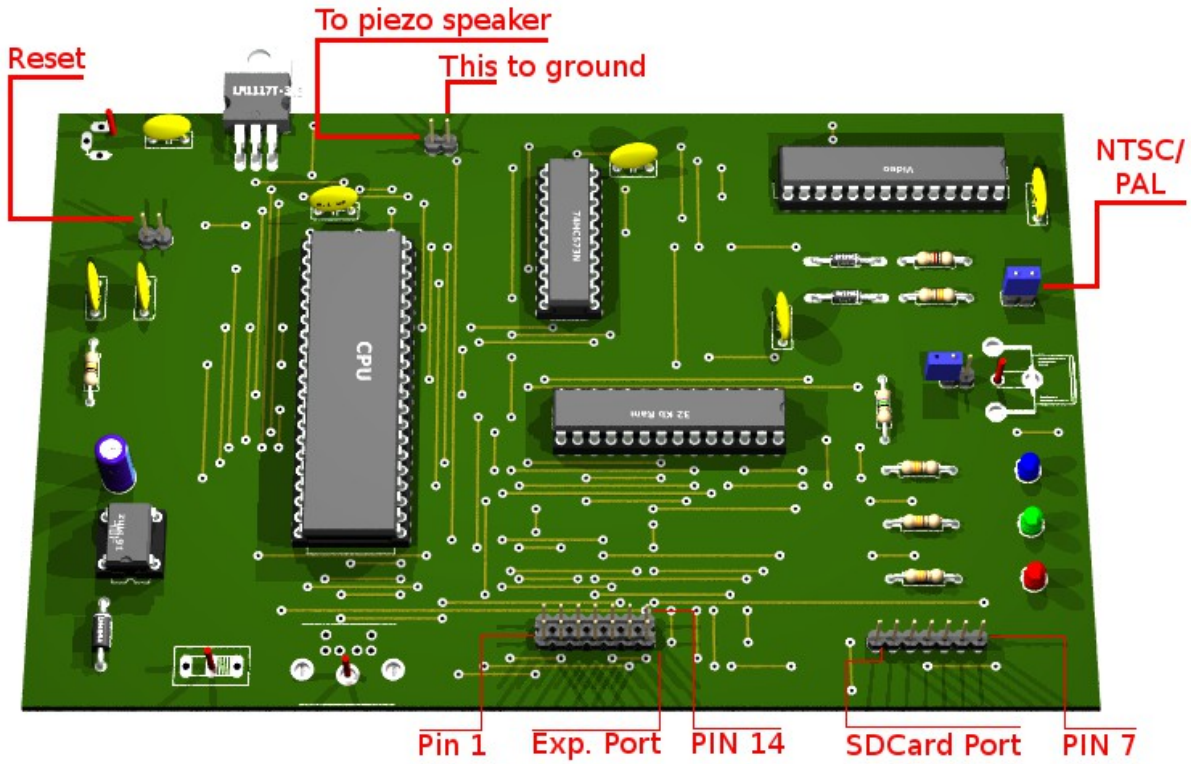
CX80 – Homebrew 8bit computer



The CX80 - A Brief Description

The CX80 is a computer based on Z80 CPU that you can build at house with a really low price. Of course I asked for some more in-depth knowledge of electronics such as knowing how to make a single sided pcb and know how program the atmel microcontroller. If you are not able to make it alone, do not worry, I can build it for you! I do not provide: a regulated 4.5 – 5.0 Volt DC mains adapter with a 1.3mm jack (centre positive) capable of supplying about 800 mA (or more), a sd card (FAT16/32 formatted) and an RCA cable to connect the CX80 at the television, a piezo speaker and a switch for the reset pins.

