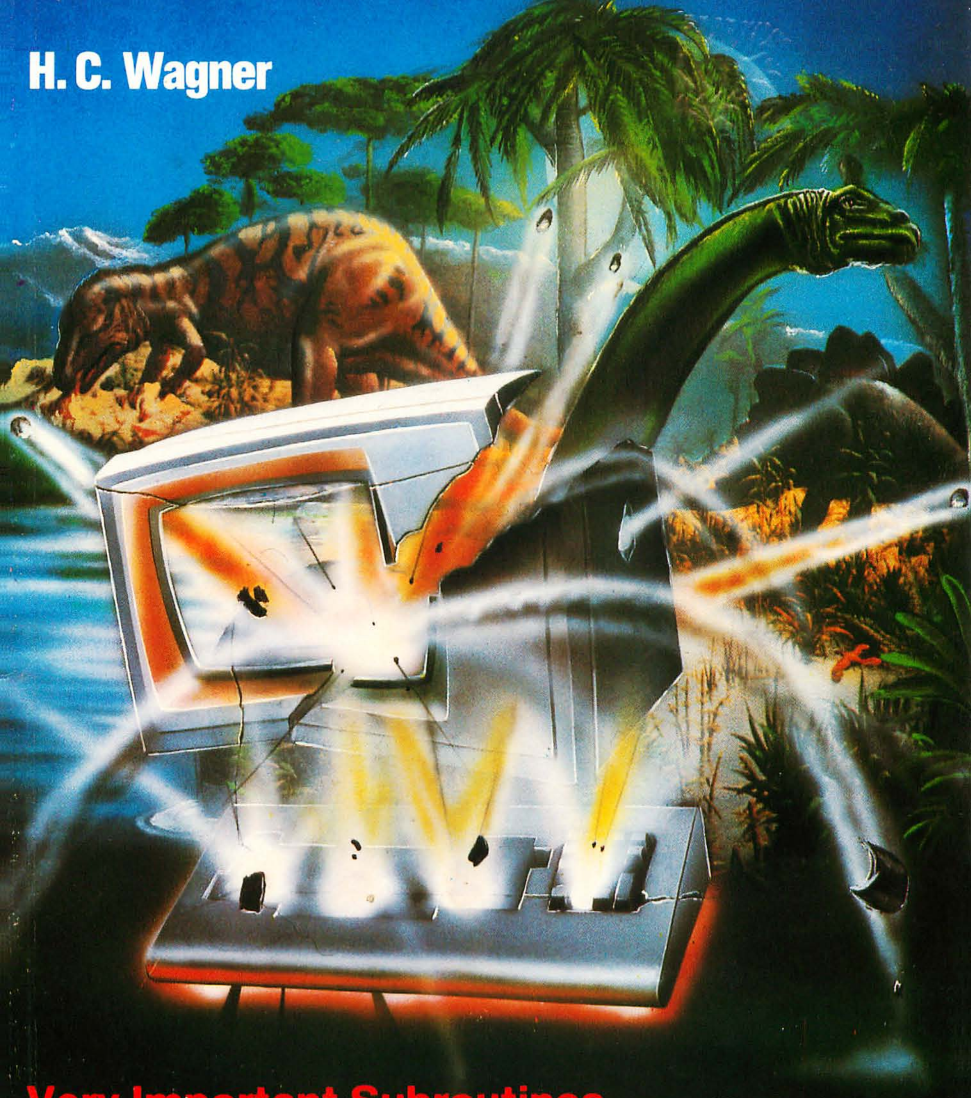
HACKERBOOK

for your ATARI®-Computer
TIPS + TRICKS

H. C. Wagner



Very Important Subroutines in Machine Language

First Edition
First Printing
1983 in the Federal Republic of Germany

Cover design by Franz Berthold

ISBN 3–88963–172–X

Publisher:
Ing. W. HOFACKER GmbH, Tegernseerstr. 18, D-8150 Holzkirchen, W.-Germany

US-Distributor:
ELCOMP Publishing, Inc., 53 Redrock Lane, Pomona CA 91766

PREFACE

Since more and more users of the ATARI personal computers write programs in machine language, more and more "workhorse"-routines, performing standard tasks, are required.

This book contains a variety of programs for the real computer "Hacker" and the machine language programmer.
All the programs have been fully tested and a complete source code is provided.
I extend my thanks to Franz Ende for the translation and Karl Wagner for his proofreading.

Munich, Spring 1983 H. C. Wagner

IMPORTANT NOTICE

This book is written for the experienced ATARI Personal Computer owner. To run the programs you need a symbolic Editor/Assembler or the ATAS/ATMAS from ELCOMP Publishing. For details please refer to the OS-Manual from ATARI.

CONTENTS

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 this 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: A200      HEXIN     LDX #0
A802: 8680                STX EXPR
A804: 8681                STX EXPR+1
A806: 202CA8    HEXIN1    JSR NEXTCH
A809: C930                CMP '0
A80B: 901E                BCC HEXRTS
A80D: C93A                CMP '9+1
A80F: 900A                BCC HEXIN2
A811: C941                CMP 'A
A813: 9016                BCC HEXRTS
A815: C947                CMP 'F+1
A817: B012                BCS HEXRTS
A819: E936                SBC 'A-10-1
A81B: 0A        HEXIN2    ASL
A81C: 0A                  ASL
A81D: 0A                  ASL
A81E: 0A                  ASL
A81F: A204                LDX #4
A821: 0A        HEXIN3    ASL
A822: 2680                ROL EXPR
A824: 2681                ROL EXPR+1
A826: CA                  DEX
A827: D0F8                BNE HEXIN3
A829: F0DB                BEQ HEXIN1  ALWAYS !!
A82B: 60        HEXRTS    RTS

A82C: 20DDF6    NEXTCH    JSR GETCHR
A82F: 20A4F6              JSR SCROUT  SHOW CHARACTE
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

                    SCROUT     EQU $F6A4

                               ORG $A800

A800: A581          PRWORD     LDA EXPR+1
A802: 200BA8                   JSR PRBYTE
A805: A580                     LDA EXPR
A807: 200BA8                   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: 2016A8                   JSR HEXOUT
A813: 68                       PLA
A814: 290F                     AND #%00001111
A816: C90A          HEXOUT     CMP #10
A818: B004                     BCS ALFA
A81A: 0930                     ORA '0
A81C: D002                     BNE HXOUT
A81E: 6936          ALFA       ADC 'A-10-1
A820: 4CA4F6        HXOUT      JMP SCROUT

PHYSICAL ENDADDRESS: $A823

*** NO WARNINGS


EXPR        $80
PRWORD      $A800      UNUSED          SCROUT     $F6A4
HEXOUT      $A816                      PRBYTE     $A80B
HXOUT       $A820                      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: A200       DECIN     LDX #0
A802: 8680                 STX EXPR
A804: 8681                 STX EXPR+1
A806: 2026A8     DEC1      JSR NEXTCH
A809: C930                 CMP '0
A80B: 9018                 BCC DECEND
A80D: C93A                 CMP '9+1
A80F: B014                 BCS DECEND
```

```
A811: 290F                  AND #%00001111
A813: A211                  LDX #17
A815: D005                  BNE DEC3    ALWAYS TAKEN

A817: 9002     DEC2         BCC *+4
A819: 6909                  ADC #10-1
A81B: 4A                    LSR
A81C: 6681     DEC3         ROR EXPR+1
A81E: 6680                  ROR EXPR
A820: CA                    DEX
A821: D0F4                  BNE DEC2
A823: F0E1                  BEQ DEC1    ALWAYS !!
A825: 60       DECEND       RTS

A826: 20DDF6 NEXTCH         JSR GETCHR
A829: 20A4F6                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 ZEROS          *
*                                    *
**************************************

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

                SCROUT   EQU $F6A4

                         ORG $A800


A800: A007      DECOUT   LDY #7
A802: A230      DECOUT1  LDX '0
A804: 38        DECOUT2  SEC
A805: A580               LDA DECL
A807: F92EA8             SBC DECTAB-1,Y
A80A: 48                 PHA
A80B: 88                 DEY
A80C: A581               LDA DECH
A80E: F930A8             SBC DECTAB+1,Y
A811: 9009               BCC DECOUT3
A813: 8581               STA DECH
A815: 68                 PLA
A816: 8580               STA DECL
A818: E8                 INX
A819: C8                 INY
```

```
A81A: D0E8                    BNE DECOUT2
A81C: 68          DECOUT3     PLA
A81D: 8A                      TXA
A81E: 8482                    STY TEMP
A820: 20A4F6                  JSR SCROUT
A823: A482                    LDY TEMP
A825: 88                      DEY
A826: 10DA                    BPL DECOUT1
A828: A580                    LDA DECL
A82A: 0930                    ORA '0
A82C: 4CA4F6                  JMP SCROUT

A82F: 0A00        DECTAB      DFW 10
A831: 6400                    DFW 100
A833: E803                    DFW 1000
A835: 1027                    DFW 10000

PHYSICAL ENDADDRESS: $A837

*** NO WARNINGS

DECL              $80
TEMP              $82
DECOUT            $A800     UNUSED
DECOUT2           $A804
DECTAB            $A82F


DECH              $81
SCROUT            $F6A4
DECOUT1           $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             *
*                                     *
*        EXPR1 := EXPR1 + EXPR2       *
*                                     *
***************************************

                EXPR1     EPZ $80.1
                EXPR2     EPZ $82.3

                          ORG $A800

A800: 18        ADD       CLC
A801: A580                LDA EXPR1
A803: 6582                ADC EXPR2
A805: 8580                STA EXPR1
A807: A581                LDA EXPR1+1
A809: 6583                ADC EXPR2+1
A80B: 8581                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              *
*                                      *
*        EXPR1 := EXPR1 - EXPR2        *
*                                      *
****************************************

                EXPR1     EPZ $80.1
                EXPR2     EPZ $82.3

                          ORG $A800

A800: 38        SUB       SEC
A801: A580                LDA EXPR1
A803: E582                SBC EXPR2
A805: 8580                STA EXPR1
A807: A581                LDA EXPR1+1
A809: E583                SBC EXPR2+1
A80B: 8581                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. Let'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             *
        *                                    *
        *       EXPR1 := EXPR1 * EXPR2       *
        *                                    *
        **************************************

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

                           ORG $A800

A800: A200       MUL       LDX #0
A802: 8684                 STX SCRATCH
A804: 8685                 STX SCRATCH+1
A806: A010                 LDY #16
A808: D00D                 BNE MUL2      ALWAYS !!
A80A: 18         MUL1      CLC
A80B: A584                 LDA SCRATCH
A80D: 6582                 ADC EXPR2
A80F: 8584                 STA SCRATCH
A811: A585                 LDA SCRATCH+1
A813: 6583                 ADC EXPR2+1
A815: 8585                 STA SCRATCH+1
A817: 4685       MUL2      LSR SCRATCH+1
A819: 6684                 ROR SCRATCH
A81B: 6681                 ROR EXPR1+1
A81D: 6680                 ROR EXPR1
A81F: 88                   DEY
A820: 3004                 BMI MULRTS
A822: 90F3                 BCC MUL2
A824: B0E4                 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. Therefor, 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: A200     DIV      LDX #0
A802: 8684              STX REMAIN
```

```
A804: 8685              STX REMAIN+1
A806: A010              LDY #16
A808: 0680    DIV1      ASL EXPR1
A80A: 2681              ROL EXPR1+1
A80C: 2684              ROL REMAIN
A80E: 2685              ROL REMAIN+1
A810: 38                SEC
A811: A584              LDA REMAIN
A813: E582              SBC EXPR2
A815: AA                TAX
A816: A585              LDA REMAIN+1
A818: E583              SBC EXPR2+1
A81A: 9006              BCC DIV2
A81C: 8684              STX REMAIN
A81E: 8585              STA REMAIN+1
A820: E680              INC EXPR1
A822: 88      DIV2      DEY
A823: D0E3              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 the 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 RTS-command is executed. By 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: 2016A8      EXAMPLE   JSR PRINT
A803: 544849                ASC \THIS IS AN EXAMPLE\
A806: 532049
A809: 532041
A80C: 4E2045
A80F: 58414D
A812: 504CC5
A815: 60                    RTS

                  *         THE VERY PRINTROUTINE
A816: 68          PRINT     PLA
A817: 8580                  STA AUX
A819: 68                    PLA
A81A: 8581                  STA AUX+1
A81C: A200                  LDX #0
A81E: E680        PRINT1    INC AUX
A820: D002                  BNE *+4
A822: E681                  INC AUX+1
A824: A180                  LDA (AUX,X)
A826: 297F                  AND #$7F
A828: 20A4F6                JSR SCROUT
A82B: A200                  LDX #0
A82D: A180                  LDA (AUX,X)
A82F: 10ED                  BPL PRINT1
A831: A581                  LDA AUX+1
A833: 48                    PHA
A834: A580                  LDA AUX
A836: 48                    PHA
A837: 60                    RTS
```

PHYSICAL ENDADDRESS: $A838

*** NO WARNINGS

```
AUX             $80                    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 IO-ControlBlocks (IOCB). This means that there are 8 IOCB'S starting from $0340. Each of the IOCB's is 16 bytes long.

```
+-----------------+
¶     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:

```
+-----------------+
¶     ICHID       ¶ HANDLER ID
+-----------------+
¶     ICDNO       ¶ DEVICE NUMBER
+-----------------+
¶     ICCMD       ¶ COMMAND
+-----------------+
```

```
¶       ICSTA        ¶ STATUS
+-------------------+
¶       ICBAL        ¶
+-                  -+ BUFFERADR
¶       ICBAH        ¶
+-------------------+
¶       ICPTL        ¶
+-                  -+ PUTADR
¶       ICPTH        ¶
+-------------------+
¶       ICBLL        ¶
+-                  -+ BUFFERLEN
¶       ICBLH        ¶
+-------------------+
¶       ICAX1        ¶ AUX1
+-------------------+
¶       ICAX2        ¶ AUX2
+-------------------+
¶       ICAX3        ¶ Remaining 4 byte
+-------------------+
¶       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 devicenames and pointers to the device-specific handlertable.

```
+----------------+  --+
¶  device name   ¶    ¶
+----------------+    ¶
¶  handler table ¶    +-- one entry
+-             -+     ¶
¶     address    ¶    ¶
+----------------+  --+
¶     other      ¶
*                *
¶     entries    ¶
+----------------+
¶  zero fill to  ¶
*                *
¶  end of table  ¶
+----------------+
```

