Introduction to the Z84 (Z80 based microcontroller)

Jean-Michel FRIEDT, June 2001-June 27, 2001

1 Introduction

The kit Aki-80 sold the Akizuke shop in Akihabara, Tokyo, Japan (`www.akizuke.ne.jp`) is based on a Z84C15 built by Toshiba (TMP-Z84C015) which is basically a Z80 core to which peripherals [1, p.590] (SIO, PIO, CTC, CGC...) have been added. Its running frequency of 12 MHz makes it much faster than the original Z80, while still keeping code compatibility with this CPU.

Although lacking A/D or D/A converters useful for robotics applications, the Z84 can be used for digital control and processing projects.

2 Compiler

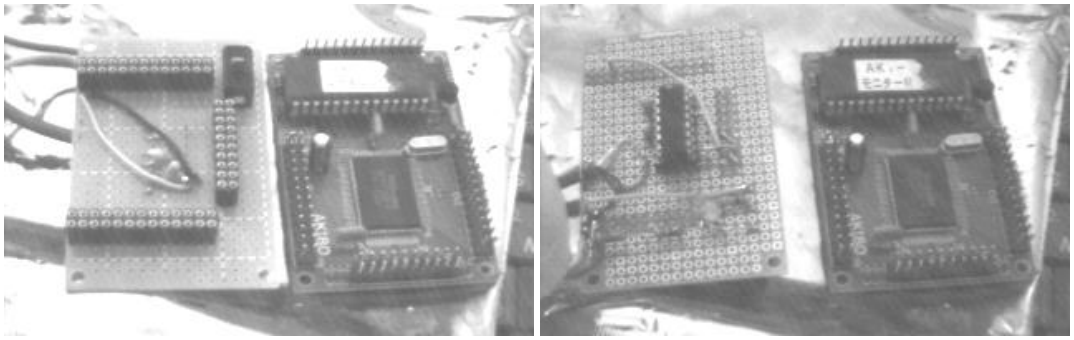
The assembler program used in this document is available by anonymous ftp from `ftp.cmf.nrl.navy.mil` in the directory `/pub/kenh/` (the file is `asxxxx-v1.51.tar.gz` in November 1999, although the version might change in the future). This assembly program has the advantage of being available for most CISC CPUs (including the 6809 and the Z80).

The compilation method used, under Linux, is the following:

- write a program in assembly language using your favorite editor (in this example the resulting file will be called `prg.asm`). Only type the right side of the examples given in this document (for example for a line like `000A 31 00 90 ld sp,#0h9000 ; stack` one should only type `ld sp,#0h9000`, eventually followed by comments (the text after '`;`'). The beginning of the line gives the adress to which the code will be located (in this example, this address would be `000A`), followd by the opcodes of the commands after assembly (in this example `31 00 90`). These opcodes are the informations sent through an RS232 connexion to the Z84.
- assemble the program by executing the command `asz80 prg.asm` under Linux in order to generate the file `prg.o` which includes the opcodes in hexadecimal notation and some additional informations which are of no interest to us.
- remove the lines starting with the letter '`T`' (`grep ^T`) and the 8 first charactes of the resulting lines (`cut -c9-60`) and store the result in the file `prg.out`. The resulting command line for all these processing is: `grep T %1.rel | cut -c9-60 > %1.out` under DOS or `grep T $1.rel | cut -c9-60 > $1.out` under Linux.
- finally, program the Z84 based Aki80 board by executing `./z80 prg.out`. This last step will require writing the bootstrap program presented later in this document (section 5) in an EEPROM and replacing the original EPROM sold with the board (containing the BASIC interepreter) with it. Indeed, the Z84 does not include a bootstrap program and it will always start executing the opcode located at address 0x0000 after reset. The aim of the bootstrap program presented in section 5 is to offer a way of testing programs on the Z84 similar to that found on the 68HC11F1 or the H8/3048: after reset, the board listens to the serial port and waits for the program length and the data to be stored in RAM, and upon receiving the whole program will jump to RAM to execute the newly loaded series of opcodes and data. This program testing method is much more flexible than programming an EEPROM for each new modification made to the software under development.