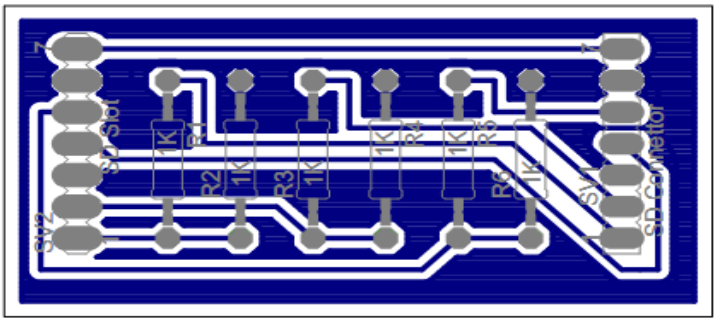
Jumpers

On CX80 board there are four jumpers that are used to configure the system or optional connections:

- JP1 – Reset (if you short this pins, CX80 reset)
- JP2 – PAL/NTSC Video (closed = NTSC, open PAL output)
- JP3 – Rvideo (if shorted, enabled the 75 ohm resistor on video signal. Short it if you see grey color on video)

- JP4 – Piezo sound (connect a piezo speaker for sound, pin2 = GND)

The SDBoard for CX80



The CX80 required the SDBoard to properly access the SD card. In project zip file you can find the schematic necessary to its construction. IMPORTANT! The signals of the data bus of the CX80 are at 5 volt, the SDBoard is necessary to reduce the voltage to 3.3 volts, for a correct operation of the SD card.

The pinout of SD card port on CX80 is:

- PIN1 = SD_SEL
- PIN2 = DI
- PIN3 = GND
- PIN4 = 3.3 Vcc (from voltage regulator on CX80 board)
- PIN5 = SCLK
- PIN6 = GND
- PIN7 = DO

I also added a pdf file that explains how to create an sd card breadboard socket. The file is "Cheap-DIY-SD-card-breadboard-socket.pdf" (from Instructables site) and is located in the folder "Schematics\SDBoard for CX80".
Be very careful to observe the orientation of the pin, otherwise it will not work at all.

The female header SV2 must be connected to CX80 SDCard port.
The female header SV1 must be connected to SD card breadboard socket.

When you mount the SDBoard on the CX80 you need to turn it upside down, so as to see the bottom part of the electrical circuit.

The Boot Procedure

When the power is switched on (use sliding switch S1) CX80 perform a memory test, sd card and sd card filesystem test. If all is ok you can see a prompt ">" on screen. Congratulations, the CX80 is waiting for your command!

The CX80 Video

The video output is provided by ATMega8A, reconfigurated as a simplified version of TellyMate video adapter. The firmware of ATMega8A is original from Batsocks. Main features are:

- PAL or NTSC Composite Video
- 38x25 characters
- Black and White
- Simple Graphics
- Double width / height text

The CX80 OS

In fact we are not dealing with a real operating system, but at a shell that allows you to use the system. The command set is very limited and commands are:

- CLS : Clear the screen
- DREG : Print on screen the value of Z80 registers
- DEL <file/folder> : Delete a file or a folder
- REN <file/folder> : Rename a file or a folder
- MKDIR <foldername>: Create a new folder
- COPY <old> <new> : Copy a old file into new file
- RUN <prgname> : Load and run a binary Z80 program. If the program not exists into actual folder, search for it into "\CMD\" folder.

Programming the CX80

To programm CX80 you can use zDevStudio v0.8 (for Z80 assembler coder), a free ide and compiler downloadable at http://sourceforge.net/projects/zdevstudio/
If you want to use SDCC (c compiler) download it at http://sdcc.sourceforge.net/.
Examples of assembler and C programmind are in DEV folder of project zip file.

CX80 memory organizzation

CX80 has available 32 Kb of ram, mostly all usable for making programs. The addresses reserved for system ranging from location 0x0000 to 0x0007. The first location to load a user's program must is 0x0008. As you perform a system call, the CX80 reads the locations of six memory cells to configure itself and to communicate with the user.

The reserved memory is configured as follows:

- Address 0x0000 – Must contain the number of system function (see below)
- Address 0x0001 – REG1L (Register 1 low value)
- Address 0x0002 – REG1H (Register 1 high value)
- Address 0x0003 – REG2L (Register 2 low value)
- Address 0x0004 – REG2H (Register 2 high value)
- Address 0x0005 – REG3L (Register 3 low value)
- Address 0x0006 – REG3H (Register 3 high value)

After setting the data correctly for the system feature required you must execute a call to address 0x0007. See examples on DEV folder of project zip file for further details.