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 ($9B)
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/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 databuffer
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 RECORD-operation
BUFFERLEN = number of bytes read to buffer 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
BUFFERADR = 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/file-controller 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 after 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 signflag 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 cassette-device 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).

```
+---------------+
¶ ¶ ¶ ¶ ¶W¶R¶ ¶ ¶   AUX1
+---------------+
 7 6 5 4 3 2 1 0
```

ICCMD will be changed into the PUT CHARACTERS-command ($0B), BUFADR 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 self-explaining.

```
*************************************
*                                   *
*       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 12

                WMODE     EQU 8
                RECL      EQU 128

                CIO       EQU $E456

                EOL       EQU $9B

                IOCBNUM   EQU 1


                          ORG $A800

                *         OPEN FILE

A800: A210                LDX #IOCBNUM*16
A802: A903                LDA #OPEN
A804: 9D4203              STA ICCMD,X
A807: A908                LDA #WMODE
A809: 9D4A03              STA ICAX1,X
A80C: A980                LDA #RECL
A80E: 9D4B03              STA ICAX2,X
```

```
A811: A956              LDA #CFILE:L
A813: 9D4403            STA ICBAL,X
A816: A9A8              LDA #CFILE:H
A818: 9D4503            STA ICBAH,X
A81B: 2056E4            JSR CIO
A81E: 3029              BMI CERR

                *       PUT BUFFER IN RECORDS
                *       TO CASSETTE

A820: A90B              LDA #PUTCHR
A822: 9D4203            STA ICCMD,X
A825: A580              LDA BUFFER
A827: 9D4403            STA ICBAL,X
A82A: A581              LDA BUFFER+1
A82C: 9D4503            STA ICBAH,X
A82F: A582              LDA BUFLEN
A831: 9D4803            STA ICBLL,X
A834: A583              LDA BUFLEN+1
A836: 9D4903            STA ICBLH,X
A839: 2056E4            JSR CIO
A83C: 300B              BMI CERR
                *       CLOSE CASSETTE FILE

A83E: A90C              LDA #CLOSE
A840: 9D4203            STA ICCMD,X
A843: 2056E4            JSR CIO
A846: 3001              BMI CERR

                *       RETURN TO SUPERVISOR

A848: 60                RTS

                *       RETURN WITH ERRORCODE IN
                *       ACCUMULATOR
```

```
A849: 98      CERR    TYA
A84A: 48              PHA
A84B: A90C            LDA #CLOSE
A84D: 9D4203          STA ICCMD,X
A850: 2056E4          JSR CIO
A853: 68              PLA
A854: A8              TAY
A855: 60              RTS

A856: 433A    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: A210            LDX #IOCBNUM*16
A802: A903            LDA #OPEN
A804: 9D4203          STA ICCMD,X
A807: A904            LDA #RMODE
A809: 9D4A03          STA ICAX1,X
A80C: A980            LDA #RECL
A80E: 9D4B03          STA ICAX2,X
A811: A956            LDA #CFILE:L
A813: 9D4403          STA ICBAL,X
A816: A9A8            LDA #CFILE:H
A818: 9D4503          STA ICBAH,X
A81B: 2056E4          JSR CIO
A81E: 3029            BMI CERR

            *         GET BUFFER IN RECORDS
            *         FROM CASSETTE

A820: A907            LDA #GETCHR
A822: 9D4203          STA ICCMD,X
A825: A580            LDA BUFFER
A827: 9D4403          STA ICBAL,X
A82A: A581            LDA BUFFER+1
A82C: 9D4503          STA ICBAH,X
A82F: A582            LDA BUFLEN
A831: 9D4803          STA ICBLL,X
```

```
A834: A583              LDA BUFLEN+1
A836: 9D4903            STA ICBLH,X
A839: 2056E4            JSR CIO
A83C: 300B              BMI CERR

              *         CLOSE CASSETTE FILE
A83E: A90C              LDA #CLOSE
A840: 9D4203            STA ICCMD,X
A843: 2056E4            JSR CIO
A846: 3001              BMI CERR

              *         RETURN TO SUPERVISOR

A848: 60                RTS

              *         RETURN WITH ERRORCODE IN
              *         ACCUMULATOR

A849: 98      CERR      TYA
A84A: 48                PHA
A84B: A90C              LDA #CLOSE
A84D: 9D4203            STA ICCMD,X
A850: 2056E4            JSR CIO
A853: 68                PLA
A854: A8                TAY
A855: 60                RTS

A856: 433A    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
CFILE       $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 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
+-----------------+
¶     DCBBUF   LO ¶
+-              -+ IO Buffer address
¶     DCBBUF   HI ¶
+-----------------+
¶     DCBTO    LO ¶
+-              -+ Time out count
¶     DCBTO    HI ¶
+-----------------+
```

```
¶     DCBCNT  LO ¶
+-               -+ IO Buffer length
¶     DCBCNT  HI ¶
+-----------------+
¶     DCBSEC  LO ¶
+-               -+ IO Sector number
¶     DCBSEC  HI ¶
+-----------------+
```

- 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 & H) is a special word containing the maximum time for executing a command, so called timeout.
- DCBCNT (L & 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.

The DCB-commands

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
DRIVE 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 READ SECTOR-operation

- PUT SECTOR writes one sector specified by the user without verify.

DCBCMD = $50 = 'P
DCBBUF = points to databuffer
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 and 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 often 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 ($40 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 $A800
```

```
A800: A582   WRITSECT LDA BUFFER
A802: 8D0403          STA DCBBUF
A805: A583            LDA BUFFER+1
A807: 8D0503          STA DCBBUF+1
A80A: A580            LDA SECTR
A80C: 8D0A03          STA DCBSEC
A80F: A581            LDA SECTR+1
A811: 8D0B03          STA DCBSEC+1
A814: A957            LDA 'W     REPLACE "W"
A816: 8D0203          STA DCBCMD BY A "P" IF
A819: A980            LDA #$80   YOU WANT IT
A81B: 8D0303          STA DCBSTA  FAST
A81E: A931            LDA '1
A820: 8D0003          STA DCBSBI
A823: A901            LDA #1
A825: 8D0103          STA DCBDRV
A828: A90F            LDA #15
A82A: 8D0603          STA DCBTO
A82D: A904            LDA #4
A82F: 8584            STA RETRY
A831: A980            LDA #128
A833: 8D0803          STA DCBCNT
A836: A900            LDA #0
A838: 8D0903          STA DCBCNT+1
A83B: 2059E4 JMPSIO   JSR SIO
A83E: 100C            BPL WRITEND
A840: C684            DEC RETRY
A842: 3008            BMI WRITEND
A844: A280            LDX #$80
A846: 8E0303          STX DCBSTA
A849: 4C3BA8          JMP JMPSIO
A84C: AC0303 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: A582     READSECT LDA BUFFER
A802: 8D0403            STA DCBBUF
A805: A583              LDA BUFFER+1
A807: 8D0503            STA DCBBUF+1
A80A: A580              LDA SECTR
A80C: 8D0A03            STA DCBSEC
A80F: A581              LDA SECTR+1
A811: 8D0B03            STA DCBSEC+1
A814: A952              LDA 'R
A816: 8D0203            STA DCBCMD
A819: A940              LDA #$40
A81B: 8D0303            STA DCBSTA
A81E: A931              LDA '1
A820: 8D0003            STA DCBSBI
```

```
A823: A901                LDA #1
A825: 8D0103              STA DCBDRV
A828: A90F                LDA #15
A82A: 8D0603              STA DCBTO
A82D: A904                LDA #4
A82F: 8584                STA RETRY
A831: A980                LDA #128
A833: 8D0803              STA DCBCNT
A836: A900                LDA #0
A838: 8D0903              STA DCBCNT+1
A83B: 2059E4 JMPSIO       JSR SIO
A83E: 100C                BPL READEND
A840: C684                DEC RETRY
A842: 3008                BMI READEND
A844: A240                LDX #$40
A846: 8E0303              STX DCBSTA
A849: 4C3BA8              JMP JMPSIO
A84C: AC0303 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 :

```
+----------------+
¶     unused     ¶  first byte
+----------------+
¶ # of 128 bytes ¶
+----------------+
¶     store      ¶
+-              -+
¶    address     ¶
+----------------+
¶ initialization ¶
+-              -+
¶    address     ¶  sixth byte
+----------------+
¶      boot      ¶
¶  continuation  ¶
¶      code      ¶
```

- The first byte is unused, and should equal zero.
- The second byte contains the length of the program, in records (128 bytes), (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 to 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 DOSVEC ($0A. B) 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 normal 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 OF PROGRAM

         LDA #RESTART:L
         STA DOSVEC
         LDA #RESTART:H
         STA DOSVEC+1        SET RESTART VECTOR IN DOSVECTOR

         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 write-sector 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 <ADR> . 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.

```
+----------------+   $BFFA or
|   cartridge    |   $9FFA
+-              -+
| start address  |
+----------------+
|       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 CARTR.)

         ORG $A000          $8000 FOR RIGHT CARTRIDGE

* THE INITIALIZATION ADDRESS

INIT     RTS                RTS IS THE SHORTEST INITIALIZATION

* THE MAIN PROGRAM

RESTART  EQU *

* THE CARTRIDGE HEADER

         ORG $BFFA          $9FFA FOR RIGHT CARTTRIDGE

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

Sample program for a cartridge: **MEMORY TEST**

```
                   *         MEMORY TEST
                   AUXE      EPZ $FE
                   TEST      EPZ $F0
                   OUTCH     EQU $F6A4

                             ORG $BF00,$9000

BF00: A97D         START     LDA #$7D
BF02: 20A4F6                 JSR OUTCH
BF05: 2064BF                 JSR MESS
BF08: 4D454D                 ASC \MEMORY TEST\
BF0B: 4F5259
BF0E: 205445
BF11: 53D4
BF13: A000                   LDY #00
BF15: 84F0                   STY TEST
BF17: A902                   LDA #02
BF19: 85F1                   STA TEST+1
BF1B: B1F0         TEST1     LDA (TEST),Y
BF1D: 85F2                   STA TEST+2
BF1F: A9AA                   LDA #$AA
BF21: 2059BF                 JSR TST
BF24: A955                   LDA #$55
BF26: 2059BF                 JSR TST
BF29: A900                   LDA #00
BF2B: 2059BF                 JSR TST
BF2E: A9FF                   LDA #$FF
BF30: 2059BF                 JSR TST
BF33: A5F2                   LDA TEST+2
BF35: 91F0                   STA (TEST),Y
BF37: E6F0                   INC TEST
BF39: D0E0                   BNE TEST1
BF3B: E6F1                   INC TEST+1
BF3D: 18                     CLC
BF3E: 90DB                   BCC TEST1

BF40: 2064BF FIN             JSR MESS
BF43: 455252                 ASC \ERROR IN \
```

```
BF46: 4F5220
BF49: 494EA0
BF4C: A5F1              LDA TEST+1
BF4E: 2086BF            JSR PRTBYT
BF51: A5F0              LDA TEST
BF53: 2086BF            JSR PRTBYT
BF56: 4C56BF FINI       JMP FINI

BF59: 85F3   TST        STA TEST+3
BF5B: 91F0              STA (TEST),Y
BF5D: B1F0              LDA (TEST),Y
BF5F: C5F3              CMP TEST+3
BF61: D0DD              BNE FIN
BF63: 60     FRTS       RTS

BF64: 68     MESS       PLA
BF65: 85FE              STA AUXE
BF67: 68                PLA
BF68: 85FF              STA AUXE+1
BF6A: A200              LDX #0
BF6C: E6FE   MS1        INC AUXE
BF6E: D002              BNE *+4
BF70: E6FF              INC AUXE+1
BF72: A1FE              LDA (AUXE,X)
BF74: 297F              AND #$7F
BF76: 20A4F6            JSR OUTCH
BF79: A200              LDX #0
BF7B: A1FE              LDA (AUXE,X)
BF7D: 10ED              BPL MS1
BF7F: A5FF              LDA AUXE+1
BF81: 48                PHA
BF82: A5FE              LDA AUXE
BF84: 48                PHA
BF85: 60                RTS

