HACKERBOOK

for your ATARI®-Computer
TIPS + TRICKS

H. C. Wagner

Very Important Subroutines in Machine Language
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.
CHAPTER 1
1-1 Input and output of numbers .......................... 1
  1-1-1 Hexadecimal input ................................ 1
  1-1-2 Hexadecimal output ................................ 2
  1-1-3 Decimal input ...................................... 5
  1-1-4 Decimal output ..................................... 6
1-2 16-bit arithmetic without sign ....................... 8
  1-2-1 16-bit addition .................................... 8
  1-2-2 16-bit subtraction ................................ 10
  1-2-3 16-bit multiplication ............................... 10
  1-2-4 16-bit division ..................................... 13

CHAPTER 2
  STRINGOUTPUT ............................................ 15
  2-1 Output of text ....................................... 15

CHAPTER 3
  INTRODUCTION TO CIO ................................. 18
  The standard CIO commands ............................... 21
  How to read or write data in machine-language ....... 24

CHAPTER 4
  INTRODUCTION TO THE DISK-CONTROLLER ............ 31
  The DCB-commands ....................................... 32
  How to write a sector to disk ........................... 34

CHAPTER 5
  HOW TO MAKE A BOOTABLE PROGRAM .................. 39
  How to make a bootable disk ............................ 42
| CHAPTER 6 | HOW TO MAKE A BOOTABLE CARTRIDGE | 43 |
| Sample program for a cartridge: MEMORY TEST | 47 |
| EPROM-BURNER for the ATARI 800/400 | 50 |
| Hexdump of the EPROM BURNER software | 59 |
| Using the EPROM board Kit from Hofacker | 65 |

| CHAPTER 7 | HOW TO ADD OR CHANGE A DEVICE | 67 |

| CHAPTER 8 | A BOOTABLE TAPE GENERATOR PROGRAM | 75 |

| CHAPTER 9 | A DIRECT CASSETTE TO DISK COPY PROGRAM | 89 |

| CHAPTER 10 | HOW TO CONNECT YOUR ATARI WITH ANOTHER COMPUTER | 98 |

| CHAPTER 11 | 300 Baud serial interface via the Atari joystick ports | 103 |

| CHAPTER 12 | Printer Interface | 106 |
| Differences between the ATARI Editor/Assembler cartridge and ATAS-1 and ATMAS-1 | 115 |
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 preceding 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 HEXIN3 LDX #4
A821: 0A ASL
A822: 2680 ROL EXPR
A824: 2681 ROL EXPR+1
A826: CA DEX
A827: D0F8 BNE HEXIN3
A829: 0F0B BEQ HEXIN1 ALWAYS !
A82B: 60 HEXRTS RTS
A82C: 20DDF6 NEXTCH JSR GETCHR
A82F: 20A4F6 JSR SCROUT SHOW CHARACTER
A832: 60 RTS
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

EXPREPZ $80.1
SCROUTEQU $F6A4
ORG $A800

A800: A581 PRWORD LDA EXPR+1
A802: 200BA8JSR PRBYTE
A805: A580LDA EXPR
A807: 200BA8JSR PRBYTE
A80A: 60RTS

THE VERY PRBYTE ROUTINE

A80B: 48 PRBYTEPHA
A80C: 4ALSR
A80D: 4ALSR
A80E: 4ALSR
A80F: 4ALSR
A810: 2016A8JSR HEXOUT
A813: 68PLA
A814: 290FAND #00001111
A816: C90AHEXOUTCMP #10
A818: B004BCS ALFA
A81A: 0930ORA '0
A81C: D002BNE HXOUT
A81E: 6936 ALFAADC 'A-10-1
A820: 4CA4F6 HXOUTJMP SCROUT

PHYSICAL ENDDADDRESS: $A823

*** NO WARNINGS

EXPR $80
PRWORD $A800 UNUSED
HEXOUT $A816
HXOUT $A820
SCROUT $F6A4
PRBYTE $A80B
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 #00000111
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
A81C: 68
A81D: 8A
A81E: 8482
A820: 20A4F6
A823: A482
A825: 88
A826: 10DA
A828: A580
A82A: 0930
A82C: 4CA4F6
A82F: 0A00
A831: 6400
A833: E803
A835: 1027

BNE DECOUT2
DECOUT3 PLA
TXA
STY TEMP
JSR SCROUT
LDY TEMP
DEY
BPL DECOUT1
LDA DECL
ORA '0
JMP SCROUT

DECTAB
DFW 10
DFW 100
DFW 1000
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

EXPRI \ 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 ENDAADDRESS: $A80E

*** NO WARNINGS

EXPR1 $80
ADD $A800 UNUSED
EXPR2 $82
1-2-2 16-bit subtraction

********************************************************************************
* 16 BIT SUBTRACTION *
* UNSIGNED INTEGER *
* EXPRI := EXPRI - EXPR2 *
********************************************************************************

EXPRI  EPZ $80.1
EXPR2  EPZ $82.3
ORG $A800

A800: 38  SUB  SEC
A801: A580  LDA EXPRI
A803: E582  SBC EXPR2
A805: 8580  STA EXPRI
A807: A581  LDA EXPRI+1
A809: E583  SBC EXPR2+1
A80B: 8581  STA EXPRI+1
A80D: 60  RTS

PHYSICAL ENDADDRESS: $A80E

*** NO WARNINGS

EXPR1  $80  EXPRI  $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 ?
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

************ 16 BIT MULTIPLICATION ************

| Address | Value  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A800</td>
<td>A200</td>
</tr>
<tr>
<td>A802</td>
<td>8684</td>
</tr>
<tr>
<td>A804</td>
<td>8685</td>
</tr>
<tr>
<td>A806</td>
<td>A010</td>
</tr>
<tr>
<td>A808</td>
<td>D00D</td>
</tr>
<tr>
<td>A80A</td>
<td>18</td>
</tr>
<tr>
<td>A80B</td>
<td>A584</td>
</tr>
<tr>
<td>A80D</td>
<td>6582</td>
</tr>
<tr>
<td>A80F</td>
<td>8584</td>
</tr>
<tr>
<td>A811</td>
<td>A585</td>
</tr>
<tr>
<td>A813</td>
<td>6583</td>
</tr>
<tr>
<td>A815</td>
<td>8585</td>
</tr>
<tr>
<td>A817</td>
<td>4685</td>
</tr>
<tr>
<td>A819</td>
<td>6684</td>
</tr>
<tr>
<td>A81B</td>
<td>6681</td>
</tr>
<tr>
<td>A81D</td>
<td>6680</td>
</tr>
<tr>
<td>A81F</td>
<td>88</td>
</tr>
<tr>
<td>A820</td>
<td>3004</td>
</tr>
<tr>
<td>A822</td>
<td>90F3</td>
</tr>
<tr>
<td>A824</td>
<td>B0E4</td>
</tr>
<tr>
<td>A826</td>
<td>60</td>
</tr>
</tbody>
</table>

LDX #0
STX SCRATCH
STX SCRATCH+1
LDY #16
BNE MUL2 ALWAYS !!

