More on the Sixtyfour 64

- Recursive Programming
- The Filesystem
- RS232 Interface
- Realtime Clock
- Terminal Program
- The A/D Converters
- Centronics Interface

H. C. Wagner

First Edition
First Printing
1983 in the Federal Republic of Germany

Cover design by Daniel Schwarz

ISBN 3-88963–183–5

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

US-Distributor:
ELCOMP PUBLISHING, INC., 53 Redrock Lane, Pomona, CA 91766

More on the Sixtyfour 64

PREFACE

Since more and more users of the Commodore-64 computer write their programs in machine language, more and more "workhorse" routines performing special tasks are required.

This book contains a variety of useful programs for the person with a knowledge of machine language programming. All the programs have been fully tested on the MACROFIRE Editor/ Assembler from Hofacker.
For your convenience a complete source code is provided. You can use them as ready-to-run programs or make your own changes.

Important notice

This book is written for the experienced Commodore-64 Personal Computer user. To run the programs you need a symbolic Editor/ Assembler or the MACROFIRE from ELCOMP PUBLISHING,INC.

TABLE OF CONTENT

1.1 Input and output of numbers 1
1.1.1 Hexadecimal input 1
Hexinput Routine 2
1.1.2 Hexadecimal output 3
Hexout Routine 3
1.1.3 Decimal input 4
Decinput Routine 5
Decout Routine 6
1.2 16 bit arithmetic without sign 7
1.2.1 16 bit addition and subtraction 7
1.2.3 16 bit multiplication 9
1.2.4 16 bit division 11
2.1 Output of text 14
3.1 Recursion 17
4.1 The Filesystem 27
4.2 Save and load programs in OS 34
5.1 The RS-232 Interface 43
6.1 The realtime clock 48
7.1 How to add new BASIC commands? 67
8.1 Hires Assistent 70
9.1 Diskutility 80
10.1 How to add devicehandlers? 91
11 An inexpensive centronix interface 94
12.1 Prettyprint 103
13.1 Screenprint on printer 105
13.2 Screenprint via RS-232 108
14.1 Terminal 113
15 How to connect your C-64 with an ATARI? 128
16.1 How to use the A/D converters? 135
17 The restore key 137
18 Fast and compact hexdumper 141
19 Two tiny ones 146

1.1 INPUT AND OUTPUT OF NUMBERS

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 zero, even if an invalid number is entered. Next, the program reads a character and checks wether or not it is a hexadecimal number. If it is, then the accumulator is erased and the lower bits are shifted up. Now, these four bits can be shifted to EXPR from the right. The preceeding number in EXPR is shifted to the left by doing so.
If you enter a number with more than four digits, only the last four digits are used.

```
Example: ABCDEF => CDEF

****************************************
*                                      *
*         HEXINPUT ROUTINE             *
*                                      *
****************************************

              EXPR      EQU $8A.B

              BASIN     EQU $FFCF

                        ORG $C000

C000: A200    HEXIN     LDX #0
C002: 868A              STX EXPR
C004: 868B              STX EXPR+1
C006: 20CFFF  HEXIN1    JSR BASIN
C009: C930              CMP '0
C00B: 901E              BCC HEXRTS
C00D: C93A              CMP '9+1
C00F: 900A              BCC HEXIN2
C011: C941              CMP 'A
C013: 9016              BCC HEXRTS
C015: C947              CMP 'F+1
C017: B012              BCS HEXRTS
C019: E936              SBC 'A-10-1
C01B: 0A      HEXIN2    ASL
C01C: 0A                ASL
C01D: 0A                ASL
C01E: 0A                ASL
C01F: A204              LDX #4
C021: 0A      HEXIN3    ASL
C022: 268A              ROL EXPR
C024: 268B              ROL EXPR+1
C026: CA                DEX
C027: D0F8              BNE HEXIN3
C029: F0DB              BEQ HEXIN1    ALWAYS !
C02B: 60      HEXRTS    RTS

PHYSICAL ENDADDRESS: $C02C

*** NO WARNINGS
```

```
EXPR            $8A
BASIN           $FFCF
HEXIN           $C000    UNUSED
HEXIN1          $C006
HEXIN2          $C01B
HEXIN3          $C021
HEXRTS          $C02B
```

1.1.2 Hexadecimal output

The next program explains the output process of the calculated numerals. You will recognize, that portion of the program which controls the output is a 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 MSBits and then the LSBits will be printed.

```
****************************************
*                                      *
*         HEXOUT ROUTINE               *
*                                      *
****************************************

              EXPR     EQU $8A.B

              BSOUT    EQU $FFD2

                       ORG $C000

C000: A58B    PRWORD   LDA EXPR+1
C002: 200BC0           JSR PRBYTE
C005: A58A             LDA EXPR
C007: 200BC0           JSR PRBYTE
C00A: 60               RTS

              * THE PRBYTE ROUTINE

C00B: 48      PRBYTE   PHA
```

```
C00C: 4A                LSR
C00D: 4A                LSR
C00E: 4A                LSR
C00F: 4A                LSR
C010: 2016C0            JSR HEXOUT
C013: 68                PLA
C014: 290F              AND #%00001111
C016: C90A    HEXOUT    CMP #10
C018: B004              BCS ALFA
C01A: 0930              ORA '0
C01C: D002              BNE HXOUT    ALWAYS!!
C01E: 6936    ALFA      ADC 'A-10-1
C020: 4CD2FF  HXOUT     JMP BSOUT

PHYSICAL ENDADDRESS: $C023

*** NO WARNINGS

EXPR                $8A
BSOUT               $FFD2
PRWORD              $C000     UNUSED
PRBYTE              $C00B
HEXOUT              $C016
ALFA                $C01E
HXOUT               $C020
```

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 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. 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).

```
*****************************************
*                                       *
*         DECINPUT ROUTINE              *
*                                       *
*****************************************

              EXPR      EQU $8A.B
              BASIN     EQU $FFCF

                        ORG $C000

C000: A200    DECIN     LDX #0
C002: 868A              STX EXPR
C004: 868B              STX EXPR+1
C006: 20CFFF  DEC1      JSR BASIN
C009: C930              CMP '0
C00B: 9018              BCC DECEND
C00D: C93A              CMP '9+1
C00F: B014              BCS DECEND
C011: 290F              AND #%00001111
C013: A211              LDX #17
C015: D005              BNE DEC3 ALWAYS !!

C017: 9002    DEC2      BCC *+4
C019: 6909              ADC #10-1
C01B: 4A                LSR
C01C: 668B    DEC3      ROR EXPR+1
C01E: 668A              ROR EXPR
C020: CA                DEX
C021: D0F4              BNE DEC2
C023: F0E1              BEQ DEC1   ALWAYS!!
C025: 60      DECEND    RTS

PHYSICAL ENDADDRESS: $C026
*** NO WARNINGS

EXPR                $8A
BASIN               $FFCF
DECIN               $C000     UNUSED
DEC1                $C006
DEC2                $C017
DEC3                $C01C
DECEND              $C025
```

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 zeros.

```
****************************************
*                                      *
*         DECOUT ROUTINE               *
*                                      *
****************************************

              DECL      EQU $8A
              DECH      EQU DECL+1

              TEMP      EQU $8C

              BSOUT     EQU $FFD2

                        ORG $C000

C000: A007    DECOUT    LDY #7
C002: A230    DECOUT1   LDX '0
C004: 38      DECOUT2   SEC
C005: A58A              LDA DECL
C007: F92EC0            SBC DECTAB-1,Y
C00A: 48                PHA
C00B: 88                DEY
C00C: A58B              LDA DECH
C00E: F930C0            SBC DECTAB+1,Y
C011: 9009              BCC DECOUT3
C013: 858B              STA DECH
C015: 68                PLA
```

```
C016: 858A              STA DECL
C018: E8                INX
C019: C8                INY
C01A: D0E8              BNE DECOUT2
C01C: 68      DECOUT3   PLA
C01D: 8A                TXA
C01E: 848C              STY TEMP
C020: 20D2FF            JSR BSOUT
C023: A48C              LDY TEMP
C025: 88                DEY
C026: 10DA              BPL DECOUT1
C028: A58A              LDA DECL
C02A: 0930              ORA '0
C02C: 4CD2FF            JMP BSOUT

C02F: 0A00    DECTAB    DFW 10
C031: 6400              DFW 100
C033: E803              DFW 1000
C035: 1027              DFW 10000

PHYSICAL ENDADDRESS: $C037

*** NO WARNINGS

DECL                $8A
DECH                $8B
TEMP                $8C
BSOUT               $FFD2
DECOUT              $C000     UNUSED
DECOUT1             $C002
DECOUT2             $C004
DECOUT3             $C01C
DECTAB              $C02F
```

1.2 16 bit arithmetic without sign

1.2.1 16 bit addition and subtraction

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

```
****************************************
*                                      *
*         16 BIT ADDITION              *
*                                      *
*         UNSIGNED INTEGER             *
*                                      *
*         EXPR1 := EXPR1 + EXPR2       *
*                                      *
****************************************

               EXPR1     EPZ $8A.B
               EXPR2     EPZ $8C.D

                         ORG $C000

C000: 18       ADD       CLC
C001: A58A               LDA EXPR1
C003: 658C               ADC EXPR2
C005: 858A               STA EXPR1
C007: A58B               LDA EXPR1+1
C009: 658D               ADC EXPR2+1
C00B: 858B               STA EXPR1+1
C00D: 60                 RTS

PHYSICAL ENDADDRESS: $C00E

*** NO WARNINGS

EXPR1                $8A
EXPR2                $8C
ADD                  $C000     UNUSED

****************************************
*                                      *
*         16 BIT SUBTRACTION           *
*                                      *
*         UNSIGNED INTEGER             *
*                                      *
*         EXPR1 := EXPR1 - EXPR2       *
*                                      *
****************************************
                   EXPR1     EPZ $8A.B
                   EXPR2     EPZ $8C.D
```

```
                        ORG $C000

C000: 38      SUB       SEC
C001: A58A              LDA EXPR1
C003: E58C              SBC EXPR2
C005: 858A              STA EXPR1
C007: A58B              LDA EXPR1+1
C009: E58D              SBC EXPR2+1
C00B: 858B              STA EXPR1+1
C00D: 60                RTS

PHYSICAL ENDADDRESS: $C00E

*** NO WARNINGS

EXPR1               $8A
EXPR2               $8C
SUB                 $C000     UNUSED
```

1.2.3 16 bit multiplication

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

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

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

For example:

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

As you can see it is simpler in the binary system than in the decimal system. Since the highest possible number of each digit is 1 the highest interim result is equal to the multiplicand.
The following program in principle does the same as the procedure descibed above, except that the interim result is shifted to the right, and if required the multiplicand is added.
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 two numbers to be multiplied (EXPR1 and EXPR1+1, EXPR2 and EXPR2+1). After the calculations the results are in locations EXPR1 (LSB) and EXPR1+1 (MSB).

```
****************************************
*                                      *
*        16 BIT MULTIPLICATION         *
*                                      *
*        UNSIGNED INTEGER              *
*                                      *
*        EXPR1 := EXPR1 * EXPR2        *
*                                      *
****************************************

EXPR1    EPZ $8A.B
EXPR2    EPZ $8C.D
SCRATCH  EPZ $8E.F
```

```
                        ORG $C000

C000: A200    MUL       LDX #0
C002: 868E              STX SCRATCH
C004: 868F              STX SCRATCH+1
C006: A010              LDY #16
C008: D00D              BNE MUL2    ALWAYS !!
C00A: 18      MUL1      CLC
C00B: A58E              LDA SCRATCH
C00D: 658C              ADC EXPR2
C00F: 858E              STA SCRATCH
C011: A58F              LDA SCRATCH+1
C013: 658D              ADC EXPR2+1
C015: 858F              STA SCRATCH+1
C017: 468F    MUL2      LSR SCRATCH+1
C019: 668E              ROR SCRATCH
C01B: 668B              ROR EXPR1+1
C01D: 668A              ROR EXPR1
C01F: 88                DEY
C020: 3004              BMI MULRTS
C022: 90F3              BCC MUL2
C024: B0E4              BCS MUL1
C026: 60      MULRTS    RTS

PHYSICAL ENDADDRESS: $C027

*** NO WARNINGS

EXPR1                   $8A
EXPR2                   $8C
SCRATCH                 $8E
MUL                     $C000     UNUSED
MUL1                    $C00A
MUL2                    $C017
MULRTS                  $C026
```

1.2.4 16 bit division

The division of two numbers are actually is just the opposite of the multiplication. Therefor, you can see in the program below, that the divisor is subtracted and the dividend is shifted to the left rather

than to the right. The memory loactions used are the same as with the multiplication, except that the 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 $8A.B
              EXPR2     EPZ $8C.D
              REMAIN    EPZ $8E.F

                        ORG $C000

C000: A200    DIV       LDX #0
C002: 868E              STX REMAIN
C004: 868F              STX REMAIN+1
C006: AC3C01            LDY 316
C009: 068A    DIV1      ASL EXPR1
C00B: 268B              ROL EXPR1+1
C00D: 268E              ROL REMAIN
C00F: 268F              ROL REMAIN+1
C011: 38                SEC
C012: A58E              LDA REMAIN
C014: E58C              SBC EXPR2
C016: AA                TAX
C017: A58F              LDA REMAIN+1
C019: E58D              SBC EXPR2+1
C01B: 9006              BCC DIV2
C01D: 868E              STX REMAIN
C01F: 858F              STA REMAIN+1
C021: E68A              INC EXPR1
```

```
C023: 88      DIV2      DEY
C024: D0E3              BNE DIV1
C026: 60                RTS

PHYSICAL ENDADDRESS: $C027

*** NO WARNINGS

EXPR1               $8A
EXPR2               $8C
REMAIN              $8E
DIV                 $C000     UNUSED
DIV1                $C009
DIV2                $C023
```

2.1 OUTPUT OF TEXT

2.1 Output of text

With the most programs it is necessary to display text (menues etc.).
The following program allows you to display strings of any length at any location you desire. The output command can be located at any place within your program.

How does this 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 a RTS command is executed. By doing so, the string is jumped and the command after it is executed.

```
*****************************************
*                                       *
*         STRINGOUTPUT FOR              *
*                                       *
*         VARIOUS LENGTH                *
*                                       *
*****************************************
              AUX       EPZ $02.3

              BSOUT     EQU $FFD2

                        ORG $C000

              *         EXAMPLE

C000: 2012C0  EXAMPLE   JSR PRINT
C003: 544849            ASC \THIS IS A TEST\
C006: 532049
C009: 532041
C00C: 205445
C00F: 53D4
C011: 00                BRK

              *         THE VERY PRINTROUTINE
C012: 68      PRINT     PLA
C013: 8502              STA AUX
C015: 68                PLA
C016: 8503              STA AUX+1
C018: A200              LDX #0
C01A: E602    PRINT1    INC AUX
C01C: D002              BNE *+4
C01E: E603              INC AUX+1
C020: A102              LDA (AUX,X)
C022: 297F              AND #$7F
C024: 20D2FF            JSR BSOUT
C027: A200              LDX #0
C029: A102              LDA (AUX,X)
C02B: 10ED              BPL PRINT1
C02D: A503              LDA AUX+1
C02F: 48                PHA
C030: A502              LDA AUX
C032: 48                PHA
C033: 60                RTS
```

```
PHYSICAL ENDADDRESS: $C034

*** NO WARNINGS

AUX                    $02
BSOUT                  $FFD2
EXAMPLE                $C000      UNUSED
PRINT                  $C012
PRINT1                 $C01A
```

3.1 RECURSION

3.1 Recursion

A method of programming which is becoming more common is the so called recursive programming. But what is recursion?
Recursion is often used in mathematics. An example is the faculty function which can be defined with recursion as follows:

```
0! = 1  ;

n! = n * (n-1)!
```

With these two definitions the whole function is described and can be evaluated. For example we will calculate 6! using the recursive definition above:

```
6! = 6*5!
6! = 6*(5*4!)
6! = 6*(5*(4*3!))
6! = 6*(5*(4*(3*2!)))
6! = 6*(5*(4*(3*(2*1!))))
6! = 6*(5*(4*(3*(2*(1*0!)))))
                          :
                         +- 0! = 1
```

So.. 6! = 6*5*4*3*2*1*1 = 720

As you can see we have evaluated the function by stepwise using our definition. The evaluation stopped because we reached the definition of 0!. That definition is very important. Without it we would have evaluated to infinity.

In general we can say recursion works only fine if there is a condition, upon which the recursion can terminate (in this case if n becomes zero).
Why do we need recursion in programming? We have seen that it is very easy, when using recursion, to formulate functions and it is very easy to find a algorithm to evaluate the recursion. We can do this because there is a definition which terminates the alogorithm (read program).
There are many problems we can define using recursion and solve them with simple algorithms. The only thing that looks uncommon is the fact you have to call the program itself. Formulating a general outline of a recursive program in a higher level language looks as follows:

```
PROC R:
IF C
THEN D
ELSE S; R
FI
```

As you can see, if condition C were true, the program would perform instruction D and stop. Otherwise it would perform S and call itself (R) again. Our example would then look like:

```
PROC fac IN n OUT f
IF n = 0
THEN f = 1
ELSE f = n * fac( n-1 )
FI.
```

This procedure (subroutine) recalls itself until n=0 and will return 1 as a result. It will then return to the previous call and so on. At last the first call of 'fac' will return to the caller n*(n-1)*(n-2)*... .*(1)*1. Which is exactly the value of n! .

The next two programs will show another purpose of recursion, namely the output of a binary number in decimal form. We did this earlier, but not recursive.
The first program prints the contents of the locations EXPR (LSB) and EXPR+1 (MSB) as an unsigned integer. As you can see the program stops when the contents of EXPR and EXPR+1 become smaller than 10 (otherwise it will divide them by 10, then call itself again). It will print the most significant digit first followed by the other digits. This process automatically removes leading zeros.