BF86: 48     PRTBYT     PHA
BF87: 4A                LSR
BF88: 4A                LSR
BF89: 4A                LSR
BF8A: 4A                LSR
BF8B: 2091BF            JSR HEX21
BF8E: 68                PLA
BF8F: 290F              AND #$0F
```

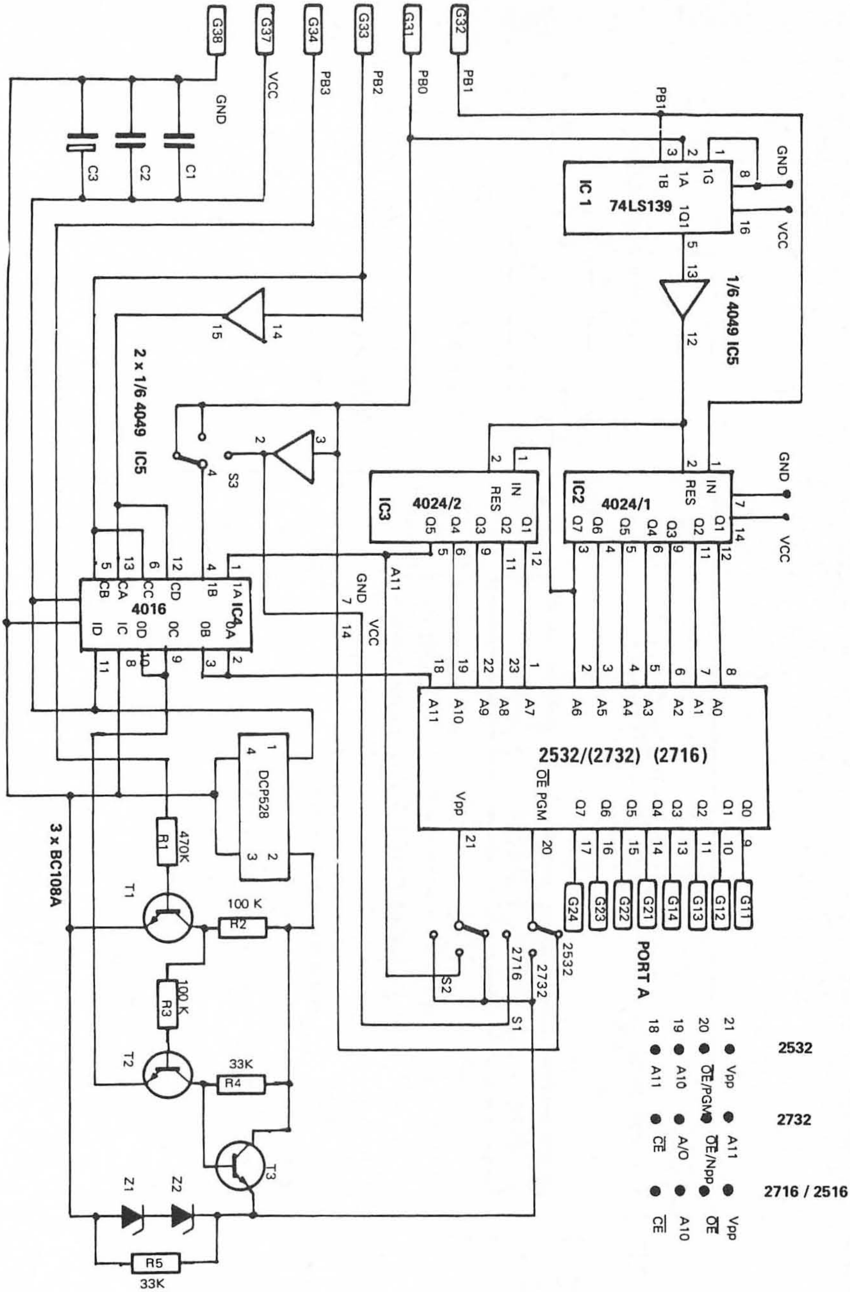
```
BF91: C90A    HEX21     CMP #9+1
BF93: B004              BCS BUCHST
BF95: 0930              ORA '0
BF97: D003              BNE HEXOUT
BF99: 18      BUCHST    CLC
BF9A: 6937              ADC 'A-10
BF9C: 4CA4F6 HEXOUT     JMP OUTCH

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

PHYSICAL ENDADDRESS: $9100

*** NO WARNINGS
```

EPROM-BURNER FOR THE ATARI 800/400®

With this epromburner you can burn your own EPROMS.It is possible to burn four different types. The four types are the 2532(4k), the 2732(4k),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 burning voltage of 25 Volts.This voltage is converted from the 5 Volts of the game port to 28 Volt by the DCDC converter DCP 528. This voltage is limited to 25 Volts by two Zener diodes in serie (ZN 24 and ZN 1). Three universal NPN transistors are used to switch between low level voltages and the high level of the burning voltage.


G38
G37
G34
G33
G31
G32
GND
VCC
PB3
PB2
PB0
PB1
C1
C2
C3
IC 1
74LS139
1/6 4049 IC5
2 x 1/6 4049 IC5
IC2
4024/1
IC3
4024/2
IN
RES
IC4
4016
S3
DCP528
2532/(2732) (2716)
OE PGM
Vpp
R1
470K
T1
100 K
R2
R3
100 K
T2
33K
R4
T3
Z1
Z2
R5
33K
3 x BC108A
S1
S2
2532
2716
2732
PORT A
2532
2732
2716 / 2516
Vpp
OE/PGM
A10
A11
OE/Vpp
A/O
CE

2) ASSEMBLING THE BURNER.

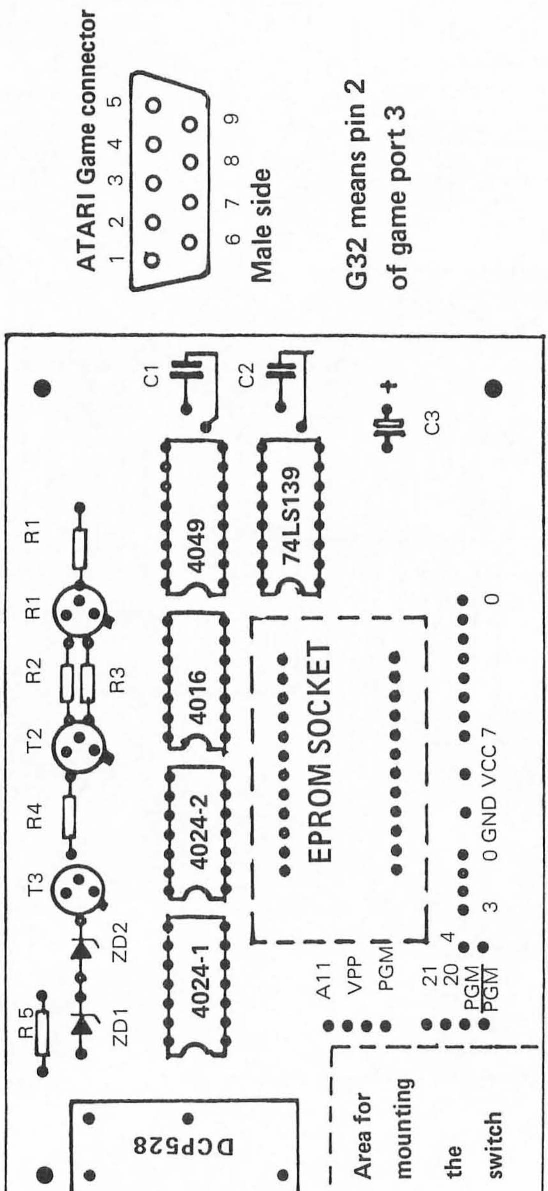


Fig. 2: Parts Layout

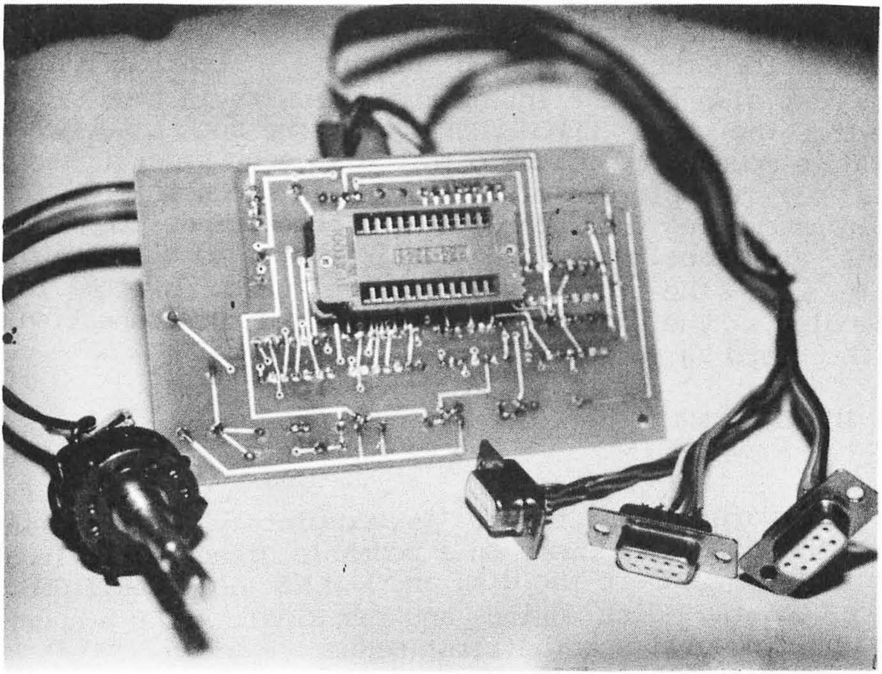


Figure 3

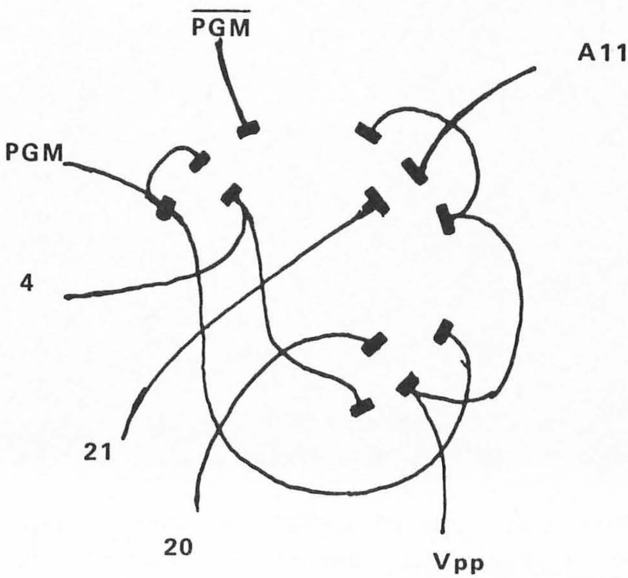
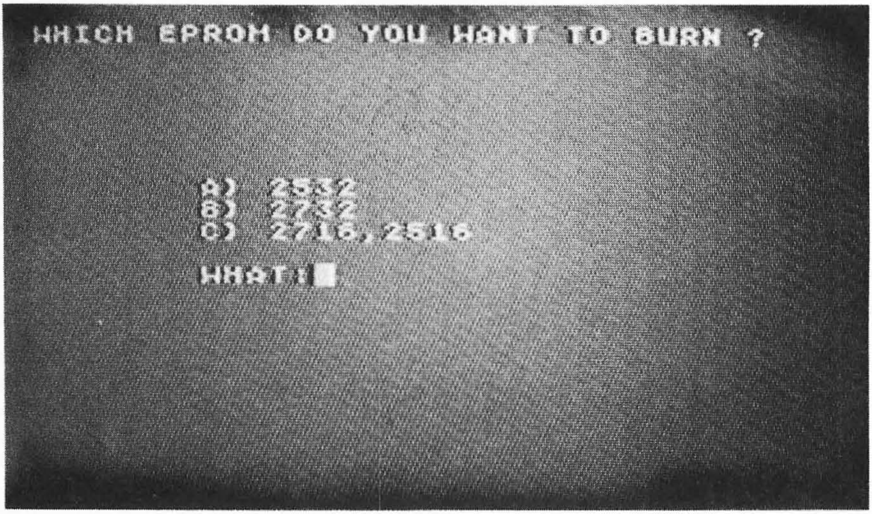


Fig. 4: Rear side of the 3P2T switch

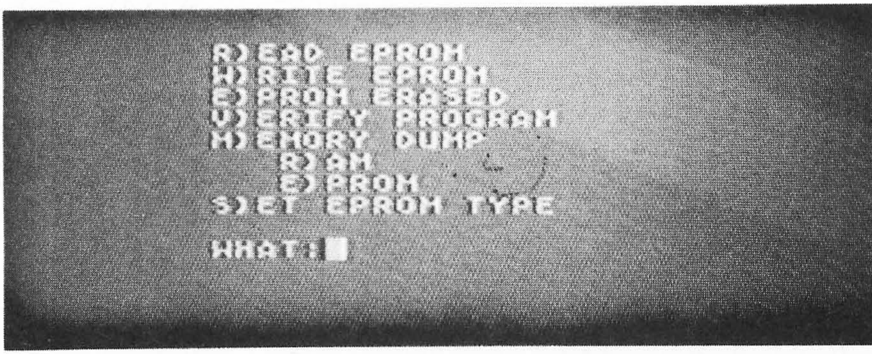
FIG.2 shows the parts layout.It is recomended to use sockets for the integrated circuits. Attention !.The component side for the integrated circuits is the side showing the text EPROMBURNER, but the socket for the EPROM is mounted opposite to this component side. (see FIG. 3) The picture of the burner is shown in FIG. 3. After assembling the board, the connections to the ATARI are made. Use three female plugs and a flatband cable. Last the three pole double throw switch is assembled. The wiring of the switch and the connection to the board is shown in FIG.4.

3) THE SOFTWARE

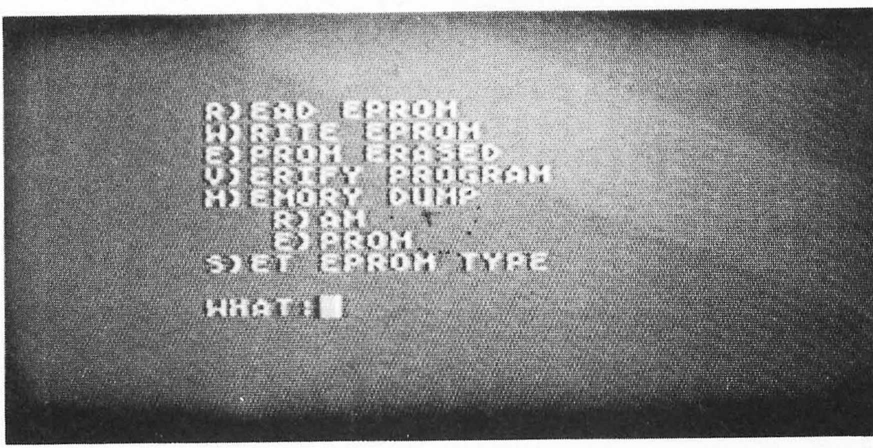
The software for the burner is completely written in machine code. It comes on a bootable diskette. To load the program, insert the disk and REMOVE ALL CARTIDGES. Turn on the disk drive and the ATARI. After a short moment, you will see the first menue:



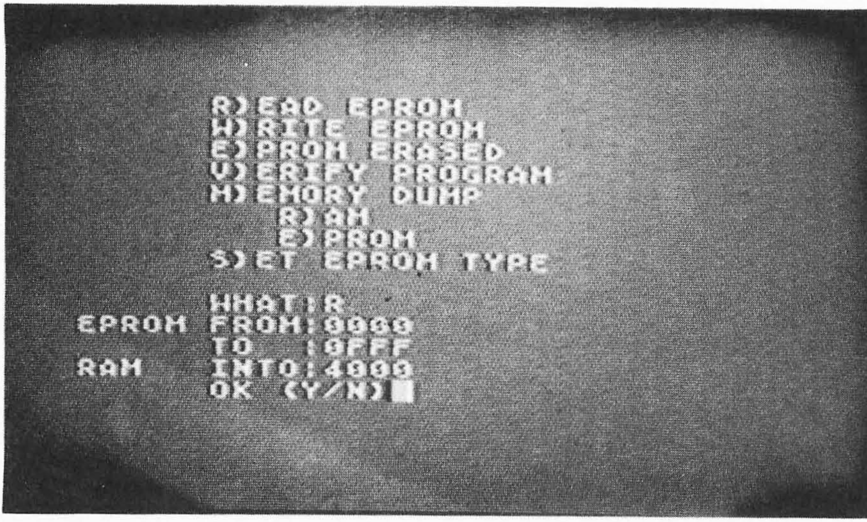
You are asked what type of EPROM you want to burn. After typing the appropriate character, you get the message to set the switch to the correct position and insert the EPROM. This is shown in the following example:



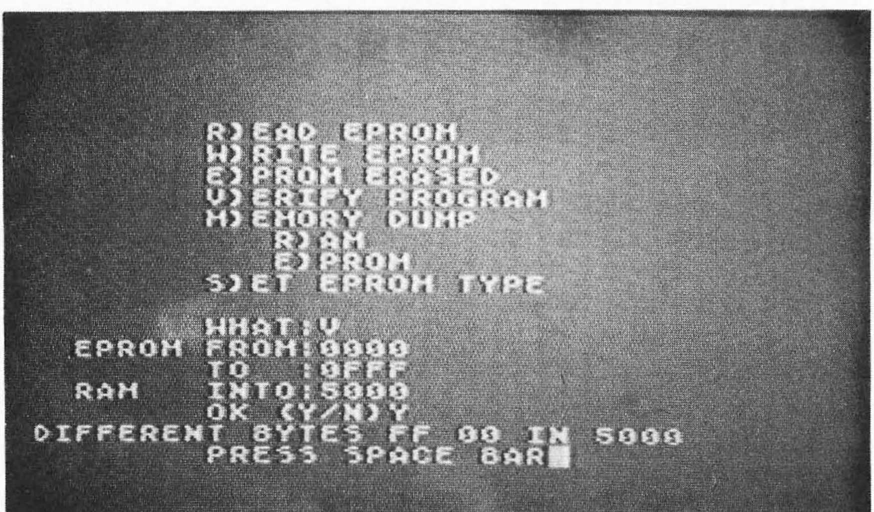
Then, pressing the space bar, you see the main menue:



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 have to type the address INTO which the content of the EPROM is read. All adresses 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, the X key. This key cancels the input and leads back into the main menue. An example of reading an EPROM is shown in the next figure:



To verify that the content of the RAM is idetical to the content of the EPROM, type V. After specifing the adresses 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 it 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 recieve 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 changing the type of EPROM you want to burn, type S. The first menue is shown and you can begin a new burning procedure.

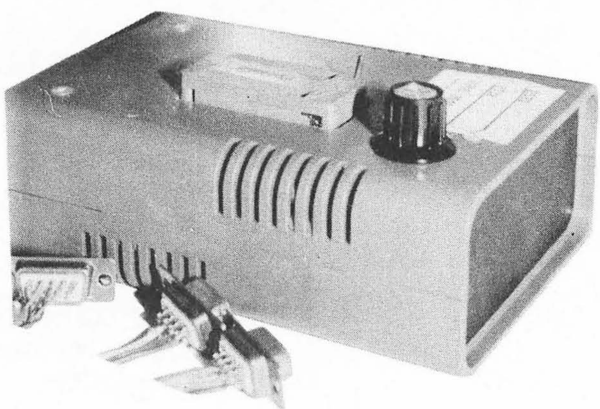
4) PARTS LIST.

IC1	74LS139	
IC2,IC3	4024	
IC4	4016	
IC5	4049	
T1,T2,T3	UNIVERSAL NPN TRANSISTOR 30V,0.3W (2N 3390 % 2N3399)	
R1	470 K	RESISTOR
R2,R3	100 K	RESISTOR
R4,R5	33 K	RESISTOR
Z1	1 V	ZENER DIODE
Z2	24 V	ZENER DIODE
M1	DCP528	DCDC CONVERTER ELPAC POWER SYSTEMS
C1,C2	100 NF	CAPACITOR

C3	10 MF	TANTAL CAPACITOR
S1	3P2T	SWITCH
1	24 PIN	TEXTOOL SOCKET
3	14 PIN	IC SOCKET
2	16 PIN	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 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 TEXTOOL socket.
* The knob should be in the
* upper right corner. Solder the socket.
9. Make the connections on the switch. (FIG.4)
* 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)  $v"
A820    00A1EE10EDA5EF48    @!nPm%oH
A828    A5EE4860A5FD2940    %nH`% )@
A830    F006A5FE0901D004    pF% IAPD
A838    A5FE290E8D01D348    % )NMASH
A840    68AD00D348A5FE8D    h-@SH% M
A848    01D36860A90085F0    ASh`)@Ep
A850    85F185F8A9308D03    EqEx)0MC
A858    D3A90F8D01D385F5    S)OMASEu
A860    A9348D03D3A9FF85    )4MCS) E
A868    F4A9B085F9A9028D    t)0Ey)BM
A870    01D360A99B4CA4F6    AS`)[L$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     $vlJ@)
A978    20A4F62073A8200A     $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    292032373136 2C32   ) 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    323731362C323531    2716,251
AA60    B62073A82073A8A9    6 s( s()
AA68    0A8555200AA84E4F    JEU J(NO
AA70    5720494E53455254    W INSERT
AA78    20455052 4FCD20D7    EPROM W
AA80    AB208FAA4C03A8A9    + O*LC()
AA88    FD20A4F64CEDA920     $vLm)
AA90    73A8A90A8555200A    s()JEU J
AA98    A850524553532053    (PRESS S
```