CLC
LDA SCRATCH
ADC EXPR2
STA SCRATCH
LDA SCRATCH+1
ADC EXPR2+1
STA SCRATCH+1
LSR SCRATCH+1
ROR SCRATCH
ROR EXPR1+1
ROR EXPR1
DEY
BMI MULRTS
BCC MUL2
BCS MUL1
MULRTS RTS
1-2-4 16-bit division

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

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

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

ORG $A800

A800: 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
2-1 Output of text

With most programs it is necessary to display text (menus 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
```

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

<table>
<thead>
<tr>
<th>AUX</th>
<th>$80</th>
<th>SCROUT</th>
<th>$F6A4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>$A800</td>
<td>Unused</td>
<td>PRINT</td>
</tr>
<tr>
<td>PRINT1</td>
<td>$A81E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

\[
\begin{array}{c|c}
\text{IOCB} & \text{Address} \\
\hline
0 & \$0340 \\
1 & \$0350 \\
2 & \$0360 \\
3 & \$0370 \\
4 & \$0380 \\
5 & \$0390 \\
6 & \$03A0 \\
7 & \$03B0 \\
\end{array}
\]

A single IOCB has the following internal scheme:

\[
\begin{array}{c|c}
\text{ICCID} & \text{HANDLER ID} \\
\hline
\text{ICDNO} & \text{DEVICE NUMBER} \\
\hline
\text{ICCMD} & \text{COMMAND} \\
\end{array}
\]
There are just a few locations which are important to the user:

- The command byte which contains the command to be executed by the CIO.
- The buffer address which contains the address of the actual databuffer.
- The buffer length 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 |     |    address     |
+----------------+     +----------------+     +----------------+
|    handler table |     |    address       |     |    other       |
+----------------+     +----------------+     +----------------+
|    other        |     |    other        |     |    entries     |
+----------------+     +----------------+     +----------------+
|    zero fill to |     |    zero fill to |
|    entries      |     |    entries      |
+----------------+     +----------------+     +----------------+
|    zero fill to |     |    zero fill to |
|    entries      |     |    entries      |
+----------------+     +----------------+     +----------------+
|    end of table |     |    end of table |
+----------------+     +----------------+     +----------------+
```

A handlertable looks like:

```
+----------------+
| OPEN-1         |
+----------------+
| CLOSE-1        |
+----------------+
| GETBYTE-1      |
+----------------+
| PUTBYTE-1      |
+----------------+
| GETSTATUS-1    |
+----------------+
| SPECIAL-1      |
+----------------+
| JMP INIT       |
+----------------+
1 & 00           |
+----------------+
```
The CIA 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 CIA the status will be altered. A 1 means a successful operation while a value greater than 128 means an error has occurred.
- PUTADR is used internally by the CIA
- 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 CIA 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 device-dependent information.
AUX2 = device-dependent 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 (B MI 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).

```
+---------------+                                     +---------------+
| 1 1 1 1 | W | R | 1 1 |                                   | 1 1 1 1 | R | 1 1 |
+---------------+                                     +---------------+
  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
A813: 9D4403
A816: A9A8
A818: 9D4503
A81B: 2056E4
A81E: 3029
LDA #CFILE:L
STA ICBAI,X
LDA #CFILE:H
STA ICBAH,X
JSR CIO
BMI CERR

* PUT BUFFER IN RECORDS TO CASSETTE *

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

A83E: A90C
A840: 9D4203
A843: 2056E4
A846: 3001
LDA #CLOSE
STA ICCMD,X
JSR CIO
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
A802: A903
A804: 9D4203
A807: A904
A809: 9D4A03
A80C: A980
A80E: 9D4B03
A811: A956
A813: 9D4403
A816: A9A8
A818: 9D4503
A81B: 2056E4
A81E: 3029

LDX #IOCBNUM*16
LDA #OPEN
STA ICCMD,X
LDA #RMODE
STA ICAX1,X
LDA #RECL
STA ICAX2,X
LDA #CFILE:L
STA ICBAL,X
LDA #CFILE:H
STA ICBAH,X
JSR CIO
BMI CERR

* GET BUFFER IN RECORDS
FROM CASSETTE

A820: A907
A822: 9D4203
A825: A580
A827: 9D4403
A82A: A581
A82C: 9D4503
A82F: A582
A831: 9D4803

LDA #GETCHR
STA ICCMD,X
LDA BUFFER
STA ICBAL,X
LDA BUFFER+1
STA ICBAH,X
LDA BUFLEN
STA ICBLL,X
A834: A583    LDA BUFLEN+1
A836: 9D4903  STA ICBLH,X
A839: 2056E4  JSR CIO
A83C: 300B    BMI CERR

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

* RETURN TO SUPERVISOR
A848: 60      BMI CERR

* RETURN WITH ERRORCODE IN ACCUMULATOR
A849: 98      CERR    TYA
A84A: 48      LDA #CLOSE
A84B: A90C    STA ICCMD,X
A84D: 9D4203  JSR CIO
A850: 2056E4  PLA
A853: 68      TAY
A854: A8      RTS
A855: 60      CFILE ASC "C:"
A856: 433A    DFB EOL
A858: 9B      PHYSICAL ENDDADDRESS: $A859

*** NO WARNINGS
BUFFER      $80    BUFSIZE      $82
ICCMDC      $0342  ICBAH        $0344
ICBAH       $0345  ICBAL        $0348
ICBLH       $0349  ICBLH        $034A
ICAX2       $034B  ICAXI        $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 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      |        |    DCBDRV      |        |    DCBCMD      |
+----------------+        +----------------+        +----------------+
|    Serial bus ID |        |    Disk drive # |        |    Command     |
+----------------+        +----------------+        +----------------+
|    DCBSTA      |        +----------------+        |    IO Status    |
+----------------+        +----------------+        +----------------+
|    DCBBUF LO   |        |    DCBBUF HI   |        |    Time out count |
+----------------+        +----------------+        +----------------+
|    DCBTO LO    |        |    DCBTO HI    |        +----------------+
+----------------+        +----------------+        +----------------+
```