3 Additional hardware

The Aki80 board comes with the CPU including its SIO communication port, and a EPROM containing a BASIC interpreter which has to be connected to an RS232-compatible port to be run. Before being able to connect the board to an IBM compatible PC in this case, a converter from TTL levels to RS232 levels (using a MAX232 chip) has to be used and interfaced to the Aki80 board.



4 The BASIC interpreter

The BASIC interpreter includes the following commands (as observed by disassembling the program stored in the EPROM sold with the boards): `ABS AND BLANK CALL CONT END FOR GOSUB IF.N.GOTO IMP INPUT LET LIST LOAD MOD MODEB MODEN NEXT NEW OR OUT PEEK POKE PRINT REM RETURN RND ROMNAME RUN SAVE SIZE STOP SYSTEM WRITE XOR`.

Switching the power on while the pin 26 of connector 1 (port A) is grounded will start the test routine in which a LED connected to port A will blink with a period of 1 second. If pin 26 of connector 1 is connected to +5 V, the boot program will jump to the BASIC interpreter and listen to the serial port for a carriage return (CR) character to be received (communication at 9600 baud, N81). The following sequence is an example as logged by the communication software `procom` under DOS (after hitting the Return key once to start the BASIC interpreter):

```
@Start Z80 remote basic Ver.1.0 made by System Load
                                ..since 1992.
ok
>10 LET A=10
>20 LET B=20
>30 LET C=A+B
>40 PRINT C
>LIST
    10 LET A=10
    20 LET B=20
    30 LET C=A+B
    40 PRINT C
ok
>RUN
      30
ok
>PRINT C
      30
ok
>SYSTEM
@Start Z80 remote basic Ver.1.0 made by System Load
                                ..since 1992.
ok
>LIST
ok
>list
ok
>10 a=10
>20 b=30
>30 c=a+b
>40 print c
>run
      40
ok
>system
@
```

Disassembling the EPROM is in fact an efficient way for learning how to program the Z84 and how to initialize the serial port and the PIO (which is based on the Z80 PIO originally built as a separate circuit).

5 The bootstrap program

The aim of the bootstrap mode is to get data and opcodes from a binary file (hexadecimal representation in a `.out` file) and store in RAM so that the Z80 can execute the program to be tested. As the RAM address space starts at 0x8000, all programs to be tested this way should start with the `.area code (ABS)` and `.org 0h8000` statements.

Once compiled using `asz80`, the bootstrap program was stored in a 28C64 EEPROM using the 68HC11F1 based circuit.

The protocol for sending a program to the on-board memory is as follows:

- send the number of bytes containing the program length (in bytes), MSB first
- send the actual data as read in the `.out` file generated as described in section 2.