AAA0	504143452042 41D2	PACE BAR
AAA8	20F0AE602073A8A9	p.` s()
AAB0	0A855520 0AA84F4B	JEU J(OK
AAB8	2028592F4EA920F0	(Y/N) p
AAC0	AE4820A4F668C94E	.H $vhIN
AAC8	F003A90060A90160	pC)@`)A`
AAD0	484A4A4A4A20DBAA	HJJJJ [*
AAD8	68290FC90AB00409	h)OIJODI
AAE0	30D0031869374CA4	0PCXi7L$
AAE8	F6A90085F285F385	v)@ErEsE
AAF0	FEA90485FC20F0AE	)DE| p.
AAF8	48C99BF00320A4F6	HI[pC $v
AB00	68C9303025C94710	hI00%IGP
AB08	21C93A3007C94130	!I:0GIA0
AB10	191869090A0A0A0A	YXiIJJJJ
AB18	A0042A26F226F388	D*&r&sH
AB20	D0F8A98085FEC6FC	Px)@E F|
AB28	D0CB60A9308D02D3	PK`)0MBS
AB30	A9FF8D00D3A9348D	) M@S)4M
AB38	02D360A9308D02D3	BS`)0MBS
AB40	A9008D00D3A9348D	)@M@S)4M
AB48	02D3602073A820FD	BS` s(
AB50	AEA90A8555200AA8	.)JEU J(
AB58	46524F4DBA20E9AA	FROM: i*
AB60	A5FE300DA5F120D0	% 0M%q P
AB68	AAA5F020D0AA4C79	*%p P*Ly
AB70	ABA5F285F0A5F385	+%rEp%sE
AB78	F12073A8A90A8555	q s()JEU
AB80	200AA8544F2020BA	J(TO :
AB88	20E9AAA5FE300DA5	i*% 0M%
AB90	F520D0AAA5F420D0	u P*%t P
AB98	AA4CA4ABA5F285F4	*L$+%rEt
ABA0	A5F385F5A5FB302E	%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   20494EA0A5F120D0    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   @1pM@S )
AD38   ADA5F1C5F59004A5   -%qEuPD%
AD40   F0C5F4F013E6F0D0   pEtpSfpP
AD48   02E6F1A5FC8D01D3   Bfq%|MAS
```

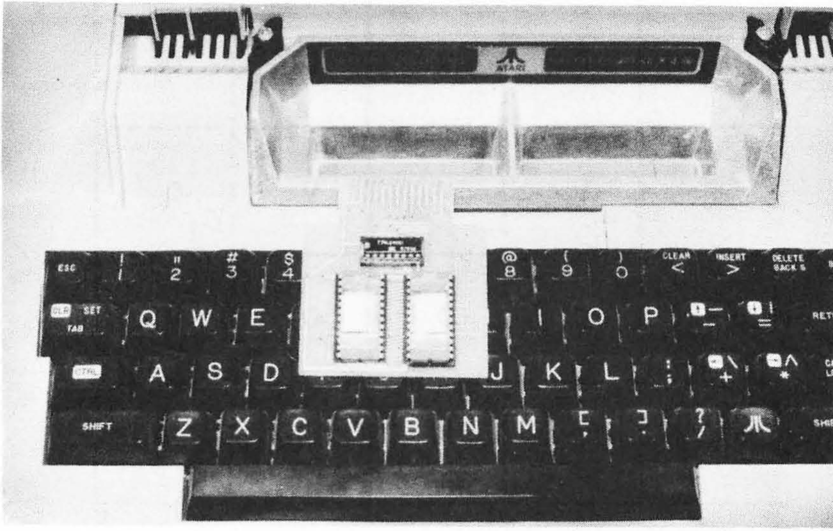
```
AD50    A5FE8D01D31890D7    % MASXPW
AD58    2073A8A90A855520     s()JEU
AD60    0AA850524F475241    J(PROGRA
AD68    4D4D45C4208FAA4C    MMED O*L
AD70    03A82073A8A90A85    C( s()JE
AD78    55200AA843454C4C    U J(CELL
AD80    A0A5F120D0AAA5F0     %q P*%p
AD88    20D0AA200AA8204E     P* J( N
AD90    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)@Ez0H)
AE18    FD20A4F64CFBAD20      $vL{-
AE20    3BABA98085FB204B    ;+)@E{ K
AE28    AB20ACAAD0F820D7    + ,*Px W
AE30    AB2037AE4C03A8A5    + 7.LC(%
AE38    FA10032009ACA97D    zPC I,)
AE40    20A4F6A90085F620     $v)@Ev
AE48    73A8A90085F7A5F1    s()@Ew%q
AE50    85F320D0AAA5F085    Es P*%pE
AE58    F220D0AA20DBAEA5    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    F2D002E6F3D0DBA5    rPBfsP[%
AEB0    F1C5F59004A5F0C5    qEuPD%pE
AEB8    F49006208FAA4C03    tPF O*LC
AEC0    A8E6F0D002E6F1E6    (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    204BAB20ACAAD0F8     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 J(
AF80    56455259464945C4    VERYFIED
AF88    208FAA4C03A82073     O*LC( s
AF90    A8200AA844494646    ( J(DIFF
AF98    4552454E54204259    ERENT BY
AFA0    544553A06820D0AA    TES h P*
AFA8    20DBAEA000B1F820     [. @1x
AFB0    D0AA200AA820494E    P* J( IN
AFB8    A0A5F920D0AAA5F8     %y P*%x
AFC0    20D0AA208FAA4C03     P* O*LC
AFC8    A800000000000000    (@@@@@@@
```

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.



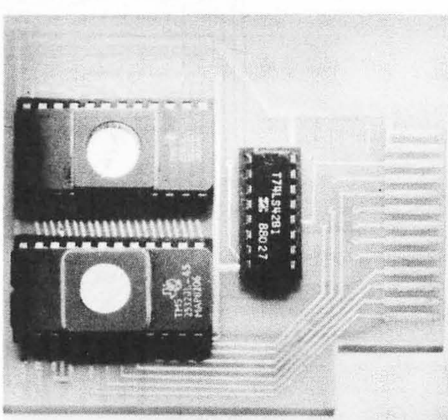
With this versatile ROM module you can use

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.

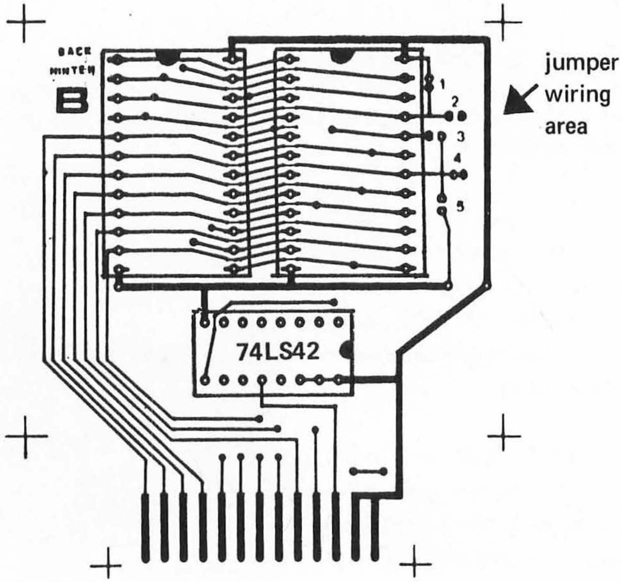
If you use only one EPROM, insert it into the right socket. If you use two EPROMs, put the one with the higher address into the right socket.

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	V	O	V	V
2	O	V	O	O
3	V	V	V	O
4	O	O	O	V
5	0	V	0	0

V = means connected (jumper)
0 = 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 a 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 $D301

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


                          ORG $0600,$A800

                 *        THE HANDLERTABLE

0600: 0F06       HANDLTAB DFW OPEN-1
0602: 2306                DFW CLOSE-1
0604: 2606                DFW GETBYTE-1
0606: 2906                DFW PUTBYTE-1
0608: 2606                DFW STATUS-1
060A: 2606                DFW SPECIAL-1

060C: 000000              DFB 0,0,0,0       FILL REST WITH ZERO
060F: 00

                 *        THE OPEN ROUTINE

                 OPEN     EQU *
0610: A930       INIT     LDA #$30
0612: 8D03D3              STA PACTL
0615: A9FF                LDA #$FF
0617: 8D01D3              STA PORTA
061A: A934                LDA #$34
061C: 8D03D3              STA PACTL
061F: A980                LDA #$80
0621: 8D01D3              STA $D301
0624: A001       SUCCES   LDY #1
0626: 60                  RTS

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

                 CLOSE    EQU SUCCES

0627: A092       NOTIMPL  LDY #146
0629: 60                  RTS

                 *        THE FOLLOWING COMMANDS ARE
                 *        NOT IMPLEMENTED SO GET ERROR
                 *        CODE 146

                 GETBYTE  EQU NOTIMPL
                 STATUS   EQU NOTIMPL
                 SPECIAL  EQU NOTIMPL

                 *        THE PUTBYTE ROUTINE
```

```
062A: C99B    PUTBYTE  CMP #EOL
062C: D007             BNE NOEOL
              *        IF EOL THEN CRLF TO PRINTER

062E: A90D             LDA #CR
0630: 203B06           JSR PARAOUT
0633: A90A             LDA #LF
0635: 203B06  NOEOL    JSR PARAOUT
0638: A001             LDY #1
063A: 60               RTS

              *        THE PARALLEL OUT

063B: AC13D0  PARAOUT  LDY TRIG3
063E: D0FB             BNE PARAOUT       WAIT IF BUSY
0640: A080             LDY #%10000000
0642: 0980             ORA #%10000000
0644: 8D01D3           STA PORTA         STROBE ON AND PUT DATA ON
0647: 297F             AND #%01111111                           BUS
0649: 8D01D3           STA PORTA         STROBE OFF
064C: 8C01D3           STY PORTA         CLEAR BUS
064F: 60               RTS

              *        PUT NEW ADDRESS IN HANDLERVECTOR

0650: A900    INITHAN  LDA #HANDLTAB:L
0652: 8D1B03           STA PRTENTRY+1
0655: A906             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
LF               $0A            HANDLTAB   $0600
OPEN             $0610          INIT       $0610     UNUSED
SUCCES           $0624          CLOSE      $0624
NOTIMPL          $0627          GETBYTE    $0627
STATUS           $0627          SPECIAL    $0627
PUTBYTE          $062A          NOEOL      $0635
PARAOUT          $063B          INITHAN    $0650     UNUSED
```