31
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 ENDDADDRESS: $A850

*** NO WARNINGS

SECTR  $80  BUFFER  $82
RETRY  $84  DCBSBI  $0300
DCBDRV  $0301  DCBCMD  $0302
DCBSTA  $0303  DCBBUF  $0304
**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
A825: 8D0103
A828: A90F
A82A: 8D0603
A82D: A904
A82F: 8584
A831: A980
A833: 8D0803
A836: A900
A838: 8D0903
A83B: 2059E4 JMPSIO
A83E: 100C
A840: C684
A842: 3008
A844: A240
A846: 8E0303
A849: 4C3BA8
A84C: AC0303 READEND
A84F: 60

LDA #1
STA DCBDRV
LDA #15
STA DCBTO
LDA #4
STA RETRY
LDA #128
STA DCBCNT
LDA #0
STA DCBCNT+1
JSR SIO
BPL READEND
DEC RETRY
BMI READEND
LDX #$40
STX DCBSTA
JMP JMPSIO
LDY DCBSTA
RTS

LDA #15
STA DCBTO
LDA #4
STA RETRY
LDA #128
STA DCBCNT
LDA #0
STA DCBCNT+1
JSR SIO
BPL READEND
DEC RETRY
BMI READEND
LDX #$40
STX DCBSTA
JMP JMPSIO
LDY DCBSTA
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
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         |
+----------------+
| # of 128 bytes |
+----------------+
| store          |
+--             +
| address        |
+----------------+
| initialization  |
+--             +
| address        |
+----------------+
| 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 INITIALIZATION 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 (SUCCESSFUL 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

\[
\begin{align*}
10101010 & = $AA \\
01010101 & = $55 \\
00000000 & = $00 \\
11111111 & = $FF
\end{align*}
\]

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.

```
+----------------+          +----------------+
l   cartridge    l          l   cartridge    l
+-             +-          +-             +-           
|start address|           | start address |
+----------------+          +----------------+           
| 00            |           | 00            |
+----------------+          +----------------+           
| option byte   |           | option byte   |
+----------------+          +----------------+           
| cartridge     |           | cartridge     |
+-             +-          +-             +-           
| init address  | $BFFF or| init address  |
+----------------+          +----------------+
```

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 CARTRIDGE

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
BF4E: 2086BF
BF51: A5F0
BF53: 2086BF
BF56: 4C56BF FINI
BF59: 85F3 TST
BF5B: 91F0
BF5D: B1F0
BF5F: C5F3
BF61: D0DD
BF63: 60 PRTS
BF64: 68 MESS
BF65: 85FE
BF67: 68
BF68: 85FF
BF6A: A200
BF6C: E6FE
BF6E: D002
BF70: E6FF
BF72: A1FE
BF74: 297F
BF76: 20A4F6
BF79: A200
BF7B: A1FE
BF7D: 10ED
BF7F: A5FF
BF81: 48
BF82: A5FE
BF84: 48
BF85: 60
BF86: 48 PRTBYT
BF87: 4A
BF88: 4A
BF89: 4A
BF8A: 4A
BF8B: 2091BF
BF8E: 68
BF8F: 290F

LDA TEST+1
JSR PRTBYT
LDA TEST
JSR PRTBYT
JMP FINI
STA TEST+3
STA (TEST),Y
LDA (TEST),Y
CMP TEST+3
BNE FIN
RTS
PLA
STA AUXE
PLA
STA AUXE+1
LDX #0
INC AUXE
BNE *+4
INC AUXE+1
LDA (AUXE,X)
AND #$7F
JSR OUTCH
LDX #0
LDA (AUXE,X)
BPL MS1
LDA AUXE+1
PHA
LDA AUXE
PHA
RTS
PHA
LSR
LSR
LSR
LSR
JSR HEX21
PLA
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
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.
Fig. 2: Parts Layout

- ATARI Game connector
- Male side
- G32 means pin 2 of game port 3

Area for mounting
the switch

EPROM SOCKET

- 4024-1
- 4024-2
- 4016
- 4049
- 74LS139
Fig. 4: Rear side of the 3P2T switch
FIG. 2 shows the parts layout. It is recommended 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 CARTRIDGES. Turn on the disk drive and the ATARI. After a short moment, you will see the first menu:

![Menu](image)

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 menu:

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 OFFF. 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 addresses which are not used by the system or the burner software (A800 to AFFF) are accessible. By typing Y after the question OK (Y/N), the program is loaded. There is a very important key, the X key. This key cancels the input and leads back into the main menu. An example of reading an EPROM is shown in the next figure:
To verify that the content of the RAM is identical to the content of the EPROM, type V. After specifying the addresses for the EPROM and the RAM and typing Y, the contents are compared. If there are any differences, you get an error message, such as the following:
You may then make a memory dump. Type M for MEMORY, either R for RAM or E for EPROM, 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 menu is shown and you can begin a new burning procedure.

4) PARTS LIST.

