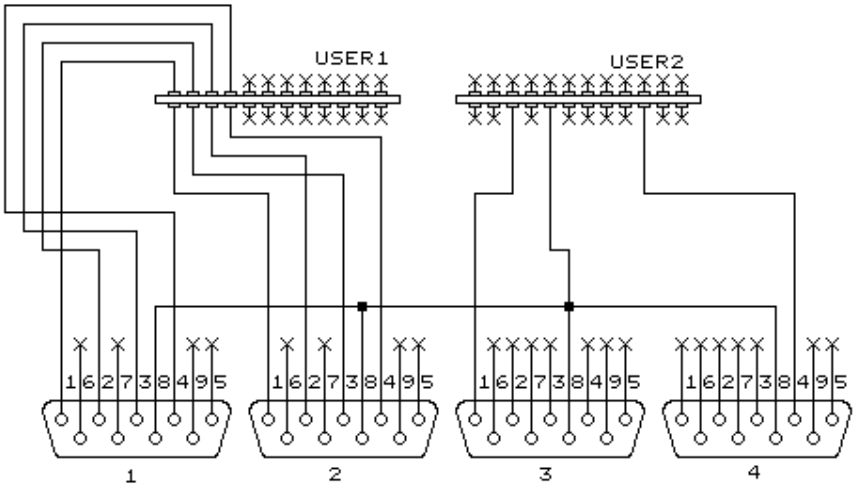
```
A883: 00                    BRK

           *                TEXT FOR ERROR MESSAGE

A884: 20 2D 52    ERROR     ASC   " -RORRE"
A887: 4F 52 52
A88A: 45
A88B: 9B 9B                 DFB   $9B,$9B

                  LENGTH    EQU (*-1)-ERROR

           *                RECURSIVE PRINT FOR DECIMAL ERRORCODE

A88D: 48          PUTINT    PHA
A88E: 8A                    TXA
A88F: C9 0A                 CMP   #10
A891: 90 0D                 BCC   PUTDIG
A893: A2 FF                 LDX   #-1
*** WARNING: OPERAND OVERFLOW
A895: E9 0A       DIV       SBC   #10
A897: E8                    INX
A898: B0 FB                 BCS   DIV
A89A: 69 0A                 ADC   #10
A89C: 20 8D A8              JSR   PUTINT      RECURSION STEP
A89F: 18                    CLC
A8A0: 6930        PUTDIG    ADC   '0
A8A2: 20 A4 F6              JSR   EOUTCH
A8A5: 68                    PLA
ABAG: 60                    RTS

      *                THE WELL KNOWN CASSETTE READ SECTION JUST A LITTLE
      *                MODIFIED

A939: 8D 08 03              STA   DCBCNT
A93C: A9 00                 LDA   #0
A93E: 8D 09 03              STA   DCBCNT+1
A941: 20 59 E4    JMPSIO    JSR   SIO
A944: 10 0C                 BPL   WRITEND
A946: C6 88                 DEC   RETRY
A948: 30 08                 BMI   WRITEND
A94A: A280                  LDX   #$80
A94C: 8E 03 03              STX   DCBSTA
A94F: 4C 41 A9              JMP   JMPSIO
A952: AC 03 03    WRITEND   LDY   DCBSTA
A955: 60                    RTS

                  BUFADR    EQU   *
PHYSICAL ENDADDRESS: $A956

SECTR       $80                         BUFFER      $84
RETRY       $88                         DCBSBI      $0300
DCBCMD      $0302                       DCBBUF      $0304
DCBCNT      $0308                       ICCMD       $0342
ICBAH       $0345                       ICBLH       $0349
ICAX2       $034B                       GETCHR      $07
RMODE       $04                         CIO         $E456
EOUTCH      $F6A4                       EOF         $88
MAIN        $A800       UNUSED          QEOF        $A82A
```

```
READY       $A867
LENGTH      $08
OPENCASS    $A8A7
CFILE       $A903
BL!FADR     $A956
BUFLEN      $86
DCBDRV      $0301
DCBTO       $0306
ICBAL       $0344
ICAX1       $034A
CLOSE       $0C
SIO         $E459
LOCBNUM     $0l
WRITLOOP    $A843
ERROR       $A884
PUTDIG      $A8A0
CERR        $A8F6
WRITEND     $A952
ERRROOP     $A86C
DIV         $A895
CLOSCASS    $A8E9
JMPSIO      $A941
DBUFFER     $82
XSAVE       $89
DCBSTA      $0303
DCBSEC      $030A
ICBLL       $0348
OPEN        $03
RECL        $80
EOL         $9B
READLOOP    $A815
IOERR       $A868
PUTINT      $A88D
READCASS    $A8C8
WRITSECT    $A906
```