For more information about the parallel interface refer to page 106.

```
               **************************************
               *                                    *
               *       RS232 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: 0F06     HANDLTAB DFW OPEN-1
0602: 2906              DFW CLOSE-1
0604: 2C06              DFW GETBYTE-1
0606: 2F06              DFW PUTBYTE-1
0608: 2C06              DFW STATUS-1
060A: 2C06              DFW SPECIAL-1
060C: 000000            DFB 0,0,0,0       JUST FILL WITH ZERO
060F: 00

               *        THE OPEN ROUTINE

               OPEN     EQU *
0610: A930     INIT     LDA #$30
0612: 8D03D3            STA PACTL
0615: A901              LDA #%00000001
0617: 8D01D3            STA PORTA
061A: A934              LDA #$34
061C: 8D03D3            STA PACTL
061F: A900              LDA #$00
0621: 8D01D3            STA PORTA
0624: 208506            JSR BITWAIT
0627: 208506            JSR BITWAIT
062A: A001     SUCCES   LDY #1
062C: 60                RTS

               *        THE CLOSE ROUTINE IS A DUMMY
               *        BUT Y=1 (SUCCESSFULL CLOSE)

               CLOSE    EQU SUCCES

062D: A092     NOTIMPL  LDY #146          RETURN WITH Y=146
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: C99B                CMP #EOL
0633: D007                BNE NOEOL

                *         IF EOL GIVE CRLF TO DEVICE

0635: A90D                LDA #CR
0637: 204306              JSR SEROUT
063A: A90A                LDA #LF
063C: 204306 NOEOL        JSR SEROUT
063F: 68                  PLA
0640: A001                LDY #1
0642: 60                  RTS

                *         SERIALOUT FIRST REVERSE BYTE

0643: 49FF      SEROUT    EOR #%11111111
0645: 8DA206              STA BUFFER

                *         DISABLE INTERRUPTS

0648: 78                  SEI
0649: A900                LDA #0
064B: 8D0ED4              STA NMIEN
064E: 8D00D4              STA DMACTL

                *         SEND STARTBIT

0651: A901                LDA #%00000001
0653: 8D01D3              STA PORTA
0656: 208506              JSR BITWAIT

                *         SEND BYTE

0659: A008                LDY #8
065B: 841F                STY COUNT

065D: ADA206 SENDBYTE     LDA BUFFER
0660: 8D01D3              STA PORTA
0663: 6A                  ROR
0664: 8DA206              STA BUFFER
0667: 208506              JSR BITWAIT
066A: C61F                DEC COUNT
066C: D0EF                BNE SENDBYTE

                *         SEND TWO STOPBITS

066E: A900                LDA #%00000000
0670: 8D01D3              STA PORTA
0673: 208506              JSR BITWAIT
0676: 208506              JSR BITWAIT

                *         ENABLE INTERRUPTS
```

```
0679: A922              LDA #$22
067B: 8D00D4            STA DMACTL
067E: A9FF              LDA #$FF
0680: 8D0ED4            STA NMIEN
0683: 58                CLI
0684: 60                RTS

               *        THE BITTIME ROUTINE FOR
               *        AN EXACT BAUDRATE

0685: A296     BITWAIT  LDX #K
0687: A006     LOOPK    LDY #L
0689: 88       LOOPL    DEY
068A: D0FD              BNE LOOPL
068C: CA                DEX
068D: D0F8              BNE LOOPK
068F: 60                RTS

               *        ROUTINE FOR INSTALLING THE
               *        RS232 HANDLER

0690: A952     INITHAN  LDA 'R           DEVICE NAME
0692: 8D2C03            STA RSENTRY
0695: A900              LDA #HANDLTAB:L
0697: 8D2D03            STA RSENTRY+1
069A: A906              LDA #HANDLTAB:H
069C: 8D2E03            STA RSENTRY+2
069F: 4C1006            JMP OPEN

               BUFFER   EQU *            ONE BYTE BUFFER

PHYSICAL ENDADDRESS: $A8A2

*** NO WARNINGS

COUNT          $1F          RSENTRY        $032C
PACTL          $D303        PORTA          $D301
NMIEN          $D40E        DMACTL         $D400
EOL            $9B          CR             $0D
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
SEROUT         $0643        SENDBYTE       $065D
BITWAIT        $0685        LOOPK          $0687
LOOPL          $0689        INITHAN        $0690    UNUSED
BUFFER         $06A2
```

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: A97D      START     LDA #CLS
A802: 20A4F6              JSR SCROUT

                *         PRINT MESSAGE

A805: 2000AA              JSR PRINT
A808: 0D0D                DFB CR,CR
A80A: 424F4F              ASC \BOOTGENERATOR FROM
A80D: 544745                                 HOFACKER\
A810: 4E4552
A813: 41544F
A816: 522046
A819: 524F4D
A81C: 20484F
A81F: 464143
A822: 4B45D2
                *         GET STOREADDRESS
A825: 2000AA              JSR PRINT
A828: 0D0D                DFB CR,CR
A82A: 53544F              ASC \STOREADDRESS :$\
A82D: 524541
A830: 444452
A833: 455353
A836: 203AA4
A839: 2028AA              JSR HEXIN
A83C: 84F0                STY STOREADR
A83E: 85F1                STA STOREADR+1


                *         GET ENDADDRESS
```

```
A840: 2000AA         JSR PRINT
A843: 0D0D0D         DFB CR,CR,CR
A846: 454E44         ASC \ENDADDRESS    :$\
A849: 414444
A84C: 524553
A84F: 532020
A852: 203AA4
A855: 2028AA         JSR HEXIN
A858: 84F2           STY ENDADR
A85A: 85F3           STA ENDADR+1

                *    GET LOGICAL STORE

A85C: 2000AA         JSR PRINT
A85F: 0D0D0D         DFB CR,CR,CR
A862: 4C4F47         ASC \LOGICAL STOREADDRESS :$\
A865: 494341
A868: 4C2053
A86B: 544F52
A86E: 454144
A871: 445245
A874: 535320
A877: 3AA4
A879: 2028AA         JSR HEXIN
A87C: 84FA           STY LOGSTORE
A87E: 85FB           STA LOGSTORE+1

                *    GET JUMP

A880: 2000AA         JSR PRINT
A883: 0D0D0D         DFB CR,CR,CR
A886: 4A554D         ASC \JUMPADDRESS     :$\
A889: 504144
A88C: 445245
A88F: 535320
A892: 202020
A895: 3AA4
A897: 2028AA         JSR HEXIN
A89A: 84F6           STY JMPADR
A89C: 85F7           STA JMPADR+1
```

```
                    *     CALCULATE NEW STORE

A89E: A5F0              LDA STOREADR
A8A0: 38                SEC

A8A1: E920              SBC #(HEADEND-HEAD)+1
A8A3: 85F0              STA STOREADR
A8A5: B002              BCS *+4
A8A7: C6F1              DEC STOREADR+1

                    *     CALCULATE LOGICAL STORE


A8A9: A5FA              LDA LOGSTORE
A8AB: 38                SEC
A8AC: E920              SBC #(HEADEND-HEAD)+1
A8AE: 85FA              STA LOGSTORE
A8B0: B002              BCS *+4
A8B2: C6FB              DEC LOGSTORE+1

                    *     MOVE HEADER IN FRONT OF
                                          PROGRAM

A8B4: 20F5A9            JSR MOVEHEAD


                    *     CALCULATE LENGTHE OF PROGR.

A8B7: A5F2              LDA ENDADR
A8B9: 38                SEC
A8BA: E5F0              SBC STOREADR
A8BC: 85F4              STA PROGLEN
A8BE: A5F3              LDA ENDADR+1
A8C0: E5F1              SBC STOREADR+1
A8C2: 85F5              STA PROGLEN+1
A8C4: B003              BCS *+5
A8C6: 4CDAA9            JMP ADRERR


                    *     ROUND UP TO 128 RECORDS

A8C9: A5F4              LDA PROGLEN
A8CB: 18                CLC
```

```
A8CC: 697F                  ADC #127
A8CE: 2980                  AND #128
A8D0: 85F4                  STA PROGLEN
A8D2: 9002                  BCC *+4
A8D4: E6F5                  INC PROGLEN+1

                  *         CALCULATE NUMBER OF
                                      RECORDS

A8D6: 0A                    ASL
A8D7: A5F5                  LDA PROGLEN+1
A8D9: 2A                    ROL
A8DA: A001                  LDY #RECN-HEAD
A8DC: 91F0                  STA (STOREADR),Y

A8DE: A002                  LDY #PST-HEAD
A8E0: A5FA                  LDA LOGSTORE
A8E2: 91F0                  STA (STOREADR),Y
A8E4: A5FB                  LDA LOGSTORE+1
A8E6: C8                    INY
A8E7: 91F0                  STA (STOREADR),Y

A8E9: A004                  LDY #PINITADR-HEAD
A8EB: 18                    CLC
A8EC: A5FA                  LDA LOGSTORE
A8EE: 691F                  ADC #PINIT-HEAD

A8F0: 91F0                  STA (STOREADR),Y
A8F2: C8                    INY
A8F3: A5FB                  LDA LOGSTORE+1
A8F5: 6900                  ADC #0
A8F7: 91F0                  STA (STOREADR),Y

A8F9: A00C                  LDY #PNDLO-HEAD
A8FB: A5FA                  LDA LOGSTORE
A8FD: 18                    CLC
A8FE: 65F4                  ADC PROGLEN
A900: 91F0                  STA (STOREADR),Y
A902: A011                  LDY #PNDHI-HEAD
A904: A5FB                  LDA LOGSTORE+1
A906: 65F5                  ADC PROGLEN+1
A908: 91F0                  STA (STOREADR),Y
```

```
A90A: A016           LDY #JUMPADRL-HEAD
A90C: A5F6           LDA JMPADR
A90E: 91F0           STA (STOREADR),Y
A910: A01A           LDY #JUMPADRH-HEAD
A912: A5F7           LDA JMPADR+1
A914: 91F0           STA (STOREADR),Y

               *     BOOTTAPE GENERATION PART
               *

               *     GIVE INSTRUCTIONS

A916: 2000AA         JSR PRINT
A919: 0D0D           DFB CR,CR
A91B: 505245         ASC "PRESS PLAY & RECORD"
A91E: 535320
A921: 504C41
A924: 592026
A927: 205245
A92A: 434F52
A92D: 44
A92E: 0D0D           DFB CR,CR
A930: 414654         ASC \AFTER THE BEEPS
A933: 455220                             'RETURN'\
A936: 544845
A939: 204245
A93C: 455053
A93F: 202752
A942: 455455
A945: 524EA7


               *          OPEN CASSETTE FOR WRITE

A948: A210     OPENIOCB LDX #IOCBNUM*16
A94A: A903              LDA #OPEN
A94C: 9D4203            STA ICCOM,X
A94F: A908              LDA #OPNOT
A951: 9D4A03            STA ICAX1,X
A954: A980              LDA #128
A956: 9D4B03            STA ICAX2,X
A959: A9F2              LDA #CFILE:L
```

```
A95B: 9D4403                STA ICBAL,X
A95E: A9A9                  LDA #CFILE:H
A960: 9D4503                STA ICBAH,X
A963: 2056E4                JSR CIOV
A966: 3028                  BMI CERR

                *           PUT PROGRAM ON TAPE

A968: A90B      PUTPROG     LDA #PUTCHR
A96A: 9D4203                STA ICCOM,X
A96D: A5F0                  LDA STOREADR
A96F: 9D4403                STA ICBAL,X
A972: A5F1                  LDA STOREADR+1
A974: 9D4503                STA ICBAH,X
A977: A5F4                  LDA PROGLEN
A979: 9D4803                STA ICBLL,X
A97C: A5F5                  LDA PROGLEN+1
A97E: 9D4903                STA ICBLH,X
A981: 2056E4                JSR CIOV
A984: 300A                  BMI CERR

                *           CLOSE IOCB

A986: A90C      CLOSIOCB    LDA #CLOSE
A988: 9D4203                STA ICCOM,X
A98B: 2056E4                JSR CIOV
A98E: 1024                  BPL SUCCES

                *           IF ERROR OCCURS
                *           SHOW THE ERRORNUMBER

A990: 98        CERR        TYA
A991: 48                    PHA
A992: A210                  LDX #IOCBNUM*16
A994: A90C                  LDA #CLOSE
A996: 9D4203                STA ICCOM,X
A999: 2056E4                JSR CIOV
A99C: 2000AA                JSR PRINT
A99F: 0D0D                  DFB CR,CR
A9A1: 455252                ASC \ERROR- \
A9A4: 4F522D
A9A7: A0
A9A8: 68                    PLA
```

```
A9A9: AA                TAX
A9AA: 2088AA            JSR PUTINT
A9AD: 2000AA            JSR PRINT
A9B0: 8D                DFB CR+128
A9B1: 4CA2AA            JMP WAIT

               *        IF NO ERROR OCCURS
               *        TELL IT THE USER

A9B4: 2000AA SUCCES     JSR PRINT
A9B7: 0D0D              DFB CR,CR
A9B9: 535543            ASC "SUCCESFULL BOOTTAPE GENERATION"
A9BC: 434553
A9BF: 46554C
A9C2: 4C2042
A9C5: 4F4F54
A9C8: 544150
A9CB: 452047
A9CE: 454E45
A9D1: 524154
A9D4: 494F4E
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 SUB-

               *        ROUTINE.