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC1</td>
<td>74LS139</td>
</tr>
<tr>
<td>IC2, IC3</td>
<td>4024</td>
</tr>
<tr>
<td>IC4</td>
<td>4016</td>
</tr>
<tr>
<td>IC5</td>
<td>4049</td>
</tr>
<tr>
<td>T1, T2, T3</td>
<td>UNIVERSAL NPN TRANSISTOR</td>
</tr>
<tr>
<td></td>
<td>30V, 0.3W (2N3390 &amp; 2N3399)</td>
</tr>
<tr>
<td>R1</td>
<td>470 K RESISTOR</td>
</tr>
<tr>
<td>R2, R3</td>
<td>100 K RESISTOR</td>
</tr>
<tr>
<td>R4, R5</td>
<td>33 K RESISTOR</td>
</tr>
<tr>
<td>Z1</td>
<td>1 V ZENER DIODE</td>
</tr>
<tr>
<td>Z2</td>
<td>24 V ZENER DIODE</td>
</tr>
<tr>
<td>M1</td>
<td>DCP528 DC/DC CONVERTER</td>
</tr>
<tr>
<td></td>
<td>ELPAC POWER SYSTEMS</td>
</tr>
<tr>
<td>C1, C2</td>
<td>100 NF CAPACITOR</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Address</th>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A800</td>
<td>2076A9204CA82078</td>
<td></td>
<td>v) L( x</td>
</tr>
<tr>
<td>A808</td>
<td>A8006885EE6885EF</td>
<td></td>
<td>(@hEnhEo</td>
</tr>
<tr>
<td>A810</td>
<td>A200E6EED002E6EF</td>
<td></td>
<td>&quot;@fnPBfo</td>
</tr>
<tr>
<td>A818</td>
<td>A1EE297F204A4F6A2</td>
<td>!n) $v&quot;</td>
<td></td>
</tr>
<tr>
<td>A820</td>
<td>00A1EE10EDA5EF48</td>
<td></td>
<td>@!nPm%oH</td>
</tr>
<tr>
<td>A828</td>
<td>A5EE4860A5FD2940</td>
<td></td>
<td>%nH`` ) @</td>
</tr>
<tr>
<td>A830</td>
<td>F006A5FE0901D004</td>
<td></td>
<td>pF% IAPD</td>
</tr>
<tr>
<td>A838</td>
<td>A5FE290E8D01D348</td>
<td></td>
<td>% ) NMASH</td>
</tr>
<tr>
<td>A840</td>
<td>68AD00D348A5FE8D</td>
<td></td>
<td>h-@SH% M</td>
</tr>
<tr>
<td>A848</td>
<td>01D36860A90085F0</td>
<td></td>
<td>AS`)(L$ v</td>
</tr>
<tr>
<td>A850</td>
<td>85F185F89308D03</td>
<td></td>
<td>) $v)EE</td>
</tr>
<tr>
<td>A858</td>
<td>D3A90F8D01D385F5</td>
<td></td>
<td>S)OMASEEu</td>
</tr>
<tr>
<td>A860</td>
<td>A9348D03D3A9FF85</td>
<td></td>
<td>)4MCS) E</td>
</tr>
<tr>
<td>A868</td>
<td>F4A9B085F9A9028D</td>
<td></td>
<td>t)0Ey)BM</td>
</tr>
<tr>
<td>A870</td>
<td>01D360A99B4CA4F6</td>
<td></td>
<td>AS`)(L$ v</td>
</tr>
<tr>
<td>A878</td>
<td>A97D20A4F6A90585</td>
<td></td>
<td>s()JEU J</td>
</tr>
<tr>
<td>A880</td>
<td>54A90A8555A90085</td>
<td></td>
<td>(W)RITE</td>
</tr>
<tr>
<td>A888</td>
<td>56200AA852294541</td>
<td></td>
<td>V J(R)EA</td>
</tr>
<tr>
<td>A890</td>
<td>44204550524FCD20</td>
<td></td>
<td>D EPROM</td>
</tr>
<tr>
<td>A898</td>
<td>73A8A90A8555200A</td>
<td></td>
<td>s()JEU J</td>
</tr>
<tr>
<td>A8A0</td>
<td>A857295249545420</td>
<td></td>
<td>(W)RITE</td>
</tr>
<tr>
<td>A8A8</td>
<td>4550524FCD2073A8</td>
<td></td>
<td>EPROM s(</td>
</tr>
<tr>
<td>A8B0</td>
<td>A90A8555200AA845</td>
<td></td>
<td>JEU J(E</td>
</tr>
<tr>
<td>A8B8</td>
<td>2950524F4D204552</td>
<td></td>
<td>)PROM ER</td>
</tr>
<tr>
<td>A8C0</td>
<td>415345C42073A8A9</td>
<td></td>
<td>ASED s()</td>
</tr>
<tr>
<td>A8C8</td>
<td>0A8555200AA85629</td>
<td></td>
<td>JEU J(V)</td>
</tr>
<tr>
<td>A8D0</td>
<td>4552494659205052</td>
<td></td>
<td>ERIFY PR</td>
</tr>
<tr>
<td>A8D8</td>
<td>4F475241CD2073A8</td>
<td></td>
<td>OGRAM s(</td>
</tr>
<tr>
<td>A8E0</td>
<td>A90A8555200AA845</td>
<td></td>
<td>JEU J(M</td>
</tr>
<tr>
<td>A8E8</td>
<td>29454D4F52592044</td>
<td></td>
<td>)EMORY D</td>
</tr>
<tr>
<td>A8F0</td>
<td>554DD02073A8A90D</td>
<td></td>
<td>UMP s()M</td>
</tr>
<tr>
<td>A8F8</td>
<td>8555200AA8522941</td>
<td></td>
<td>EU J(R)A</td>
</tr>
<tr>
<td>A900</td>
<td>CD2073A8A90D8555</td>
<td></td>
<td>M s()MEU</td>
</tr>
<tr>
<td>A908</td>
<td>200AA8452950524F</td>
<td></td>
<td>J(E)PRO</td>
</tr>
<tr>
<td>A910</td>
<td>CD2073A8A90A8555</td>
<td></td>
<td>M s()JEU</td>
</tr>
<tr>
<td>A918</td>
<td>200AA85329455420</td>
<td></td>
<td>J(S)ET</td>
</tr>
<tr>
<td>A920</td>
<td>4550524F4D205459</td>
<td></td>
<td>EPROM TY</td>
</tr>
<tr>
<td>A928</td>
<td>50C52073A82073A8</td>
<td></td>
<td>PE s( s(</td>
</tr>
<tr>
<td>A930</td>
<td>A90A8555200AA857</td>
<td></td>
<td>)JEU J(W</td>
</tr>
<tr>
<td>A938</td>
<td>484154BA20F0AE48</td>
<td></td>
<td>HAT: p.H</td>
</tr>
<tr>
<td>A940</td>
<td>20A4F668C952D003</td>
<td></td>
<td>$vhIRPC</td>
</tr>
</tbody>
</table>