CX80 System Functions

The system function number (SFN from now on) must be loaded into address 0x0000 before

performing “call $0007” instruction.

SFN $00 – Print a string on video
REG1 must point to a 0 terminated string into ram

Example in assembler:

```
ld      iy, $0000
ld      (iy), $00       ;Function number

ld      hl, StringToPrint
ld      (iy + 1), l     ;REG1L
ld      (iy + 2), h     ;REG1H

call    $0007           ;Perform the requested system function

StringToPrint           db “Test Line” $A, $D, 0
```

SFN $01 - Print a single character on screen
REG1L is ASCII value of character to print

Example in assembler:

```
ld      iy, $0000
ld      (iy), $01       ;Function number

ld      l, “C”          ; Print 'C' character in screen
ld      (iy + 1), l     ;REG1L
call    $0007           ;Perform the requested system function
```

SFN $02 - Move the cursor up
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $02       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $03 - Move the cursor down
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $03       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $04 - Move the cursor right

No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $04       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $05 - Move the cursor left
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $05       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $06 - Clear the screen
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $06       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $07 - Move cursor to home
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $07       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $08 - Set Reverse Index on screen
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $08       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $09 - Clears from the cursor (inclusive) to the end of the screen
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $09       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $0A - Clears from the cursor (inclusive) to the end of the line
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0A       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $0B - Moves the cursor to the specified row and column.
REG1L = row to move, REG1H = column to move

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0B       ;Function number

ld      (iy + 1), 10    ;REG1L
ld      (iy + 2), 14    ;REG1H

; Move the cursor at row 10 and column 14
call    $0007           ;Perform the requested system function
```

SFN $0C - Set single line dimension. Must be call before print string or character.
REG1L = type of line on screen. Load into REG1L character from '0' to '5'
'0': single width, single height
'1': double width, single height
'2': single width, double height (top half)
'3': single width, double height (bottom half)
'4': double width, double height (top half)
'5': double width, double height (bottom half)

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0C       ;Function number

ld      (iy + 1), '1'   ;REG1L

; Set the current row at double width, single height
call    $0007           ;Perform the requested system function
```

SFN $0D - Clears from the cursor (inclusive) to the start of the screen
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0D       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $0E - Show or hidden the cursor
REG1L = 1 show the cursor, 0 hidden the cursor

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0E       ;Function number

ld      (iy + 1), 0     ;REG1L

;Hidden the cursor
call    $0007           ;Perform the requested system function
```

SFN $0F - The current row/column position is stored into video chip ram
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $0F       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $10 - The current row/column position is restored from video chip ram
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $10       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $11 - Clears the current row and moves the cursor to column 0
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $11       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $12 - Clears from the cursor (inclusive) to column 0 on the current row

No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $12       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $13 - Enable or disable line overflow. When Line Overflow is enabled, any character output to column 37 (the last column) will cause the cursor to be moved to the begining of the next row. When Line Overflow is disabled, any character output to column 37 will not cause the cursor to be moved. The next character will be output to the same position.
REG1L = 1 enable line overflow, 0 disable

Example in assembler:

```
ld      iy, $0000
ld      (iy), $13       ;Function number

ld      (iy + 1), 1     ;REG1L

; Enable line overflow
call    $0007           ;Perform the requested system function
```

SFN $14 - Set cursor to block shape
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $14       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $15 - Set cursor to underline shape
No parameters required

Example in assembler:

```
ld      iy, $0000
ld      (iy), $15       ;Function number

call    $0007           ;Perform the requested system function
```

SFN $16 - Set the invert output (black on white background) or normal output (white on black background).
REG1L = 1 invert output, 0 normal output

Example in assembler:

```
ld      iy, $0000
ld      (iy), $16       ;Function number

ld      (iy + 1), 1     ;REG1L

;Enable inverted output
call    $0007           ;Perform the requested system function
```

SFN $17 - Open or create a file for input/output
REG1L = FA_READ, FA_WRITE, FA_CREATE_NEW, FA_CREATE_ALWAYS, FA_OPEN_EXISTING (see below)
REG2 = pointer to 0 terminated string with file name

Return value on REG1H:
- 0xEF = error on file
- 0xFF = no free file slot
- 0 to 2 = number on memory ID to file (max 3 file in use)

Example in assembler:

```
ld      iy, $0000
ld      (iy), $17       ;Function number

; FA_WRITE or FA_CREATE_ALWAYS = open file to write in it, if
; it exist overwrite it.
ld      (iy + 1), FA_WRITE or FA_CREATE_ALWAYS

ld      hl, FileName
ld      (iy + 3), l             ; REG2L = low address of FileName
ld      (iy + 4), h             ; REG2L = high address of FileName

;Create file
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)
```

SFN $18 - Read one byte from file
REG1L = File ID
REG1H = readed value

Return value on REG2L:
- 0xEF = wrong file slot
- 0xAA = error on reading
- 0xFF = EOF
- 0x00 = read ok

Example in assembler (suppose you've already opened a file for reading and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $18       ;Function number
```

```
ld      (iy + 1), a     ;REG1L now contain the file ID

;Perform the requested system function
call    $0007           ;Perform the requested system function

; store in register B the byte readed
ld      b, (iy + 2)     ; REG1H contain the byte readed

;store in register A the result of operation
ld      a, (iy + 3)
```

SFN $19 - Write one byte to file
REG1L = File ID
REG1H = value to write

Return value on REG2L: 0xEF = wrong file slot
0xAA = error on writing
0x00 = write ok

Example in assembler (suppose you've already opened a file for writing and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $19       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;REG1H contain the character 'F' to write on file
ld      (iy + 2), 'F'

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 3)
```

SFN $1A - Close an opened file
REG1L = File ID

Return value on REG2L: 0xEF = wrong file slot
0x00 = close ok

Example in assembler (suppose you've already opened a file for reading/writing and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $1A       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;Perform the requested system function
call    $0007           ;Perform the requested system function
```

```
;store in register A the result of operation
ld      a, (iy + 3)
```

SFN $1B - Execute a seek on a file
REG1L = File ID
REG2 = Most Significant Word
REG3 = Least Significant Word

Return value on REG1H: 0xEF = wrong file slot
0xAA = error on seek
0x00 = seek ok

Example in assembler (suppose you've already opened a file for reading/writing and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $1B       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;move to position 500 on file
;the most significant word is 0
ld      (iy + 3), 0
ld      (iy + 4), 0

;least significant word is 500
ld      hl, 500
ld      (iy + 5), l
ld      (iy + 6), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)
```

SFN $1C - Read from file into ram buffer
REG1L = File ID
REG2 = Pointer to file buffer in ram
REG3 = Number of bytes to read

Return value REG1H: 0x00 = read ok
0xFF = EOF before REG3 = 0
if REG1H = 0xFF, REG3 = bytes read before EOF
0xAA = Error while reading
0xEF = Wrong file slot

Example in assembler (suppose you've already opened a file for reading and its ID is stored in register A):

```
ld      iy, $0000
```

```
ld      (iy), $1C       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;read 50 bytes
ld      hl, 50
ld      (iy + 5), l
ld      (iy + 6), h

;REG2 point to buffer into ram
ld      hl, ReadBuffer
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

ReadBuffer     db 50
```

SFN $1D - Write into file from ram buffer
REG1L = File ID
REG2 = Pointer to file buffer in ram
REG3 = Number of bytes to write

Return value REG1H: 0x00 = read ok
0xAA = Error while reading
0xEF = Wrong file slot

Example in assembler (suppose you've already opened a file for writing and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $1D       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;write 50 bytes
ld      hl, 50
ld      (iy + 5), l
ld      (iy + 6), h

;REG2 point to buffer into ram
ld      hl, WriteBuffer
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
```

```
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

WriteBuffer     db 50
```

SFN $1E - Change the current directory
REG2 = Pointer to path buffer in ram (0 terminated string)