A9D9: 00                BRK

               *        IF ERROR IN THE ADDRESSES
               *        TELL IT THE USER

A9DA: 2000AA ADRERR     JSR PRINT
A9DD: 0D0D              DFB CR,CR
A9DF: 414444            ASC \ADDRESSING ERROR\
A9E2: 524553
A9E5: 53494E
A9E8: 472045
A9EB: 52524F
A9EE: D2
A9EF: 4CA2AA            JMP WAIT

               *        THESE 3 CHARACTERS ARE NEEDED
               *        TO OPEN A CASSETTE IOCB.

A9F2: 433A   CFILE      ASC "C:"
A9F4: 9B                DFB EOL

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

```
A9F5: A01F   MOVEHEAD LDY #HEADEND-HEAD
A9F7: B9A8AA MOVELOOP LDA HEAD,Y
A9FA: 91F0            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: 85F8            STA EXPR
AA03: 68              PLA
AA04: 85F9            STA EXPR+1
AA06: A200            LDX #0
AA08: E6F8   PRINT1   INC EXPR
AA0A: D002            BNE *+4
AA0C: E6F9            INC EXPR+1
AA0E: A1F8            LDA (EXPR,X)
AA10: 297F            AND #%01111111
AA12: C90D            CMP #CR
AA14: D002            BNE NOCR
AA16: A99B            LDA #EOL
AA18: 20A4F6 NOCR     JSR SCROUT
AA1B: A200            LDX #0
AA1D: A1F8            LDA (EXPR,X)
AA1F: 10E7            BPL PRINT1
AA21: A5F9            LDA EXPR+1
AA23: 48              PHA
AA24: A5F8            LDA EXPR
AA26: 48              PHA
AA27: 60              RTS

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


AA28: A900   HEXIN    LDA #0
AA2A: 85F8            STA EXPR
AA2C: 85F9            STA EXPR+1
AA2E: A903            LDA #3
AA30: 85FC            STA HEXCOUNT
AA32: 3031   HEXIN1   BMI HEXRTS
AA34: 20DDF6          JSR GETCHR
AA37: 48              PHA
AA38: 20A4F6          JSR SCROUT
AA3B: 68              PLA
```

```
AA3C: C99B                  CMP #EOL
AA3E: F025                  BEQ HEXRTS
AA40: C958                  CMP 'X
AA42: F096                  BEQ ADRERR
AA44: C930                  CMP '0
AA46: 9022                  BCC HEXERR
AA48: C93A                  CMP '9+1
AA4A: B008                  BCS ALFA
AA4C: 290F                  AND #%00001111
AA4E: 2075AA                JSR HEXROT
AA51: 4C32AA                JMP HEXIN1

AA54: C941    ALFA          CMP 'A
AA56: 9012                  BCC HEXERR
AA58: C947                  CMP 'F+1
AA5A: B00E                  BCS HEXERR
AA5C: 38                    SEC
AA5D: E937                  SBC 'A-10
AA5F: 2075AA                JSR HEXROT
AA62: 4C32AA                JMP HEXIN1

AA65: A4F8    HEXRTS        LDY EXPR
AA67: A5F9                  LDA EXPR+1
AA69: 60                    RTS

              *             IF WRONG DIGIT
              *             RINGS THE  BUZZER
              *             AND PRINT BACKSTEP


AA6A: 200AF9 HEXERR         JSR BELL
AA6D: A91E                  LDA #BST
AA6F: 20A4F6                JSR SCROUT
AA72: 4C32AA                JMP HEXIN1

AA75: C6FC    HEXROT        DEC HEXCOUNT
AA77: 08                    PHP
AA78: A204                  LDX #4
AA7A: 0A                    ASL
AA7B: 0A                    ASL
AA7C: 0A                    ASL
AA7D: 0A                    ASL
AA7E: 0A      HEXROT1       ASL
```

```
AA7F: 26F8                 ROL EXPR
AA81: 26F9                 ROL EXPR+1
AA83: CA                   DEX
AA84: D0F8                 BNE HEXROT1
AA86: 28                   PLP
AA87: 60                   RTS

              *            THE RECURSIVE PUTINT
              *            FOR PRINTING ONE BYTE
```

```
             *         IN DECIMAL FORM

AA88: 48      PUTINT   PHA
AA89: 8A               TXA
AA8A: C90A             CMP #10
AA8C: 900D             BCC PUTDIG—IF A<10 THEN STOP RECURSION
AA8E: A2FF             LDX #-1
*** WARNING: OPERAND OVERFLOW
AA90: E90A    DIV      SBC #10
AA92: E8               INX
AA93: B0FB             BCS DIV
AA95: 690A             ADC #10
AA97: 2088AA           JSR PUTINT— THE RECURSION STEP
AA9A: 18               CLC
AA9B: 6930    PUTDIG   ADC '0
AA9D: 20A4F6           JSR SCROUT
AAA0: 68               PLA
AAA1: 60               RTS

             *         WAIT FOR ANY KEY

AAA2: 20DDF6 WAIT      JSR GETCHR
AAA5: 4C00A8           JMP START

             *         THE BARE CODE FOR THE HEADER
             *         TO PUT IN FRONT OF PROGRAM

             *         THE DUMMY HEADER

             DUMMY     EQU 0

AAA8: 00     HEAD      DFB 0
AAA9: 00     RECN      DFB DUMMY
AAAA: 0000   PST       DFW DUMMY
AAAC: 0000   PINITADR  DFW DUMMY

             *         THE BOOT CONTINUATION CODE

AAAE: A93C             LDA #$3C
AAB0: 8D02D3           STA PACTL
```

```
AAB3: A900                LDA #DUMMY
            PNDLO     EQU *-1
AAB5: 8DE702              STA MEMLO
AAB8: A900                LDA #DUMMY
            PNDHI     EQU *-1
AABA: 8DE802              STA MEMLO+1

AABD: A900                LDA #DUMMY
            JUMPADRL  EQU *-1
AABF: 850A                STA DOSVEC
AAC1: A900                LDA #DUMMY
            JUMPADRH  EQU *-1
AAC3: 850B                STA DOSVEC+1

AAC5: 18                  CLC
AAC6: 60                  RTS

            HEADEND   EQU *
AAC7: 60    PINIT     RTS

PHYSICAL ENDADDRESS: $AAC8

STOREADR      $F0     ENDADR      $F2
PROGLEN       $F4     JMPADR      $F6

EXPR              $F8
HEXCOUNT          $FC
MEMLO             $02E7
ICBAL             $0344
ICBLL             $0348
ICAX1             $034A
OPEN              $03
CLOSE             $0C
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	
HEXIN1	$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	
PRINT1	$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 succesful read the program will be written on the disk. If, during one of these IO's an error occurrs, 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: 20A7A8 MAIN         JSR OPENCASS
A803: 3063                BMI IOERR

                 *        INITIALIZE BUFFERLENGTH &
                 *        BUFFER POINTER

A805: A956                LDA #BUFADR:L
A807: 8584                STA BUFFER
A809: A9A9                LDA #BUFADR:H
A80B: 8585                STA BUFFER+1
A80D: A980                LDA #128
A80F: 8586                STA BUFLEN
A811: A900                LDA #0
A813: 8587                STA BUFLEN+1

                 *        READ RECORD BY RECORD
                 *        TO BUFFER UNTILL EOF REACHED

A815: 20C8A8 READLOOP     JSR READCASS
A818: 3010                BMI QEOF

                 *        IF NO ERROR OR EOF INCREASE
                 *        THE BUFFERPOINTER

A81A: A584                LDA BUFFER
A81C: 18                  CLC
A81D: 6980                ADC #128
A81F: 8584                STA BUFFER
A821: A585                LDA BUFFER+1
A823: 6900                ADC #0
A825: 8585                STA BUFFER+1
A827: 4C15A8              JMP READLOOP

                 *        IF EOF REACHED THEN WRITE
                 *        BUFFER TO DISK
                 *        ELSE ERROR

A82A: C088   QEOF         CPY #EOF
A82C: D03A                BNE IOERR
A82E: 20E9A8              JSR CLOSCASS
A831: 3035                BMI IOERR
```

```
                 *        INIT POINTERS FOR
                 *        SECTOR WRITE

A833: A901                LDA #1
A835: 8580                STA SECTR
A837: A900                LDA #0
A839: 8581                STA SECTR+1
A83B: A956                LDA #BUFADR:L
A83D: 8582                STA DBUFFER
A83F: A9A9                LDA #BUFADR:H
A841: 8583                STA DBUFFER+1

                 *        WRITE SECTOR BY SECTOR
                 *        BUFFER TO DISK

A843: 2006A9 WRITLOOP JSR WRITSECT
A846: 3020                BMI IOERR

                 *        IF BUFFER IS WRITTEN THEN
                 *        STOP PROGRAM

A848: A582                LDA DBUFFER
A84A: C584                CMP BUFFER
A84C: A583                LDA DBUFFER+1
A84E: E585                SBC BUFFER+1
A850: B015                BCS READY

                 *        INCREASE BUFFER AND SECTOR
                 *        POINTERS

A852: A582                LDA DBUFFER
A854: 18                  CLC
A855: 6980                ADC #128
A857: 8582                STA DBUFFER
A859: A583                LDA DBUFFER+1
A85B: 6900                ADC #0
A85D: 8583                STA DBUFFER+1

A85F: E680                INC SECTR
A861: D002                BNE *+4
A863: E681                INC SECTR+1
A865: D0DC                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: 8689       ERRLOOP  STX XSAVE
A86E: BD84A8              LDA ERROR,X
A871: 20A4F6              JSR EOUTCH
```

```
A874: A689              LDX XSAVE
A876: CA                DEX
A877: 10F3              BPL ERRLOOP
A879: 68                PLA
A87A: AA                TAX
A87B: 208DA8            JSR PUTINT
A87E: A99B              LDA #EOL
A880: 20A4F6            JSR EOUTCH

             *          THE BREAK FOR RETURNING
             *          TO THE CALLING MONITOR

A883: 00                BRK

             *          TEXT FOR ERROR MESSAGE


A884: 202D52 ERROR      ASC " -RORRE"
A887: 4F5252
A88A: 45
A88B: 9B9B              DFB $9B,$9B

             LENGTH     EQU (*-1)-ERROR



             *          RECURSIVE PUTINT FOR
             *          DECIMAL ERRORCODE


A88D: 48     PUTINT     PHA
A88E: 8A                TXA
A88F: C90A              CMP #10
A891: 900D              BCC PUTDIG
A893: A2FF              LDX #-1
*** WARNING: OPERAND OVERFLOW
A895: E90A   DIV        SBC #10
A897: E8                INX
A898: B0FB              BCS DIV
A89A: 690A              ADC #10
A89C: 208DA8            JSR PUTINT        RECURSION STEP
A89F: 18                CLC
A8A0: 6930   PUTDIG     ADC '0
A8A2: 20A4F6            JSR EOUTCH
A8A5: 68                PLA
A8A6: 60                RTS

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

```
                 *        OPEN FILE

A8A7: A210    OPENCASS LDX #IOCBNUM*16
A8A9: A903             LDA #OPEN
A8AB: 9D4203           STA ICCMD,X
A8AE: A904             LDA #RMODE
A8B0: 9D4A03           STA ICAX1,X
A8B3: A980             LDA #RECL
A8B5: 9D4B03           STA ICAX2,X
A8B8: A903             LDA #CFILE:L
A8BA: 9D4403           STA ICBAL,X
A8BD: A9A9             LDA #CFILE:H
A8BF: 9D4503           STA ICBAH,X
A8C2: 2056E4           JSR CIO
A8C5: 302F             BMI CERR
A8C7: 60               RTS

                 *        GET BUFFER IN RECORDS
                 *        FROM CASSETTE

A8C8: A210    READCASS LDX #IOCBNUM*16
A8CA: A907             LDA #GETCHR
A8CC: 9D4203           STA ICCMD,X
A8CF: A584             LDA BUFFER
A8D1: 9D4403           STA ICBAL,X
A8D4: A585             LDA BUFFER+1
A8D6: 9D4503           STA ICBAH,X
A8D9: A586             LDA BUFLEN
A8DB: 9D4803           STA ICBLL,X
A8DE: A587             LDA BUFLEN+1
A8E0: 9D4903           STA ICBLH,X
A8E3: 2056E4           JSR CIO
A8E6: 300E             BMI CERR
A8E8: 60               RTS