59
A948 4C30ACC957D0034C  L0,IWPCL
A950 10ADC945D0034C8B  P-IEPCLK
A958 ACC956D0034C2DAF  ,IVPCL-/ISPClV)
A960 C953D0034C76A9C9  MPCL{(-)
A968 4DD0034CFBADA9FD  $v1J@)
A970 20A4F66C0A00A97D  $v s( J
A978 20A4F62073A8200A  WHICH E
A980 A857484943482045  PROM DO
A988 50524F4D2044F20  YOU WANT
A990 594F552057414E54  TO BURN
A998 20544F204255524E  (?)HET)J
A9A0 20BFA9088554A90A  EU J(A)
A9A8 8555200AA8412920  2532 s()
A9B0 323533B22073A8A9  JEU J(B)
A9B8 0A8555200AA84229  2732 s(
A9C0 20323733B22073A8  )JEU J(C)
A9C8 A90A8555200A843  ( )2716,2
A9D0 2920323731362C32  516 s( s
A9D8 3531B62073A82073  ()JEU J(6
A9E0 A8A90A8555200AA8  WHAT: p.
A9E8 57484154BA20F0AE  H $vhE|I
A9F0 4820A4F66885FFC9  APF|E p
A9F8 41D006A90085FDF0  RIBPF|E
AA00 12C942D006A98085  0HICPx)
AA08 FD3008C943D078A9  @E s( s
AA10 C085FD2073A82073  ( J(SET
AA18 A8200AA853455420  SWITCH T
AA20 5357495443482054  O POSITI
AA28 4F20504F53495449  ON %|IAP
AA30 4F4EA0A5FFC941D0  J J(2532
AA38 0A200AA8323533B2  XP*IBPJ
AA40 18901EC942D00A20  J(2732XP
AA48 0AA8323733B21890  PICP2 J(
AA50 10C943D032200AA8  2716,251
AA58 323731362C323531  6 s( s(
AA60 B62073A82073A8A9  JEU J(NO
AA68 0A8555200AA84E4F  W INSERT
AA70 5720494E53455254  EPROM W
AA78 204550524FCD20D7  + O*LC()
AA80 AB208FA4C03A8A9  $vLm)
AA88 FD20A4F64CEDA920  s()JEU J
AA90 73A8A90A8555200A  (PRESS S
AA98 A850524553532053
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.
<table>
<thead>
<tr>
<th>EPROM</th>
<th>I 2716</th>
<th>I 2732</th>
<th>I 2516</th>
<th>I 2532</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>V 0 V 1</td>
<td>V 1 V 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 V 0 O 1</td>
<td>O 1 O 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>V V V O 1</td>
<td>V O 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>O O O O 1</td>
<td>O V 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 V 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V = means connected (jumper)
0 = means open

[Diagram of circuit board with labels and connections]

jumper
wiring area
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.
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 *
  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 BUS
0647: 297F  AND  #01111111  STROBE OFF
0649: 8D01D3  STA  PORTA  CLEAR BUS
064C: 8C01D3  STY  PORTA  RTS
064F: 60  RTS
  * PUT NEW ADDRESS IN HANDLERVECTOR

0650: A900  INITL  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
SUCCESS   $0624  CLOSE   $0624
NOTIMPL   $0627  GETBYTE $0627
STATUS    $0627  SPECIAL $0627
PUTBYTE   $062A  NOEOL   $0635
PARAOUT   $063B  INITL   $0650  UNUSED

For more information about the parallel interface refer to page 106.
RS232 SERIAL INTERFACE

COUNT EPZ $1F
RSENTRY EQU $032C
PACTL EQU $D303
PORTA EQU $D301
NMIE N 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

HANDLT AB DFW OPEN-l
DFW CLOSE-l
DFW GETBYTE-l
DFW PUTBYTE-l
DFW STATUS-l
DFW SPECIAL-l
DPB 0,0,0,0 JUST FILL WITH ZERO

THE OPEN ROUTINE

OPEN EQU *
INIT LDA #$30
STA PACTL
LDA #$0000001
STA PORTA
LDA #$34
STA PACTL
LDA #$00
STA PORTA
JSR BITWAIT
JSR BITWAIT
LDY #1
RTS

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

CLOSE EQU SUCCES

LDY #146 RETURN WITH Y=146
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
THE BITTIME ROUTINE FOR AN EXACT BAUDRATE

BITWAIT LDX #K
LOOPK LDY #$L
DEY BNE LOOPL
DEX BNE LOOPK
RTS

THE ROUTINE FOR INSTALLING THE RS232 HANDLER

INITIAN LDA 'R

DEVICE NAME

STA RSENTRY
LDA #HANDLTAB:L
STA RSENTRY+1
LDA #HANDLTAB:H
STA RSENTRY+2
JMP OPEN

BUFFER EQU *

ONE BYTE BUFFER

NO WARNINGS

COUNT $1F
PACTL $D303
NMEN $D40E
EOL $9B
LF $06
OPEN $0610
SUCES $062A
NOTIMPL $062D
STATUS $062D
PUTBYTE $0630
SEROUT $0643
BITWAIT $0685
LOOPL $0689
BUFFER $06A2

RSENTRY $D30C
PORTA $D301
DMACRL $D400
CR $0D
K $96
HANDLTAB $0600
INIT $0610
CLOSE $062A
GETBYTE $062D
SPECIAL $062D
NOEOL $063C
SENDBYTE $065D
LOOP $0687
INITHAN $0690