HOW TO CONNECT YOUR ATARI WITH ANOTHER COMPUTER

CHAPTER 10

The following programs make it possible to communicate between an ATARI and a PET/CBM. The output ports are referenced as PORTA and DATABUS between the two computers. Bit 0 on the ATARI PORTB is the 'hand' of the ATARI and bit 7 on the same port is the 'hand' of the CBM. Now a handshake communication between both can be started. The routines PUT and GET are, in this case, dummies. Further, you need a stop criterium to stop the transfer. See these routines merely as a general outlines and not as complete transfer programs.



The ATARI - CBM / PET connection-wiring diagram

RECEIVE FOR ATARI

```
                PORTB       EQU   $D301
                PBCTL       EQU   $D303
                PORTA       EQU   $D300
                PACTL       EQU   $D302

                PUT         EQU   $3309

                            ORG   $A800

            *               SET BIT 0 ON PORTB AS OUTPUT

A800: A9 30                 LDA   #$30
A802: 8D 03 D3              STA   PBCTL
A805: A9 01                 LDA   #%00000001
A807: 8D 01 D3              STA   PORTB
```

```
A80A: A9 34                 LDA   #$34
A80C: 8D 03 D3              STA   PBCTL

           *                GIVE YOUR 'HAND' TO THE PET

A80F: A9 01       RFD       LDA   #1
A811: 8D 01 D3              STA   PORTB

           *           WAIT UNTIL PET TAKES YOUR 'HAND'

A814: 2C 01 D3    WAITDAV   BIT   PORTB
A817: 30 FB                 BMI   WAITDAV

           *                GET DATA FROM BUS & PUT THEM SOMEWHERE

A819: AD 00 D3              LDA   PORTA
A81C: 20 09 33              JSR   PUT

           *                TAKE YOUR 'HAND' BACK

A81F: A9 00                 LDA   #0
A821: 8D 01 D3              STA   PORTB

           *                WAIT UNTIL 'PETS HAND' IS IN HIS POCKET

A824: 2C01D3      WAITDAVN  BIT   PORTB
A827: 10 FB                 BPL   WAITDAVN

           *                START AGAIN

A829: 4C0FA8                JMP   RFD

PHYSICAL ENDADDRESS: $A82C
*** NO WARNINGS

PORTB       $D301                PORTA       $D300
PUT         $3309                WAITDAV     $A814
PBCTL       $D303                PACTL       $D302       UNUSED
RFD         $A80F                WAITDAVN    $A824
```

SEND FOR PET CBM

```
           PORTB       EQU   $E84F
           PBCTL       EQU   $E843
           PORTA       EQU   $A822
           GET         EQU   $FFCF       USER GET BYTE
```

```
            *                ROUTINE

                             ORG   $033A,$A800

            *                SET BIT 7 ON PET TO OUTPUT

033A: A9 80                  LDA   #%10000000
033C: 8D 43 E8               STA   PBCTL

            *                GET DATA FROM USER PUT IT ON BUS

033F: 20 CF FF    GETDATA    JSR   GET
0342: 8D 22 A8               STA   PORTA

            *                TELL ATARI DATA VALID

0345: A9 00       DAV        LDA   #0
0347: 8D 4F E8               STA   PORTB

            *                WAIT UNTIL ATARI GIVES HIS 'HAND'

034A: AD 4F E8    WAITNRFD   LDA   PORTB
034D: 29 01                  AND   #%00000001
034F: D0 F9                  BNE   WAITNRFD

            *                SHAKE 'HANDS' WITH ATARI

0351: A9 80       DANV       LDA   #%10000000
0353: 8D 4F E8               STA   PORTB

            *                WAIT UNTIL ATARI RELEASE HIS 'HAND'

0356: AD 4F E8    WAITRFD    LDA   PORTB
0359: 29 01                  AND   #%00000001
035B: F0 F9                  BEQ   WAITRFD

            *                START AGAIN WITH DATA

035D: 4C 3F 03               JMP   GETDATA

PHYSICAL ENDADDRESS: $A826
*** NO WARNINGS

PORTB       $E84F
PORTA       $A822
GETDATA     $033F
WAITNRFD    $034A
WAITRfD     $0356
PBCTL       $E843
GET         $FFCF
DAV         $0345       UNUSED
DANV        $0351       UNUSED
```