Return value on REG1H: 0xAA = error on chdir
0x00 = all ok

Example in assembler:

```
;Enter into "\CMD\" folder
ld      iy, $0000
ld      (iy), $1E       ;Function number

ld      hl, NewPathString
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

NewPathString           db "\CMD\", 0
```

SFN $1F - Create a new directory
REG2 = Pointer to name buffer in ram (0 terminated string)

Return value on REG1H: 0xAA = error on creation
0x00 = all ok

Example in assembler:

```
;Create folder "NEWDIR"
ld      iy, $0000
ld      (iy), $1F       ;Function number

ld      hl, NewDirString
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
```

```
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

NewDirString            db "NEWDIR", 0
```

SFN $20 - Delete a file/directory
REG2 = Pointer to name of file/directory buffer in ram (0 terminated string)

Return value on REG1H: 0xAA = error on delete
0x00 = all ok

Example in assembler:

```
;Delete file "TEXT.TXT"
ld      iy, $0000
ld      (iy), $20       ;Function number

ld      hl, DeleteString
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

DeleteString            db "TEXT.TXT", 0
```

SFN $21 - Rename or move a file/directory
REG2 = Pointer to to old file/dir name (0 terminated)
REG3 = Pointer to new file/dir name (0 terminated)

Return value on REG1H: 0xAA = error on rename
0x00 = all ok

Example in assembler:

```
;Rename file "TEXT.TXT" into "TEXT.NEW"
ld      iy, $0000
ld      (iy), $21       ;Function number

ld      hl, OldNameString
ld      (iy + 3), l
ld      (iy + 4), h
```

```
ld      hl, NewNameString
ld      (iy + 5), l
ld      (iy + 6), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

OldNameString       db "TEXT.TXT", 0
NewNameString       db "TEXT.NEW", 0
```

SFN $22 - Get current directory
REG1L = Length of path string into ram (max 128 chars)
REG2 = Pointer to path string

Example in assembler:

```
ld      iy, $0000
ld      (iy), $22       ;Function number

; length of CurrPathString is 80 bytes
ld      (iy + 1), 80

ld      hl, CurrPathString
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
call    $0007           ;Perform the requested system function

...

CurrPathString      db 80
```

SFN $23 - Return the pressed key
REG1L = ASCII code of key pressed
REG1H = Scan code of key pressed

Example in assembler:

```
ld      iy, $0000
ld      (iy), $23       ;Function number

call    $0007           ;Perform the requested system function

;Store ASCII code into B register
;Store Scan code int C register
ld      b, (iy + 1)
```

```
ld      c, (iy + 2)
```

SFN $24 - Wait for a keypress event
REG1L = ASCII code of key pressed
REG1H = Scan code of key pressed

Example in assembler:

```
ld      iy, $0000
ld      (iy), $24       ;Function number

call    $0007           ;Perform the requested system function

;Store ASCII code into B register
;Store Scan code int C register
ld      b, (iy + 1)
ld      c, (iy + 2)
```

SFN $25 - Milliseconds delay
REG2 = delay in milliseconds

Example in assembler:

```
ld      iy, $0000
ld      (iy), $25       ;Function number

; wait for 600 milliseconds
ld      hl, 600
ld      (iy + 3), l
ld      (iy + 4), h

call    $0007           ;Perform the requested system function
```

SFN $26 - Get input string typed by user. End typing with return
REG1L = Length of path string into ram (max 128 chars)
REG2 = Pointer to input buffer

Example in assembler:

```
ld      iy, $0000
ld      (iy), $26       ;Function number

; length of InputBuffer is 80 bytes
ld      (iy + 1), 80

ld      hl,  InputBuffer
ld      (iy + 3), l
ld      (iy + 4), h

;Perform the requested system function
call    $0007           ;Perform the requested system function
```

```
...

 InputBuffer          db 80
```

SFN $27 - Play sound via piezo speaker
REG1 = sound duration
REG2 = sound frequency

Note	Frequency
A	880
A#	932
B	988
C	1047
C#	1109
D	1175
D#	1244
E	1319
F	1397
F#	1480
G	1568
G#	1660

