Introduction to Z-80 Assembly Language Programming

VCF SE 3.0
Malcolm Macleod
3 May 2015
malcolm@avitech.com.au
Topics Covered Today

• Overview of the Z80 – History & Features
• Pinouts and Architecture
• Instruction Set
• Assembly Language Example 1 - Toggle output port
• Development Environment
  • For assembling under Windows
  • For assembling under CP/M 2.2
• Assembly Language Example 2 – Output string to console
  • Assembling Example 2 under Windows
  • Assembling Example 2 in CP/M emulator
• Links:
  • Reference Cards and Manuals
  • Useful Websites
Overview of the Zilog Z-80 CPU

- Released in 1976
- 16-bit address space, 8 bit data bus
- Each instruction stored as 1, 2, 3 or 4 bytes
- “binary upwards compatible” with 8080 machine code – e.g. CP/M 2.2
- Developed by ex-Intel employees: Federico Faggin, Ralph Unger mann and Masatoshi Shima.
- Less than 50% of all Z80 CPUs were produced by Zilog
- Second sourced (licensed) manufactures included: Mostek, Toshiba, Sharp, NEC and SGS-Thomson
- NMOS versions were 2.5 MHz to 8 MHz
- CMOS versions are 4 MHz to 20 MHz
Improvements over the Intel 8080

Relative to the 8080 the Z80 has:

• An enhanced instruction set
• Two new 16-bit index registers (IX & IY)
• 4 new “alternate” 16-bit registers (AF’, BC’, DE’ and HL’)
• Two new interrupt modes (Modes 1 & 2)
• Register I = Interrupt vector base, for Mode 2 interrupts
• Register R = Refresh register
• A non-maskable interrupt input
• Single supply rail (+5V), rather than +5, -5 & +12
• Built in DRAM refresh (only 16k RAMs and smaller)
Z-80 Pin Configuration (40-Pin DIP)
### Registers

<table>
<thead>
<tr>
<th>Main Register Set</th>
<th>Alternate Register Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accumulator</strong></td>
<td><strong>Accumulator</strong></td>
</tr>
<tr>
<td>A</td>
<td>A'</td>
</tr>
<tr>
<td>B</td>
<td>B'</td>
</tr>
<tr>
<td>D</td>
<td>D'</td>
</tr>
<tr>
<td>H</td>
<td>H'</td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td><strong>Flags</strong></td>
</tr>
<tr>
<td>F</td>
<td>F'</td>
</tr>
<tr>
<td>C</td>
<td>B'</td>
</tr>
<tr>
<td>E</td>
<td>E'</td>
</tr>
<tr>
<td>L</td>
<td>L'</td>
</tr>
</tbody>
</table>

#### General Purpose Registers
- **Interrupt Vector**
- **Memory Refresh**

#### Special Purpose Registers
- **Index Register**
- **IX**
- **Index Register**
- **IY**
- **Stack Pointer**
- **SP**
- **Program Counter**
- **PC**

*Source: Zilog, Z80 Family CPU User Manual, Document Number UM008002-0202, Figure 2 (Page 3). Unmodified.*
Instruction Overview

The Z-80 can execute 158 different (published) instruction types, including all 78 of the 8080A CPU.

The instructions fall into these categories:

• Load and Exchange
• Block Transfer and Search
• Arithmetic and Logical
• Rotate and Shift
• Bit Manipulation (Set, Reset, Test)
• Jump, Call, and Return
• Input and Output
• Basic CPU Control
Addressing Modes

Most instructions need to access data in external memory or internal CPU registers. The various “addressing modes” describe the way in which this can occur:

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Assembly Language Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>LD A,FFH -or- LD BC,1234H</td>
</tr>
<tr>
<td>Modified Page Zero Addressing</td>
<td>RST 30H</td>
</tr>
<tr>
<td>Relative Addressing</td>
<td>JR Z,EXIT</td>
</tr>
<tr>
<td>Extended Addressing</td>
<td>JP EXIT -or- LD A,(TIMER)</td>
</tr>
<tr>
<td>Indexed Addressing</td>
<td>LD A,(IX+9H)</td>
</tr>
<tr>
<td>Register Addressing</td>
<td>LD A,B</td>
</tr>
<tr>
<td>Implied Addressing</td>
<td>SUB 30H</td>
</tr>
<tr>
<td>Register Indirect Addressing</td>
<td>LD A,(HL)</td>
</tr>
</tbody>
</table>
Flags

The flag registers (F and F') supply information to the user about the status of the Z80 at any given time. The bit positions for each flag is listed below:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Z</td>
<td>X</td>
<td>N</td>
<td>X</td>
<td>P/V</td>
<td>N</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Carry Flag</td>
</tr>
<tr>
<td>N</td>
<td>Add/Subtract</td>
</tr>
<tr>
<td>P/V</td>
<td>Parity/Overflow Flag</td>
</tr>
<tr>
<td>H</td>
<td>Half Carry Flag</td>
</tr>
<tr>
<td>Z</td>
<td>Zero Flag</td>
</tr>
<tr>
<td>S</td>
<td>Sign Flag</td>
</tr>
<tr>
<td>X</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

Notes:

1. When starting out, focus on learning how to use the C and Z flags, then S.
2. Flags H and N cannot be tested – they are only used for BCD arithmetic.
Example of Zilog’s instruction tables

Table 4. Exchanges EX and EXX

<table>
<thead>
<tr>
<th>IMPLIED</th>
<th>AF</th>
<th>BC', DE', and HL'</th>
<th>HL</th>
<th>IX</th>
<th>IY</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLIED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPLIED</td>
<td>AF</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REG.</td>
<td>(SP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IND.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implied Addressing

- AF': 08
- BC', DE', and HL': D9
- HL': EB
- IX: E3
- IY: DD E3 FD E3
Assembly Language Example 1 – Toggle Output

CYCLS EQU 40000D ; 10 msec = 4,000,000 divided by 100
PORT EQU 0FFH ; We're going to toggle output port 0FFH
ORG 0000H ; Our program goes in low memory
START
  LD SP,8000H ; Initialise the stack
LOOP
  LD A,D ; Get current value of D
  XOR 1H ; Toggle the least significant bit
  LD D,A ; Save new value of D
  OUT (PORT),A ; Output new value of D
  CALL DELAY ; Delay for 10 msec
JR LOOP ; Loop back to toggle again

DELAY
  LD BC,CYCLS/26D ; Number of loops required
DELLOOP
  LD A,B ; Put upper 8 bits of BC into A
  OR C ; Logical or A with lower 8 bits of BC
  DEC BC ; Decrement loop counter
  JR NZ,DELLOOP ; Loop unless BC=0
RET ; Return to main program loop

END START
Assembly Language Example 1 – Toggle Output

19: - 9C40 CYCLS EQU 40000D
20: - 00FF PORT EQU 0FFH
21: 
22: - 0000 ORG 0000H
23: 
24: 0+10 0000 310080 START LD SP, 8000H
25: 10+4 0003 7A LOOP LD A, D
26: 14+7 0004 EE01 XOR 1H
27: 21+4 0006 57 LD D, A
28: 25+11 0007 D3FF OUT (PORT), A
29: 36+17 0009 CD0E00 CALL DELAY
30: 53+12 000C 18F5 JR LOOP
31: 
32: 65+10 000E 010206 DELAY LD BC, CYCLS/26D
33: 75+4 0011 78 DELLOOP LD A, B
34: 79+4 0012 B1 OR C
35: 83+6 0013 0B DEC BC
36: 89+7+5 0014 20FB JR NZ, DELLOOP
37: 96+10 0016 C9 RET
38: 
39: - 0000 END START
Suggested Windows Environment

Editors:

• Crimson Editor (v. 3.72 – 2008) :
  http://www.crimsoneditor.com/

• Notepad++ (v6.7.5)
  http://notepad-plus-plus.org

Assemblers:

• George Phillips’ ZMAC (version 19sep2013):
  http://members.shaw.ca/gp2000/zmac.html

• Matthew Reed’s Z80ASM command-line assembler:
  http://www.trs-80emulators.com/z80asm/
Configuring Crimson Editor and ZMAC

Crimson Editor:

• Under Tools -> Conf. User Tools, for Hotkey “Ctrl+1”:
  • Set Menu Text = zmac
  • Set Command = [directory containing zmac]
  • Set Argument = $(FileName)
  • Set Initial Dir = $(FileDir)
  • Use “.z80” as suffix for your source code file

ZMAC:

• To assemble, press Ctrl+1 from within Crimson Editor
• Assembled listing will appear as “.lst” file in the ./zout directory
• Any assembly errors will also show in the “Capture Output” panel
Suggested Emulated CP/M 2.2 Environment

Editors:
- Crimson Editor
- Notepad++ (v6.7.5)

CP/M Emulator:
- CP/M 2.2 or C/M 3.0 on Peter Schorn’s “AltairZ80” SIMH-based emulator
  
  http://schorn.ch/altair.html

Z80 Assembler:
- SLR Systems’ Z80ASM (run this under CP/M)
  
  http://www.s100computers.com/Software%20Folder/Assembler%20Collection/Assembler%20Collection.htm
Assembling with SLR’s Z80ASM under AltairZ80

1. Download altairz80 from Peter Schorn’s website. The website has versions available for PC, Mac and Linux.

2. Configure a “cpm2” file (on your host computer) for altairz80 that attaches “cpm2.dsk” and “i.dsk” as hdsk0.

3. Create/Edit your “PROG.Z80” source file on your host computer.

4. Run altairz80. You’ll get a SIMH “sim >” prompt.

5. Type “do cpm2” to start CP/M 2.2. [use Ctrl-E later to exit to SIMH]

6. Use “R.COM” (on Drive I) to import SLR’s “Z80ASM.COM” from your host computer and store it on Drive I.

7. On Drive I, Use “R PROG.Z80” to import your source file from the host file system and store it on Drive I.

8. On Drive I, type “Z80ASM PROG/F” to assemble your program.

9. On Drive I, type “W SOURCE.LST” to export a copy of your “PROG.LST” file back to the host file system.
Example 2: Output String to Console

Note: This will be an on-screen demonstration using Crimson Editor, altairz80 and other applications:

2. Assembling under emulated CP/M 2.2 environment using Peter Schorn’s altairz80 emulator and SLR’s Z80ASM assembler.
Tips & Tricks

- Execution starts at 0x0000
- Remember to initialize SP before doing any calls or push/pop
- Stack grows downwards (and doesn’t store at initial value of SP)
- JR can only jump +127/-128. Use JP for longer jumps
- Some instructions do NOT update flags - e.g., "LD A,(HL)"
- Have a strategy about preserving registers - e.g., "caller saves"
- Document your assembly code thoroughly
- There are two interrupt inputs available: /NMI and /INT
- The Z80 is Little Endian (16 bit values are stored LSB first)
- You can assemble to a ROM address, but need to use EPROM programmer to write the program to the chip.
- You can’t store variables in ROM!
Key Reference Documents

Z-80 Instant Reference Card:

Z-80 Family CPU User Manual

Rodney Zaks – How to Program the Z80
Useful Websites

Documentation:
http://www.ballyalley.com/ml/z80_docs/z80_docs.html

Home of the Z80 CPU – Official Support Page:
http://www.z80.info/

Wikipedia Page on the Z80:

John Monahan’s guide to Peter Schorn’s altairz80:
http://www.s100computers.com/Software%20Folder/Altair%20Simulator/Altair%20Software.htm