300 BAUD SERIAL INTERFACE VIA THE ATARI JOYSTICK PORTS

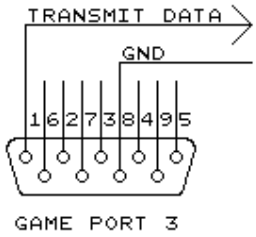
CHAPTER 11

The following construction article allows you to build your own RS232 interface for the ATARI computer. The interface only works with 300 Baud and just in one direction (output).

The interface consists of:

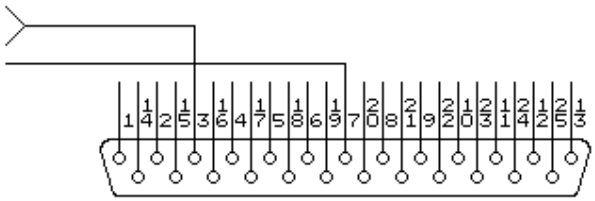
a) RS232 serial interface driver on a bootable cassette or as a SYS file on disk.

b) Two wires hooked up to game port 3 on your ATARI.

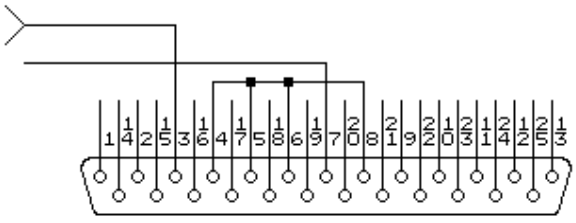


We used this interface with a DEC-writer, a NEC spinwriter, and a Brother HR-15. The DEC-writer worked with just the two wires connected (Transmit DATA and GND).

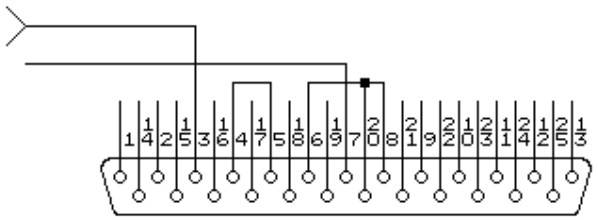
The Spinwriter and the Brother needed some jumper wires as shown below:



Receive data on DEC-writer



Receive DATA on Brother HR-15



Receive DATA on NEC Spinwriter

Depending on the printer you use you will have to make the appropriate wiring according to the instructions in the manual.
The source code for the RS232 driver is listed on a previous page in this book.

This is a sample printout in BASIC:

```
10 OPEN #1,8,0,"R:"
20 FOR X=1 TO 10
30 PRINT #1, "ELCOMP-RS232",X
40 NEXT X
50 CLOSE #1
```

will generate the following printout:

```
ELCOMP-RS232      1
ELCOMP-RS232      1
ELCOMP-RS232      3
ELCOMP-RS232      4
ELCOMP-RS232      5
ELCOMP-RS232      6
ELCOMP-RS232      7
ELCOMP-RS232      8
ELCOMP-RS232      9
ELCOMP-RS232      10
```

The source code for the RS-232 Interface you will find on page 46.

PRINTER INTERFACE

CHAPTER 12

Screen to Printer Interface for the ATARI 400/800

Many ATARI users would like to connect a parallel interface to the computer. For many people buying an interface is too expensive. On the other hand, they may not have the experience to build one by their own. Also a lot of software is needed.

The following instructions make it easy to hook up an EPSON or Centronics printer to the ATARI.

Only seven of the eight DATA bits are used for a printout.