```
*****************************************
*                                       *
*         RECURSIVE PUTINT              *
*                                       *
*         UNSIGNED INTEGER              *
*                                       *
*****************************************
EXPR     EPZ $02.3
REMAIN   EPZ $04.5

BSOUT    EQU $FFD2

CR       EQU 13

* TEST MACRO TO TEST ONLY
* THE ROUTINE

TEST     MACRO N
         LDA #N:L
         STA EXPR
         LDA #N:H
         STA EXPR+1
         JSR PUTINT
         LDA #CR
         JSR BSOUT
         MEND

         ORG $C000
```

```
C000: A9FF85+EXAMPLES TEST 65535
C003: 02A9FF+
C006: 850320+
C009: 51C0A9+
C00C: 0D20D2+
C00F: FF    +
C010: A90085+          TEST 0
C013: 02A900+
C016: 850320+
C019: 51C0A9+
C01C: 0D20D2+
C01F: FF    +
C020: A90085+          TEST -32768
C023: 02A980+
C026: 850320+
C029: 51C0A9+
C02C: 0D20D2+
C02F: FF    +
C030: A93985+          TEST 12345
C033: 02A930+
C036: 850320+
C039: 51C0A9+
C03C: 0D20D2+
C03F: FF    +
C040: A9FF85+          TEST -1
C043: 02A9FF+
C046: 850320+
C049: 51C0A9+
C04C: 0D20D2+
C04F: FF    +
C050: 00               BRK

                * THE PRINT ROUTINE

C051: 48      PUTINT   PHA
C052: A503             LDA EXPR+1
C054: D006             BNE NODIG
C056: A502             LDA EXPR
C058: C90A             CMP #10
C05A: 9006             BCC PUTDIG
C05C: 2069C0 NODIG     JSR DIV10
              * ACCU = MOST SIGNIFICANT
              *        DIGIT
```

```
                * GO IN RECURSION

C05F: 2051C0            JSR PUTINT

C062: 0930    PUTDIG    ORA #'0
C064: 20D2FF            JSR BSOUT
C067: 68                PLA
C068: 60                RTS

                * DIVIDE BY TEN ROUTINE
                * SEE FIRST CHAPTER


C069: A200    DIV10     LDX #0
C06B: 8604              STX REMAIN
C06D: 8605              STX REMAIN+1
C06F: A010              LDY #16
C071: 0602    DIV1      ASL EXPR
C073: 2603              ROL EXPR+1
C075: 2604              ROL REMAIN
C077: 2605              ROL REMAIN+1
C079: 38                SEC
C07A: A504              LDA REMAIN
C07C: E90A              SBC #10
C07E: AA                TAX
C07F: A505              LDA REMAIN+1
C081: E900              SBC #0
C083: 9006              BCC DIV2
C085: 8604              STX REMAIN
C087: 8505              STA REMAIN+1
C089: E602              INC EXPR
C08B: 88      DIV2      DEY
C08C: D0E3              BNE DIV1

                * LOAD ACCU WITH THE
                * REMAINDER

C08E: A504              LDA REMAIN
C090: 60                RTS


PHYSICAL ENDADDRESS: $C091
```

```
*** NO WARNINGS

EXPR                $02
REMAIN              $04
BSOUT               $FFD2
CR                  $0D
TEST                MACRO
EXAMPLES            $C000     UNUSED
PUTINT              $C051
NODIG               $C05C
PUTDIG              $C062
DIV10               $C069
DIV1                $C071
DIV2                $C08B
```

With a few extra instructions you can change the program which prints signed integer values. It first checks the sign of the number to be printed, then prints a space (>=0) or a dash (<0). If the number was less than zero it will invert it and print it as if it where unsigned. Now you can print numbers between -32768 and 32767.

```
****************************************
*                                      *
*        RECURSIVE PUTINT              *
*                                      *
*        SIGNED INTEGER                *
*                                      *
****************************************
               EXPR     EPZ $02.3
               REMAIN   EPZ $04.5

               BSOUT    EQU $FFD2

               CR       EQU 13

               * MACRO USED ONLY FOR TEST
               * THE ROUTINES

               TEST     MACRO N
                        LDA #N:L
```

```
                        STA EXPR
                        LDA #N:H
                        STA EXPR+1
                        JSR PUTINTS
                        LDA #CR
                        JSR BSOUT
                        MEND


                        ORG $C000

C000: A9FF85+EXAMPLES TEST 65535
C003: 02A9FF+
C006: 850320+
C009: 71C0A9+
C00C: 0D20D2+
C00F: FF    +
C010: A90085+         TEST 0
C013: 02A900+
C016: 850320+
C019: 71C0A9+
C01C: 0D20D2+
C01F: FF    +
C020: A90085+         TEST -32768
C023: 02A980+
C026: 850320+
C029: 71C0A9+
C02C: 0D20D2+
C02F: FF    +
C030: A9FF85+         TEST 32767
C033: 02A97F+
C036: 850320+
C039: 71C0A9+
C03C: 0D20D2+
C03F: FF    +
C040: A93985+         TEST 12345
C043: 02A930+
C046: 850320+
C049: 71C0A9+
C04C: 0D20D2+
C04F: FF    +
C050: A9FF85+         TEST -1
C053: 02A9FF+
```

```
C056: 850320+
C059: 71C0A9+
C05C: 0D20D2+
C05F: FF    +
C060: A9A785+           TEST -345
C063: 02A9FE+
C066: 850320+
C069: 71C0A9+
C06C: 0D20D2+
C06F: FF    +
C070: 00                BRK

                * THE PRINT ROUTINE
C071: A920    PUTINTS   LDA '
C073: 2403              BIT EXPR+1
C075: 100F              BPL PUTSGN

                * IF < 0 MAKE 2'S COMPLEMENT
                * AND PRINT DASH '-'

C077: 38                SEC
C078: A900              LDA #0
C07A: E502              SBC EXPR
C07C: 8502              STA EXPR
C07E: A900              LDA #0
C080: E503              SBC EXPR+1
C082: 8503              STA EXPR+1
C084: A92D              LDA #'-
C086: 20D2FF  PUTSGN    JSR BSOUT

                * THE KNOWN RECURSIVE ROUTINE

C089: 48      PUTINT    PHA
C08A: A503              LDA EXPR+1
C08C: D006              BNE NODIG
C08E: A502              LDA EXPR
C090: C90A              CMP #10
C092: 9006              BCC PUTDIG
C094: 20A1C0  NODIG     JSR DIV10

                * MOST SIGNIFICANT DIGIT IN
                * ACCUMULATOR
                * GO IN RECURSION
```

```
C097: 2089C0            JSR PUTINT
C09A: 0930    PUTDIG    ORA #'0
C09C: 20D2FF            JSR BSOUT
C09F: 68                PLA
C0A0: 60                RTS

C0A1: A200    DIV10     LDX #0
C0A3: 8604              STX REMAIN
C0A5: 8605              STX REMAIN+1
C0A7: A010              LDY #16
C0A9: 0602    DIV1      ASL EXPR
C0AB: 2603              ROL EXPR+1
C0AD: 2604              ROL REMAIN
C0AF: 2605              ROL REMAIN+1
C0B1: 38                SEC
C0B2: A504              LDA REMAIN
C0B4: E90A              SBC #10
C0B6: AA                TAX
C0B7: A505              LDA REMAIN+1
C0B9: E900              SBC #0
C0BB: 9006              BCC DIV2
C0BD: 8604              STX REMAIN
C0BF: 8505              STA REMAIN+1
C0C1: E602              INC EXPR
C0C3: 88      DIV2      DEY
C0C4: D0E3              BNE DIV1

                * REMAINDER IN ACCU

C0C6: A504              LDA REMAIN
C0C8: 60                RTS

PHYSICAL ENDADDRESS: $C0C9

*** NO WARNINGS

EXPR                    $02
REMAIN                  $04
BSOUT                   $FFD2
CR                      $0D
TEST                    MACRO
EXAMPLES                $C000     UNUSED
```

PUTINTS	$C071
PUTSGN	$C086
PUTINT	$C089
NODIG	$C094
PUTDIG	$C09A
DIV10	$C0A1
DIV1	$C0A9
DIV2	$C0C3

4.1 THE FILESYSTEM

4.1 the filesytem

The commodore 64 has the usual file system for a COMMODORE computer. This system handles up to 10 files simultaneously. In BASIC there is enough support to easily handle the IO (OPEN, PRINT# etc.). How can you do this in machine language ?
The OS gives you the highest ROM locations in a jumptable (through vectors) in order to perform the different subroutines easily.
The following is a brief explanation of the various routines.

Name : SETFPA
Address : $FFBA
Influence: none

Set file parameters. The logical file number has to be in the accumulator. The X-register contains the devicenumber and the Y-register the secondary address.

Name : SETFNA
Address : $FFBD
Influence: none

Set file name. Accu contains name length, X-register the lowbyte of address of the file name, Y-register the highbyte of that address.

Name : OPEN
Address : $FFC0
Influence: all registers

OPEN file. First you have set file parameters as well as the filename. If carry is set an error occurred during opening of the file.

Name : CLOSE
Address : $FFC3
Influence: all registers

CLOSE file. Accu contains the file number of file to be closed.

Name : CHKIN
Address : $FFC6
Influence: all registers

Set input to file/device. X-register contains the file number of an already opened file. OS will henceforth input data from the device belonging to the file. A true carrybit indicates an error.

Name : CKOUT
Address : $FFC9
Influence: all registers

Set output to file/device. X-register contains the file number of an already opened file. OS will henceforth output data to the device belonging to the file. A true carrybit indicates an error.

Name : CLRCH
Address : $FFCC
Influence: all registers

Reset input/output to default devices. Input from keyboard (device 0) and output to screen (device 3).

Name : BASIN
Address : $FFCF
Influence: Y and Accu

Input byte from the input device. Default from keyboard. BASIN waits till a carriage return and will then transfer all bytes via the accu included the Carriage Return (CR). Carry indicates an error.

Name : BSOUT
Address : $FFD2
Influence: none

Output byte to the output device. Default to screen. Accu must contain byte to be output. If an error occurred the carry will be set.

Name : GET
Address : $FFE4
Influence: Y and Accu

GET byte from the input device. Default obtained from keyboard. If no data is available, a zero will be returned. Carry indicates error.

Name : CLRALL
Address : $FFE7
Influence: all registers

Close all files and perform a CLRCH.

The following minor programs shows you some examples of opening files on different devices in machine language with equivalent BASIC statements.

```
****************************************
*                                      *
*         OPEN CASSETTE FOR            *
*                                      *
*         INPUT  IN BASIC:             *
*                                      *
*         OPEN1,1,0,"NAME"             *
*                                      *
****************************************

              SETFPA   EQU $FFBA
              SETFNA   EQU $FFBD
              OPEN     EQU $FFC0

                       ORG $C000

C000: A901             LDA #1
C002: A201             LDX #1
C004: A000             LDY #0
C006: 20BAFF           JSR SETFPA
C009: A215             LDX #FNAME:L
C00B: A0C0             LDY #FNAME:H
C00D: A904             LDA #FNLEN
C00F: 20BDFF           JSR SETFNA
C012: 20C0FF           JSR OPEN

              * USER MAINFRAME

              *
              *
              *

              * THE FILENAME

C015: 4E414D FNAME     ASC "NAME"
C018: 45
              FNLEN    EQU *-FNAME
```

```
PHYSICAL ENDADDRESS: $C019

*** NO WARNINGS

SETFPA              $FFBA
SETFNA              $FFBD
OPEN                $FFC0
FNAME               $C015
FNLEN               $04

*****************************************
*                                       *
*         OPEN CASSETTE FOR             *
*                                       *
*         OUTPUT  IN BASIC:             *
*                                       *
*         OPEN1,1,1,"NAME"              *
*                                       *
*****************************************

               SETFPA   EQU $FFBA
               SETFNA   EQU $FFBD
               OPEN     EQU $FFC0

                        ORG $C000

C000: A901              LDA #1
C002: A201              LDX #1
C004: A001              LDY #1
C006: 20BAFF            JSR SETFPA
C009: A215              LDX #FNAME:L
C00B: A0C0              LDY #FNAME:H
C00D: A904              LDA #FNLEN
C00F: 20BDFF            JSR SETFNA
C012: 20C0FF            JSR OPEN

               * USER MAINFRAME

               *
               *
               *

               * THE FILENAME
```

```
C015: 4E414D FNAME     ASC "NAME"
C018: 45
               FNLEN    EQU *-FNAME

PHYSICAL ENDADDRESS: $C019

*** NO WARNINGS

SETFPA             $FFBA
SETFNA             $FFBD
OPEN               $FFC0
FNAME              $C015
FNLEN              $04

****************************************
*                                      *
*         OPEN DISK FOR                *
*                                      *
*         INPUT  IN BASIC:             *
*                                      *
*         OPEN1,8,6,"0:NAME"           *
*                                      *
****************************************

               SETFPA   EQU $FFBA
               SETFNA   EQU $FFBD
               OPEN     EQU $FFC0

                        ORG $C000

C000: A901             LDA #1
C002: A208             LDX #8
C004: A006             LDY #6
C006: 20BAFF           JSR SETFPA
C009: A215             LDX #FNAME:L
C00B: A0C0             LDY #FNAME:H
C00D: A906             LDA #FNLEN
C00F: 20BDFF           JSR SETFNA
C012: 20C0FF           JSR OPEN

               * USER MAINFRAME

               *
```

```
                *
                *
                * THE FILENAME

C015: 303A4E FNAME     ASC "0:NAME"
C018: 414D45
                FNLEN     EQU *-FNAME

PHYSICAL ENDADDRESS: $C01B

*** NO WARNINGS

SETFPA              $FFBA
SETFNA              $FFBD
OPEN                $FFC0
FNAME               $C015
FNLEN               $06

*****************************************
*                                       *
*         OPEN DISK FOR                 *
*                                       *
*         OUTPUT  IN BASIC:             *
*                                       *
*         OPEN1,8,6,"0:NAME,S,W"        *
*                                       *
*****************************************
                SETFPA    EQU $FFBA
                SETFNA    EQU $FFBD
                OPEN      EQU $FFC0

                          ORG $C000

C000: A901                LDA #1
C002: A208                LDX #8
C004: A006                LDY #6
C006: 20BAFF              JSR SETFPA
C009: A215                LDX #FNAME:L
C00B: A0C0                LDY #FNAME:H
C00D: A90A                LDA #FNLEN
C00F: 20BDFF              JSR SETFNA
C012: 20C0FF              JSR OPEN
```

```
                * USER MAINFRAME

                *
                *
                *

                * THE FILENAME

C015: 303A4E FNAME     ASC "0:NAME,S,W"
C018: 414D45
C01B: 2C532C
C01E: 57
                FNLEN     EQU *-FNAME

PHYSICAL ENDADDRESS: $C01F

*** NO WARNINGS

SETFPA              $FFBA
SETFNA              $FFBD
OPEN                $FFC0
FNAME               $C015
FNLEN               $0A
```

The built-in RS232 interface will be described in the next chapter.
Programmers often want to stop an IO activity by pressing the STOP key. OS supports this by a scanroutine called STOPQ ($FFE1). The zeroflag will be true if the STOP key is pressed. With BEQ and BNE it is easy to decide what to do in your program.

4.2 Save and load programs in OS.

If you wish to save or to load a range of the memory, you can use the build in SAVE and LOAD routines.

```
Name      : LOAD
Address   : $FFD5
Influence: all registers
```

Load a program in memory (X contains lowbyte, Y highbyte of start addres). The accu indicates, if a verify (A=1) or a load (A=0) shall be performed. It is necessary to first set the different types of parameters (device, secondary address) and the filename. If the secondary address is equal zero the program will be loaded as a relative BASIC program. A one as secondary address will force an absolute load. The carry indicates an error.

Name : SAVE
Address : $FFD8
Influence: all registers

SAVE a program (X contains lowbyte of end address, Y the highbyte and the accu points to the two locations in the zeropage, which contain the start address). The parameters and file name must be set first. You can save the program with different secondary addresses:

sa = 0 save as BASIC program
sa = 1 save as machine language program
sa = 2 save as BASIC with EOT block.
sa = 3 save as machine language with EOT.

The carrybit indicates an error.

The following programming examples are for disk and cassette.

```
****************************************
*                                      *
*         LOAD A PROGRAM               *
*                                      *
*         FROM CASSETTE                *
*                                      *
****************************************

                START     EPZ $02.3

                SETFPA    EQU $FFBA
                SETFNA    EQU $FFBD
                BSOUT     EQU $FFD2
                LOAD      EQU $FFD5

                SECADR    EQU 0
                LOADF     EQU 0
                DEVNUM    EQU 1

                          ORG $C000

                * SET SECADR & DEVICENUMBER

C000: A000                LDY #SECADR
C002: A201                LDX #DEVNUM
C004: 20BAFF              JSR SETFPA

                * SET FILENAME & LENGTH

C007: A222                LDX #FNAME:L
C009: A0C0                LDY #FNAME:H
C00B: A904                LDA #FLEN
C00D: 20BDFF              JSR SETFNA

                * ADDRESS WHERE TO PUT THE
                * PROGRAM IN (X,Y)
                * ACCU = 0 FOR LOAD

C010: A602                LDX START
C012: A403                LDY START+1
C014: A900                LDA #LOADF
C016: 20D5FF              JSR LOAD
C019: B001                BCS LOADERR
```

```
                * BACK TO MONITOR

C01B: 00                BRK

                * C=1 THEN ERROR PUT '?

C01C: A93F    LOADERR   LDA #'?
C01E: 20D2FF            JSR BSOUT
C021: 00                BRK

                * THE FILENAME
C022: 544553  FNAME     ASC "TEST"
C025: 54
                FLEN    EQU *-FNAME

PHYSICAL ENDADDRESS: $C026

*** NO WARNINGS

START               $02
SETFPA              $FFBA
SETFNA              $FFBD
BSOUT               $FFD2
LOAD                $FFD5
SECADR              $00
LOADF               $00
DEVNUM              $01
LOADERR             $C01C
FNAME               $C022
FLEN                $04

****************************************
*                                      *
*         LOAD A PROGRAM               *
*                                      *
*         FROM DISK                    *
*                                      *
****************************************
                START   EPZ $02.3

                SETFPA  EQU $FFBA
                SETFNA  EQU $FFBD
```

```
                BSOUT   EQU $FFD2
                LOAD    EQU $FFD5

                SECADR  EQU 0
                LOADF   EQU 0
                DEVNUM  EQU 8

                        ORG $C000

                * SET SECADR & DEVICENUMBER

C000: A000              LDY #SECADR
C002: A208              LDX #DEVNUM
C004: 20BAFF            JSR SETFPA

                * SET FILENAME & LENGTH

C007: A222              LDX #FNAME:L
C009: A0C0              LDY #FNAME:H
C00B: A906              LDA #FLEN
C00D: 20BDFF            JSR SETFNA

                * ADDRESS WHERE TO PUT THE
                * PROGRAM IN (X,Y)
                * ACCU = 0 IS LOAD

C010: A602              LDX START
C012: A403              LDY START+1
C014: A900              LDA #LOADF
C016: 20D5FF            JSR LOAD
C019: B001              BCS LOADERR

                * RETURN TO MONITOR

C01B: 00                BRK

                * C=1 THEN ERROR PUT '?

C01C: A93F      LOADERR LDA #'?
C01E: 20D2FF            JSR BSOUT
C021: 00                BRK

                * THE FILENAME
```

```
C022: 303A54 FNAME     ASC "0:TEST"
C025: 455354
               FLEN      EQU *-FNAME

PHYSICAL ENDADDRESS: $C028

*** NO WARNINGS

START              $02
SETFPA             $FFBA
SETFNA             $FFBD
BSOUT              $FFD2
LOAD               $FFD5
SECADR             $00
LOADF              $00
DEVNUM             $08
LOADERR            $C01C
FNAME              $C022
FLEN               $06

****************************************
*                                      *
*         SAVE A PROGRAM               *
*                                      *
*         TO CASSETTE                  *
*                                      *
****************************************

               START     EPZ $02.3
               END       EPZ $04.5

               SETFPA    EQU $FFBA
               SETFNA    EQU $FFBD
               BSOUT     EQU $FFD2
               SAVE      EQU $FFD8
               SECADR    EQU 1
               DEVNUM    EQU 1


                         ORG $C000

               * SET SECADR & DEVICENUMBER
```

```
C000: A201              LDX #DEVNUM
C002: A001              LDY #SECADR
C004: 20BAFF            JSR SETFPA

                * SET FILENAME & LENGTH

C007: A222              LDX #FNAME:L
C009: A0C0              LDY #FNAME:H
C00B: A904              LDA #FLEN
C00D: 20BDFF            JSR SETFNA

                * ENDADDRES IN (X,Y)
                * POINTER TO STARTADDRESS
                * IN ACCUMULATOR

C010: A604              LDX END
C012: A405              LDY END+1
C014: A902              LDA #START
C016: 20D8FF            JSR SAVE
C019: B001              BCS SAVEERR

                * RETURN TO MONITOR

C01B: 00                BRK

                * C=1 THEN ERROR PUT '?