                 *        CLOSE CASSETTE FILE

A8E9: A210    CLOSCASS LDX #IOCBNUM*16
A8EB: A90C             LDA #CLOSE
A8ED: 9D4203           STA ICCMD,X
A8F0: 2056E4           JSR CIO
A8F3: 3001             BMI CERR
```

```
                *         RETURN TO SUPERVISOR

A8F5: 60                  RTS

                *         RETURN WITH ERRORCODE IN
                *         ACCUMULATOR

A8F6: 98       CERR       TYA
A8F7: 48                  PHA
A8F8: A90C                LDA #CLOSE
A8FA: 9D4203              STA ICCMD,X
A8FD: 2056E4              JSR CIO
A900: 68                  PLA
A901: A8                  TAY
A902: 60                  RTS

A903: 433A     CFILE      ASC "C:"
A905: 9B                  DFB EOL

                *         THE WELL KNOWN WRITE SECTOR
                *         ROUTINE

A906: A582     WRITSECT   LDA DBUFFER
A908: 8D0403              STA DCBBUF
A90B: A583                LDA DBUFFER+1
A90D: 8D0503              STA DCBBUF+1
A910: A580                LDA SECTR
A912: 8D0A03              STA DCBSEC
A915: A581                LDA SECTR+1
A917: 8D0B03              STA DCBSEC+1
A91A: A957                LDA 'W Replace "W" by a "P" if you want it fast
A91C: 8D0203              STA DCBCMD
A91F: A980                LDA #$80
A921: 8D0303              STA DCBSTA
A924: A931                LDA '1
A926: 8D0003              STA DCBSBI
A929: A901                LDA #1
A92B: 8D0103              STA DCBDRV
A92E: A90F                LDA #15
A930: 8D0603              STA DCBTO
A933: A904                LDA #4
A935: 8588                STA RETRY
A937: A980                LDA #128
```

```
A939: 8D0803          STA DCBCNT
A93C: A900            LDA #0
A93E: 8D0903          STA DCBCNT+1
A941: 2059E4 JMPSIO   JSR SIO
A944: 100C            BPL WRITEND
A946: C688            DEC RETRY
A948: 3008            BMI WRITEND
A94A: A280            LDX #$80
A94C: 8E0303          STX DCBSTA
A94F: 4C41A9          JMP JMPSIO
A952: AC0303 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
ERRLOOP          $A86C
LENGTH           $08
DIV              $A895
OPENCASS         $A8A7
CLOSCASS         $A8E9
CFILE            $A903
```

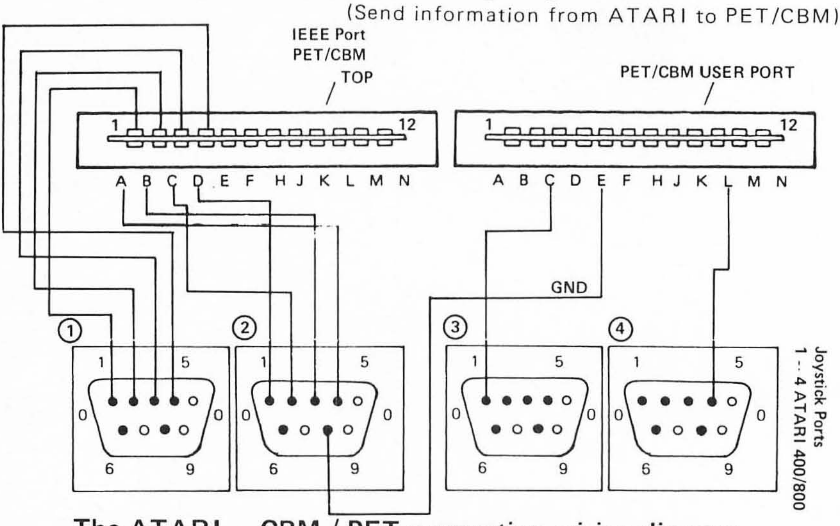
JMPSIO	$A941
BUFADR	$A956
DBUFFER	$82
BUFLEN	$86
XSAVE	$89
DCBDRV	$0301
DCBSTA	$0303
DCBTO	$0306
DCBSEC	$030A
ICBAL	$0344
ICBLL	$0348
ICAX1	$034A
OPEN	$03
CLOSE	$0C
RECL	$80
SIO	$E459
EOL	$9B
IOCBNUM	$01
READLOOP	$A815
WRITLOOP	$A843
IOERR	$A868
ERROR	$A884
PUTINT	$A88D
PUTDIG	$A8A0
READCASS	$A8C8
CERR	$A8F6
WRITSECT	$A906
WRITEND	$A952

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.

(Send information from ATARI to PET/CBM)



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: A930              LDA #$30
A802: 8D03D3            STA PBCTL
A805: A901              LDA #%00000001
A807: 8D01D3            STA PORTB
A80A: A934              LDA #$34
A80C: 8D03D3            STA PBCTL

               *        GIVE YOUR 'HAND' TO THE
               *        PET

A80F: A901     RFD      LDA #1
A811: 8D01D3            STA PORTB

               *        WAIT UNTIL PET TAKES
               *        YOUR 'HAND'

A814: 2C01D3 WAITDAV    BIT PORTB
A817: 30FB              BMI WAITDAV

               *        GET DATA FROM BUS
               *        & PUT THEM SOMEWHERE

A819: AD00D3            LDA PORTA
A81C: 200933            JSR PUT

               *        TAKE YOUR 'HAND' BACK
```

```
A81F: A900                LDA #0
A821: 8D01D3              STA PORTB

                *         WAIT UNTIL 'PETS HAND'
                *         IS IN HIS POCKET

A824: 2C01D3 WAITDAVN BIT PORTB
A827: 10FB                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: A980                  LDA #%10000000
033C: 8D43E8                STA PBCTL

               *            GET DATA FROM USER
               *            PUT IT ON BUS

033F: 20CFFF GETDATA        JSR GET
0342: 8D22A8                STA PORTA

               *            TELL ATARI DATA VALID

0345: A900   DAV            LDA #0
0347: 8D4FE8                STA PORTB

               *            WAIT UNTIL ATARI
               *            GIVES HIS 'HAND'

034A: AD4FE8 WAITNRFD       LDA PORTB
034D: 2901                  AND #%00000001
034F: D0F9                  BNE WAITNRFD

               *            SHAKE 'HANDS' WITH ATARI

0351: A980   DANV           LDA #%10000000
0353: 8D4FE8                STA PORTB

               *            WAIT UNTIL ATARI RELEASE
               *            HIS 'HAND'

0356: AD4FE8 WAITRFD        LDA PORTB
0359: 2901                  AND #%00000001
035B: F0F9                  BEQ WAITRFD

               *            START AGAIN WITH DATA

035D: 4C3F03                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
```

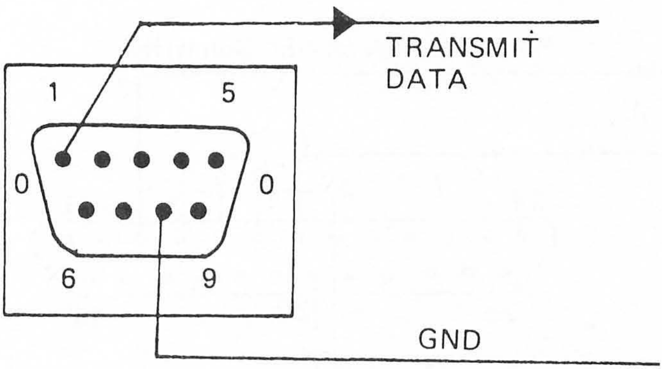
300 BAUD SERIAL INTERFACE VIA THE ATARI JOYSTICK PORTS

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:

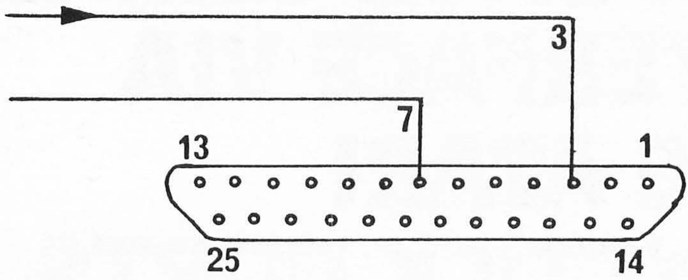
a) RS232 serial interface driver on a bootable cassette or as a SYS file on disk.
b) Two wires hooked up to game port 3 on your ATARI.



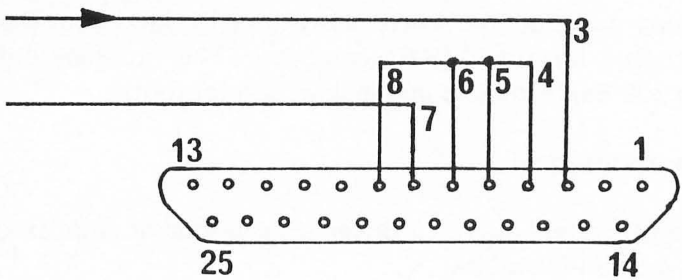
GAME PORT 3

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:

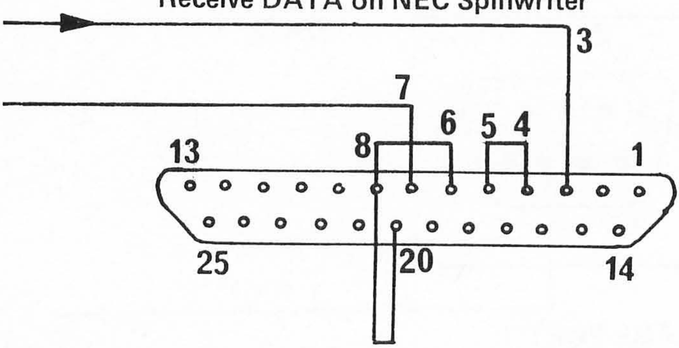
Receive data on DEC-writer



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        2
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 72.

Printer Interface

Chapter 12

Screen to Printer Interface for the ATARI 400/800

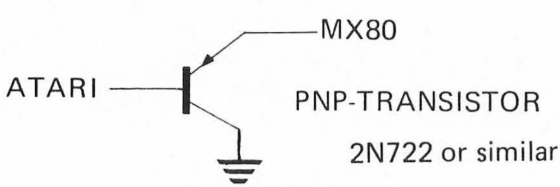
Many ATARI users would like to connect a parallel interface to the computer. For many people buying an interface is too expensive. On the other hand, they may not have the experience to build one by their own. Also a lot of software is needed.

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 lines). 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 (1670) and by the assembler with a GOTO $0687.

You may install pnp transistors between the game output and the printer, as it is shown in this little figure and in the schematic on page 112.



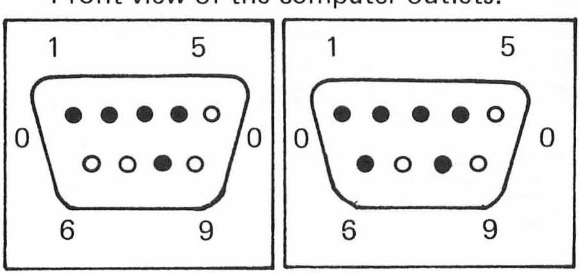
The next figure shows the connection of the ATARI game outlets and the connector for the MX-80 printer. This is a so-called Centronics interface and the program can be used with each printer and this interface.

EPSON MX80 – ATARI 400/800
Interconnection-Scheme

MX80-Connector	ATARI-Connectors Port3	Port 4
Pin#	Pin#	Pin#
1 (19) $\overline{\text{STROBE}}$		4
2 (20) DATA 1	1	
3 (21) DATA 2	2	
4 (22) DATA 3	3	
5 (23) DATA 4	4	
6 (24) DATA 5		1
7 (25) DATA 6		2
8 (26) DATA 7		3
9 (27) DATA 8		8
11 (29) BUSY		6
(GND)	8	8

(19)–(29) = Ground (GND)

Plugs seen from the rear view.
Front view of the computer outlets. [1]



PORT 3 PORT 4

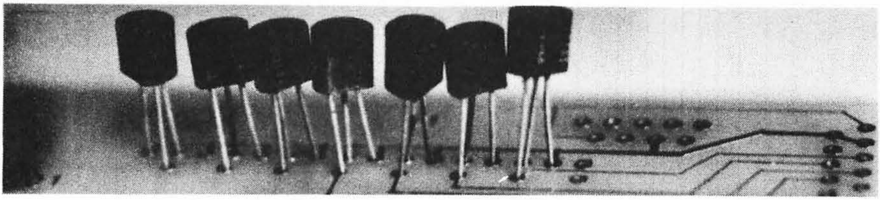
The next figure shows the program.