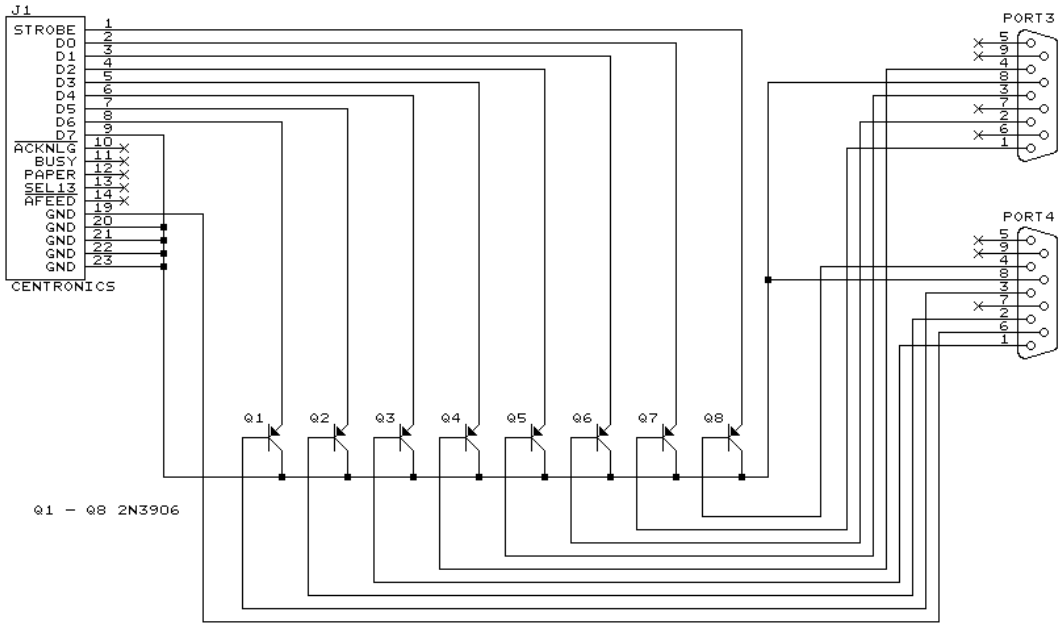
DATA 8 is grounded. BUSY and STROBE are used for handshake. There is an automatic formfeed every 66 lines. Thus it is necessary to adjust the paper before starting to print. You may need to make several trials to find the best position of the paper. For a different form-length you may POKE 1768, ... (number of linesl. After system reset the line counter is set to zero, so you have to provide your own formfeed for a correct paper position.

You can control the length of a line by a POKE 1770,xxx. After doing so, press system reset and enter LPRINT.

The program SCREENPRINT is called by BASIC thru an USR (16701 and by the assembler with a GOTO $0687.

You may install pnp transistors between the game output and the printer.

The figure shows the connection of the ATARI game outlets and the connector for the MX-80 printer. This is a so-called Centronics interface and the program can be used with each printer and this interface.



EPSON MX80 - ATARI 400/800 Interconnection Schematic

The next figure shows the program.