C01C: A93F    SAVEERR   LDA #'?
C01E: 20D2FF            JSR BSOUT
C021: 00                BRK

                * THE FILENAME

C022: 544553  FNAME     ASC "TEST"
C025: 54
                FLEN    EQU *-FNAME

PHYSICAL ENDADDRESS: $C026
```

```
*** NO WARNINGS

START               $02
END                 $04
SETFPA              $FFBA
SETFNA              $FFBD
BSOUT               $FFD2
SAVE                $FFD8
SECADR              $01
DEVNUM              $01
SAVEERR             $C01C
FNAME               $C022
FLEN                $04

****************************************
*                                      *
*         SAVE A PROGRAM               *
*                                      *
*         TO DISK                      *
*                                      *
****************************************
              START     EPZ $02.3
              END       EPZ $04.5

              SETFPA    EQU $FFBA
              SETFNA    EQU $FFBD
              BSOUT     EQU $FFD2
              SAVE      EQU $FFD8

              SECADR    EQU 1
              DEVNUM    EQU 8



                        ORG $C000

              * SET SECADR & DEVICENUMBER

C000: A208                LDX #DEVNUM
C002: A001                LDY #SECADR
C004: 20BAFF              JSR SETFPA

              * SET FILENAME & LENGTH
```

```
C007: A222              LDX #FNAME:L
C009: A0C0              LDY #FNAME:H
C00B: A906              LDA #FLEN
C00D: 20BDFF            JSR SETFNA

                * ENDADDRESS IN (X,Y)
                * POINTER TO STARTADDRESS
                * IN ACCUMULATOR

C010: A604              LDX END
C012: A405              LDY END+1
C014: A902              LDA #START
C016: 20D8FF            JSR SAVE
C019: B001              BCS SAVEERR

                * RETURN TO MONITOR

C01B: 00                BRK
                * C=1 THEN ERROR PUT '?

C01C: A93F    SAVEERR   LDA #'?
C01E: 20D2FF            JSR BSOUT
C021: 00                BRK

                * THE FILENAME

C022: 303A54  FNAME     ASC "0:TEST"
C025: 455354
                FLEN    EQU *-FNAME
PHYSICAL ENDADDRESS: $C028

*** NO WARNINGS
START               $02
END                 $04
SETFPA              $FFBA
SETFNA              $FFBD
BSOUT               $FFD2
SAVE                $FFD8
SECADR              $01
DEVNUM              $08
SAVEERR             $C01C
FNAME               $C022
FLEN                $06
```

5.1 THE RS232 INTERFACE

5.1 the rs232 interface

One powerful feature of the COMMODORE 64 is the built-in RS232 interface. This bidirectional interface allows you to hook up printers as well as modems and terminals. How do you connect into the RS232 and program with it?
The RS232 interface uses the userport on the rear of the C64. Do not mistake this with the expansion port on the right side. The pinout of the userport is as follows:

Userport lines

PIN ID	6526 ID	description	in/out
C	PB0	received data	in
D	PB1	request to send	out
E	PB2	data term. ready	out
F	PB3	ring indicator	in
H	PB4	recvd. line signal	in
J	PB5	unassigned	in
K	PB6	clear to send	in
L	PB7	data set ready	in

B	FLG2	received data	in
M	PA2	transmitted data	out
A	GND	protective ground	
N	GND	signal ground	

As you can see, there are two lines with the same meaning: to receive data. To have the RS232 function properly both of these lines have to be connected. Further, you have to invert all in and outgoing lines, otherwise nothing will happen.

The RS232 interface needs a filename consisting of two bytes. The setting of the bits of these bytes is essential for the way the RS232 interface has to work.

In BASIC you have to convert the values of these bytes into two decimal numbers and use these numbers within CHR$ commands to build the filename. In machine language the X and Y register have to contain the address where both bytes are located. The accumulator has to be loaded with 2 (length of filename) then you have to call the SETFNA routine. The devicenumber of the RS232 interface is two. The secondary address should be zero.

The first byte is the control register byte. Its bits have the following meanings:

bit 0-3 Baud rate 3:2:1:0

		-+-+-+-
50	baud	0:0:0:1
75	baud	0:0:1:0
110	baud	0:0:1:1
134.5	baud	0:1:0:0
150	baud	0:1:0:1
300	baud	0:1:1:0
600	baud	0:1:1:1
1200	baud	1:0:0:0
1800	baud	1:0:0:1
2400	baud	1:0:1:0

bit 4 unused

bit 5-6 data word length

	6:5
8 bits	0:0
7 bits	0:1
6 bits	1:0
5 bits	1:1

bit 7 stopbits

	7
1 stopb.	0
2 stopb.	1

The second byte is the command register byte. It has the following mapping:

bit 0 handshake

	0
0-3 line	0
X line	1

bit 1-3 unused

bit 4 duplex

	4
full dupl	0
half dupl	1

bit 5-7 parity options

	7:6:5
disabled	*:*:0
odd parity	0:0:1
even parity	0:1:1
mark transm.	1:0:1
space transm.	1:1:1

By example you need the following setting for your printer:

2 stopbits
8 databits
300 baud
3 lines
half duplex
parity disabled

The two bytes should be $86 and $10 (hexadecimal). Now your RS232 interface will, after opening, work with these settings. You only have to invert the transmitted data line and connect it together with signal ground to your printer (i.e. QUME Sprint).
The RS232 interface builds two buffers during opening. Each has a capacity of 256 bytes (one for receiving the other for sending data). These two buffers will be allocated at the top of RAM.
The following program shows how to open files on the RS232 interface for input as well for output.

```
*****************************************
*                                       *
*        OPEN RS232 FOR IN & OUTPUT     *
*                                       *
*        IN BASIC:                      *
*                                       *
*        OPEN1,2,0,CHR$(134)+CHR$(16)   *
*                                       *
*****************************************

            SETFPA   EQU $FFBA
            SETFNA   EQU $FFBD
            OPEN     EQU $FFC0


                     ORG $C000

C000: A901           LDA #1
C002: A202           LDX #2
C004: A000           LDY #0
```

```
C006: 20BAFF            JSR SETFPA
C009: A215              LDX #FNAME:L
C00B: A0C0              LDY #FNAME:H
C00D: A902              LDA #FNLEN
C00F: 20BDFF            JSR SETFNA
C012: 20C0FF            JSR OPEN

                * USER MAIN PROGRAM

                *
                *
                *

                * THE FILENAME
                * RS232 SETTINGS

C015: 8610      FNAME   DFB 134,16
                FNLEN   EQU 2   ;2 BYTES

PHYSICAL ENDADDRESS: $C017

*** NO WARNINGS

SETFPA              $FFBA
SETFNA              $FFBD
OPEN                $FFC0
FNAME               $C015
FNLEN               $02
```

6.1 THE REALTIME CLOCK

6.1 the realtime clock

Another feature of the C64 is the built-in realtime clock. The realtime clock is in the Complex Interface Adapter (CIA) 6526. This CIA is a new peripheral chip of the 65XX family. The start address of that chip in the C64 is $DC00.
The realtime clock is an AM/PM clock with an accuracy of 1/10th of a second. It has four different registers (1/10s, seconds, minutes and hours). The contents are organized as BCD numbers, so that it is very easy to convert them to ASCII numbers. In addition to the actual time there is also an ALARM time. If both are equal the clock will force an IRQ. To use this alarm function you have to change the soft IRQ vector to your own routine and poll the Interrupt Control Register (ICR $DC0D). If bit 2 is set alarm and realtime are equal. To select the alarm registers instead of the realtime registers bit 7 of the Control Register B (CRB $DC0F) must be one. The following registers are used for the realtime clock:

TOD10TH	$DC08	bit 0-3 : 1/10th second bit 4-7 : always zero
TODSEC	$DC09	bit 0-3 : seconds bit 4-6 : 10 seconds bit 7 : zero

```
TODMIN    $DC0A   bit 0-3 : minutes
                  bit 4-6 : 10 minutes
                  bit 7   : zero

TODHR     $DC0B   bit 0-3 : hours
                  bit 4   : 10 hours
                  bit 5-6 : zero
                  bit 7   : 0=AM 1=PM

ICR       $DC0D   bit 2   : alarm = actual(1)

CRA       $DC0E   bit 7   : 1=50Hz 0=60Hz

CRB       $DC0F   bit 7   : 1=setting alarm
                          : 0=setting realt.
```

If you set the time, (hours first) the clock stops automatically until TOD10TH is also set.
Reading the time causes the clock to store all registers in temporary registers, able to read, until you have read the TOD10TH then the registers will again show the actual time.
As you can see, once the realtime clock is set, it always gives the correct time (once set).
The following program sets the time and goes into a polling loop, to scan the time registers and to display their contents.

```
*****************************************
*                                       *
*         REALTIMECLOCK                 *
*                                       *
*****************************************

              TMP       EPZ $FB
              AUX       EPZ $FD.E

              STACK     EQU $100
```

```
                CIA1      EQU $DC00
                TOD10TH   EQU CIA1+8
                TODSEC    EQU CIA1+9
                TODMIN    EQU CIA1+10
                TODHR     EQU CIA1+11
                CRB       EQU CIA1+15

                BSOUT     EQU $FFD2
                GET       EQU $FFE4

                CR        EQU 13

                          ORG $C000

                * CLEAR SCREEN

C000: A993                LDA #19+128
C002: 20D2FF              JSR BSOUT

                * SET AM OR PM

C005: 20FFC0 PMSET        JSR PRINT
C008: 0D                  DFB CR
C009: 41204F              ASC \A OR P : \
C00C: 522050
C00F: 203AA0

C012: 202CC1              JSR INPUT
C015: A200                LDX #0
C017: C941                CMP 'A
C019: F006                BEQ SETAM
C01B: C950                CMP 'P
C01D: D0E6                BNE PMSET

                * AM -> X=0
                * PM -> X=128
C01F: A280                LDX #128
C021: 86FC    SETAM       STX TMP+1


                * SET HOURS

C023: 20FFC0 HRSET        JSR PRINT
```

```
C026: 0D              DFB CR
C027: 484F55          ASC \HOURS: \
C02A: 52533A
C02D: A0
C02E: 2034C1          JSR NINPUT
C031: B0F0            BCS HRSET

                * >= 20 THEN AGAIN

C033: C902            CMP #2
C035: B0EC            BCS HRSET
C037: 85FB            STA TMP
C039: 2034C1          JSR NINPUT
C03C: B0E5            BCS HRSET

                * ACCU + TMP := BCD

C03E: 2021C1          JSR COMPATMP

                * SET AM OR PM

C041: 05FC            ORA TMP+1

                * AND STORE IN REGISTER

C043: 8D0BDC          STA TODHR

                * SET MINUTES

C046: 20FFC0 MINSET   JSR PRINT
C049: 0D              DFB CR
C04A: 4D494E          ASC \MIN   : \
C04D: 20203A
C050: A0
C051: 2034C1          JSR NINPUT
C054: B0F0            BCS MINSET

                * >= 60 THEN ERROR

C056: C906            CMP #6
C058: B0EC            BCS MINSET
C05A: 85FB            STA TMP
C05C: 2034C1          JSR NINPUT
```

```
C05F: B0E5              BCS MINSET
C061: 2021C1            JSR COMPATMP
C064: 8D0ADC            STA TODMIN

                * SET SECONDS

C067: 20FFC0 SECSET     JSR PRINT
C06A: 0D                DFB CR
C06B: 534543            ASC \SEC  : \
C06E: 20203A
C071: A0
C072: 2034C1            JSR NINPUT
C075: B0F0              BCS SECSET

                * >= 60 THEN ERROR

C077: C906              CMP #6
C079: B0EC              BCS SECSET
C07B: 85FB              STA TMP
C07D: 2034C1            JSR NINPUT
C080: B0E5              BCS SECSET
C082: 2021C1            JSR COMPATMP
C085: 8D09DC            STA TODSEC

                * SET 1/10 SECONDS

C088: 20FFC0 SET10TH    JSR PRINT
C08B: 0D                DFB CR
C08C: 312F31            ASC \1/10S: \
C08F: 30533A
C092: A0
C093: 2034C1            JSR NINPUT
C096: B0F0              BCS SET10TH

                * ONLY ONE DIGIT

C098: 8D08DC            STA TOD10TH


                * CLEAR SCREEN

C09B: A993              LDA #19+128
C09D: 20D2FF            JSR BSOUT
```

```
                * THE DISPLAY LOOP
                * CUSROR HOME

C0A0: A913    LOOP      LDA #19
C0A2: 20D2FF            JSR BSOUT

                * AM OR PM ?

C0A5: 2C0BDC            BIT TODHR
C0A8: 1004              BPL AM
C0AA: A950              LDA 'P
C0AC: D002              BNE PRINTM
C0AE: A941    AM        LDA 'A
C0B0: 20D2FF  PRINTM    JSR BSOUT
C0B3: A94D              LDA 'M
C0B5: 20D2FF            JSR BSOUT
C0B8: A920              LDA '
C0BA: 20D2FF            JSR BSOUT

                * PRINT HOURS FOLLOWED BY
                * A COLON

C0BD: AD0BDC            LDA TODHR
C0C0: 297F              AND #$7F
C0C2: 20EFC0            JSR PRTDIGS
C0C5: A93A              LDA ':
C0C7: 20D2FF            JSR BSOUT

                * DO THE SAME FOR MINUTES

C0CA: AD0ADC            LDA TODMIN
C0CD: 20EFC0            JSR PRTDIGS
C0D0: A93A              LDA ':
C0D2: 20D2FF            JSR BSOUT

                * AND FOR SECONDS

C0D5: AD09DC            LDA TODSEC
C0D8: 20EFC0            JSR PRTDIGS
C0DB: 20FFC0            JSR PRINT
C0DE: BA                ASC \:\
```

```
                    * PRINT 1/10 SECONDS
                    * AND FAKE 1/100

C0DF: AD08DC                LDA TOD10TH
C0E2: 0930                  ORA '0
C0E4: 20D2FF                JSR BSOUT
C0E7: A930                  LDA '0
C0E9: 20D2FF                JSR BSOUT


C0EC: 4CA0C0                JMP LOOP



                    * PRINT DCB BYTE AS TWO
                    * DECIMAL DIGITS

C0EF: 48          PRTDIGS   PHA
C0F0: 4A                    LSR
C0F1: 4A                    LSR
C0F2: 4A                    LSR
C0F3: 4A                    LSR
C0F4: 20FAC0                JSR DIGOUT
C0F7: 68                    PLA
C0F8: 290F                  AND #%00001111
C0FA: 0930        DIGOUT    ORA '0
C0FC: 4CD2FF                JMP BSOUT

                    * PRINT STRING ROUTINE
                    * AS DISCUSSED IN
                    * CHAPTER TWO
C0FF: 68          PRINT     PLA
C100: 85FD                  STA AUX
C102: 68                    PLA
C103: 85FE                  STA AUX+1
C105: A200                  LDX #0
C107: E6FD        PRINT1    INC AUX
C109: D002                  BNE *+4
C10B: E6FE                  INC AUX+1
C10D: A1FD                  LDA (AUX,X)
C10F: 297F                  AND #$7F
C111: 20D2FF                JSR BSOUT
C114: A200                  LDX #0
```

```
C116: A1FD              LDA (AUX,X)
C118: 10ED              BPL PRINT1
C11A: A5FE              LDA AUX+1
C11C: 48                PHA
C11D: A5FD              LDA AUX
C11F: 48                PHA
C120: 60                RTS


                * PACK ACCU AND TMP
                * IN ONE BYTE

C121: 06FB    COMPATMP  ASL TMP
C123: 06FB              ASL TMP
C125: 06FB              ASL TMP
C127: 06FB              ASL TMP
C129: 05FB              ORA TMP
C12B: 60                RTS

                * WAIT FOR ANY KEY
                * AND SHOW IT ON SCREEN

C12C: 20E4FF  INPUT     JSR GET
C12F: F0FB              BEQ INPUT
C131: 4CD2FF            JMP BSOUT

                * GET ASCII DIGIT AND
                * CONVERT TO BINARY
                * IF ILLEGAL DIGIT SET
                * CARRY BIT ON

C134: 202CC1  NINPUT    JSR INPUT
C137: C930              CMP '0
C139: 9007              BCC NERROR
C13B: C93A              CMP '9+1
C13D: B003              BCS NERROR
C13F: 290F              AND #%00001111
C141: 60                RTS
C142: 38      NERROR    SEC
C143: 60                RTS

PHYSICAL ENDADDRESS: $C144
```

```
*** NO WARNINGS

TMP             $FB
AUX             $FD
STACK           $0100     UNUSED
CIA1            $DC00
TOD10TH         $DC08
TODSEC          $DC09
TODMIN          $DC0A
TODHR           $DC0B
CRB             $DC0F     UNUSED
BSOUT           $FFD2
GET             $FFE4
CR              $0D
PMSET           $C005
SETAM           $C021
HRSET           $C023
MINSET          $C046
SECSET          $C067
SET10TH         $C088
LOOP            $C0A0
AM              $C0AE
PRINTM          $C0B0
PRTDIGS         $C0EF
DIGOUT          $C0FA
PRINT           $C0FF
PRINT1          $C107
COMPATMP        $C121
INPUT           $C12C
NINPUT          $C134
NERROR          $C142
```

This program has one big disadvantage. Your computer is doing nothing meaningful most of the time. We need a better method to show us the time continuously without wasting time. There is only one solution. Use the interrupt. The following program can be activated from BASIC.

With SYS12*4096 a dialogue will be started to set the time. After completion return to BASIC.

SYS12*4096+3 switches on the display mode. In the upper left corner a digital clock will be displayed showing you the actual time. As you can see, you can continue your normal computer session.

SYS12*4096+6 switches off the display mode, but the time still keeps running.

The clock will be reset when you turn your computer off and back on.