BUFFER EQU $A8A2
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.

```
STOREADR EPZ $F0.1
ENDADR EPZ $F2.3
PROGLN 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

START
A800: A97D LDA #CLS
A802: 20A4F6 JSR SCROUT

PRINT MESSAGE
A805: 2000AA JSR PRINT
A808: 0D0D DFB CR,CR
A80A: 424F4F ASC \BOOTGENERATOR FROM\\
A80D: 544745 HO\FACKER\\
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: 455352
A836: 203AA4
A839: 2028AA JSR HEXIN
A83C: 84F0 STY STOREADR
A83E: 85F1 STA STOREADR+1

GET ENDDADDRESS

77
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 ENADDR
A85A: 85F3 STA ENADDR+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

78
* 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
A8CE: 2980
A8D0: 85F4
A8D2: 9002
A8D4: E6F5

ADC #127
AND #128
STA PROGLLEN
BCC *+4
INC PROGLLEN+1

CALCULATE NUMBER OF RECORDS

A8D6: 0A
A8D7: A5F5
A8D9: 2A
A8DA: A001
A8DC: 91F0

ASL
LDA PROGLLEN+1
ROL
LDY #RECN-HEAD
STA (STOREADR), Y

A8DE: A002
A8E0: A5FA
A8E2: 91F0
A8E4: A5FB
A8E6: C8
A8E7: 91F0

LDY #PST-HEAD
LDA LOGSTORE
STA (STOREADR), Y
LDY #RECN-HEAD
LDA LOGSTORE+1
INY
STA (STOREADR), Y

A8E9: A004
A8EB: 18
A8EC: A5FA
A8EE: 691F

LDY #PINITADR-HEAD
CLC
LDA LOGSTORE
ADC #PINIT-HEAD

A8F0: 91F0
A8F2: C8
A8F3: A5FB
A8F5: 6900
A8F7: 91F0

STA (STOREADR), Y
INY
LDA LOGSTORE+1
ADC #0
STA (STOREADR), Y

A8F9: A00C
A8FB: A5FA
A8FD: 18
A8FE: 65F4
A900: 91F0
A902: A011
A904: A5FB
A906: 65F5
A908: 91F0

LDY #PNDLO-HEAD
LDA LOGSTORE
CLC
ADC PROGLLEN
STA (STOREADR), Y
LDY #PNDHI-HEAD
LDA LOGSTORE+1
ADC PROGLLEN+1
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  
A933: 455220  
A936: 544845  
A939: 204245  
A93C: 455053  
A93F: 202752  
A942: 455455  
A945: 524EA7  

* OPEN CASSETTE FOR WRITE  

A948: A210 OPENIOCB  
A949: A03     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 ICBAI,X
A95E: A9A9 LDA #CFILE:H
A960: 9D4503 STA ICBAL,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 ICBAL,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 IOCBO

A986: A90C CLOSIOCBO 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 PLA
A9A8: 68
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: 524154  
A9D1: 494P4E  
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 ADERR 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 IOCBI.  

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

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

83
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  HEXROTL ASL
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
AA99: 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
AAA4: 0000 PST DFW DUMMY
AAAC: 0000 PINITADR DFW DUMMY

* THE BOOT CONTINUATION CODE

AAAE: A93C LDA #$3C
AAB0: 8D02D3 STA PACTL
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
OPENIIOCB $A948  UNUSED
CLOSIOCB $A986  UNUSED
SUCCES $A9B4
CFIILE $A9F2
MOVELOOP $A9F7
PRINT1 $AA08
HEXIN $AA28
ALFA $AA54
HEXERR $AA6A
HEXROTI $AA7E
DIV $AA90
WAIT $AAA2
HEAD $AA8
PST $AAAA
PNDLO $AAB4
JUMPADRL $AABE
HEADEND $AAC7
A DIRECT CASSETTE TO DISK COPY PROGRAM

CHAPTER 9

If you have a bootable program on cassette, and you want to have it on a bootable disk, the following program will help you. This program is easy to understand if you have read the previous chapters. It allows you to copy direct from tape to disk, using a buffer.

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

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

The copy program will not be described, but it has helpful comments, and you possess the knowledge of the IO. It is important that the buffer (BUFADR) is large enough for the program.
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
EOUTCHEQU $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 UNTIL EOF REACHED

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

* IF NO ERROR OR EOF INCREASE THE BUFFER POINTER

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
LDX XSAVE
DEX
BPL ERRLOOP
PLA
TAX
JSR PUTINT
LDA #EOL
JSR EOUTCH

* THE BREAK FOR RETURNING
* TO THE CALLING MONITOR

BRK

* TEXT FOR ERROR MESSAGE

ASC " -RORRE"
DFB $9B,$9B
LENGTH EQU (*-I)-ERROR

* RECURSIVE PUTINT FOR
* DECIMAL ERRORCODE

PHA
TXA
CMP #10
BCC PUTDIG
LDX #-1
SBC #10
INX
BCS DIV
ADC #10
JSR PUTINT
RECURSION STEP

CLA
ADC 'O'
JSR EOUTCH
PLA
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
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A939:</td>
<td>8D0803</td>
<td>STA DCBCNT</td>
</tr>
<tr>
<td>A93C:</td>
<td>A900</td>
<td>LDA #0</td>
</tr>
<tr>
<td>A93E:</td>
<td>8D0903</td>
<td>STA DCBCNT+1</td>
</tr>
<tr>
<td>A941:</td>
<td>2059E4</td>
<td>JMPSIO</td>
</tr>
<tr>
<td>A944:</td>
<td>100C</td>
<td>JSR SIO</td>
</tr>
<tr>
<td>A946:</td>
<td>C688</td>
<td>BPL RETRY</td>
</tr>
<tr>
<td>A948:</td>
<td>3008</td>
<td>BMI Writend</td>
</tr>
<tr>
<td>A94A:</td>
<td>A280</td>
<td>LDX #$80</td>
</tr>
<tr>
<td>A94C:</td>
<td>8E0303</td>
<td>STX DCBSTA</td>
</tr>
<tr>
<td>A94F:</td>
<td>4C41A9</td>
<td>JMP JMPSIO</td>
</tr>
<tr>
<td>A952:</td>
<td>AC0303</td>
<td>Writend</td>
</tr>
<tr>
<td>A955:</td>
<td>60</td>
<td>RTS</td>
</tr>
</tbody>
</table>

BUFADR EQU *

PHYSICAL ENDADDRESS: $A956

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTR</td>
<td>$80</td>