Once the whole program is sent, the bootstrap program will automatically jump to the beginning RAM area (locate at address 0x8000) and keep its current RS232 port configuration (9600 bauds, N81).

```
1 ; based on an example program by qzf06013@nifty.ne.jp (cant read japanese ...)
   001C  PIOA  = 0h1C
   001D  PIOAC = 0h1D
   0018  SIOA  = 0h18
   0019  SIOAC = 0h19
   0013  CTC3  = 0h13
   0004  TIMEC = 0h04     ; for 10MHz
         ;TIMEC  equ     05h     ; for 12MHz
         ;TIMEC  equ     03h     ; for  7MHz

         .area code (ABS)
         .org 0h0000

   0000 31 FF 9F    ld sp,#0h9FFF ; stack pointer  <- AKI80 (end of RAM)
   0003 F3          di            ; disable interrupts
   0004 C3 10 00    jp start
                                  ; define byte : serial port control
   0007 18 serial:.db 0h18        ; WR0 reset SIO (reset wr0)
   0008 04 44     .db 0h04,0h44   ; WR4=0100 0100->Nx1, x16 ck
   000A 03 C1     .db 0h03,0hC1   ; WR3=1100 0001->Rx ena, 8 b/ch : N81
   000C 05 6A     .db 0h05,0h6A   ; WR5=0110 1010->Tx ena, 8b/c, (6A instead 68 : RTS=1)
   000E 01 00     .db 0h01,0h00   ; WR1=0000 0000->dis. int

   0010 3E 17 start:ld a,#0h17    ; set baudrate
   0012 D3 13       out  (CTC3),a
   0014 3E 04       ld   a,#TIMEC
   0016 D3 13       out  (CTC3),a
   0018 3E 17       ld   a,#0h17  ; ? COPIED from EPROM ?! <- what is it for ???
   001A D3 14       out  (0h14),a ; ?
   001C 3E 04       ld   a,#0h04  ; ?
   001E D3 14       out  (0h14),a ; ? secondary timer ?

   0020 21 07 00    ld   hl,#serial ; init serial port
   0023 06 09       ld   b,#0h09
   0025 0E 19       ld   c,#SIOAC
   0027 ED B3       otir            ; send 9 bytes from 'serial' to port SIOAC=0h19

   0029 3E CF       ld   a,#0hcf    ; 1100 1111 -> config port A
   002B D3 1D       out  (#0h01d),a ; |
   002D 3E 00       ld   a,#0h00    ; 0000 0000
   002F D3 1D       out  (#0h01d),a ; |
   0031 3E 07       ld   a,#0h07    ; 0000 0111
   0033 D3 1D       out  (#0h01d),a ; |

   0035 21 00 80    ld   hl,#0h8000 ; RAM start @ <- data storage area

   0038 CD 6E 00    call rcv
   003B 57          ld   d,a        ; MSB first
   003C 47          ld   b,a        ; MSB first
   003D CD 77 00    call snd        ; char to be sent in reg B

   0040 CD 6E 00    call rcv
   0043 5F          ld   e,a        ; LSB second -> de=number of bytes to rcv
   0044 47          ld   b,a        ; MSB first
   0045 CD 77 00    call snd        ; char to be sent in reg B

   0048 CD 6E 00 loop:  call    rcv ; result in reg A
   004B 77          ld  (hl),a      ; store char in RAM

   004C 3E FF       ld  a,#0hFF
   004E D3 1C       out (#0h1c),a   ; store port A : blink LED
   0050 CD 65 00    call Wait1ms
   0053 3E 00       ld  a,#0h00
   0055 D3 1C       out (#0h1c),a   ; store port A

   0057 46          ld  b,(hl)      ; check if char was stored properly
   0058 CD 77 00    call snd        ; char to be sent in reg B
   005B 23          inc hl
   005C 1B          dec de
   005D 7B          ld  a,e         ; astuce de Wait1ms pour voir si dec de reg
   005E B2          or  d           ; 16 bits est nul
   005F C2 48 00    jp  NZ,loop
   0062 C3 00 80    jp  0h8000      ; all received -> jump to RAM & exec ...

   0065 01 77 01 Wait1ms:ld      bc,#375
   0068 0B       Wait1msLoop:dec     bc
   0069 79          ld  a,c     ; test for B=C=00 (A=C;OR (A,B) !=0)
   006A B0          or  b
   006B 20 FB       jr  NZ,Wait1msLoop
   006D C9          ret

   006E DB 19   rcv:in  a,(SIOAC)
   0070 CB 47       bit 0,a
   0072 28 FA       jr  Z,rcv       ; nothing to read yet ... (other bit0=1)
   0074 DB 18       in  a,(SIOA)    ; got it !
   0076 C9          ret

   0077 DB 19  snd: in  a,(SIOAC)   ; wait for SIO to be ready
   0079 CB 57       bit 2,a         ; |                        (1=>TxD empty)
   007B 28 FA       jr  Z,snd       ; |
   007D 78          ld  a,b         ; send char to RS232
   007E D3 18       out (SIOA),a    ; |
   0080 C9          ret
```

`bootstrap.asm`: program for loading the program from the serial port, storing it to RAM and jumping to the beginning of the RAM area once the transfer is completed.