```
*****************************************
*                                       *
*         REALTIMECLOCK                 *
*                                       *
*         WITH INTERRUPT DISPLAY        *
*                                       *
*****************************************

              TMP       EPZ $FB.C
              OLD10TH   EPZ $FC
              POINT     EPZ $FD
              AUX       EPZ $FD.E

              ACTCOL    EQU $0286

              IRQVECT   EQU $0314

              SCREEN    EQU $0400
              COLOR     EQU $D800

              CIA1      EQU $DC00
              TOD10TH   EQU CIA1+8
              TODSEC    EQU CIA1+9
              TODMIN    EQU CIA1+10
              TODHR     EQU CIA1+11
              CRB       EQU CIA1+15

              BSOUT     EQU $FFD2
              GET       EQU $FFE4

              OLDIRQ    EQU $EA31
```

```
                CR          EQU 13
                HOME        EQU 19
                CLS         EQU HOME+128

                            ORG $C000

                * SET TIME SYS12*4096

C000: 4C09C0                JMP SET

                * DISPLAY TIME CONTINUOES
                * SYS 12*4096+3

C003: 4C2CC0                JMP INSTALL

                * SWITCH DISPLAY OFF
                * SYS 12*4096+6

C006: 4C39C0                JMP LEAVE

                * POINT IRQVECTOR TO OWN
                * HANDLER

                * CLEAR SCREEN
C009: A993      SET         LDA #CLS
C00B: 20D2FF                JSR BSOUT

                * SET CRB FOR SETTING
                * REALTIME

C00E: AD0FDC                LDA CRB
C011: 297F                  AND #%01111111
C013: 8D0FDC                STA CRB

                * SHOW MESSAGE
                * SET REALTIME
                * AS YOU CAN SEE EASY TO
                * EXTEND FOR ALARMTIME TOO

C016: 2047C1                JSR PRINT
C019: 0D                    DFB CR
C01A: 534554                ASC \SET TIME OF DAY\
C01D: 205449
```

```
C020: 4D4520
C023: 4F4620
C026: 4441D9
C029: 4C46C0            JMP SETCLOCK


                * SET DISPLAY ON

C02C: 78        INSTALL SEI
C02D: A9CF              LDA #OWNIRQ:L
C02F: 8D1403            STA IRQVECT
C032: A9C0              LDA #OWNIRQ:H
C034: 8D1503            STA IRQVECT+1
C037: 58                CLI
C038: 60                RTS


                * SWITCH DISPLAY OFF

C039: 78        LEAVE   SEI
C03A: A931              LDA #OLDIRQ:L
C03C: 8D1403            STA IRQVECT
C03F: A9EA              LDA #OLDIRQ:H
C041: 8D1503            STA IRQVECT+1
C044: 58                CLI
C045: 60                RTS



                * SET CONTENTS OF TOD
                * REGISTERS

                * AM -> X=0
                * PM -> X=128

C046: 2047C1 SETCLOCK JSR PRINT

C049: 0D                DFB CR
C04A: 41204F            ASC \A OR P : \
C04D: 522050
C050: 203AA0
C053: 2074C1            JSR INPUT
C056: A200              LDX #0
```

```
C058: C941              CMP 'A
C05A: F006              BEQ SETAM
C05C: C950              CMP 'P
C05E: D0E6              BNE SETCLOCK
C060: A280              LDX #128
C062: 86FC    SETAM     STX TMP+1

                * SET HOURS

C064: 2047C1  HRSET     JSR PRINT
C067: 0D                DFB CR
C068: 484F55            ASC \HOURS: \
C06B: 52533A
C06E: A0
C06F: 207CC1            JSR NINPUT
C072: B0F0              BCS HRSET

                * >= 20 THEN ERROR

C074: C902              CMP #2
C076: B0EC              BCS HRSET
C078: 85FB              STA TMP
C07A: 207CC1            JSR NINPUT
C07D: B0E5              BCS HRSET
C07F: 2069C1            JSR COMPATMP
C082: 05FC              ORA TMP+1
C084: 8D0BDC            STA TODHR

                * SET MINUTES

C087: 2047C1  MINSET    JSR PRINT
C08A: 0D                DFB CR
C08B: 4D494E            ASC \MIN   : \
C08E: 20203A
C091: A0
C092: 207CC1            JSR NINPUT
C095: B0F0              BCS MINSET

                * >= 60 THEN ERROR

C097: C906              CMP #6
C099: B0EC              BCS MINSET
C09B: 85FB              STA TMP
```

```
C09D: 207CC1           JSR NINPUT
C0A0: B0E5             BCS MINSET
C0A2: 2069C1           JSR COMPATMP
C0A5: 8D0ADC           STA TODMIN

               * SET SECONDS

C0A8: 2047C1 SECSET    JSR PRINT
C0AB: 0D               DFB CR
C0AC: 534543           ASC \SEC   : \
C0AF: 20203A
C0B2: A0
C0B3: 207CC1           JSR NINPUT
C0B6: B0F0             BCS SECSET

               * >= 60 THEN ERROR

C0B8: C906             CMP #6
C0BA: B0EC             BCS SECSET
C0BC: 85FB             STA TMP
C0BE: 207CC1           JSR NINPUT
C0C1: B0E5             BCS SECSET
C0C3: 2069C1           JSR COMPATMP
C0C6: 8D09DC           STA TODSEC

               * 1/10 SECONDS ALWAYS 0

C0C9: A900             LDA #0
C0CB: 8D08DC           STA TOD10TH
C0CE: 60               RTS


               * IRQ HANDLER
               * IF 1/10 S DIFFERENT THEN
               * SHOW NEW TIME
               * ELSE JUMP TO OS HANDLER

C0CF: AD08DC OWNIRQ    LDA TOD10TH
C0D2: C5FC             CMP OLD10TH
C0D4: F005             BEQ NORMIRQ
C0D6: 85FC             STA OLD10TH
C0D8: 20DEC0           JSR SHOWTIM
C0DB: 4C31EA NORMIRQ   JMP OLDIRQ  OS IRQ
```

```
                * DISPLAY TIME WITH POKING
                * IN THE SCREEN


C0DE: 78        SHOWTIM  SEI

                * RESET OUTPUT POINTER

C0DF: A900               LDA #0
C0E1: 85FD               STA POINT

                * AM OR PM OUT

C0E3: 2C0BDC             BIT TODHR
C0E6: 1004               BPL AM
C0E8: A910               LDA 'P-'@
C0EA: D002               BNE PRINTM
C0EC: A901      AM       LDA 'A-'@
C0EE: 2038C1    PRINTM   JSR ZOUTPUT
C0F1: A920               LDA '
C0F3: 2038C1             JSR ZOUTPUT

                * PRINT HOURS FOLLOWED BY
                * A COLON

C0F6: AD0BDC             LDA TODHR
C0F9: 297F               AND #$7F
C0FB: 2028C1             JSR PRTDIGS
C0FE: A93A               LDA ':
C100: 2038C1             JSR ZOUTPUT

                * SAME FOR MINUTES

C103: AD0ADC             LDA TODMIN
C106: 2028C1             JSR PRTDIGS
C109: A93A               LDA ':
C10B: 2038C1             JSR ZOUTPUT

                * AND SECONDS

C10E: AD09DC             LDA TODSEC
C111: 2028C1             JSR PRTDIGS
C114: A93A               LDA ':
```

```
C116: 2038C1          JSR ZOUTPUT

                * DISPLAY 1/10 SECONDS
                * AND FAKE 1/100 (ZERO)

C119: AD08DC          LDA TOD10TH
C11C: 0930            ORA '0
C11E: 2038C1          JSR ZOUTPUT
C121: A930            LDA '0
C123: 2038C1          JSR ZOUTPUT
C126: 58              CLI
C127: 60              RTS


                * PRINT BCD BYTE AS TWO
                * DIGITS

C128: 48      PRTDIGS PHA
C129: 4A              LSR
C12A: 4A              LSR
C12B: 4A              LSR
C12C: 4A              LSR
C12D: 2033C1          JSR DIGOUT
C130: 68              PLA
C131: 290F            AND #%00001111
C133: 0930    DIGOUT  ORA '0
C135: 4C38C1          JMP ZOUTPUT


                * POKE CHARACTER ON SCREEN

C138: A6FD    ZOUTPUT LDX POINT
C13A: 9D0004          STA SCREEN,X
C13D: AD8602          LDA ACTCOL
C140: 9D00D8          STA COLOR,X
C143: E8              INX
C144: 86FD            STX POINT
C146: 60              RTS

                * THE WELL KNOWN PRINT
                * STRING ROUTINE FROM
                * CHAPTER TWO
```

```
C147: 68      PRINT    PLA
C148: 85FD             STA AUX
C14A: 68               PLA
C14B: 85FE             STA AUX+1
C14D: A200             LDX #0
C14F: E6FD    PRINT1   INC AUX
C151: D002             BNE *+4
C153: E6FE             INC AUX+1
C155: A1FD             LDA (AUX,X)
C157: 297F             AND #$7F
C159: 20D2FF           JSR BSOUT
C15C: A200             LDX #0
C15E: A1FD             LDA (AUX,X)
C160: 10ED             BPL PRINT1
C162: A5FE             LDA AUX+1
C164: 48               PHA
C165: A5FD             LDA AUX
C167: 48               PHA
C168: 60               RTS

                * MAKE ONE BYTE FROM ACCU
                * AND TMP (BCD)

C169: 06FB    COMPATMP ASL TMP
C16B: 06FB             ASL TMP
C16D: 06FB             ASL TMP
C16F: 06FB             ASL TMP
C171: 05FB             ORA TMP
C173: 60               RTS

                * GET ANY KEY AND SHOW IT

C174: 20E4FF  INPUT    JSR GET
C177: F0FB             BEQ INPUT
C179: 4CD2FF           JMP BSOUT

                * GET DIGIT
                * IF INVALLID CARRYBIT ON

C17C: 2074C1  NINPUT   JSR INPUT
C17F: C930             CMP '0
C181: 9007             BCC NERROR
C183: C93A             CMP '9+1
```

```
C185: B003              BCS NERROR
C187: 290F              AND #%00001111
C189: 60                RTS
C18A: 38      NERROR    SEC
C18B: 60                RTS

PHYSICAL ENDADDRESS: $C18C

*** NO WARNINGS

TMP                 $FB
OLD10TH             $FC
POINT               $FD
AUX                 $FD
ACTCOL              $0286
IRQVECT             $0314
SCREEN              $0400
COLOR               $D800
CIA1                $DC00
TOD10TH             $DC08
TODSEC              $DC09
TODMIN              $DC0A
TODHR               $DC0B
CRB                 $DC0F
BSOUT               $FFD2
GET                 $FFE4
OLDIRQ              $EA31
CR                  $0D
HOME                $13
CLS                 $93
SET                 $C009
INSTALL             $C02C
LEAVE               $C039
SETCLOCK            $C046
SETAM               $C062
HRSET               $C064
MINSET              $C087
SECSET              $C0A8
OWNIRQ              $C0CF
NORMIRQ             $C0DB
SHOWTIM             $C0DE
AM                  $C0EC
PRINTM              $C0EE
```

PRTDIGS	$C128
DIGOUT	$C133
ZOUTPUT	$C138
PRINT	$C147
PRINT1	$C14F
COMPATMP	$C169
INPUT	$C174
NINPUT	$C17C
NERROR	$C18A

7.1 HOW TO ADD NEW BASIC COMMANDS?

7.1 How to add new BASIC commands ?

In BASIC it is possible to start your own machine language routines with the SYS command. It is also possible to add your own commands by putting a wedge in the interpreter loop of BASIC.
First let us look at the use of the SYS command for executing your own commands in BASIC. If you have a program that you want to execute from BASIC, there is a safe memory range where you can place it. This safe memory range starts at $C000 (12*4096) and ends at $CFFF. All our examples are using this area.
Sometimes you want your routines to have parameters to be given by the BASIC program they are started from. We will discuss some of these routines and show you a sample program.

Name : CHECKCOM
Address : $AEFD

This routine checks for a comma (', ') and skips it. The BASIC pointer now points directly after the comma. If no comma is found BASIC will stop and print an error message.

Name : GETCOORD
Address : $B7EB

This routine will obtain two numbers seperated by a comma. The first number (unsigned integer) is a 16 bit number and will be stored in locations $14 (LSB) and $15 (MSB). The second number is a byte expression. This means it is a number in the range 0-255. The value will be in the X-register. If the two numbers are out of their ranges an error will occurr and BASIC stops. It also stops by a missing comma.

Name : GETBYTE
Address : $B79E

GETBYTE will fetch a byte expression (0-255) and transfer it to the X-register. If the expression is out of range BASIC will stop and an error occurs.

Name : GETPARAM
Address : $E1D4

GETPARAM retrieves the file name followed by the device number etc. It will give an error if something is incorrect with the filespecification.

The program in the next chapter will use these routines.
Instead of the SYS command you can use your own keywords.
You can do this by placeing a wedge into the interpreter loop. The next figure shows us a part of that loop:

```
A7E1 6C0803  JMP ($0308)    ; points to next
                            ; instr. $A7E4
```

```
A7E4 207300   JSR $0073      ; CHRGET gets
                             ; next character
A7E7 20EDA7   JSR $A7ED      ; execute BASIC
                             ; statement
A7EA 4CAEA7   JMP $A7AE      ; jmp to begin
                             ; of loop
```

As you can see, you can change the pointer ($0308, $0309) and let it point to your own decoder. This decoder program checks for one of your own keywords. If you do not wish to do this mearly jump to $A7E7 and the interpreter will continue. You may execute your own program and jump after completion to $A7AE (begin of interpreter loop).

A common method of recognizing your own commands, is to preceed them with an unique symbol such as '@,'! etc. You have only to check that symbol, look in your own command table, and if it is a legal command, perform it.

Chapter 9 will discuss a program (diskutiliyties) that uses additional commands.

If you want to know more about routines in BASIC, please refer a C64 memory map and ROM listing.

8.1 HIRES ASSISTENT

8.1 Hires Assistent

The following program allows you to plot in the HirRes graphic mode of your C64. This means you can rapidly set or reset graphic dots in BASIC by using this programming aid.
It is also an example of using SYS commands to perform your own BASIC routines.
As you can see the program uses a jumptable, so you have only to add 3 to get the start address of the next command. For example, SYS12*4096 will initialize the screen. SYS12*4096+3 clears the screen and SYS12*4096+6,5 will set the background color 5.
You can save the object code of this program via a monitor program (i. e. SUPERMON) and load in BASIC with the LOAD-command followed by file name, device number and secondary address equal to one. The program will then be loaded absolute at address $C000. After loading, type NEW. Now you can enter or load the BASIC program that will use the plot routines etc.
This HiRes program allows you to load and save the screen on cassette or disk.
The program uses routines discussed earlier in this book. A good exercise would be to extend the program with a drawto command, so you can quickly draw lines.

The following BASIC program is a test for the HiRes Assistent.

```
5 REM HIRES TESTDEMO
10 INPUT"X1=VALUE ";X1
20 INPUT"Y1=VALUE ";Y1
30 INPUT"X2=VALUE ";X2
40 INPUT"Y2=VALUE ";Y2
45 SYS12*4096:SYS12*4096+3
55 DX=X2-X1:DY=Y2-Y1
57 IF ABS(DY)<ABS(DX) THEN 64
58 FOR YL=Y1 TO Y2 STEP SGN(DY)
60 SYS12*4096+9,DX/DY*YL+X1,YL
62 NEXT YL
63 GOTO 300
64 FOR XL=X1 TO X2 STEP SGN(DX)
65 SYS12*4096+9,XL,DY/DX*XL+Y1
68 NEXT XL
300 IF ABS(DY)<ABS(DX) THEN 350
310 FOR YL=Y1 TO Y2 STEP SGN(DY)
320 SYS12*4096+12,DX/DY*YL+X1,YL
330 NEXT YL
340 GOTO 380
350 FOR XL=X1 TO X2 STEP SGN(DX)
360 SYS12*4096+12,XL,DY/DX*XL+Y1
370 NEXT XL
380 GOTO 57
```

```
*****************************************
*                                       *
*         HIRES GRAPHIC                 *
*                                       *
*         ASSISTENT                     *
*                                       *
*****************************************

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

              COLORLOW  EQU $0400
              COLORHI   EQU $0800
```

```
                GRAPHICL EQU $2000
                GRAPHICH EQU $4000

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

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

                VIDEO    EQU $D000

                FALSE    EQU 255
                TRUE     EQU 0

                CLS      EQU 19+128

                         ORG $C000

                * INIT HIRES GRAPHIC
                * SYS12*4096

C000: 4C18C0             JMP INIT

                * CLEAR HIRES SCREEN
                * SYS12*4096+3

C003: 4C33C0             JMP CLEAR

                * SET BACKGROUND COLOR
                * SYS12*4096+6,COLOR
C006: 4C4AC0             JMP COLOR

                * PLOT X,Y (0 <= X < 320)
                *           (0 <= Y < 200)
                * SYS12*4096+9,X,Y

C009: 4C6BC0             JMP SET

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

```
C00C: 4C67C0            JMP RESET

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

C00F: 4CD8C0            JMP SWTCHOFF

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

C012: 4CE9C0            JMP SCREENSA

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

C015: 4C00C1            JMP SCREENLO


                * INIT HIRES SCREEN

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


                * CLEAR HIRES SCREEN

C033: A000   CLEAR      LDY #0
C035: A920              LDA #GRAPHICL:H
C037: 84FD              STY TEMP
C039: 85FE              STA TEMP+1
C03B: 98     CLEAR1     TYA
C03C: 91FD   CLEAR2     STA (TEMP),Y
C03E: C8                INY
C03F: D0FB              BNE CLEAR2
```

```
C041: E6FE              INC TEMP+1
C043: A5FE              LDA TEMP+1
C045: C940              CMP #GRAPHICH:H
C047: D0F2              BNE CLEAR1
C049: 60                RTS

                * SET BACK COLOR

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

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

                * (RE)SET DOT AT X,Y

C067: A9FF   RESET      LDA #FALSE
C069: D002              BNE SET1      A.T.
C06B: A900   SET        LDA #TRUE
C06D: 8D53C1 SET1       STA RSFLG
C070: 20FDAE            JSR CHECKCOM
C073: 20EBB7            JSR GETCOORD
C076: E0C8              CPX #200
C078: B0EC              BCS OUTRANGE
C07A: A514              LDA XCOORD
C07C: C940              CMP #320:L
C07E: A515              LDA XCOORD+1
C080: E901              SBC #320:H
C082: B0E2              BCS OUTRANGE
C084: 8A                TXA
```

```
C085: 4A                LSR
C086: 4A                LSR
C087: 4A                LSR
C088: 0A                ASL
C089: A8                TAY
C08A: B90FC1            LDA MUL320,Y
C08D: 85FD              STA ADDRESS
C08F: B910C1            LDA MUL320+1,Y
C092: 85FE              STA ADDRESS+1
C094: 8A                TXA
C095: 2907              AND #%00000111
C097: 18                CLC
C098: 65FD              ADC ADDRESS
C09A: 85FD              STA ADDRESS
C09C: A5FE              LDA ADDRESS+1
C09E: 6900              ADC #0
C0A0: 85FE              STA ADDRESS+1
C0A2: A514              LDA XCOORD
C0A4: 2907              AND #%00000111
C0A6: A8                TAY
C0A7: A514              LDA XCOORD
C0A9: 29F8              AND #%11111000
C0AB: 18                CLC
C0AC: 65FD              ADC ADDRESS
C0AE: 85FD              STA ADDRESS
C0B0: A5FE              LDA ADDRESS+1
C0B2: 6515              ADC XCOORD+1
C0B4: 85FE              STA ADDRESS+1
C0B6: A5FD              LDA ADDRESS
C0B8: 18                CLC
C0B9: 6900              ADC #GRAPHICL:L
C0BB: 85FD              STA ADDRESS
C0BD: A5FE              LDA ADDRESS+1
C0BF: 6920              ADC #GRAPHICL:H
C0C1: 85FE              STA ADDRESS+1
C0C3: A200              LDX #0
C0C5: A1FD              LDA (ADDRESS,X)
C0C7: 2C53C1            BIT RSFLG
C0CA: 1006              BPL SET2
C0CC: 3949C1            AND ANDMASK,Y
C0CF: 4CD5C0            JMP SET3
C0D2: 1941C1  SET2      ORA ORMASK,Y
C0D5: 81FD    SET3      STA (ADDRESS,X)
```

```
C0D7: 60                RTS

                * SWITCH GRAPHIC OFF BACK
                * TO NORMAL MODE

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

                * SAVE HIRES SCREENMEMORY

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

                * LOAD HIRES SCREENMEMORY

C100: 20FDAE SCREENLO JSR CHECKCOM
C103: 20D4E1          JSR GETPARAM
C106: A961            LDA #6*16+1
C108: 85B9            STA SECADR
C10A: A900            LDA #0
C10C: 4CD5FF          JMP LOAD

                N       EQU 320

                * MULTIPLY TABLE

C10F: 000040 MUL320 DFW 0*N,1*N,2*N,3*N,4*N
C112: 018002
C115: C00300
C118: 05
C119: 400680        DFW 5*N,6*N,7*N,8*N,9*N
```

```
C11C: 07C008
C11F: 000A40
C122: 0B
C123: 800CC0   DFW 10*N,11*N,12*N,13*N,14*N
C126: 0D000F
C129: 401080
C12C: 11
C12D: C01200   DFW 15*N,16*N,17*N,18*N,19*N
C130: 144015
C133: 8016C0
C136: 17
C137: 001940   DFW 20*N,21*N,22*N,23*N,24*N
C13A: 1A801B
C13D: C01C00
C140: 1E

               * SET MASK FOR BIT WITHIN
               * THE SELECTED BYTE

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


               * CLEAR MASK FOR BIT WITHIN
               * THE SELECTED BYTE

C149: 7F       ANDMASK   DFB %01111111
C14A: BF                 DFB %10111111
C14B: DF                 DFB %11011111
C14C: EF                 DFB %11101111
C14D: F7                 DFB %11110111
C14E: FB                 DFB %11111011
C14F: FD                 DFB %11111101
C150: FE                 DFB %11111110
C151: 0000     SCRATCH   DFW 0  SAVE OLD VALUS
C153: 00       RSFLG     DFB 0  SET OR RESET
C154: 00       YCOORD    DFB 0  YCOORDINATES
```

```
PHYSICAL ENDADDRESS: $C155

*** NO WARNINGS

XCOORD          $14
SECADR          $B9
TEMP            $FD
ADDRESS         $FD
COLORLOW        $0400
COLORHI         $0800
GRAPHICL        $2000
GRAPHICH        $4000
CHECKCOM        $AEFD
GETBYTE         $B79E
GETCOORD        $B7EB
GETPARAM        $E1D4
BSOUT           $FFD2
LOAD            $FFD5
SAVE            $FFD8
VIDEO           $D000
FALSE           $FF
TRUE            $00
CLS             $93
INIT            $C018
CLEAR           $C033
CLEAR1          $C03B
CLEAR2          $C03C
COLOR           $C04A
COLOR1          $C050
COLOR2          $C058
COLOR3          $C059
OUTRANGE        $C066
RESET           $C067
SET             $C06B
SET1            $C06D
SET2            $C0D2
SET3            $C0D5
SWTCHOFF        $C0D8
SCREENSA        $C0E9
SCREENLO        $C100
N               $0140
MUL320          $C10F
ORMASK          $C141
```

ANDMASK	$C149	
SCRATCH	$C151	
RSFLG	$C153	
YCOORD	$C154	UNUSED

9.1 DISKUTILITY

9.1 Diskutility

The program discussed here, adds following new disk commands to BASIC:

DLOAD

Like the well known LOAD command but the devicenumber is always 8. The filename may be followed by the secondary address. Refer to your COMMODORE manual for those details.