UNIVERSAL PRINT FOR ATARI
400 / 800 VERSION ELCOMP
BY HANS CHRISTOPH WAGNER

```
0600: 00                        DFB   0
0601: 02                        DFB   2
0602: 00 06                     DFW   PST
0604: 6E 06                     DFW   INIT
0606: A9 3C                     LDA   #$3C
0608: 8D 02 D3                  STA   $D302
060B: A9 EB                     LDA   #PND
060D: 8D E7 02                  STA   $02E7
0610: A9 06                     LDA   #PND/256
0612: 8D E8 02                  STA   $02E8
0615: A9 6E                     LDA   #INIT
0617: 85 0A                     STA   $0A
0619: A9 06                     LDA   #INIT/256
061B: 85 0B                     STA   $0B
061D: 18                        CLC
061E: 60                        RTS

061F: 2B 06 42
0622: 06 3F 06
0625: 42 06 3F
0628: 06 3F 06    HANDLTAB DFW DUMMY,
      WRITE-1,RTS1-1,WRITE-1,RTS1-1, RTS1-1

062B: 01          DUMMY       DFB   1
062C: A9 30       OPEN        LDA   #$30
062E: 8D 03 D3                STA   $D303
0631: A9 FF                   LDA   #$FF
0633: 8D 03 D3                STA   $D301
0636: A9 34                   LDA   #$34
0688: 8D 03 D3                STA   $D303
063B: A9 80                   LDA   #$80
063D: 8D 01 D3                STA   $D301
0640: A0    01    RTS1        LDY   #1
0642: 60                      RTS
0643: C9 9B       WRIT        CMP   #$9B
0645: D0 1D                   BNE   PRINT
0647: AD EA 06    CARR        LDA   LINLEN
064A: 8D E9 06                STA   LCOUNT
064D: CE E8 06                DEC   COUNT
0650: 10 0D                   BPL   NOFF
0652: A9 0C                   LDA   #12
0654: 20 64 06                JSR   PRINT
0657: EE E9 06                INC   LCOUNT
065A: A9 41                   LDA   #65
065C: 8D E8 06                STA   COUNT
065F: EE E9 06    NOFF        INC   LCOUNT
0662: A9 0D                   LDA   #13
0664: 20 D1 06    PRINT       JSR   OUTCHAR
0667: CE E9 06                DEC   LCOUNT
```

```
066A: F0 DB                   BEQ   CARR
066C: D0 D2                   BNE   RTS1
066E: A9 1F       INIT        LDA   #HANDLTAB
0670: 8D 1B 03                STA   $031B
0673: A9 06                   LDA   #HANDLTAB/256
0675: 8D 1C 03                STA   $031C
0678: A9 41                   LDA   #65
067A: 8D E8 06                STA   COUNT
067D: AD EA 06                LDA   LINLEN
0680: 8D E9 06                STA   LCOUNT
0683: 4C 2C 06                JMP   OPEN

0686: 68          BASIC       PLA
0687: A5 58       NORMAL      LDA   BASIS
0689: 85 FE                   STA   PT
068B: A5 59                   LDA   BASIS+1
068D: 85 FF                   STA   PT+1
068F: A9 17                   LDA   #23
0691: BD E6 06                STA   ROW
0694: A9 27       ROWLOOP     LDA   #39
0696: 8D E7 06                STA   COLOMN
0699: A2 00                   LDX   #0
069w: A1 FE       LOOP        LDA   (PT,X)
069D: 29 7F                   AND   #$7F
069F: E9 60                   CMP   #$60
06A1: 80 02                   BCS   LOOP1
06A3: 69 20                   ADC   #$20
06A5: 20 D1 06    LOOP1       JSR   OUTCHAR
06A8: E6 FE                   INC   PT
06AA: D0 02                   BNE   *+4
06AE: E6 FF                   INC   PT+1
06AE: CE E7 06                DEC   COLUMN
0681: 10 E8                   BPL   LOOP
06B3: A9 0D                   LDA   #13
06B5: 20 01 06                JSR   OUTCHAR
06B8: EE E6 06                DEC   ROW
06BB: 10 D7                   BPL   ROWLOOP
06BD: 60                      RTS
068E: 48 41 4E
06C1: 53 20 57
06C4: 41 47 4E
06C7: 45 52 20
06CA: 32 37 2E
06CD: 37 2E 38
06D0: 31          AUTHOR      ASC   "HANS WAGNER”

06D1: AC 13 D0    OUTCHAR     LDY   $D013
06D4: DO FB                   BNE   OUTCHAR
06D6: A0 80                   LDY   #$80
06D8: 09 80                   ORA   #$80
O6DA: 8D 01 D3                STA   $D301
06DD: 29 7F                   AND   #$7F
06DF: 8D 01 D3                STA   $D301
06E2: 8C 01 D3                STY   $D301
06E5: 60                      RTS

06E6: 17          ROW         DFB   23
06E7: 27          COLUMN      DFB   39
06E8: 41          COUNT       DFB   6S
06E9: FF          LCOUNT      DFB   255
06EA: FF          LINLEN      DFB   255
```

```
                  PND         EQU   *

BASIS       $58               PT          $FE
PST         $0600             HANDLTAB    $061F
DUMMY       $062B             OPEN        $062C
RTS1        $0640             WRITE       $0643
CARR        $0647             NOFF        $065F
PRlNT       $0664             INIT        $066E
BASIC       $0686 UNUSED      NORMAL      $0687 UNUSED
ROWLOOP     $0694             LOOP        $069B
LOOP1       $06A5             AUTHOR      $06BE UNUSED
OUTCHAR     $06D1             ROW         $06E6
COLUMN      $06E7             COUNT       $06E8
LCOUNT      $06E9             LINLEN      $06EA
PND         $06EB
```

Program description:

Address

0600-061E end of the booting start

0610-082b HANTAB for the ATARI OS

062C-0642 opens the ports for output

0643-066D printer driver

066E-0685 initialize. Now LPRINT and PRINT “P” use thee printer driver.

0686-06BD Label BASIC starting address for a call by BASIC.
Label NORMAL starting address for a call by assembler.

068E-06D0 Copyright notice

06D1-06E5 Subroutine, brings one ASCII character from the accumulator to the printer

O6E6-06EA values for the various counters
ROW sets the number of horizontal lines to 23.
COLUMN sets the number of characters of one line to 39.
COUNT sets the number of lines between two formfeeds to 65
LCOUNT, LINLEN contains the actual parameters for the number of characters and lines.