6 Basic examples

PIO initialization and programming is described in [1, p.365]. The first example simply makes a LED connected to port A blink with a period of 1 second.

```
; based on an example program by qzf06013@nifty.ne.jp (cant read japanese ...)
; PIOA          equ     01ch
; PIOAC         equ     01dh

               .area code (ABS)
               .org 0h8000

   8000       start:
   8000 31 00 90    ld   sp,#0h9000      ; stack pointer  <- AKI80
                    ; ld sp,#0h00        ; stack en 0000 et decroit => FFFF ?

   8003 3E CF       ld   a,#0hcf         ; 1100 1111 -> config port A
   8005 D3 1D       out  (#0h01d),a      ; |
   8007 3E 00       ld   a,#0h00         ; 0000 0000
   8009 D3 1D       out  (#0h01d),a      ; |
   800B 3E 07       ld   a,#0h07         ; 0000 0111
   800D D3 1D       out  (#0h01d),a      ; |

   800F 3E AA  loop:ld   a,#0haa         ; 1010 1010
   8011 D3 1C       out  (#0h1c),a       ; store port A
   8013 CD 20 80    call Wait
   8016 3E 55       ld   a,#0h55         ; 0101 0101
   8018 D3 1C       out  (#0h1c),a       ; store port A
   801A CD 20 80    call Wait
   801D C3 0F 80    jp   loop

   8020 11 E8 03  Wait:ld de,#1000
   8023 CD 2C 80  Wait_loop:call Wait1ms
   8026 1B          dec  de
   8027 7A          ld   a,d
   8028 B3          or   e
   8029 20 F8       jr   NZ,Wait_loop
   802B C9          ret

   802C 01 77 01 Wait1ms:ld bc,#375
   802F 0B       Wait1msLoop:dec bc
   8030 79          ld   a,c
   8031 B0          or   b
   8032 20 FB       jr   NZ,Wait1msLoop
   8034 C9          ret
```

`led1.asm`: test program for initializing and controlling port A

7 The LCD display

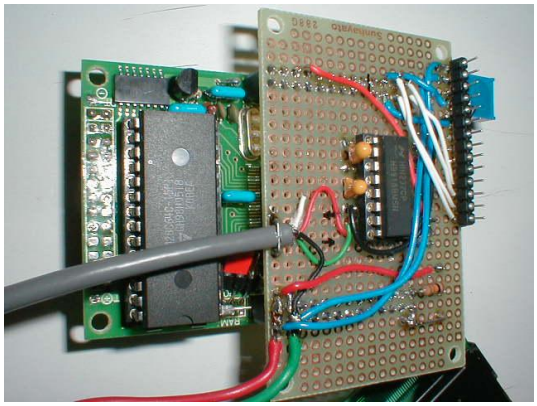
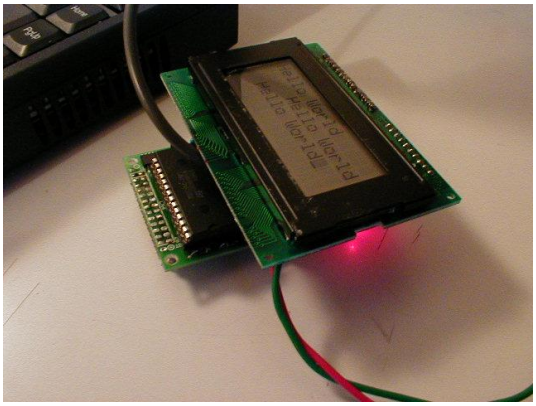
The LCD is run in 4 bits (6 wires) mode using port B. We thus connect PB0-PB3 of the Z84 to D4-D7 of the LCD, and PB4 and PB5 to the control bits E(nable) and RS (data/command) respectively [1].

```
 ; LCD display, 4 bit mode, on port B
 ;
 ; PIOA         equ  01ch
 ; PIOAC        equ  01dh
 ; PIOB         equ  01eh
 ; PIOBC        equ  01fh

    .area code (ABS)
    .org 0h8000

8000 31 00 90    ld sp,#0h9000      ; stack pointer  <- AKI80
              ; ld sp,#0h00        ; stack en 0000 et decroit => FFFF ?
   8003       start:
   8003 3E CF       ld   a,#0hcf         ; 1100 1111 -> config port A
   8005 D3 1D       out  (#0h1d),a       ; |
   8007 3E 00       ld   a,#0h00         ; 0000 0000
   8009 D3 1D       out  (#0h1d),a       ; |
   800B 3E 07       ld   a,#0h07         ; 0000 0111
   800D D3 1D       out  (#0h1d),a       ; |    A OPTIMISER EN METTANT LES 2 INITS en //
   800F 3E CF       ld   a,#0hcf         ; 1100 1111 -> config port A
   8011 D3 1F       out  (#0h1f),a       ; config port B (idem)
   8013 3E 00       ld   a,#0h00         ; 0000 0000
   8015 D3 1F       out  (#0h1f),a       ; config port B (idem)
   8017 3E 07       ld   a,#0h07         ; 0000 0111
```