DSAVE

The SAVE command for device 8. Like DLOAD.

DPRINT

This command allows you directly to send commands to your disk in one statement. Like NEW, COPY, RENAME etc. It is similar to the following sequence in BASIC:

```
OPEN15,8,15,"<command>":CLOSE15
```

DERROR

Shows you the disk status. You will no longer need following program:

```
10 OPEN15,8,15
20 INPUT#15,A$,B$,C$,D$
30 PRINTA$,B$,C$,D$
40 CLOSE15
```

Thus the DERROR command will print the ERROR No. , ERROR MESSAGE, TRACK No. and SECTOR No.

DLIST

This command shows you the directory of the disk. But it doesn't destroy the actual BASIC program.

You load the utility program with the secondary address equal to one (absolute loading). After loading, type NEW and SYS12*4096. Now you can use the new commands.
SYS12*4096 calls a program, that places a wedge in the interpreter loop (by changing the vector $0308-$0309 to the new decoder routine).
The program may have some helpful comments.

```
****************************************
*                                      *
*         DISK UTILITIES               *
*                                      *
****************************************
              FLAG      EPZ $0A
              BASICP    EPZ $7A.B
              ST        EPZ $90
              FNLENGTH  EPZ $B7
              SECADR    EPZ $B9
              DEVNUM    EPZ $BA
              FNADRESS  EPZ $BB.C
              TMP       EPZ $FB.C

              CHRGET    EQU $0073
```

```
                EXECUTE  EQU $A7E7
                INTERP   EQU $A7AE
                LNPRT    EQU $BDCD
                PARGET   EQU $E200
                GETNAME  EQU $E257
                NEXTQ    EQU $E206
                GIVERR   EQU $E0F9

                SENDFNAM EQU $F3D5
                CLOSEFIL EQU $F642
                SENDSEC  EQU $FF96
                IECINP   EQU $FFA5
                UNTALK   EQU $FFAB
                IECTALK  EQU $FFB4
                SETFPA   EQU $FFBA
                SETFNA   EQU $FFBD
                OPEN     EQU $FFC0
                CLOSE    EQU $FFC3
                CHKIN    EQU $FFC6
                CLRCH    EQU $FFCC
                BASIN    EQU $FFCF
                BSOUT    EQU $FFD2
                LOAD     EQU $FFD5
                BLOAD    EQU $E16F
                BSAVE    EQU $E159

                LOADTOK  EQU 147
                SAVETOK  EQU 148
                VERITOK  EQU 149
                PRINTTOK EQU 153
                ORTOKEN  EQU 176
                LISTTOK  EQU 155
                CR       EQU 13

                         ORG $C000


                * ADD NEW COMMANDS
                * BY SYS12*4096

C000: A90B      INSTALL  LDA #DECODE:L
C002: 8D0803             STA $0308
C005: A9C0               LDA #DECODE:H
```

```
C007: 8D0903            STA $0309
C00A: 60                RTS

                * OWN DECODER
                * ALL ADDITIONAL COMMANDS
                * START WITH A 'D

C00B: 207300 DECODE     JSR CHRGET
C00E: C944              CMP #'D
C010: F003              BEQ FOUND
C012: 4CE7A7            JMP EXECUTE


                * CHECK FOR OWN COMMAND

C015: A001   FOUND      LDY #1
C017: B17A              LDA (BASICP),Y

                * DLOAD ?

C019: C993              CMP #LOADTOK
C01B: F031              BEQ DLOAD

                * NO! DSAVE ?

C01D: C994              CMP #SAVETOK
C01F: F040              BEQ DSAVE

                * NO! DVERIFY ?

C021: C995              CMP #VERITOK
C023: F026              BEQ DVERIFY

                * NO! DPRINT ?

C025: C999              CMP #PRINTTOK
C027: F01C              BEQ DOPRINT

                * NO! DLIST ?

C029: C99B              CMP #LISTTOK
C02B: F01B              BEQ DODLIST
```

```
                * NO! DERROR

C02D: A200              LDX #0
C02F: B17A    COMPLOOP  LDA (BASICP),Y
C031: DD64C1            CMP DERRTAB,X
C034: D008              BNE NDERROR
C036: C8                INY
C037: E8                INX
C038: E004              CPX #4
C03A: 90F3              BCC COMPLOOP
C03C: B051              BCS DERROR

                * NO! RETURN WITH CORRECT
                * TOKEN IN ACCU

C03E: A000    NDERROR   LDY #0
C040: B17A              LDA (BASICP),Y
C042: 4CE7A7            JMP EXECUTE


C045: 4CC5C0  DOPRINT   JMP DPRINT


C048: 4CF4C0  DODLIST   JMP DLIST

                * DVERIFY VERIFIES PROGRAM
                * ON DISK WITH THAT IN
                * MEMORY (SA OPTIONAL)
                * USE: DVERIFY"NAME",SA

C04B: A901    DVERIFY   LDA #1
C04D: 2C                DFB $2C

                * DLOAD LOADS PROGRAM FROM
                * DISK INTO MEMORY
                * (SA OPTIONAL)
                * USE: DLOAD"NAME",SA

C04E: A900    DLOAD     LDA #0
C050: 850A              STA FLAG
C052: 207300            JSR CHRGET
C055: 207300            JSR CHRGET
C058: 2070C0            JSR GETPAR
```

```
C05B: 206FE1          JSR BLOAD
C05E: 4CAEA7          JMP INTERP

                * DSAVE SAVES PROGRAM TO
                * DISK FROM MEMORY
                * (SA OPTIONAL)
                * USE: DSAVE"NAME",SA

C061: 207300 DSAVE    JSR CHRGET
C064: 207300          JSR CHRGET
C067: 2070C0          JSR GETPAR
C06A: 2059E1          JSR BSAVE
C06D: 4CAEA7          JMP INTERP
                * GETPARAMETER ROUTINE
                * TO GET FILENAME AND
                * SECONDARY ADDRESS
                * DEVICENUMBER ALWAYS 8

C070: A900   GETPAR   LDA #0
C072: 20BDFF          JSR SETFNA
C075: A208            LDX #8
C077: A000            LDY #0
C079: 20BAFF          JSR SETFPA
C07C: 2006E2          JSR NEXTQ
C07F: 2057E2          JSR GETNAME
C082: 2006E2          JSR NEXTQ
C085: 2000E2          JSR PARGET
C088: 8A              TXA
C089: A8              TAY
C08A: A208            LDX #8
C08C: 4CBAFF          JMP SETFPA

                * DERROR READS THE ERROR
                * CHANNEL (15) OF THE DISK
                * AND SHOWS ERROR NUMBER
                * MESSAGES,TRACK AND SECTOR
                * USE: DERROR

                * FIRST SKIP COMMAND

C08F: A57A   DERROR   LDA BASICP
C091: 18              CLC
C092: 6905            ADC #5
```

```
C094: 857A              STA BASICP
C096: A57B              LDA BASICP+1
C098: 6900              ADC #0
C09A: 857B              STA BASICP+1

                * OPEN CHANNEL AND READ IT

C09C: A900              LDA #0
C09E: 8590              STA ST
C0A0: A908              LDA #8
C0A2: 85BA              STA DEVNUM
C0A4: 20B4FF            JSR IECTALK
C0A7: A96F              LDA #15+6*16
C0A9: 85B9              STA SECADR
C0AB: 2096FF            JSR SENDSEC
C0AE: A490    DERRLOOP  LDY ST
C0B0: D00A              BNE DERR4
C0B2: 20A5FF            JSR IECINP
C0B5: 20D2FF            JSR BSOUT
C0B8: C90D              CMP #CR
C0BA: D0F2              BNE DERRLOOP
C0BC: 20ABFF  DERR4     JSR UNTALK
C0BF: 4CAEA7            JMP INTERP

C0C2: 4CF9E0  DERRERR   JMP GIVERR
                * DPRINT SENDS COMMANDS VIA
                * THE COMMAND CHANNEL 15
                * USE: DPRINT"<COMM>"
                * PLEASE REFER TO DISKMANUAL

C0C5: 207300  DPRINT    JSR CHRGET
C0C8: 207300            JSR CHRGET
C0CB: A90F              LDA #15
C0CD: 20C3FF            JSR CLOSE
C0D0: 20E0C0            JSR GETFPAR
C0D3: 20C0FF            JSR OPEN
C0D6: B0EA              BCS DERRERR
C0D8: A90F              LDA #15
C0DA: 20C3FF            JSR CLOSE
C0DD: 4CAEA7            JMP INTERP

                * SPECIAL GET PARAMETER
                * ROUTINE FOR DPRINT
```

```
C0E0: A900    GETFPAR  LDA #0
C0E2: 20BDFF           JSR SETFNA
C0E5: A90F             LDA #15
C0E7: A8               TAY
C0E8: A208             LDX #8
C0EA: 20BAFF           JSR SETFPA
C0ED: 2006E2           JSR NEXTQ
C0F0: 2057E2           JSR GETNAME
C0F3: 60               RTS

                * DLIST SHOWS DIRECTORY
                * OF THE DISK WITHOUT
                * DISTURBING MEMORY
                * USE: DLIST

C0F4: 207300  DLIST    JSR CHRGET
C0F7: 207300           JSR CHRGET
C0FA: A900             LDA #0
C0FC: 8590             STA ST
C0FE: A924             LDA '$
C100: 85FB             STA TMP
C102: A9FB             LDA #TMP:L
C104: 85BB             STA FNADRESS
C106: A900             LDA #TMP:H
C108: 85BC             STA FNADRESS+1
C10A: A901             LDA #1
C10C: 85B7             STA FNLENGTH
C10E: A908             LDA #8
C110: 85BA             STA DEVNUM
C112: A960             LDA #6*16
C114: 85B9             STA SECADR
C116: 20D5F3           JSR SENDFNAM
C119: A5BA             LDA DEVNUM
C11B: 20B4FF           JSR IECTALK
C11E: A5B9             LDA SECADR
C120: 2096FF           JSR SENDSEC
C123: A490             LDY ST
C125: D037             BNE DLIST4
C127: A006             LDY #6
C129: 84FB    DLIST1   STY TMP
C12B: 20A5FF           JSR IECINP
C12E: A6FC             LDX TMP+1
```

```
C130: 85FC              STA TMP+1
C132: A490              LDY ST
C134: D028              BNE DLIST4
C136: A4FB              LDY TMP
C138: 88                DEY
C139: D0EE              BNE DLIST1
C13B: A4FC              LDY TMP+1
C13D: 20CDBD            JSR LNPRT
C140: A920              LDA '
C142: 20D2FF            JSR BSOUT
C145: 20A5FF DLIST3     JSR IECINP
C148: A690              LDX ST
C14A: D012              BNE DLIST4
C14C: AA                TAX
C14D: F006              BEQ DLIST2
C14F: 20D2FF            JSR BSOUT
C152: 4C45C1            JMP DLIST3
C155: A90D   DLIST2     LDA #CR
C157: 20D2FF            JSR BSOUT
C15A: A004              LDY #4
C15C: D0CB              BNE DLIST1
C15E: 2042F6 DLIST4     JSR CLOSEFIL
C161: 4CAEA7            JMP INTERP

             * TABLE FOR 'ERROR' FOR
             * THE DERROR COMMAND

C164: 455252 DERRTAB    ASC "ERR"
C167: B0                DFB ORTOKEN

PHYSICAL ENDADDRESS: $C168

*** NO WARNINGS

FLAG                $0A
BASICP              $7A
ST                  $90
FNLENGTH            $B7
SECADR              $B9
DEVNUM              $BA
FNADRESS            $BB
TMP                 $FB
CHRGET              $73
```

EXECUTE	$A7E7	
INTERP	$A7AE	
LNPRT	$BDCD	
PARGET	$E200	
GETNAME	$E257	
NEXTQ	$E206	
GIVERR	$E0F9	
SENDFNAM	$F3D5	
CLOSEFIL	$F642	
SENDSEC	$FF96	
IECINP	$FFA5	
UNTALK	$FFAB	
IECTALK	$FFB4	
SETFPA	$FFBA	
SETFNA	$FFBD	
OPEN	$FFC0	
CLOSE	$FFC3	
CHKIN	$FFC6	UNUSED
CLRCH	$FFCC	UNUSED
BASIN	$FFCF	UNUSED
BSOUT	$FFD2	
LOAD	$FFD5	UNUSED
BLOAD	$E16F	
BSAVE	$E159	
LOADTOK	$93	
SAVETOK	$94	
VERITOK	$95	
PRINTTOK	$99	
ORTOKEN	$B0	
LISTTOK	$9B	
CR	$0D	
INSTALL	$C000	UNUSED
DECODE	$C00B	
FOUND	$C015	
COMPLOOP	$C02F	
NDERROR	$C03E	
DOPRINT	$C045	
DODLIST	$C048	
DVERIFY	$C04B	
DLOAD	$C04E	
DSAVE	$C061	
GETPAR	$C070	
DERROR	$C08F	

DERRLOOP	$C0AE
DERR4	$C0BC
DERRERR	$C0C2
DPRINT	$C0C5
GETFPAR	$C0E0
DLIST	$C0F4
DLIST1	$C129
DLIST3	$C145
DLIST2	$C155
DLIST4	$C15E
DERRTAB	$C164

10.1 HOW TO ADD DEVICEHANDLERS?

10.1 how to add devicehandlers ?

The C64 OS jumps by IO handling through vectors. These vectors point to OS routines which handle the IO. Therefore it is possible for the user to change these vectors and let point them to his own routines.
Now it is possible to add your own handler. First check if it is your own device. If it is your device, do your own devicehandling, else jump to the OS handler. The next chapter will describe such a handler (Centronics).
Now a description follows of the vectors used for IO.

031A - 031B OPEN vector.
points to the routine that opens and initializes the device stored in location $BA

031C - 031D CLOSE vector.
closes the file (# in accu) and the device belonging to file.

031E - 031F CHKIN vector.
stores the device number belonging to the file #

transferred in the
X-register in location $99.
It also, if needed,initializes
the device.

0320 - 0321 CKOUT vector.
same as CHKIN but now stores
the device number in loaction
$9A (actual outdevice)

0322 - 0323 CLRCH vector.
restores default in/outdevice
and resets the actual ones.

0324 - 0325 INPUT vector.
inputs character from device
(stored in $99). character has
to be in Accu no other reg.
may be changed.

0326 - 0327 OUTPUT vector.
outputs character (in Accu)
to the device (stored in $9A).
No registers may be changed.

032A - 032B GET vector.
Gets character from inputdev.
stored in $99. If no data is
available a zero will be
returned.
Accu contains the character,
other register may not be
changed.

032C - 032D CLALL vector.
closes all files. After
closing like CLRCH.

0330 - 0331 LOAD vector.
Loads or verifies program.
Refer to chapter 4.

0332 - 0333 SAVE vector.
Refer to chapter 4.

Possibly not all vectors are needed for your application. See the sample program in the next chapter for implementation of your own routines.
Leave all routines with carry clear if no error occurred.