Boot-Routine

```
PST         EQU   $0600
PND         EQU   $0700
FLEN        EQU   PND-PST+127/128*128
                            ORG   $6000

6000: A2 10       BOOTB     LDX   #$10
6002: A9 03                 LDA   #3
6004: 9D 42 03              STA   $0342,X
6007: A9 08                 LDA   #8
6009: 9D 4A 03              STA   $034A,X
600C: A9 80                 LDA   #$80
600E: 9D 4B 03              STA   $034B,X
6011: A9 4A                 LDA   #CFILE
6013: 9D 44 03              STA   $0344,X
6016: A9 60                 LDA   #CFILE/256
6018: 9D 45 03              STA   $0345,X
```

```
6018: 20 56 E4                JSR   $E456
601E: 30 29                   BMI   CERR
6020: A9 0B                   LDA   #$0B
6022: 9D 42 03                STA   $0322,X
6025: A9 00                   LDA   #PST
6027: 9D 44 03                STA   $0344,X
602A: A9 06                   LDA   #PST/256
602C: 9D 45 03                STA   $0345,X
602F: A9 00                   LDA   #FLEN
6031: 9D 48 03                STA   $0348,X
6034: A9 01                   LDA   #FLEN/256
6036: 9D 49 03                STA   $0349,X
6039: 20 56 E4                JSR   $E456
603C: 30 0B                   BMI   CERR
603E: A9 0C                   LDA   #$0C
6040: 9D 42 03                STA   $0342,X
6043: 20 56 E4                JSR   $E456
6046: 30 01                   BMI   CERR
6048: 00                      BRK
6049: 00            CERR      BRK
604A: 43 3A         CFILE     ASC   "C: “
604E: 9B                      DFB   *9B

PST          $0600
PND          $0700
FLEN         $0100
BOOTB        $6000       UNUSED
CERR         $6049
CFILE        $604A
```

If you want to use this program, it has to be bootable. Therefore you must enter both programs and start the boot routine at address $6000. This will create a bootable cassette, you can use afterwards in the following manner, to enter the SCREENPRINT in your computer.

- turn off the computer
- press the start key
- turn on the computer
- release the start key
- press PLAY on the recorder and
- press RETURN

BASIC or assembler-editor cartridge must be in the left slot of your ATARI computer.

COMMENT:

DIFFERENCES BETWEEN THE ATARI EDITOR/ASSEMBLER CARTRIDGE AND ATAS-1 AND ATMAS-1

The programs in this book are developed using the ATMAS (ATAS) syntax. In the following I would like to explain the difference of some mnemonics of the ATARI Editor/Assembler cartridge and the Editor/Assembler and ATMAS-1 from Elcomp Publishing.
Instead of the asterisk the ATAS uses the pseudo op-codes ORG. Another difference is that the ATAS is screen oriented (no line numbers needed). Instead of the equal sign ATAS uses EQU. Additionally ATAS allows you the pseudo op-codes EPZ: Equal Page Zero.

There is also a difference in using the mnemonics regarding storage of strings within the program.

ATARI		ELCOMP
- BYTE "STRING"	=	ASC "STRING"
- BYTE $	=	DFB $ (Insertion of a byte)
- WORD	=	DFW (Insertion of a word Lower byte, higher byte)

The end of string marker of the ATARI 800/400 output routine is hex 9B.
In the listing you can see, how this command is used in the two assemblers:
ATARI Assembler: - BYTE $9B
ATMAS from ELCOMP - DFB $9B
Depending on what Editor/Assembler from ELCOMP you use, the stringoutput is handled as follows:

1. ATAS 32K and ATAS 48K cassette version

```
LDX # TEXT
LDY # TEXT/256
TEXT ASC "STRING"
DFB$9B
```

2. ATMAS 48K

```
LDX # TEXT:L
LDY # TEXT:H
TEXT ASC "STRING"
DFB $9B
```

There is also a difference between other assemblers and the ATAS-1 or ATMAS-1 in the mnemonic code for shift and relocate commands for the accumulator.
(ASL A = ASL) = OA
(LSR A = LSR) = 4A
ROL A = ROL = 2A
ROR A = ROR = 6A

The ATMAS/ATAS also allows you to comment consecutive bytes as follows:

```
JUMP EQU $F5.7
```

$F5 = Label Jump
$F6 and $F7 are empty locations.

This is a comment and not an instruction.