Example in assembler:

```
ld      iy, $0000
ld      (iy), $27       ;Function number

; REG1 = sound duration (600 milliseconds)
ld      hl, 600
ld      (iy + 1), l
ld      (iy + 2), h

; REG2 = sound frequency (1047, note C)
ld      hl, 1047
ld      (iy + 3), l
ld      (iy + 4), h

call    $0007           ;Perform the requested system function
```

SFN $28 - Write On Expansion Port
REG1L = value to write
REG1H = Port address (0..3)
REG2 = wait time before write (in milliseconds)

Example in assembler:

```
ld      iy, $0000
ld      (iy), $28       ;Function number

;write value $AB on expansion port
ld      (iy + 1), $AB
```

```
;use port 2
ld      (iy + 2), 2

; wait for 100 milliseconds
ld      hl, 100
ld      (iy + 3), l
ld      (iy + 4), h

call    $0007           ;Perform the requested system function
```

SFN $29 - Read from Expansion Port
REG1L = Readed value
REG1H = Port address (0..3)
REG2 = wait time before read (in milliseconds)

Example in assembler:

```
ld      iy, $0000
ld      (iy), $29       ;Function number

;read on port 2
ld      (iy + 2), 2

; wait for 100 milliseconds
ld      hl, 100
ld      (iy + 3), l
ld      (iy + 4), h

call    $0007           ;Perform the requested system function

;store the value readed into A register
ld      a, (iy + 1)
```

SFN $2A - Read string from file (until '\n' character or REG3 value reached)
REG1L = File ID
REG2 = Pointer to string buffer in ram
REG3 = Number of bytes to read

Return value REG1H:
- 0x00 = Read ok
- 0xAA = Error while reading
- 0xEF = Wrong file slot
- 0xFF = End of File reached

Example in assembler (suppose you've already opened a file for reading and its ID is stored in register A):

```
ld      iy, $0000
ld      (iy), $2A       ;Function number

ld      (iy + 1), a     ;REG1L now contain the file ID

;REG2 point to StringFromFile
```

```
ld      hl, StringFromFile
ld      (iy + 3), l
ld      (iy + 4), h

;read a maximun of 300 bytes from file
ld      hl, 300
ld      (iy + 5), l
ld      (iy + 6), h

call    $0007           ;Perform the requested system function

;store in register A the result of operation
ld      a, (iy + 2)

...

StringFromFile          db 300
```

SFN $2B - Get OS CommandLine (0 terminated string, max 38 characters)
REG2 = Pointer to buffer in ram

Example in assembler (suppose you are started a program with command "RUN MYPROG.BIN"):

```
ld      iy, $0000
ld      (iy), $2B       ;Function number

ld      hl, OSCommandLine
ld      (iy + 3), l
ld      (iy + 4), h

call    $0007           ;Perform the requested system function

;now OSCommandLine contain "RUN MYPROG.BIN")

...

OSCommandLine       db 38
```

SFN $FF - Stop user program and return to OS.
IMPORTANT!! User program must be terminated with this system call!!!
SDCC framework insert by default this system call into binary file, simply use instruction "return <val>" into main.
REG1L = Return value to OS

If REG1L <> 0 the CX80 OS print on video the value passed

Example in assembler:

```
ld      iy, $0000
ld      (iy), $FF       ;Function number
```

```
; Exit with no error
ld      (iy + 1), 0

call    $0007           ;Perform the requested system function
```

CX80 Expansion Port

The expansion port allows you to read/write eight bits of 4 different ports, in fact the pinout is:

- PIN1 = 5 Vcc
- PIN3 = D0
- PIN5 = D1
- PIN7 = D2
- PIN9 = D3
- PIN11 = D4
- PIN13 = D5
- PIN2 = D6
- PIN4 = D7
- PIN6 = ADR0
- PIN8 = ADR1
- PIN10 = IOREQ (negated)
- PIN12 = WR/RD
- PIN14 = GND

So you can experiment with some small external circuit (I think LED, buttons, flip-flop, serial buffer and much more).