11 AN INEXPENSIVE CENTRONICS INTERFACE

11 An inexpensive centronics interface

The following program allows you to connect a printer with a centronics interface at the userport of the C64.
The program will switch off the RS232 interface and install a centronix interface as device 2.
The centronics interface sends 7 bit ASCII. Bit 7 is always low.
You can use the normal IO commands. A A secondary address unequal zero will switch the linefeed mode on. That means an additional linefeed after a carriage return will be sent.
Following lines have to be connected:

Userport	to	Printer	description
L		1	STROBE
C		2	DATA1
D		3	DATA2
E		4	DATA3
F		5	DATA4
H		6	DATA5
J		7	DATA6
K		8	DATA7
N		9	DATA8
M		11	BUSY
N		19	GROUND

As you see, no components except two connectors are needed. For the userport you need a TRW CINCH 251-12-50-170/50-24sn-98124 connector. For the printer a printer dependent connector (see printermanual). You install the centronics interface with SYS12*4096.

```
****************************************
*                                      *
*        CENTRONICS VIA USERPORT       *
*                                      *
*        SYS 12*4096                   *
*                                      *
****************************************

             IOVECT   EQU $031A.2D

             CIA2     EQU $DD00
             PORTA    EQU CIA2
             PORTB    EQU CIA2+1
             DDRA     EQU CIA2+2
             DDRB     EQU CIA2+3

             CLRCH    EQU $F333
             STOPQ    EQU $F6ED
             CLALL    EQU $F32F

             CR       EQU 13
             LF       EQU 10


                      ORG $C000

             * SET OWN VECTORS

C000: A213   INSTALL  LDX #TABLEN
C002: BD0CC0 INSTLOOP LDA HANDTAB,X
C005: 9D1A03          STA IOVECT,X
C008: CA              DEX
C009: 10F7            BPL INSTLOOP
C00B: 60              RTS
```

```
                    * OWN VECTOR TABLES

C00C: 20C0    HANDTAB   DFW OPEN
C00E: 6BC0              DFW CLOSE
C010: 86C0              DFW CHKIN
C012: 9DC0              DFW CKOUT
C014: 33F3              DFW CLRCH
C016: B5C0              DFW BASIN
C018: C1C0              DFW BSOUT
C01A: EDF6              DFW STOPQ
C01C: FCC0              DFW GET
C01E: 2FF3              DFW CLALL

                    TABLEN    EQU (*-1)-HANDTAB


                    * THE OPEN ROUTINE

C020: A6B8    OPEN      LDX $B8
C022: D003              BNE NOTZER

                    * NOT INPUT FILE

C024: 4C0AF7            JMP $F70A

                    * SEARCH FOR FILE

C027: 200FF3  NOTZER    JSR $F30F
C02A: D003              BNE NOTFOUND

                    * ERROR 'FILE OPEN'

C02C: 4CFEF6            JMP $F6FE

C02F: A698    NOTFOUND  LDX $98
C031: E00A              CPX #10
C033: 9003              BCC NOTFULL
                    * #FILES > 10 THEN
                    * ERROR 'TOO MANY FILES'

C035: 4CFBF6            JMP $F6FB
```

```
C038: E698    NOTFULL  INC $98

                * FILE NUMBER

C03A: A5B8             LDA $B8
C03C: 9D5902           STA $0259,X

                * SEC ADDRESS

C03F: A5B9             LDA $B9
C041: 8D09C1           STA LFFLG
C044: 0960             ORA #%01100000
C046: 85B9             STA $B9
C048: 9D6D02           STA $026D,X

                * DEVICE NUMBER

C04B: A5BA             LDA $BA
C04D: 9D6302           STA $0263,X

                * DEVICE 2 ?

C050: C902             CMP #2
C052: F003             BEQ CTRXOPEN

                * NO DO OS ROUTINE

C054: 4C72F3           JMP $F372


                * OPEN CIA FOR PARALLEL

C057: A9FF    CTRXOPEN LDA #%11111111
C059: 8D03DD           STA DDRB
C05C: AD02DD           LDA DDRA
C05F: 29FB             AND #%11111011
C061: 8D02DD           STA DDRA
C064: A980             LDA #%10000000
C066: 8D01DD           STA PORTB
C069: 18               CLC
C06A: 60               RTS

                * OWN CLOSE
```

```
C06B: 2014F3 CLOSE    JSR $F314
C06E: F002            BEQ FOUND

                * NO FILE FOUND THEN
                * DO NOTHING

C070: 18              CLC
C071: 60              RTS

                * FOUND THEN SETPARA'S

C072: 201FF3 FOUND    JSR $F31F
C075: 8A              TXA
C076: 48              PHA
C077: A5BA            LDA $BA

                * DEVICE 2 ?

C079: C902            CMP #2
C07B: F003            BEQ CTRXCLOS

                * PERFORM OS ROUTINE

C07D: 4C9DF2          JMP $F29D


                * DO OWN CLOSE

C080: 68      CTRXCLOS PLA
C081: 20F2F2          JSR $F2F2

                * DO NOTHING

C084: 18              CLC
C085: 60              RTS


C086: 200FF3 CHKIN    JSR $F30F
C089: F003            BEQ FOUND2

                * ERROR FILE NOT OPEN

C08B: 4C01F7          JMP $F701
```

```
C08E: 201FF3 FOUND2    JSR $F31F
C091: A5BA             LDA $BA
C093: C902             CMP #2
C095: F003             BEQ CTRXCKIN
C097: 4C19F2           JMP $F219

               * CENTRONIX THEN
               * ERROR NO INPUT FILE

C09A: 4C0AF7 CTRXCKIN JMP $F70A

C09D: 200FF3 CKOUT     JSR $F30F
C0A0: F003             BEQ FOUND3

               * ERROR FILE NOT FOUND
C0A2: 4C01F7           JMP $F701

C0A5: 201FF3 FOUND3    JSR $F31F
C0A8: A5BA             LDA $BA

               * DEVICE 2 ?

C0AA: C902             CMP #2
C0AC: F003             BEQ CTRXCKOT

               * NO THEN OS

C0AE: 4C5BF2           JMP $F25B

               * YES THEN SET OUTPUT
               * FILE

C0B1: 859A   CTRXCKOT  STA $9A
C0B3: 18               CLC
C0B4: 60               RTS

C0B5: A599   BASIN     LDA $99
C0B7: C902             CMP #2
C0B9: F003             BEQ CTRXBSIN
C0BB: 4C57F1           JMP $F157

               * IF INPUT DEVICE 2
               * THEN 'NOT INPUT FILE'
```

```
C0BE: 4C0AF7 CTRXBSIN JMP $F70A

C0C1: 48     BSOUT    PHA
C0C2: A59A            LDA $9A
C0C4: C902            CMP #2
C0C6: F004            BEQ CTRXBSOT
C0C8: 68              PLA
C0C9: 4CCAF1          JMP $F1CA


              * THE CENTRONIX OUTPUT

C0CC: 8E08C1 CTRXBSOT STX XSAVE
C0CF: 68              PLA
C0D0: 20E8C0          JSR PAROUT
C0D3: C90D            CMP #CR
C0D5: D00C            BNE NOLF

              * IF SEC. ADDRESS > 0
              * THEN EXTRA LINEFEED

C0D7: AE09C1          LDX LFFLG
C0DA: F007            BEQ NOLF
C0DC: A90A            LDA #LF
C0DE: 20E8C0          JSR PAROUT
C0E1: A90D            LDA #CR
C0E3: AE08C1 NOLF     LDX XSAVE
C0E6: 18              CLC
C0E7: 60              RTS

              * THE PARALLEL OUTPUT

C0E8: 48     PAROUT   PHA

              * WAIT IF PRINTER BUSY
C0E9: AD00DD WAIT     LDA PORTA
C0EC: 2904            AND #%00000100
C0EE: D0F9            BNE WAIT
C0F0: 68              PLA

              * SET STROBE (BIT 7)
              * ON AND PUT DATA (BIT 0-6)
              * ON BUS
```

```
C0F1: 0980              ORA #%10000000
C0F3: 8D01DD            STA PORTB

                * SET STROBE OFF
                * AND HOLD IT DOWN TILL
                * NEXT SEND

C0F6: 297F              AND #%01111111
C0F8: 8D01DD            STA PORTB
C0FB: 60                RTS

C0FC: A599    GET       LDA $99
C0FE: C902              CMP #2
C100: F003              BEQ CTRXGET
C102: 4C3EF1            JMP $F13E

                * GET THEN ERROR
                * 'NOT INPUT FILE'

C105: 4C0AF7  CTRXGET   JMP $F70A

                * TEMPORARY XSAVE

C108: 00      XSAVE     DFB 0

                * COPY OF SEC. ADDRESS
                * FOR ADDITIONAL LINEFEED

C109: 00      LFFLG     DFB 0

PHYSICAL ENDADDRESS: $C10A

*** NO WARNINGS

IOVECT              $031A
CIA2                $DD00
PORTA               $DD00
PORTB               $DD01
DDRA                $DD02
DDRB                $DD03
CLRCH               $F333
STOPQ               $F6ED
CLALL               $F32F
```

```
CR          $0D
LF          $0A
INSTALL     $C000     UNUSED
INSTLOOP    $C002
HANDTAB     $C00C
TABLEN      $13
OPEN        $C020
NOTZER      $C027
NOTFOUND    $C02F
NOTFULL     $C038
CTRXOPEN    $C057
CLOSE       $C06B
FOUND       $C072
CTRXCLOS    $C080
CHKIN       $C086
FOUND2      $C08E
CTRXCKIN    $C09A
CKOUT       $C09D
FOUND3      $C0A5
CTRXCKOT    $C0B1
BASIN       $C0B5
CTRXBSIN    $C0BE
BSOUT       $C0C1
CTRXBSOT    $C0CC
NOLF        $C0E3
PAROUT      $C0E8
WAIT        $C0E9
GET         $C0FC
CTRXGET     $C105
XSAVE       $C108
LFFLG       $C109
```

12.1 PRETTYPRINT

12.1 Prettyprint

The next program is very helpful if you want to list your program within margins. This means no line exceeds a predefined maximum of characters.
The program changes the OUTPUT vector ($0326-$0327). With SYS12*4096 this vector will point to our own routine.
The default right margin is 40 but you can substitute any other value (0-255). The listing will be "shorter" but longer, because the program inserts additional CR LF's when a line reaches the maximum length. Thus lines longer than the defined maximum will be folded.

```
*****************************************
*                                       *
*         PRETTYPRINT                   *
*                                       *
*****************************************
              OUTVEC   EQU $0326

              CR       EQU 13

                       ORG $C000

              * SET UP NEW OUTVECTOR
              * SAVING THE OLD ONE
              * IN OLDOUT
              * CALL: SYS12*4096

C000: AD2603 INSTALL   LDA OUTVEC
C003: 8D33C0           STA OLDOUT
C006: AD2703           LDA OUTVEC+1
C009: 8D34C0           STA OLDOUT+1
```

```
C00C: A917              LDA #NEWOUT:L
C00E: 8D2603            STA OUTVEC
C011: A9C0              LDA #NEWOUT:H
C013: 8D2703            STA OUTVEC+1
C016: 60                RTS

                * THE NEW OUTROUTINE

C017: C90D    NEWOUT    CMP #CR
C019: F008              BEQ DOCR
C01B: CE35C0            DEC COUNT
C01E: D00B              BNE NADDCR
C020: 202FC0            JSR BSOUT2
C023: AD32C0  DOCR      LDA CPERL
C026: 8D35C0            STA COUNT
C029: A90D              LDA #CR
C02B: 202FC0  NADDCR    JSR BSOUT2
C02E: 60                RTS
                * DO OLDOUT

C02F: 6C33C0  BSOUT2    JMP (OLDOUT)

                * POKE IN THIS LOCATION
                * THE RIGHTMARGIN (49202)

C032: 28        CPERL   DFB 40
                OLDOUT  EQU *
                COUNT   EQU OLDOUT+2

PHYSICAL ENDADDRESS: $C033

*** NO WARNINGS

OUTVEC              $0326
CR                  $0D
INSTALL             $C000     UNUSED
NEWOUT              $C017
DOCR                $C023
NADDCR              $C02B
BSOUT2              $C02F
CPERL               $C032
OLDOUT              $C033
COUNT               $C035
```

13.1 SCREENPRINT ON PRINTER

13.1 screenprint on printer

The next program will make a hardcopy of the screen. You can start it directly from BASIC with SYS12*4096. The printer must have devicenumber 4. All graphic characters will be printed correctly because the program translates the screen code into ASCII. The only exception is reverse characters, which will be printed as non reverse ones.
Possibly it is a good exercise to implement reverse characters too, but it is not simple to solve.

```
****************************************
*                                      *
*         SCREENPRINT                  *
*                                      *
*         SYS12*4096                   *
*                                      *
****************************************

              TEMP      EPZ $FC
              PT        EPZ $FD.E

              SCREEN    EQU $0400

              SETFPA    EQU $FFBA
              SETFNA    EQU $FFBD
              OPEN      EQU $FFC0
              CLOSE     EQU $FFC3
              CKOUT     EQU $FFC9
```

```
                CLRCH     EQU $FFCC
                BSOUT     EQU $FFD2
                STOPQ     EQU $FFE1

                COLUMN    EQU 40
                ROW       EQU 25


                CR        EQU 13

                          ORG $C000

                * SET FILE#, DEVICE AND
                * SEC. ADDRESS

C000: A97F                LDA #127
C002: A204                LDX #4
C004: A000                LDY #0
C006: 20BAFF              JSR SETFPA

                * NO FILENAME

C009: A900                LDA #0
C00B: 20BDFF              JSR SETFNA

                * OPEN FILE

C00E: 20C0FF              JSR OPEN
C011: B04A                BCS QUIT

                * SET SCREEN START

C013: A900                LDA #SCREEN:L
C015: 85FD                STA PT
C017: A904                LDA #SCREEN:H
C019: 85FE                STA PT+1

                * SET OUTPUT DEVICE

C01B: A27F                LDX #127
C01D: 20C9FF              JSR CKOUT

                * #ROWS OF SCREEN
```

```
C020: A219              LDX #ROW

                * THE PRINTLOOP

C022: A90D    ROWLOOP   LDA #CR
C024: 20D2FF            JSR BSOUT
C027: 20E1FF            JSR STOPQ
C02A: F031              BEQ QUIT

                * NO STOPKEY

C02C: A000              LDY #0
C02E: B1FD    COLLOOP   LDA (PT),Y

                * CONVERT TO SCREENCODE

C030: 85FC              STA TEMP
C032: 293F              AND #%00111111
C034: 06FC              ASL TEMP
C036: 24FC              BIT TEMP
C038: 1002              BPL NOOR
C03A: 0980              ORA #%10000000
C03C: 7002    NOOR      BVS NOADDOR
C03E: 0940              ORA #%01000000
C040: 20D2FF  NOADDOR   JSR BSOUT
C043: C8                INY
C044: C028              CPY #COLUMN
C046: D0E6              BNE COLLOOP
C048: 98                TYA
C049: 18                CLC
C04A: 65FD              ADC PT
C04C: 85FD              STA PT
C04E: 9002              BCC *+4
C050: E6FE              INC PT+1
C052: CA                DEX
C053: D0CD              BNE ROWLOOP

                * ALL ROWS ? THEN CLOSE
C055: A90D              LDA #CR
C057: 20D2FF            JSR BSOUT
C05A: 20CCFF            JSR CLRCH
C05D: A97F    QUIT      LDA #127
C05F: 4CC3FF            JMP CLOSE
```

```
PHYSICAL ENDADDRESS: $C062

*** NO WARNINGS

TEMP            $FC
PT              $FD
SCREEN          $0400
SETFPA          $FFBA
SETFNA          $FFBD
OPEN            $FFC0
CLOSE           $FFC3
CKOUT           $FFC9
CLRCH           $FFCC
BSOUT           $FFD2
STOPQ           $FFE1
COLUMN          $28
ROW             $19
CR              $0D
ROWLOOP         $C022
COLLOOP         $C02E
NOOR            $C03C
NOADDOR         $C040
QUIT            $C05D
```

13.2 screenprint via RS232

To obtain a hardcopy of the screen on a RS232 printer, you can use following program. As you can see, it appears familiar to the one just described. There are some differences. First, the graphic characters will not be printed. Because the printer can't print them. The QUME Sprint, we were using, printed capitols instead of the graphic characters.
We used the following settings for the interface:

300 baud
8 databits
2 stopbits
3 line handshake
half duplex
parity disabled

Of course you are to change them.

```
****************************************
*                                      *
*        SCREENPRINT RS232             *
*                                      *
*        SYS12*4096                    *
*                                      *
****************************************
             TEMP      EPZ $FC
             PT        EPZ $FD.E

             SCREEN    EQU $0400

             SETFPA    EQU $FFBA
             SETFNA    EQU $FFBD
             OPEN      EQU $FFC0
             CLOSE     EQU $FFC3
             CKOUT     EQU $FFC9
             CLRCH     EQU $FFCC
             BSOUT     EQU $FFD2
             STOPQ     EQU $FFE1

             COLUMN    EQU 40
             ROW       EQU 25

             LF        EQU 10
             CR        EQU 13

             * RS232 SETTINGS SEE TEXT

             CONTROL   EQU %10000110
             COMMAND   EQU %00010000

                       ORG $C000

             * SET FILE#, DEVICE AND
             * SEC. ADDRESS

C000: A97F                LDA #127
C002: A202                LDX #2
C004: A000                LDY #0
C006: 20BAFF              JSR SETFPA
```

```
                * SET FILENAME (SETTINGS)
                * AND LENGTH (2 BYTES)

C009: A902              LDA #2
C00B: A26E              LDX #FNAME:L
C00D: A0C0              LDY #FNAME:H
C00F: 20BDFF            JSR SETFNA

                * OPEN FILE

C012: 20C0FF            JSR OPEN
                * STORE START OF SCREEN

C015: A900              LDA #SCREEN:L
C017: 85FD              STA PT
C019: A904              LDA #SCREEN:H
C01B: 85FE              STA PT+1

                * SET OUTDEVICE

C01D: A27F              LDX #127
C01F: 20C9FF            JSR CKOUT

                * THE PRINTLOOP

C022: A219              LDX #ROW
C024: A90D    ROWLOOP   LDA #CR
C026: 20D2FF            JSR BSOUT
C029: A90A              LDA #LF
C02B: 20D2FF            JSR BSOUT
C02E: 20E1FF            JSR STOPQ
C031: F036              BEQ QUIT
C033: A000              LDY #0
C035: B1FD    COLLOOP   LDA (PT),Y
C037: 85FC              STA TEMP
C039: 293F              AND #%00111111
C03B: 06FC              ASL TEMP
C03D: 24FC              BIT TEMP
C03F: 1002              BPL NOOR
C041: 0980              ORA #%10000000
C043: 7002    NOOR      BVS NOADDOR
C045: 0940              ORA #%01000000
C047: 20D2FF NOADDOR    JSR BSOUT
```

```
C04A: C8                INY
C04B: C028              CPY #COLUMN
C04D: D0E6              BNE COLLOOP
C04F: 98                TYA
C050: 18                CLC
C051: 65FD              ADC PT
C053: 85FD              STA PT
C055: 9002              BCC *+4
C057: E6FE              INC PT+1
C059: CA                DEX
C05A: D0C8              BNE ROWLOOP

                * ALL ROWS ?
                * THEN CLOSE

C05C: A90D              LDA #CR
C05E: 20D2FF            JSR BSOUT
C061: A90A              LDA #LF
C063: 20D2FF            JSR BSOUT
C066: 20CCFF            JSR CLRCH
C069: A97F    QUIT      LDA #127
C06B: 4CC3FF            JMP CLOSE

                * THE CONTROL & COMMAND BYTE

C06E: 8610    FNAME     DFB CONTROL,COMMAND

PHYSICAL ENDADDRESS: $C070

*** NO WARNINGS

TEMP                    $FC
PT                      $FD
SCREEN                  $0400
SETFPA                  $FFBA
SETFNA                  $FFBD
OPEN                    $FFC0
CLOSE                   $FFC3
CKOUT                   $FFC9
CLRCH                   $FFCC
BSOUT                   $FFD2
STOPQ                   $FFE1
COLUMN                  $28
```

```
ROW         $19
LF          $0A
CR          $0D
CONTROL     $86
COMMAND     $10
ROWLOOP     $C024
COLLOOP     $C035
NOOR        $C043
NOADDOR     $C047
QUIT        $C069
FNAME       $C06E
```

14.1 TERMINAL

14.1 terminal

The next program will simulate a terminal on your C64. It uses menus for the different settings. How is the menu used ? The menu technique is derived from the well known decision tree. You can compare it with dialing telephone numbers.
A menu exists of three different parts.