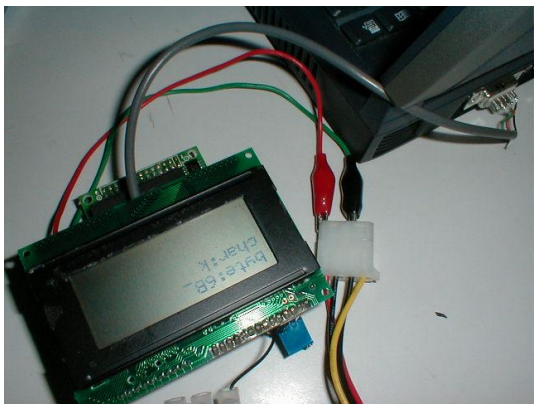
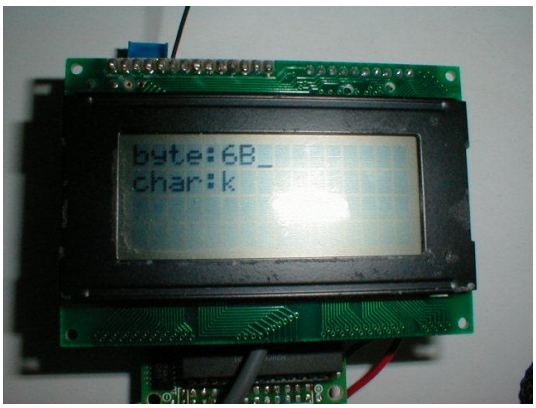
[1] A first design, connecting the data lines of the LCD to the data but of the Z84, RS to A0 (address 0) of the Z84 and E to a combination of the address lines and `IOW#` did not work as the timing requirements are not verified by the Z80 signals.

```
8019 D3 1F       out  (#0h1f),a       ; config port B (idem)

801B CD 34 80    call Wait
801E CD 49 80    call setuplcd
8021 CD 72 80    call printlcd

8024 3E AA  loop:ld  a,#0haa         ; 1010 1010
8026 D3 1C       out (#0h1c),a       ; store port A
8028 CD 34 80    call Wait
802B 3E 55       ld  a,#0h55         ; 0101 0101
802D D3 1C       out (#0h1c),a       ; store port A
802F CD 34 80    call Wait
8032 18 F0       jr  loop

8034 11 E8 03 Wait:ld  de,#1000
8037 CD 40 80 Wait_loop:call Wait1ms
803A 1B          dec de
803B 7A          ld  a,d
803C B3          or  e
803D 20 F8       jr  NZ,Wait_loop
803F C9          ret

8040 01 77 01 Wait1ms:ld bc,#375
8043 0B       Wait1msLoop:dec bc
8044 79          ld   a,c
8045 B0          or   b
8046 20 FB       jr   NZ,Wait1msLoop
8048 C9          ret

8049    setuplcd:
8049 3E 03       ld   a,#0h03
804B CD 9A 80    call cmd4lcd
804E 3E 03       ld   a,#0h03
8050 CD 9A 80    call cmd4lcd
8053 3E 03       ld   a,#0h03
8055 CD 9A 80    call cmd4lcd
8058 3E 02       ld   a,#0h02
805A CD 9A 80    call cmd4lcd ; end of 4 bits interface init.
805D 3E 28       ld   a,#0h28 ; .11: 5x8 font
805F CD AD 80    call cmd8lcd
8062 3E 08       ld   a,#0h08 ; .13: disp=curs.=blink off
8064 CD AD 80    call cmd8lcd
8067 3E 0F       ld   a,#0h0f ; .15: disp=curs=blink on
8069 CD AD 80    call cmd8lcd
806C 3E 06       ld   a,#0h06 ; 17: mv right, no shift
806E CD AD 80    call cmd8lcd
8071 C9          ret

        ; DEBUT DE L'AFFICHAGE CST :
8072     printlcd:           ; no input, modif. D, HL, A, B, C
8072 3E 80       ld   a,#0h80 ; pos. (starts @ 0) <- cursor pos=0 (8=gotoxy)
8074 CD AD 80    call cmd8lcd
8077 CD 8B 80    call aff_msg
807A 3E C5       ld   a,#0hC5 ; pos. (starts @ 0) <- cursor pos=0 (8=gotoxy)
807C CD AD 80    call cmd8lcd
807F CD 8B 80    call aff_msg
8082 3E 92       ld   a,#0h92 ; pos. (starts @ 0) <- cursor pos=0 (8=gotoxy)
8084 CD AD 80    call cmd8lcd
8087 CD 8B 80    call aff_msg
808A C9          ret

808B 16 0B aff_msg:ld d,#11   ; number of chars to be displayed