</tr>
<tr>
<td>BUFFER</td>
<td>$84</td>
</tr>
<tr>
<td>RETRY</td>
<td>$88</td>
</tr>
<tr>
<td>DCBSBI</td>
<td>$0300</td>
</tr>
<tr>
<td>DCBCMD</td>
<td>$0302</td>
</tr>
<tr>
<td>DCBBUF</td>
<td>$0304</td>
</tr>
<tr>
<td>DCBCNT</td>
<td>$0308</td>
</tr>
<tr>
<td>ICCMD</td>
<td>$0342</td>
</tr>
<tr>
<td>ICBAH</td>
<td>$0345</td>
</tr>
<tr>
<td>ICBLH</td>
<td>$0349</td>
</tr>
<tr>
<td>ICAX2</td>
<td>$034B</td>
</tr>
<tr>
<td>GETCHR</td>
<td>$07</td>
</tr>
<tr>
<td>RMODE</td>
<td>$04</td>
</tr>
<tr>
<td>CIO</td>
<td>$E456</td>
</tr>
<tr>
<td>EOUTCH</td>
<td>$F6A4</td>
</tr>
<tr>
<td>EOF</td>
<td>$88</td>
</tr>
<tr>
<td>MAIN</td>
<td>$A800</td>
</tr>
<tr>
<td>QEOF</td>
<td>$A82A</td>
</tr>
<tr>
<td>READY</td>
<td>$A867</td>
</tr>
<tr>
<td>ERRLOOP</td>
<td>$A86C</td>
</tr>
<tr>
<td>LENGTH</td>
<td>$08</td>
</tr>
<tr>
<td>DIV</td>
<td>$A895</td>
</tr>
<tr>
<td>OPENCASS</td>
<td>$A8A7</td>
</tr>
<tr>
<td>CLOSCASS</td>
<td>$A8E9</td>
</tr>
<tr>
<td>CFILE</td>
<td>$A903</td>
</tr>
</tbody>
</table>

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

99
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
The following construction article allows you to build your own RS232 interface for the ATARI computer. The interface only works with 300 Baud and just in one direction (output).

The interface consists of:

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

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

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

The Spinwriter and the Brother needed some jumper wires as shown below:
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:\n"
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.
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**

<table>
<thead>
<tr>
<th>MX80-Connector</th>
<th>ATARI-Connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 3</td>
<td>Port 4</td>
</tr>
<tr>
<td>Pin#</td>
<td>Pin#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Pin #</th>
<th>Pin #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (19) STROBE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2 (20) DATA 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3 (21) DATA 2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4 (22) DATA 3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5 (23) DATA 4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6 (24) DATA 5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7 (25) DATA 6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8 (26) DATA 7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9 (27) DATA 8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11 (29) BUSY</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

(GND) 8 8
(19)–(29) = Ground (GND)

Plugs seen from the rear view.
Front view of the computer outlets.
The next figure shows the program.

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

<table>
<thead>
<tr>
<th>Basis</th>
<th>EPZ $58</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>EPZ $FE</td>
</tr>
<tr>
<td>PST</td>
<td>EQU $600</td>
</tr>
</tbody>
</table>

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: 1B  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: B03D3 STA $D303
0631: A9FF LDA #$FF
0633: B01D3 STA $D301
0636: A934 LDA #$34
0638: B03D3 STA $D303
063B: A980 LDA #$80
063D: B01D3 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: BDE906 STA LCOUNT
064D: CEE806 DEC COUNT
0650: 100D BPL NOFF LDA #12
0652: A90C LDA #65
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: B03D3 STA $031B
0673: A906 LDA #HANDLTAB/256 STA $031C
0675: B1DC03 STA #65
0678: A941 STA COUNT
067A: BDE806 LDA LINLEN
067D: ADEA06 STA LCOUNT
0680: BDE906 JMP OPEN
0683: 4C2C06
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: BDE606 STA ROW
0694: A927 ROWLOOP LDA #39
0696: BDE706 STA COLUMN
0699: A200 LDX #0
069B: 297F LOOP LDA (PT,X)
069D: C960 AND #$7F
069F: A002 CMP #$60
06A1: B002 BCS LOOP1
06A3: 6920 ADC #$20
06A5: 2D106 LOOP1 JSR OUTCHAR
06A8: E6FE OUTCHAR INC PT
06AA: D002 BNE #+4
06AC: E6FF INC PT+1
06AE: CEE706 DEC COLUMN
06B1: 10E8 BPL LOOP
06B3: A90D LDA #13
06B5: 2D106 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: DOFB BNE OUTCHAR
06D6: A080 LDY #$80
06D8: 0980 ORA #$80
06DA: BD01D3 STA #D301
06DD: 297F AND #$7F
06DF: BD01D3 STA #D301
06E2: BC01D3 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
Program description:
Address
0600 – 061E 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.
In this case go by the name of signal rather than by the numbers.
The numbers on the printer connector may vary with the different parallel printer used.

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

In this case go by the name of signal rather than by the numbers.
The numbers on the printer connector may vary with the different parallel printer used.
Copyright notice
Subroutine, brings one ASCII character from the accumulator to the printer
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

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

6000: A210 BOOTB LDX #$10
       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
```
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 Cartridge and ATAS-1 and ATMAS-1