1. Show the diverse choices, which can be made.

2. Get the choice and check if it is legal.

3. Execute the choice.

First there will be a menu overview. Then you have to make a choice and the program continues according to your choice.
As you can see, it is now possible to make menues and submenues. So you have only to chose your "way" through them.
The following program is menu organized. After entering your choice the setting will be made and the program asks for the next setting.
After entering the last setting, the terminal will start.
You can now connect the C64 with a modem or a host computer.
To connect it with a smartmodem 300 from Hayes using half duplex, following settings have to be made:

300 baud
7 bits
1 stopbit
3 line handshake
half duplex
even parity

You can now dial the number of the computer, you will be connected with. As an exercise you can add an up and download facility. Do this by using the functionkeys.
You start the terminal by SYS12*4096 from BASIC. Pressing RUN/STOP together with RESTORE returns you to BASIC.

```
*****************************************
*                                       *
*         TERMINAL                      *
*                                       *
*         SYS12*4096                    *
*                                       *
*****************************************

ST          EPZ $90
AUX         EPZ $8E.F

SHFTCOM     EQU $0291

CHARGEN     EQU $D018

SCRNOUT     EQU $E716
FROMKBD     EQU $F142
SETFPA      EQU $FFBA
SETFNA      EQU $FFBD
OPEN        EQU $FFC0
CLOSE       EQU $FFC3
CHKIN       EQU $FFC6
CKOUT       EQU $FFC9
BSOUT       EQU $FFD2
GET         EQU $FFE4
CLALL       EQU $FFE7
```

```
                BS      EQU 08
                LF      EQU 10
                CR      EQU 13
                DEL     EQU 20
                CLS     EQU 19+128
                CURS    EQU 175
                BACKS   EQU 157


                        ORG $C000

                * CLEAR BOTH BYTES

C000: A900              LDA #0
C002: 8D0FC3            STA CONTROL
C005: 8D10C3            STA COMMAND

                * SET BAUDRATE

C008: 20BFC2            JSR CLRSCRN
C00B: 20C4C2            JSR PRINT
C00E: 302E20            ASC "0. 50     BAUD"
C011: 353020
C014: 202020
C017: 424155
C01A: 44
C01B: 0D                DFB CR
C01C: 312E20            ASC "1. 75     BAUD"
C01F: 373520
C022: 202020
C025: 424155
C028: 44
C029: 0D                DFB CR
C02A: 322E20            ASC "2. 110    BAUD"
C02D: 313130
C030: 202020
C033: 424155
C036: 44
C037: 0D                DFB CR
C038: 332E20            ASC "3. 134.5 BAUD"
C03B: 313334
C03E: 2E3520
C041: 424155
```

```
C044: 44
C045: 0D              DFB CR
C046: 342E20          ASC "4. 150   BAUD"
C049: 313530
C04C: 202020
C04F: 424155
C052: 44
C053: 0D              DFB CR
C054: 352E20          ASC "5. 300   BAUD"
C057: 333030
C05A: 202020
C05D: 424155
C060: 44
C061: 0D              DFB CR
C062: 362E20          ASC "6. 600   BAUD"
C065: 363030
C068: 202020
C06B: 424155
C06E: 44
C06F: 0D              DFB CR
C070: 372E20          ASC "7. 1200  BAUD"
C073: 313230
C076: 302020
C079: 424155
C07C: 44
C07D: 0D              DFB CR
C07E: 382E20          ASC "8. 1800  BAUD"
C081: 313830
C084: 302020
C087: 424155
C08A: 44
C08B: 0D              DFB CR
C08C: 392E20          ASC "9. 2400  BAUD"
C08F: 323430
C092: 302020
C095: 424155
C098: 44
C099: 8D              DFB CR+128
C09A: 20E6C2          JSR NINPUT
C09D: A8              TAY
C09E: AD0FC3          LDA CONTROL
C0A1: 19F6C2          ORA BAUDTAB,Y
C0A4: 8D0FC3          STA CONTROL
```

```
                * SET DATA BITS

C0A7: 20BFC2            JSR CLRSCRN
C0AA: 20C4C2            JSR PRINT
C0AD: 302E20            ASC "0. 8 DATABITS"
C0B0: 382044
C0B3: 415441
C0B6: 424954
C0B9: 53
C0BA: 0D                DFB CR
C0BB: 312E20            ASC "1. 7 DATABITS"
C0BE: 372044
C0C1: 415441
C0C4: 424954
C0C7: 53
C0C8: 0D                DFB CR
C0C9: 322E20            ASC "2. 6 DATABITS"
C0CC: 362044
C0CF: 415441
C0D2: 424954
C0D5: 53
C0D6: 0D                DFB CR
C0D7: 332E20            ASC "3. 5 DATABITS"
C0DA: 352044
C0DD: 415441
C0E0: 424954
C0E3: 53
C0E4: 8D                DFB CR+128
C0E5: 20E6C2 GETDAT     JSR NINPUT
C0E8: C904              CMP #4
C0EA: B0F9              BCS GETDAT
C0EC: A8                TAY
C0ED: AD0FC3            LDA CONTROL
C0F0: 1900C3            ORA BITTAB,Y
C0F3: 8D0FC3            STA CONTROL


                * SET STOPBITS

C0F6: 20BFC2            JSR CLRSCRN
C0F9: 20C4C2            JSR PRINT
C0FC: 302E20            ASC "0. 1 STOPBIT"
C0FF: 312053
```

```
C102: 544F50
C105: 424954
C108: 0D                DFB CR
C109: 312E20            ASC "1. 2 STOPBITS"
C10C: 322053
C10F: 544F50
C112: 424954
C115: 53
C116: 8D                DFB CR+128
C117: 20E6C2 GETSTOP    JSR NINPUT
C11A: C902              CMP #2
C11C: B0F9              BCS GETSTOP
C11E: A8                TAY
C11F: AD0FC3            LDA CONTROL
C122: 1904C3            ORA STOPTAB,Y
C125: 8D0FC3            STA CONTROL


                * SET HANDSHAKE MODE

C128: 20BFC2        JSR CLRSCRN
C12B: 20C4C2        JSR PRINT
C12E: 302E20        ASC "0. 0-3 LINE HANDSHAKE
C131: 302D33
C134: 204C49
C137: 4E4520
C13A: 48414E
C13D: 445348
C140: 414B45
C143: 0D            DFB CR
C144: 312E20        ASC "1.   X LINE HANDSHAKE
C147: 202058
C14A: 204C49
C14D: 4E4520
C150: 48414E
C153: 445348
C156: 414B45
C159: 8D                DFB CR+128
C15A: 20E6C2 GETHAND    JSR NINPUT
C15D: C902              CMP #2
C15F: B0F9              BCS GETHAND
C161: A8                TAY
C162: AD10C3            LDA COMMAND
```

```
C165: 1906C3            ORA HANDTAB,Y
C168: 8D10C3            STA COMMAND


              * SET DUPLEX

C16B: 20BFC2            JSR CLRSCRN
C16E: 20C4C2            JSR PRINT
C171: 302E20            ASC "0. HALF DUPLEX"
C174: 48414C
C177: 462044
C17A: 55504C
C17D: 4558
C17F: 0D                DFB CR
C180: 312E20            ASC "1. FULL DUPLEX"
C183: 46554C
C186: 4C2044
C189: 55504C
C18C: 4558
C18E: 8D                DFB CR+128
C18F: 20E6C2 GETDUPL    JSR NINPUT
C192: C902              CMP #2
C194: B0F9              BCS GETDUPL
C196: A8                TAY
C197: AD10C3            LDA COMMAND
C19A: 1908C3            ORA DUPLTAB,Y
C19D: 8D10C3            STA COMMAND


              * SET PARITY MODE

C1A0: 20BFC2        JSR CLRSCRN
C1A3: 20C4C2        JSR PRINT
C1A6: 302E20        ASC "0. PARITY DISABLED"
C1A9: 504152
C1AC: 495459
C1AF: 204449
C1B2: 534142
C1B5: 4C4544
C1B8: 0D            DFB CR
C1B9: 312E20        ASC "1. ODD PARITY"
C1BC: 4F4444
C1BF: 205041
```

```
C1C2: 524954
C1C5: 59
C1C6: 0D          DFB CR
C1C7: 322E20      ASC "2. EVEN PARITY"
C1CA: 455645
C1CD: 4E2050
C1D0: 415249
C1D3: 5459
C1D5: 0D          DFB CR
C1D6: 332E20      ASC "3. MARK TRANSMITTED"
C1D9: 4D4152
C1DC: 4B2054
C1DF: 52414E
C1E2: 534D49
C1E5: 545445
C1E8: 44
C1E9: 0D          DFB CR
C1EA: 342E20      ASC "4. SPACE TRANSMITTED"
C1ED: 535041
C1F0: 434520
C1F3: 545241
C1F6: 4E534D
C1F9: 495454
C1FC: 4544
C1FE: 8D                  DFB CR+128
C1FF: 20E6C2 GETPARIT     JSR NINPUT
C202: C905                CMP #5
C204: B0F9                BCS GETPARIT
C206: A8                  TAY
C207: AD10C3              LDA COMMAND
C20A: 190AC3              ORA PARTAB,Y
C20D: 8D10C3              STA COMMAND

                  * LOWER CASE
                  * DON'T ALLOW <SHFT> <C=>
C210: A917                LDA #23
C212: 8D18D0              STA CHARGEN
C215: A9FF                LDA #255
C217: 8D9102              STA SHFTCOM

                  * SIGNON MESSAGE

C21A: 20BFC2              JSR CLRSCRN
```

```
C21D: 20C4C2          JSR PRINT
C220: 544552          ASC "TERMINAL"
C223: 4D494E
C226: 414C
C228: 0D8D            DFB CR,CR+128

            * CLOSE FILE# 3

C22A: A903            LDA #3
C22C: 20C3FF          JSR CLOSE

            * SET DEVICE , SEC. ADDRES
            * AND FILE#

C22F: A202            LDX #2
C231: A000            LDY #0
C233: A903            LDA #3
C235: 20BAFF          JSR SETFPA

            * SET FILENAME (SETTINGS)
            * LENGTH 2 BYTES

C238: A20F            LDX #CONTROL:L
C23A: A0C3            LDY #CONTROL:H
C23C: A902            LDA #2
C23E: 20BDFF          JSR SETFNA

            * OPEN FILE

C241: 20C0FF          JSR OPEN

            * SET IN AND OUTDEVICE

C244: A203            LDX #3
C246: 20C6FF          JSR CHKIN
C249: A203            LDX #3
C24B: 20C9FF          JSR CKOUT

            * PLACE OWN CURSOR

C24E: A9AF            LDA #CURS
C250: 2016E7          JSR SCRNOUT
C253: A99D            LDA #BACKS
```

```
C255: 2016E7            JSR SCRNOUT

                * THE TERMINAL LOOP
                * IF ERROR STATUS STOP
C258: A690    TERMLOOP  LDX ST
C25A: D048              BNE TERMERR

C25C: 2042F1            JSR FROMKBD
C25F: F012              BEQ NOKEY

                * KEY PRESSED ?
                * YES!
C261: C914              CMP #DEL
C263: D002              BNE NODEL
                * CONVERT DELETE IN STANDARD
                * ASCII CODE BS ($08)

C265: A908              LDA #BS
C267: C90D    NODEL     CMP #CR
C269: D005              BNE NOCR
C26B: 20D2FF            JSR BSOUT

                * IF CR ADDITIONAL LINEFEED

C26E: A90A              LDA #LF
C270: 20D2FF  NOCR      JSR BSOUT

                * GET FROM RS232

C273: 20E4FF  NOKEY     JSR GET
C276: 20A8C2            JSR TRANSL
C279: F0DD              BEQ TERMLOOP

                * PRINTABLE ?
                * YES!

C27B: C90D              CMP #CR
C27D: D007              BNE TOSCRN

                * IF CR CLEAR CURSOR

C27F: A920              LDA '
C281: 2016E7            JSR SCRNOUT
```

```
C284: A90D              LDA #CR

                * CONVERT CHARACTER TO
                * COMMODORE ASCII

C286: 48        TOSCRN  PHA
C287: C941              CMP 'A
C289: 9008              BCC TOSCRN2
C28B: C95B              CMP 'Z+1
C28D: B004              BCS TOSCRN2
C28F: 68                PLA
C290: 4920              EOR #%00100000
C292: 48                PHA
C293: 68        TOSCRN2 PLA
C294: 2016E7            JSR SCRNOUT
                * PRINT CURSOR AT NEXT
                * POSITION

C297: A9AF              LDA #CURS
C299: 2016E7            JSR SCRNOUT
C29C: A99D              LDA #BACKS
C29E: 2016E7            JSR SCRNOUT

C2A1: 4C58C2            JMP TERMLOOP

                * IF ERROR CLOSE RESTORE
                * ALL FILES

C2A4: 20E7FF TERMERR    JSR CLALL
C2A7: 60                RTS

                * TRANSLATES ASCII IN
                * COMMODORE CODE
                * SKIPS NON PRINTABLES

C2A8: A014      TRANSL  LDY #DEL
C2AA: 297F              AND #$7F
C2AC: C908              CMP #BS
C2AE: F00D              BEQ TRANSRTS
C2B0: A00D              LDY #CR
C2B2: C90D              CMP #CR
C2B4: F007              BEQ TRANSRTS
C2B6: A000              LDY #0
```

```
C2B8: C920              CMP '
C2BA: 9001              BCC TRANSRTS
C2BC: A8                TAY
C2BD: 98        TRANSRTS TYA
C2BE: 60                RTS

                * CLEAR SCREEN ROUTINE

C2BF: A993      CLRSCRN LDA #CLS
C2C1: 4CD2FF            JMP BSOUT


                * THE WELL KNOWN PRINT
                * STRING ROUTINE OF
                * CHAPTER 2

C2C4: 68        PRINT   PLA
C2C5: 858E              STA AUX
C2C7: 68                PLA
C2C8: 858F              STA AUX+1
C2CA: A200              LDX #0
C2CC: E68E      PRINT1  INC AUX
C2CE: D002              BNE *+4
C2D0: E68F              INC AUX+1
C2D2: A18E              LDA (AUX,X)
C2D4: 297F              AND #$7F
C2D6: 20D2FF            JSR BSOUT
C2D9: A200              LDX #0
C2DB: A18E              LDA (AUX,X)
C2DD: 10ED              BPL PRINT1
C2DF: A58F              LDA AUX+1
C2E1: 48                PHA
C2E2: A58E              LDA AUX
C2E4: 48                PHA
C2E5: 60                RTS


                * GET DIGIT FROM KEYBOARD

C2E6: 20E4FF NINPUT     JSR GET
C2E9: F0FB              BEQ NINPUT
C2EB: C930              CMP '0
C2ED: 90F7              BCC NINPUT
```

```
C2EF: C93A            CMP '9+1
C2F1: B0F3            BCS NINPUT
C2F3: 290F            AND #$0F
C2F5: 60              RTS

                * DIVERSE SETTING TABLES

                * BAUDRATE

C2F6: 01      BAUDTAB DFB %00000001
C2F7: 02              DFB %00000010
C2F8: 03              DFB %00000011
C2F9: 04              DFB %00000100
C2FA: 05              DFB %00000101
C2FB: 06              DFB %00000110
C2FC: 07              DFB %00000111
C2FD: 08              DFB %00001000
C2FE: 09              DFB %00001001
C2FF: 0A              DFB %00001010

                * DATABITS

C300: 00      BITTAB  DFB %00000000
C301: 20              DFB %00100000
C302: 40              DFB %01000000
C303: 60              DFB %01100000

                * STOPBITS

C304: 00      STOPTAB DFB %00000000
C305: 80              DFB %10000000

                * HANDSHAKE

C306: 00      HANDTAB DFB %00000000
C307: 01              DFB %00000001

                * DUPLEX

C308: 00      DUPLTAB DFB %00000000
C309: 10              DFB %00010000

                * PARITY
```

```
C30A: 00      PARTAB   DFB %00000000
C30B: 20               DFB %00100000
C30C: 60               DFB %01100000
C30D: A0               DFB %10100000
C30E: E0               DFB %11100000

               * THE 'FILENAME'

C30F: 00      CONTROL  DFB 0
C310: 00      COMMAND  DFB 0

PHYSICAL ENDADDRESS: $C311

*** NO WARNINGS

ST                     $90
AUX                    $8E
SHFTCOM                $0291
CHARGEN                $D018
SCRNOUT                $E716
FROMKBD                $F142
SETFPA                 $FFBA
SETFNA                 $FFBD
OPEN                   $FFC0
CLOSE                  $FFC3
CHKIN                  $FFC6
CKOUT                  $FFC9
BSOUT                  $FFD2
GET                    $FFE4
CLALL                  $FFE7
BS                     $08
LF                     $0A
CR                     $0D
DEL                    $14
CLS                    $93
CURS                   $AF
BACKS                  $9D
GETDAT                 $C0E5
GETSTOP                $C117
GETHAND                $C15A
GETDUPL                $C18F
GETPARIT               $C1FF
TERMLOOP               $C258
```

NODEL	$C267
NOCR	$C270
NOKEY	$C273
TOSCRN	$C286
TOSCRN2	$C293
TERMERR	$C2A4
TRANSL	$C2A8
TRANSRTS	$C2BD
CLRSCRN	$C2BF
PRINT	$C2C4
PRINT1	$C2CC
NINPUT	$C2E6
BAUDTAB	$C2F6
BITTAB	$C300
STOPTAB	$C304
HANDTAB	$C306
DUPLTAB	$C308
PARTAB	$C30A
CONTROL	$C30F
COMMAND	$C310

15 HOW TO CONNECT YOUR C-64 WITH AN ATARI?

15 How to connect your C64 with an ATARI ?

We have used the built-in RS232 interface succesfully for connecting an ATARI 800 48k with the COMMODORE 64. It should work with every computer, which has a RS232 interface.
In our case the ATARI was the sender an the C64 the receiver. Of course you can do it the other way.
The following programs are only samples. You have to add your own PUT and GET routines. Start first the receiver (C64) and then the sender (ATARI).
The C64 waits for the first nonzero data. So the first databyte has to be unequal to zero. After receiving the first character the C64 waits till there is new data received by the RS232 interface. Then the program jumps to the user defined PUT routine.
Only two lines are required in this case. You don't have to invert the signal because the ATARI program does this by software. Connect the computers as following:

On the ATARI:
GAME port 3 :

pin 1 : Transmitted data
pin 2 : Ground

on the COMMODORE 64:

USER port:

pin B & C : Received data
pin N : Ground

Connect pin 1 to pin B & C and pin 2 to pin N.
Often the first databyte will be damaged. It is better to send first a dummy byte, or to change the single byte after receiving.
The programs will send and receive with 300 baud. For this reason long data will take long time. Possibly you could speed up the transmission by changing the baudrate.

```
*****************************************
*                                       *
*        RECEIVE FROM ATARI 800         *
*                                       *
*****************************************

            RECPOINT EQU $029B

            SETFPA   EQU $FFBA
            SETFNA   EQU $FFBD
            OPEN     EQU $FFC0
            CLOSE    EQU $FFC3
            CHKIN    EQU $FFC6
            CLRCH    EQU $FFCC
            BASIN    EQU $FFCF
            BSOUT    EQU $FFD2
            GET      EQU $FFE4

            * USER DEFINED PUT

            PUT      EQU $B000

                     ORG $C000
```

```
C000: A902              LDA #2
C002: A202              LDX #2
C004: A000              LDY #0
C006: 20BAFF            JSR SETFPA

C009: A902              LDA #2
C00B: A247              LDX #NAME:L
C00D: A0C0              LDY #NAME:H
C00F: 20BDFF            JSR SETFNA

C012: 20C0FF            JSR OPEN

C015: A202              LDX #2
C017: 20C6FF            JSR CHKIN

C01A: 20E4FF WAIT       JSR GET
C01D: F0FB              BEQ WAIT

C01F: AD9B02            LDA RECPOINT
C022: 8D49C0            STA OLDPOINT
C025: 4C2DC0            JMP WAIT2


C028: 2000B0 LOOP       JSR PUT

                * CARRY MUST INDICATE
                * 'FULL BUFFER'

C02B: B011              BCS READY

                * WAIT FOR CHARACTER

C02D: AD9B02 WAIT2      LDA RECPOINT
C030: CD49C0            CMP OLDPOINT
C033: F0F8              BEQ WAIT2
C035: 8D49C0            STA OLDPOINT
C038: 20E4FF            JSR GET
C03B: 4C28C0            JMP LOOP

C03E: A902   READY      LDA #2
C040: 20C3FF            JSR CLOSE
C043: 20CCFF            JSR CLRCH
C046: 00                BRK
```

```
C047: 8610    NAME      DFB $86,$10
C049: 00      OLDPOINT  DFB 0

PHYSICAL ENDADDRESS: $C04A

*** NO WARNINGS

RECPOINT          $029B
SETFPA            $FFBA
SETFNA            $FFBD
OPEN              $FFC0
CLOSE             $FFC3
CHKIN             $FFC6
CLRCH             $FFCC
BASIN             $FFCF     UNUSED
BSOUT             $FFD2     UNUSED
GET               $FFE4
PUT               $B000
WAIT              $C01A
LOOP              $C028
WAIT2             $C02D
READY             $C03E
NAME              $C047
OLDPOINT          $C049

*****************************************
*                                       *
*         SEND PROGRAM FOR              *
*                                       *
*         ATARI                         *
*                                       *
*****************************************

              COUNT     EPZ $1F

              PT        EPZ $F0.1
              MAX       EPZ $F2.3

              GET       EQU $3303

              RSENTRY   EQU $032C

              PACTL     EQU $D303
```

```
                    PORTA     EQU $D301
                    NMIEN     EQU $D40E
                    DMACTL    EQU $D400


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

                    * K & L FOR 300 BAUD

                    K         EQU 150
                    L         EQU 6

                              ORG $AB00,$B000

                    *         THE OPEN ROUTINE

AB00: A930          INIT      LDA #$30
AB02: 8D03D3                  STA PACTL
AB05: A901                    LDA #%00000001
AB07: 8D01D3                  STA PORTA
AB0A: A934                    LDA #$34
AB0C: 8D03D3                  STA PACTL
AB0F: A901                    LDA #%00000001
AB11: 8D01D3                  STA PORTA
AB14: 2067AB                  JSR BITWAIT
AB17: 2067AB                  JSR BITWAIT


AB1A: 200333 LOOP             JSR GET
AB1D: 48                      PHA
AB1E: 2027AB                  JSR SEROUT
AB21: 68                      PLA

                    * STOP IF CTRL-Z
AB22: C91A                    CMP #26
AB24: D0F4                    BNE LOOP
AB26: 00                      BRK

                    * SERIALOUT FIRST
                    * REVERSE BYTE
```

```
AB27: 8D72AB SEROUT   STA BUFFER

              * DISABLE INTERRUPTS

AB2A: 78              SEI
AB2B: A900            LDA #0
AB2D: 8D0ED4          STA NMIEN
AB30: 8D00D4          STA DMACTL

              * SEND STARTBIT

AB33: A900            LDA #%00000000
AB35: 8D01D3          STA PORTA
AB38: 2067AB          JSR BITWAIT

              * SEND BYTE

AB3B: A008            LDY #8
AB3D: 841F            STY COUNT

AB3F: AD72AB SENDBYTE LDA BUFFER
AB42: 8D01D3          STA PORTA
AB45: 6A              ROR
AB46: 8D72AB          STA BUFFER
AB49: 2067AB          JSR BITWAIT
AB4C: C61F            DEC COUNT
AB4E: D0EF            BNE SENDBYTE

              * SEND TWO STOPBITS

AB50: A901            LDA #%00000001
AB52: 8D01D3          STA PORTA
AB55: 2067AB          JSR BITWAIT
AB58: 2067AB          JSR BITWAIT

              * ENABLE INTERRUPTS

AB5B: A922            LDA #$22
AB5D: 8D00D4          STA DMACTL
AB60: A9FF            LDA #$FF
AB62: 8D0ED4          STA NMIEN
AB65: 58              CLI
AB66: 60              RTS
```

```
                    * THE BITTIME ROUTINE FOR
                    * AN EXACT BAUDRATE

AB67: A296    BITWAIT  LDX #K
AB69: A006    LOOPK    LDY #L
AB6B: 88      LOOPL    DEY
AB6C: D0FD             BNE LOOPL
AB6E: CA               DEX
AB6F: D0F8             BNE LOOPK
AB71: 60               RTS

                    * ROUTINE FOR INSTALLING
                    * THE RS232 HANDLER

                    * ONE BYTE BUFFER

                    BUFFER   EQU *

PHYSICAL ENDADDRESS: $B072
*** NO WARNINGS

COUNT               $1F
PT                  $F0       UNUSED
MAX                 $F2       UNUSED
GET                 $3303
RSENTRY             $032C     UNUSED
PACTL               $D303
PORTA               $D301
NMIEN               $D40E
DMACTL              $D400
EOL                 $9B       UNUSED
CR                  $0D       UNUSED
LF                  $0A       UNUSED
K                   $96
L                   $06
INIT                $AB00     UNUSED
LOOP                $AB1A
SEROUT              $AB27
SENDBYTE            $AB3F
BITWAIT             $AB67
LOOPK               $AB69
LOOPL               $AB6B
BUFFER              $AB72
```

16.1 HOW TO USE THE A/D CONVERTERS?

16.1 How to use the A/D converters ?

The C64 has two A/D converters. You can use them for connecting a pair of paddles on your C64. Paddles are just variable resistors. So we can say the A/D converters of the C64 are a kind of measuring instruments. By using unijunction transistors it is possible to measure voltages.
The A/D converter works as follows. A A capacitor will be loaded with up to 5 volts. Register 25 (or 26) of the SID ($D400) will contain a value corresponding to the needed time.
This procedure goes on and on. To use the whole scale of 8 bits (0-255) the resistor must be in the range from 0 to 500 Kohm.
To scan the values of the both A/D converters, ($D419 and $D41A) a delay loop will be needed because during loading of the capacitor the valure will be instable.
Because of the two gameports, it is possible to connect two pairs of paddles. This means 4 paddles in total. A solution is needed to evaluate all the paddles, because there are only two A/D converters.
To select the paddles of gameport 1 a $80 has to be poked $DC00. To select the paddles of gameport 2 a $40. During paddle request the keyboard has to be switched

off. You can do that by disabling the interrupt (SEI) and poking $C0 in $DC02 before reading the paddle values. After reading the keyboard has to be enabled again by clearing the interruptflag (CLI) and poking $FF in $DC02.
Of course these are not professional A/D converters, but you can use them very well for small measurements and for connecting linear joysticks and paddles.

17 THE RESTORE KEY

17 the restore key

On the right side of your C64 keyboard you will find a key labled 'RESTORE'. What is that key for ?
Maybe you have used it already in BASIC together with the RUN/STOP key, to force a BASIC warmstart. What is so special about pressing the RESTORE key ?
By pressing RESTORE a NMI interrupt will be forced, and the program counter of the 6510 will be loaded with the address stored at the loactions $FFFA and $FFBA. These locations are in ROM and the address stored there is $FE43. Now let us look to the so called NMI handler located at $FE43.

```
FE43  78       SEI
FE44  6C1803   JMP ($0318)   JMP $FE47
FE47  48       PHA           save registers
FE48  8A       TXA
FE49  48       PHA
FE4A  98       TYA
FE4B  48       PHA
FE4C  A97F     LDA #$7F
FE4E  8D0DDD   STA $DD0D
FE51  AC0DDD   LDY $DD0D
FE54  301C     BMI $FE72     using RS232 ?
FE56  2002FD   JSR $FD02     ROM in $8000 ?
FE59  D003     BNE $FE5E     no !
FE5B  6C0280   JMP ($8002)   yes! jump to ROM

FE5E  20BCF6   JSR $F6BC     Set flag f. STOP
FE61  20E1FF   JSR $FFE1     STOP key ?
FE64  D00C     BNE $FE72     no! then RS232
```

```
FE66   2015FD   JSR $FD15      init IO vectors
FE69   20A3FD   JSR $FDA3      init IO
FE6C   2018E5   JSR $E518      init IO clr scrn
FE6F   6C02A0   JMP ($A002)    BASIC warmstart

FE72 - FEBB   RS232 handling

FEBC   68       PLA            rest. registers
FEBD   A8       TAY
FEBE   68       PLA
FEBF   AA       TAX
FEC0   68       PLA
FEC1   40       RTI            back to program
```

As you can see, it is possible to change the vector ($0318-$0319) and let it point to the own NMI handler. By doing so, you have to save the registers and test if RS232 is active in your own handler. If it is active, you have to jump $FE72. Otherwise you can restart a program or do something else.
The following program is a sample routine.

```
*****************************************
*                                       *
*         RESET VIA RESTORE             *
*                                       *
*         USING NMI                     *
*                                       *
*****************************************
              NMIVECT  EQU $0318

              OTHNMI   EQU $FE72
              SETSTOPF EQU $F6BC
              STOPQ    EQU $FFE1
              SETVECT  EQU $FD15
              INITIO1  EQU $FDA3
              INITIO2  EQU $E518

              USER     EQU $0810

                       ORG $C000
```

```
                * SET NMIVECT TO OWN
                * HANDLER

C000: A90B    INITNMIH LDA #NMIHNDL:L
C002: 8D1803           STA NMIVECT
C005: A9C0             LDA #NMIHNDL:H
C007: 8D1903           STA NMIVECT+1
C00A: 00               BRK



                * THE NMI HANDLER

C00B: 48      NMIHNDL  PHA
C00C: 8A               TXA
C00D: 48               PHA
C00E: 98               TYA
C00F: 48               PHA

                * RS232 ACTIVE ?

C010: A97F             LDA #$7F
C012: 8D0DDD           STA $DD0D
C015: AC0DDD           LDY $DD0D
C018: 301E             BMI RS232

                * NO! STOPKEY ?

C01A: 20BCF6           JSR SETSTOPF
C01D: 20E1FF           JSR STOPQ
C020: D016             BNE RS232

                * YES! INITIALIZE

C022: 2015FD           JSR SETVECT
C025: 20A3FD           JSR INITIO1
C028: 2018E5           JSR INITIO2

                * BUT RESTORE OWN VECTOR

C02B: A90B             LDA #NMIHNDL:L
C02D: 8D1803           STA NMIVECT
C030: A9C0             LDA #NMIHNDL:H
```

```
C032: 8D1903            STA NMIVECT+1

                * JUMP TO WARMSTART OWN
                * PROGRAM

C035: 4C1008            JMP USER

                * OTHERWISE HANDLE RS232

C038: 4C72FE RS232      JMP OTHNMI

PHYSICAL ENDADDRESS: $C03B

*** NO WARNINGS

NMIVECT             $0318
OTHNMI              $FE72
SETSTOPF            $F6BC
STOPQ               $FFE1
SETVECT             $FD15
INITIO1             $FDA3
INITIO2             $E518
USER                $0810
INITNMIH            $C000     UNUSED
NMIHNDL             $C00B
RS232               $C038
```

18 FAST AND COMPACT HEXDUMPER

18 fast and compact hexdumper

The next program is a fast and compact hexdumper. This means you only have to give start and end addresses of a memory block, you want to have dumped on a printer. In this case a RS232 printer (QUME Sprint).
You only have to poke start and end in the labled locations.

```
****************************************
*                                      *
*         FAST AND COMPACT             *
*                                      *
*         HEXDUMPER                    *
*                                      *
****************************************

              FROM      EPZ $80.1
              TO        EPZ $82.3

              RS232OUT  EQU $029D
              RS232ACT  EQU $029E

              SETFPA    EQU $FFBA
              SETFNA    EQU $FFBD
              OPEN      EQU $FFC0
              CLOSE     EQU $FFC3
              CKOUT     EQU $FFC9
              STOPQ     EQU $FFE1
              CLALL     EQU $FFE7
              BSOUT     EQU $FFD2
```

```
                CR          EQU 13
                LF          EQU 10

                * 16 BYTES PER LINE

                MAX         EQU 16

                            ORG $C000

                * CLOSE FILE# 2

C000: A902                  LDA #2
C002: 20C3FF                JSR CLOSE

                * SET FILE#, DEVICE AND
                * SEC. ADDRESS

C005: A902                  LDA #2
C007: A202                  LDX #2
C009: A000                  LDY #0
C00B: 20BAFF                JSR SETFPA
                * SET FILENAME (SETTINGS)
                * AND LENGTH (2 BYTES)

C00E: A902                  LDA #2
C010: A285                  LDX #FNAME:L
C012: A0C0                  LDY #FNAME:H
C014: 20BDFF                JSR SETFNA

                * OPEN THE FILE

C017: 20C0FF                JSR OPEN

                * SET OUTDEVICE

C01A: A202                  LDX #2
C01C: 20C9FF                JSR CKOUT


C01F: A580      LINELOOP    LDA FROM
C021: C582                  CMP TO
C023: A581                  LDA FROM+1
C025: E583                  SBC TO+1
```

```
C027: B038              BCS READY
C029: A90D              LDA #CR
C02B: 20D2FF            JSR BSOUT
C02E: A90A              LDA #LF
C030: 20D2FF            JSR BSOUT
C033: A581              LDA FROM+1
C035: 206DC0            JSR PRBYTE
C038: A580              LDA FROM
C03A: 206DC0            JSR PRBYTE
C03D: A920              LDA '
C03F: 20D2FF            JSR BSOUT
C042: 20E1FF            JSR STOPQ
C045: F01A              BEQ READY
C047: A000              LDY #0
C049: B180    PRLOOP    LDA (FROM),Y
C04B: 206DC0            JSR PRBYTE
C04E: C8                INY
C04F: C010              CPY #MAX
C051: 90F6              BCC PRLOOP

              * LINE PRINTED (MAX BYTES)

C053: 98                TYA
C054: 18                CLC
C055: 6580              ADC FROM
C057: 8580              STA FROM
C059: 90C4              BCC LINELOOP
C05B: E681              INC FROM+1
C05D: F002              BEQ READY
C05F: B0BE              BCS LINELOOP A.T.

              * ALL DUMPED !

              * WAIT TILL ALL PRINTED

C061: AD9D02 READY      LDA RS232OUT
C064: CD9E02            CMP RS232ACT
C067: D0F8              BNE READY

              * CLOSE ALL

C069: 20E7FF            JSR CLALL
C06C: 00                BRK
```

```
                    * PRINT ACCU AS HEXBYTE

C06D: 48        PRBYTE    PHA
C06E: 4A                  LSR
C06F: 4A                  LSR
C070: 4A                  LSR
C071: 4A                  LSR
C072: 2078C0              JSR HEXOUT
C075: 68                  PLA
C076: 290F                AND #%00001111
C078: C90A      HEXOUT    CMP #10
C07A: B004                BCS ALFA
C07C: 0930                ORA '0
C07E: D002                BNE HXOUT
C080: 6936      ALFA      ADC 'A-10-1
C082: 4CD2FF    HXOUT     JMP BSOUT


                    * THE RS232 CONTROL AND
                    * COMMAND BYTE

C085: 86        FNAME     DFB %10000110
C086: 10                  DFB %00010000

PHYSICAL ENDADDRESS: $C087

*** NO WARNINGS

FROM                $80
TO                  $82
RS232OUT            $029D
RS232ACT            $029E
SETFPA              $FFBA
SETFNA              $FFBD
OPEN                $FFC0
CLOSE               $FFC3
CKOUT               $FFC9
STOPQ               $FFE1
CLALL               $FFE7
BSOUT               $FFD2
CR                  $0D
LF                  $0A
MAX                 $10
```

```
LINELOOP    $C01F
PRLOOP      $C049
READY       $C061
PRBYTE      $C06D
HEXOUT      $C078
ALFA        $C080
HXOUT       $C082
FNAME       $C085
```

19 TWO TINY ONES

19 two tiny ones

19.1 ROM to RAM

As you know the C64 has 64k RAM available. In some places there is ROM over it. If you read a memory location you get the value stored in ROM instead in RAM. If you write something in a location covered by ROM, it will be written in the RAM under the ROM. If you try to read it back, you will get the value ROM.
If you switch off the ROM and try to read again, you will get the RAM data.
By reading the ROM and storing it in the RAM, you make a one to one copy of the ROM in RAM. You can now switch off the ROM and it is possible to change the kernal.
The following is a ROM to RAM copy program.

```
*****************************************
*                                       *
*         ROM TO RAM                    *
*                                       *
*         COPY PROGRAM                  *
*                                       *
*****************************************

              PT        EPZ $02.3
              MAX       EPZ $04.5

                        ORG $C000
```

```
C000: A502              LDA PT
C002: C504              CMP MAX
C004: A503              LDA PT+1
C006: E505              SBC MAX+1
C008: B00F              BCS READY
C00A: A200              LDX #0
C00C: A102    COPY      LDA (PT,X)
C00E: 8102              STA (PT,X)
C010: E602              INC PT
C012: D002              BNE *+4
C014: E603              INC PT+1
C016: 4C0CC0            JMP COPY
C019: 00      READY     BRK

PHYSICAL ENDADDRESS: $C01A

*** NO WARNINGS

PT                      $02
MAX                     $04
COPY                    $C00C
READY                   $C019
```

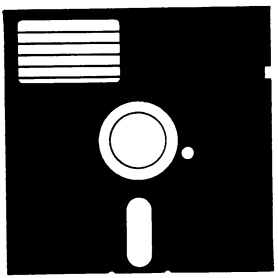
19.2 Upper and lower case on the 2022.

As you probably know, you have to send special characters (cursor up and down) to switch the printer in upper or lower case mode. Possibly you don't know, there is a special mode on the 2022 not mentioned in the manual, to print mixed upper and lower case characters.
You can do this by first opening the printer with secondary addres 7, closing it immediately, and reopen it with secondary address 0 again. All characters will be printed as they arrived at the 2022 printer, except [ ] _ ^ \. That may be the reason why not mentioned in the manual.

References and Acknowledgements

1. Commodore-64 Users Guide
   from Commodore Business Machines, Inc.
   Computer Systems Division
   487 Devon Park Drive
   Wayne, PA 19087
2. Commodore-64 Programmer' s Reference Guide from Commodore
3. Commodore 64 intern: A Data Becker Book from Data Becker, Duesseldorf, West-Germany

The programs from this book on a disk.



Use this page as an orderform

☐ All programs on disk, US $ 19.95

☐ MACROFIRE – The Editor/Assembler used in this book to edit the source code and to translate it into machine language. US $89.–.

MACROFIRE

This is the most powerful machine language tool for the C-64 we have seen so far.
The system consists of two parts:

1. A very powerful editor, almost as our wordprocessor BLIZTEXT
2. A very fast Assembler (10K of sourcecode will be translated in approx. 5 seconds).

Both programs are in memory at the same time, and from the editor you can start the assembly by just pressing two keys. After assembly the editor takes over control again. There are 24 commands available in the editor. The Editor/Assembler is available on cassette or disk (please specify). It stands to reason that the MACROFIRE has full macro capability. An INCLUDE function allows you to assemble files greater than the amount of memory.

Order-Nr. 4963 $89.00

Name: (please print).....

Address: .....

City/State/Country/Code.....

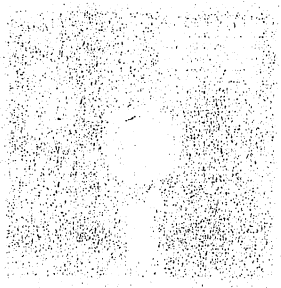
Mail it to one of these addresses:

ELCOMP PUBLISHING, INC.
53 Redrock Lane
Pomona, CA 91766
USA
Phone: (714) 623-8314
Telex: 29 81 91

ELCOMP Computer (S) Pte Ltd.
89 Short Street
Unit 03-07, Golden Wall Auto C.
Singapore 0718
Phone: 3382623,3388228
Telex: 56516

Ing.W.Hofacker GmbH
Tegernseer Str. 18
D-8150 Holzkirchen
West-Germany
Phone: 08024/7331
Telex: 52 69 73

Payment in US funds only: Check, money order, VISA, MASTER CARD, Eurocheck, ACCESS, Interbank.
Prepaid orders aid $ 3.50 for shipping (USA), $ 5.00 handling for C.O.D.
All orders outside USA: add 15% shipping, California residents add 6.5% sales tax.



NOTES

NOTES

NOTES

NOTES

NOTES

BITFIRE

BITFIRE

BITFIRE

BITFIRE

BITFIRE

BITFIRE

ISBN 3-88963-183-5

BITFIRE