808D 21 D7 80    ld   hl,#message     ; (attention: C is modif. by Wait1ms)
8090 7E      msg:ld   a,(hl)
8091 CD C0 80    call chr8lcd
8094 23          inc  hl
8095 15          dec  d
8096 C2 90 80    jp   NZ,msg
8099 C9          ret
     ; ld a,#0h61 ; call chr8lcd  <- manually load a char

809A F5  cmd4lcd:push af             ; cmd to be output in reg. A, modif. A, B, C
809B D3 1E       out  (#0h1e),a
809D CB E7       set  #4,a           ; E->1
809F D3 1E       out  (#0h1e),a
80A1 CD 40 80    call Wait1ms
80A4 F1          pop  af
80A5 CB A7       res  #4,a           ; E->0
80A7 D3 1E       out  (#0h1e),a
80A9 CD 40 80    call Wait1ms
80AC C9          ret

80AD F5  cmd8lcd:push af      ; char a envoyer dans A, modifie A, B, C
80AE CB 3F       srl  a
80B0 CB 3F       srl  a
80B2 CB 3F       srl  a
80B4 CB 3F       srl  a
80B6 CD 9A 80    call cmd4lcd
80B9 F1          pop  af
80BA E6 0F       and  #0h0f
80BC CD 9A 80    call cmd4lcd
80BF C9          ret

80C0 F5  chr8lcd:push af      ; char a envoyer dans A, modifie A, B, C
80C1 CB 3F       srl  a       ; pareil que cmd8lcd, mais avec RS=1 : OPTIMISER ?
80C3 CB 3F       srl  a
80C5 CB 3F       srl  a
80C7 CB 3F       srl  a
80C9 CB EF       set  #5,a
80CB CD 9A 80    call cmd4lcd
80CE F1          pop  af
80CF E6 0F       and  #0h0f
80D1 CB EF       set  #5,a
80D3 CD 9A 80    call cmd4lcd
80D6 C9          ret

80D7 48 65 6C 6C 6F 20 message: .db    0h48,0h65,0h6c,0h6c,0h6f,0h20,0h57,0h6f,0h72,0h6c,0h64
     57 6F 72 6C 64
```

`lcd.asm`: test program for initializing and controlling an LCD display in 4-bits mode.

A more useful use of the LCD screen: a display of the value and ASCII equivalent of data received on the serial port. The subroutine for converting from binary to ASCII values is taken from [1, p.300]

Be aware that although one might be tempted to power the negative supply (contrast) of the LCD using the unused output of the MAX232 converter, doing so will result in erratic RS232 communication. The MAX232 indeed seems not to work reliably anymore

once its unused output is connected to so as to supply the LCD screen with a negative voltage. A separate negative power supply should thus be used when the LCD and the RS232 communication are to be used simultaneously.

References

[1] Ramesh Gaonkar, *The Z80 Microprocessor (2nd Ed)*, Merrill, Macmillan Publishing Company (1993)

1, 3, 4