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

                 BASIS    EPZ $58
                 PT       EPZ $FE
                 PST      EQU $600

                          ORG PST

0600: 00                  DFB 0
0601: 02                  DFB 2
0602: 0006                DFW PST
0604: 6E06                DFW INIT
0606: A93C                LDA #$3C
0608: 8D02D3              STA $D302
060B: A9EB                LDA #PND
060D: 8DE702              STA $02E7
0610: A906                LDA #PND/256
0612: 8DE802              STA $02E8
0615: A96E                LDA #INIT
0617: 850A                STA $0A
0619: A906                LDA #INIT/256
061B: 850B                STA $0B
061D: 18                  CLC
061E: 60                  RTS

061F: 2B0642
0622: 063F06
0625: 42063F
0628: 063F06 HANDLTAB DFW DUMMY,
      WRITE-1,RTS1-1,WRITE-1,RTS1-1,
      RTS1-1
062B: 01        DUMMY     DFB 1
```

```
062C: A930   OPEN    LDA #$30
062E: 8D03D3         STA $D303
0631: A9FF           LDA #$FF
0633: 8D01D3         STA $D301
0636: A934           LDA #$34
0638: 8D03D3         STA $D303
063B: A980           LDA #$80
063D: 8D01D3         STA $D301
0640: A001   RTS1    LDY #1
0642: 60             RTS
0643: C99B   WRITE   CMP #$9B
0645: D01D           BNE PRINT
0647: ADEA06 CARR    LDA LINLEN
064A: 8DE906         STA LCOUNT
064D: CEE806         DEC COUNT
0650: 100D           BPL NOFF
0652: A90C           LDA #12
0654: 206406         JSR PRINT
0657: EEE906         INC LCOUNT
065A: A941           LDA #65
065C: 8DE806         STA COUNT
065F: EEE906 NOFF    INC LCOUNT
0662: A90D           LDA #13
0664: 20D106 PRINT   JSR OUTCHAR
0667: CEE906         DEC LCOUNT
066A: F0DB           BEQ CARR
066C: D0D2           BNE RTS1
066E: A91F   INIT    LDA #HANDLTAB
0670: 8D1B03         STA $031B
0673: A906           LDA #HANDLTAB/256
0675: 8D1C03         STA $031C
0678: A941           LDA #65
067A: 8DE806         STA COUNT
067D: ADEA06         LDA LINLEN
0680: 8DE906         STA LCOUNT
0683: 4C2C06         JMP OPEN

0686: 68     BASIC   PLA
0687: A558   NORMAL  LDA BASIS
0689: 85FE           STA PT
068B: A559           LDA BASIS+1
068D: 85FF           STA PT+1
068F: A917           LDA #23
```

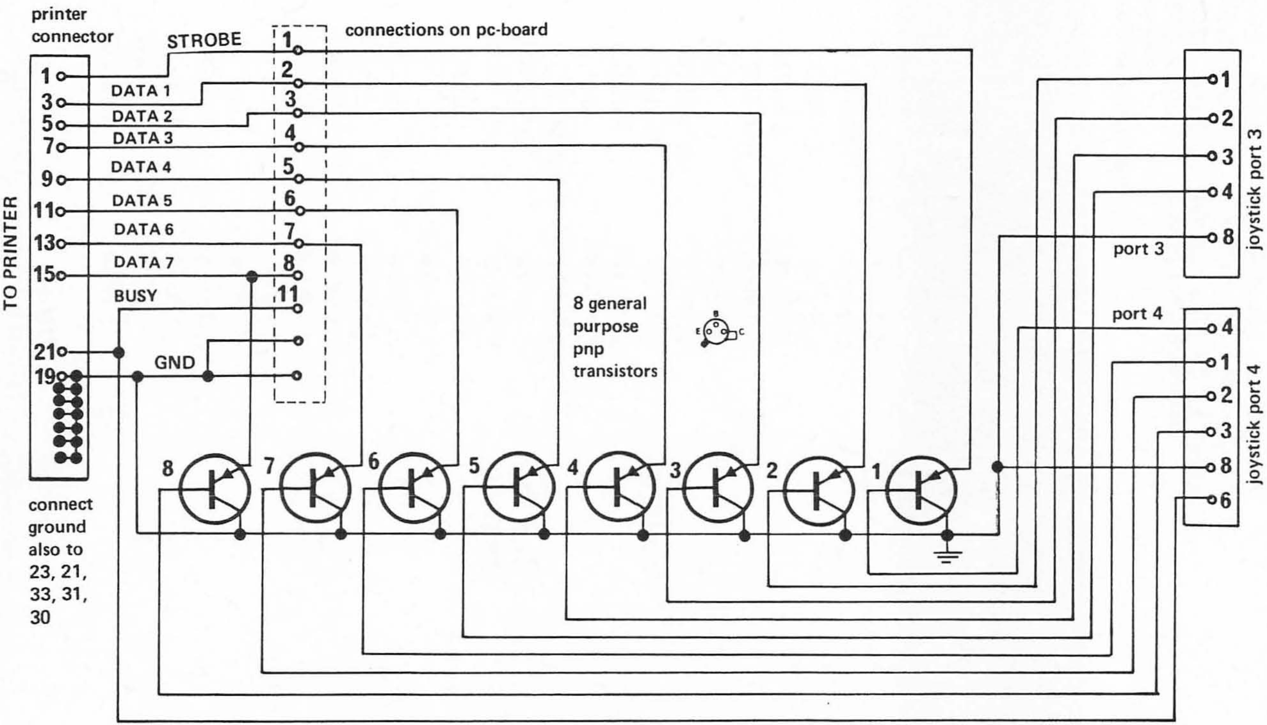
```
0691: 8DE606                    STA ROW
0694: A927   ROWLOOP   LDA #39
0696: 8DE706                    STA COLUMN
0699: A200                      LDX #0
069B: A1FE   LOOP      LDA (PT,X)
069D: 297F                      AND #$7F
069F: C960                      CMP #$60
06A1: B002                      BCS LOOP1
06A3: 6920                      ADC #$20
06A5: 20D106 LOOP1     JSR OUTCHAR
06A8: E6FE                      INC PT
06AA: D002                      BNE *+4
06AC: E6FF                      INC PT+1
06AE: CEE706                    DEC COLUMN
06B1: 10E8                      BPL LOOP
06B3: A90D                      LDA #13
06B5: 20D106                    JSR OUTCHAR
06B8: CEE606                    DEC ROW
06BB: 10D7                      BPL ROWLOOP
06BD: 60                        RTS

06BE: 48414E
06C1: 532057
06C4: 41474E
06C7: 455220
06CA: 32372E
06CD: 372E38
06D0: 31 AUTHOR     ASC "HANS WAGNER

06D1: AC13D0 OUTCHAR   LDY $D013
06D4: D0FB                      BNE OUTCHAR
06D6: A080                      LDY #$80
06D8: 0980                      ORA #$80
06DA: 8D01D3                    STA $D301
06DD: 297F                      AND #$7F
06DF: 8D01D3                    STA $D301
06E2: 8C01D3                    STY $D301
06E5: 60                        RTS

06E6: 17     ROW        DFB 23
06E7: 27     COLUMN     DFB 39
06E8: 41     COUNT      DFB 65
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
PRINT       $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 – 06IE	end of the booting part
0610 – 062B	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 the printer driver
0686 – 06BD	label BASIC starting address for a call by BASIC Label NORMAL starting address for a call by assembler.



Schematic of the parallel printer interface for the EPSON MX-80 or MX100. (Centronics like)

The numbers on the printer connector may vary with the different parallel printer used. In this case go by the name of signal rather than by the numbers.

06BE – 06D0	Copyright notice
06DL – 06E5	Subroutine, brings one ASCII character from the accumulator to the printer
06E6 – 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: A210    BOOTB       LDX #$10
6002: A903                LDA #3
6004: 9D4203              STA $0342,X
6007: A908                LDA #8
6009: 9D4A03              STA $034A,X
600C: A980                LDA #$80
600E: 9D4B03              STA $034B,X
6011: A94A                LDA #CFILE
6013: 9D4403              STA $0344,X
6016: A960                LDA #CFILE/256
6018: 9D4503              STA $0345,X
601B: 2056E4              JSR $E456
601E: 3029                BMI CERR
6020: A90B                LDA #$0B
6022: 9D4203              STA $0342,X
6025: A900                LDA #PST
6027: 9D4403              STA $0344,X
602A: A906                LDA #PST/256
602C: 9D4503              STA $0345,X
602F: A900                LDA #FLEN
```

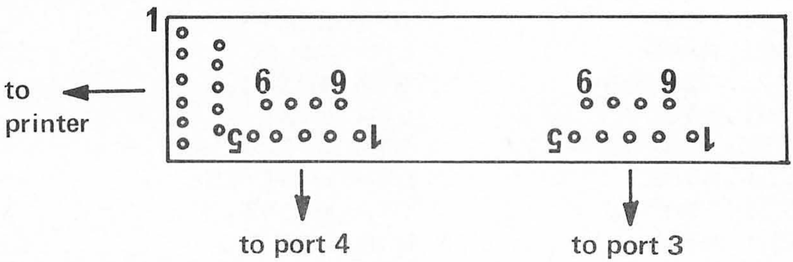
```
6031: 9D4803              STA $0348,X
6034: A901                LDA #FLEN/256
6036: 9D4903              STA $0349,X
6039: 2056E4              JSR $E456
603C: 300B                BMI CERR
603E: A90C                LDA #$0C
6040: 9D4203              STA $0342,X
6043: 2056E4              JSR $E456
6046: 3001                BMI CERR
6048: 00                  BRK
6049: 00       CERR       BRK
604A: 433A     CFILE      ASC "C:"
604C: 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.

How to wire the board:



Differences between the ATARI Editor/Assembler Cartrigde 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) = 0A
(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.

ORDER FORM
HOFACKER

ELCOMP PUBLISHING, INC., 53 Redrock Lane, Pomona, CA 91766 (Phone: (714) 623-8314)

Please make checks out to ELCOMP PUBLISHING,INC.

Name: ....

Address: ....

City / State / Zip: ....

Please bill to my Master Card or Visa account # ....

Card # ....

Expiration Date ....

Master Charge Bank Code ....

Signature ....

Qty.	Order No.	Description	Price $	Qty.	Order No.	Description	Price $
.....	29	MICROC. HARDWARE HANDB.	14.95	.....	4889	EXPANDING YOUR VIC	14.95
.....	150	CARE AND FEEDING...	9.95	.....	4894	RUNFILL	9.95
.....	151	8K MICROSOFT BASIC	9.95	.....	4896	MINIASSEMBLER	19.95
.....	152	EXP.HANDB. 6502 AND 6802	9.95	.....	6153	LEARN-FORTH	19.95
.....	153	MICROC. APPLICATION NOTES	9.95	.....	6155	POWER FORTH (APPLE)	39.95
.....	154	COMPLEX SOUND	6.95	.....	7022	ATMONA-1, CASS.	19.95
.....	156	SMALL BUSINESS PROGR.	14.90	.....	7023	ATMONA-1, DISK	24.95
.....	158	SECOND BOOK OF OHIO	7.95	.....	7024	ATMONA-1, CARTRIDGE	59.00
.....	159	THE THIRD BOOK OF OHIO	7.95	.....			
.....	160	THE FOURTH BOOK OF OHIO	9.95	.....	7042	EPROM BURNER FOR ATARI	179.00
.....	161	THE FIFTH BOOK OF OHIO	7.95	.....	7043	EPROM BOARD (CARTRIDGE)	29.95
.....	162	GAMES FOR THE ATARI	7.95	.....	7049	ATMONA-2, CASS.	49.95
.....	164	ATARI LEARNING BY USING	7.95	.....	7050	ATMONA-2, DISK	54.00
.....	166	PROGR. I.6502 MACH.LANG.	19.95	.....	7053	LEARN-FORTH	19.95
.....	169	HOW TO PROGR.IN MACH.L.	9.95	.....	7055	POWER FORTH	39.95
.....	170	FORTH ON THE ATARI	7.95	.....	7098	ATAS	49.95
.....	171	A LOOK INTO THE FUTURE	9.95	.....	7099	ATMAS	89.00
.....	173	PROGRAM DESCRIPTIONS	4.95	.....	7100	PROGRAMS FROM BOOK # 164	29.95
.....	174	ZX-81/TIMEX	9.95	.....	7200	INVOICE WRITING,DISK	39.00
.....	176	TRICKS FOR VICS	9.95	.....	7207	GUNFIGHT FOR ATARI	19.95
.....	202	JANA/1 MONITOR	29.95	.....	7210	WORDPROCESSOR, CASS.	29.95
.....	604	PROTOTYPING CARD	29.00	.....	7211	HOW TO CONNECT...	19.95
.....	605	6522 VIA I/O EXP. CARD	39.00	.....	7212	MAILING LIST, CASS.	19.95
.....	606	SLOT REPEATER	49.00	.....	7213	MAILING LIST, DISK	24.95
.....	607	2716 EPROM PROGRAMMER	49.00	.....	7214	INVENTORY CONTR., CASS.	19.95
.....	608	SOUND WITH THE GI AY3-8912	39.00	.....	7215	INVENTORY CONTR., DISK	24.95
.....	609	8K EPROM CARD (2716)	29.00	.....	7216	WORDPROCESSOR, DISK	34.95
.....	610	12 BIT A/D CONVERTER BOAR	74.00	.....	7217	WORDPROCESSOR, CARTRIDGE	69.00
.....	615	16K RAMROM-BOARD	59.95	.....	7221	PROGRAMS FROM BOOK # 162	29.95
.....	680	THE CUSTOM APPLE BOOK	24.95	.....	7222	KNAUS OGINO	29.95
.....	2398	MAILING LIST FOR ZX-81	19.95	.....	7223	ASTROLOGY PROGRAM	29.95
.....	2399	MACHINE LANG. MONITOR	9.95	.....	7224	EPROM BOARD KIT	14.94
.....	2400	ADAPTER BOARD FOR ZX-81	14.80	.....	7230	FLOATING POINT PACKAGE	299.95
.....	3276	EDITOR/ASS. FOR CBM	39.00	.....	7291	RS232-INTERFACE	19.95
.....	3475	ASSEMBLER FOR CBM	39.95	.....	7292	EPROM BURNER KIT	49.00
.....	4826	GUNFIGHT FOR PET/CBM	9.95	.....	7307	ATCASH	49.95
.....	4844	UNIVERSAL EXP. BOARD	18.95	.....	7309	MOON PHASES	19.95
.....	4848	ADAPTER BOARD	3.95	.....	7310	ATAMEMO	29.95
.....	4870	PROFI WORDPROC. F. VIC	19.95	.....	7312	SUPERMAIL	49.00
.....	4880	TIC TAC VIC	9.95	.....	7313	BUSIPACK 1	98.00
.....	4881	GAMEPACK FOR VIC	14.95	.....	7314	BIORHYTHM FOR ATARI	9.95
.....	4883	MAIL-VIC	14.95	.....	....	....	....

Payment: Check, Money Order, VISA, Mastercharge, Access, Interbank, Eurocheck
Prepaid orders add $3.50 for shipping (USA)
$5.00 handling fee for C.O.D.
ALL ORDERS OUTSIDE USA: Add 15 % shipping.
California residents add 6.5% sales tax.

BITFIRE

BITFIRE
BITFIRE
BITFIRE
BITFIRE
BITFIRE
BITFIRE