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

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

<table>
<thead>
<tr>
<th>ATARI</th>
<th>ELCOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>- BYTE &quot;STRING&quot; =</td>
<td>ASC &quot;STRING&quot;</td>
</tr>
<tr>
<td>- BYTE $        =</td>
<td>DFB $ (Insertion of a byte)</td>
</tr>
<tr>
<td>- WORD         =</td>
<td>DFW (Insertion of a word Lower byte, higher byte)</td>
</tr>
</tbody>
</table>

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 string output 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.
## HOFAKER ORDER FORM

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

Please make checks out to ELCOMP PUBLISHING, INC.
Please bill to my Master Card or Visa account # ........................
Card # ........................
Expiry Date ........................
Master Charge Bank Code ........................
Signature ........................

<table>
<thead>
<tr>
<th>Qty</th>
<th>Order No.</th>
<th>Description</th>
<th>Price $</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>150</td>
<td>MICROC. HARDWARE HANDB.</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>151</td>
<td>CARE AND FEEDING...</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>152</td>
<td>BK MICROSOFT BASIC</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>153</td>
<td>EXP.HANDB. 6582 AND 6802</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>154</td>
<td>COMPLEX SOUND</td>
<td>6.95</td>
</tr>
<tr>
<td></td>
<td>155</td>
<td>SMALL BUSINESS PRGR.</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>156</td>
<td>SECOND BOOK OF OHIO</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>157</td>
<td>THE THIRD BOOK OF OHIO</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>158</td>
<td>THE FOURTH BOOK OF OHIO</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>159</td>
<td>THE FIFTH BOOK OF OHIO</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>160</td>
<td>GAMES FOR THE Atari</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>161</td>
<td>ATARI LEARNING BY USING</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>162</td>
<td>PROGR. 1. DISK</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>HOW TO PROGR. IN MACH.L.</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>164</td>
<td>FORTH ON THE Atari</td>
<td>7.95</td>
</tr>
<tr>
<td></td>
<td>165</td>
<td>A LOOK INTO THE FUTURE</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>166</td>
<td>PROGRAM DESCRIPTIONS</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>167</td>
<td>ZX-81/TIMEX</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>168</td>
<td>TRICKS FOR VICS</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>169</td>
<td>JAMA/1 MONITOR</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>170</td>
<td>PROTOTYPING CARD</td>
<td>29.80</td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>6522 V/VI A/D EXP. CARD</td>
<td>39.00</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>SLOT REPEATER</td>
<td>49.80</td>
</tr>
<tr>
<td></td>
<td>173</td>
<td>2716 EPROM PROGRAMMER</td>
<td>49.00</td>
</tr>
<tr>
<td></td>
<td>174</td>
<td>SOUND WITH THE GI ATY3-8912</td>
<td>39.00</td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>8K EPROM CARD (2716)</td>
<td>29.00</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>12 BIT A/D CONVERTER BOAR</td>
<td>74.00</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>551-1 RAMM-RAMM-BOARD</td>
<td>39.95</td>
</tr>
<tr>
<td></td>
<td>178</td>
<td>THE CUSTOM APPLE BOARD</td>
<td>24.95</td>
</tr>
<tr>
<td></td>
<td>179</td>
<td>MAILING LIST FOR ZX-81</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>180</td>
<td>MACHINE LANG. MONITOR</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>181</td>
<td>ADAPTER BOARD FOR ZX-81</td>
<td>14.80</td>
</tr>
<tr>
<td></td>
<td>182</td>
<td>EDITOR/ASS. FOR CBM</td>
<td>39.00</td>
</tr>
<tr>
<td></td>
<td>183</td>
<td>ASSEMBLER FOR CBM</td>
<td>39.95</td>
</tr>
<tr>
<td></td>
<td>184</td>
<td>GUNFIGHT FOR PET/CBM</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>185</td>
<td>UNIVERSAL EXP. BOARD</td>
<td>18.95</td>
</tr>
<tr>
<td></td>
<td>186</td>
<td>ADAPTER BOARD</td>
<td>3.95</td>
</tr>
<tr>
<td></td>
<td>187</td>
<td>PROFI WORDPROC. F. VIC</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>188</td>
<td>TIC TAC VIC</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>189</td>
<td>GAMEPACK FOR VIC</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>190</td>
<td>MAIL-VIC</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>191</td>
<td>EXPANDING YOUR VIC</td>
<td>14.95</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>RINFFILL</td>
<td>9.95</td>
</tr>
<tr>
<td></td>
<td>193</td>
<td>MINIASSEMBLER</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>194</td>
<td>LEARN-FORTH</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>195</td>
<td>POWER FORTH (APPLE)</td>
<td>39.95</td>
</tr>
<tr>
<td></td>
<td>196</td>
<td>ATMOMA-1, CASS.</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>197</td>
<td>ATMOMA-1, DISK</td>
<td>24.95</td>
</tr>
<tr>
<td></td>
<td>198</td>
<td>ATMOMA-1, CARDITRE</td>
<td>59.00</td>
</tr>
<tr>
<td></td>
<td>199</td>
<td>EPROM BURNER FOR ATARI</td>
<td>179.00</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>EPROM BOARD (CARTIDGE)</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>201</td>
<td>ATMOMA-2, CASS.</td>
<td>49.95</td>
</tr>
<tr>
<td></td>
<td>202</td>
<td>ATMOMA-2, DISK</td>
<td>54.00</td>
</tr>
<tr>
<td></td>
<td>203</td>
<td>LEARN-FORTH</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>POWER FORTH</td>
<td>39.95</td>
</tr>
<tr>
<td></td>
<td>205</td>
<td>ATAS</td>
<td>69.95</td>
</tr>
<tr>
<td></td>
<td>206</td>
<td>ATMAS</td>
<td>89.00</td>
</tr>
<tr>
<td></td>
<td>207</td>
<td>PROGRAMS FROM BOOK H 164</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>208</td>
<td>INVOICE WRITING, DISK</td>
<td>39.00</td>
</tr>
<tr>
<td></td>
<td>209</td>
<td>GUNFIGHT FOR ATARI</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>210</td>
<td>WORDPROCESSOR, CASS.</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>211</td>
<td>HOW TO CONNECT...</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>212</td>
<td>MAILING LIST, CASS.</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>213</td>
<td>MAILING LIST, DISK</td>
<td>24.95</td>
</tr>
<tr>
<td></td>
<td>214</td>
<td>INVENTORY CONTR., CASS.</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>215</td>
<td>INVENTORY CONTR., DISK</td>
<td>24.95</td>
</tr>
<tr>
<td></td>
<td>216</td>
<td>WORDPROCESSOR, DISK</td>
<td>34.95</td>
</tr>
<tr>
<td></td>
<td>217</td>
<td>WORDPROCESSOR, CARDITRE</td>
<td>69.00</td>
</tr>
<tr>
<td></td>
<td>218</td>
<td>PROGRAMS FROM BOOK H 162</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>219</td>
<td>KNAUS CGING</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>220</td>
<td>ASTROLOGY PROGRAM</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>221</td>
<td>EPROM BOARD KIT</td>
<td>14.94</td>
</tr>
<tr>
<td></td>
<td>222</td>
<td>FLOATING POINT PACKAGE</td>
<td>299.95</td>
</tr>
<tr>
<td></td>
<td>223</td>
<td>RS232-INTERFACE</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>224</td>
<td>EPROM BURNER KIT</td>
<td>49.00</td>
</tr>
<tr>
<td></td>
<td>225</td>
<td>AICASH</td>
<td>49.95</td>
</tr>
<tr>
<td></td>
<td>226</td>
<td>ROOM PHAGES</td>
<td>19.95</td>
</tr>
<tr>
<td></td>
<td>227</td>
<td>ATARIO</td>
<td>29.95</td>
</tr>
<tr>
<td></td>
<td>228</td>
<td>SUPERMAIL</td>
<td>49.00</td>
</tr>
<tr>
<td></td>
<td>229</td>
<td>BUSIPACK 1</td>
<td>98.00</td>
</tr>
<tr>
<td></td>
<td>230</td>
<td>BIODRHYTHM FOR ATARI</td>
<td>9.95</td>
</tr>
</tbody>
</table>

Payment: Check, Money Order, VISA, Mastercharge, Access, Interbank, Eurocheque
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.

Superbord and CIP are trademarks of Ohio Scientific Co.
ATARI is a trademark of ATARI-Warner Communications
PET, CBM and VIC-20 are trademarks of Commodore Business Machines.
TRS-80 is a trademark of TANDY Radio Shack.
APPLE II is a trademark of APPLE Computer Inc.
BITFIRE
BITFIRE
BITFIRE
BITFIRE
BITFIRE
BITFIRE