University of British Columbia
Electrical and Computer Engineering
Digital Systems and Microcomputers CPEN312

Lecture 13b: Introduction to 8051 Assembly II

Dr. Jesús Calviño-Fraga
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

March 9, 2017

Objectives

- Get familiar with 8051's assembler programs.
- Write compilable assembly programs.
- Compile assembly programs using a computer.
- Load and run programs written in assembly language using a computer.

Assembling by hand…

- Assembling by hand is sometimes ‘fun’ but it is also:
  - Prone to error.
  - Slow.
  - Tedious.
  - Hard to add simple changes.
- We use a computer to ‘assembly’ our code from the assembly code mnemonics. We need:
  - A computer
  - The assembler compiler!

The Assembler

- In this course we will be using A51.
  - I wrote this assembler!
  - Syntax is compatible with original Intel ASM51. The syntax is mostly the same as the assembler described in the text book.
- We also need a text editor. Any text editor will do, for example Notepad works fine. On the other had we can use an editor with syntax highlighting as in Crosside:

http://www.ece.ubc.ca/~jesusc/crosside_setup.exe

Assembler Overview

- Symbols
- Labels
- Assembler Controls
- Assembler Directives
- ASCII Literals
- Comments
- Macros

Symbols

- The legal characters: both upper and lower case (A..Z,a..z) letters, decimal numbers (0..9) and these two special characters: question mark (?) and underscore (_).
- Symbols can not start with a number.
- The assembler converts all symbols to uppercase. For instance these two are the same symbol:
  - My_symbol
  - MY_SYMBOL
- Mnemonics, register names, assembler controls and assembler directives are reserved words and can not be used as symbols.

Labels

- Labels are symbols also: same rules apply.
- Must appear BEFORE mnemonics, the storage directives (DB and DW), or data reservation directives (DS and DBIT).
- A label is defined by a symbol name and ':'
- Example:

```
L1: DJNZ R0, L1
```


Assembler Controls

- Assembler controls are used to tell the program where to gets its input source files (include files), where it puts the object file, and how it formats the listing file.
- Example:

```
$MOD52
$TITLE(CPEN312 – Example for Lecture 14)
$LIST
$PAGEWIDTH(75)
```

Assembler Controls

$DATE(date) Places date in page header
$INCLUDE(file) Inserts file in source program
$TITLE(string) Places string in page header
$NOLIST Stops outputting the listing
$MOD52 Uses 8052 predefined symbols
$MOD51 Uses 8051 predefined symbols
$MODDE0CV Uses CV-8052 predefined symbols
$NOMOD No predefined symbols used
$NOOBJECT No object file is generated
$NOPAGING Print listing w/o page breaks
$PAGEWIDTH(n) No. of columns on a listing page
$NOPRINT Listing will not be output
$NOSYMBOLS Symbol table will not be output
$EJECT Places a form feed in listing
$LIST Allows listing to be output
$OBJECT(file) Places object output in file
$PAGING Break output listing into pages
$PAGELENGTH(n) No. of lines on a listing page
$PRINT(file) Places listing output in file
$SYMBOLS Append symbol table to listing

Actually you can use any register definition file provided with the assembler, or even write your own!

Assembler Directives

- Assembler directives are used to define symbols, reserve memory space, store values in program memory and switch between different memory spaces.
- Examples of directives:

```
TEN EQU 10
RESET CODE 0
ORG 4096
```

Assembler Directives

- Symbol Definition Directives: EQU, SET, BIT, CODE, DATA, IDATA, XDATA
- Segment Selection Directives: CSEG, BSEG, DSEG, ISEG, XSEG
- Memory Reservation and Storage Directives: DS, DBIT, DB, DW
- Miscellaneous Directives: ORG, USING, END

EQU directive

- Very useful to keep your code looking clean and professional:

```
CLK  EQU 33333333
FREQ EQU 1000
BAUD equ 115200
TIMER_0_RELOAD EQU (0x10000-(CLK/(12*FREQ)))
TIMER_2_RELOAD equ (0x10000-(CLK/(32*BAUD)))
```

Segment Selection Directives

- There are five Segment Selection directives: CSEG, BSEG, DSEG, ISEG, XSEG, one for each of the five memory spaces in the 8051 architecture.
- Mostly used to allocate variables and constants in memory. Normally combined with DS, DBIT, DB, DW.
- Examples:

  DSEG at 30H ; Variables after the register banks and bits
  BSEG at 20H ;
  CSEG at 8000H

Memory Reservation and Storage Directives

- DS Directive: Used to reserve memory for variables. Works when ISEG, DSEG or XSEG are the active segments.
- Examples:

```
          DSEG at 30H ; Select data segment
          DS      32 ; Label is optional!
BUFF1:    DS      16
AVERAGE:  DS      8
COUNT:    DS      1
```

Memory Reservation and Storage Directives

- DBIT Directive: is used to reserve bits within the BIT segment.
- Examples:

```
          BSEG      ; Select bit segment
LEDON:    DBIT      1
LEDOFF:   DBIT      8
ERROR:    DBIT      1
```


Memory Reservation and Storage Directives

- DB Directive: is used to store byte constants in the Program Memory Space. It can only be used when CSEG is the active segment.
- Examples:

```
CSEG at 8000H
COPYRGHT_MSG:
   DB          '(c) Copyright, 2017 Jesus Calvino-Fraga‘, 0
RUNTIME_CONSTANTS:
   DB        127, 13, 54, 0, 99     ; Table of constants
   DB        17, 32, 239, 163, 49   ; Second line of const.
MIXED:
   DB        2*8, 'MPG' , 2*16, 'abc' ; Can mix literals & no.
```

Memory Reservation and Storage Directives

- DW Directive: is used to store word constants in the Program Memory Space. Similar to DB, but it stores two bytes (16-bits) at once.
- Examples:

```
CSEG at 8000H
  DW        'AB', 1000H
```


Miscellaneous Directives

- ORG Directive is used to specify a value for the currently active segment's location counter.
- **Examples:**

```
CSEG
ORG 0
  LJMP MyCode
ORG 1BH ; Timer 1 ISR vector location
  LJMP 1803H
MyCode:
  MOV SP, #7FH
```

Miscellaneous Directives

"The USING Directive is used to specify which of the four General Purpose Register banks is used in the code that follows the directive. It allows the use of the predefined register symbols AR0 thru AR7 instead of the register's direct addresses. It should be noted that the actual register bank switching must still be done in the code. This directive simplifies the direct addressing of a specified register bank."

Example:

```
USING 1
PUSH AR2 ; Pushes R2 of bank 1 into the stack
```


Miscellaneous Directives

- END Directive is used to signal the end of the source program to the Cross assembler.
- Your code should have an END directive!
- Example:

```
END         ;This is the End
```

ASCII literals

- ASCII characters can be used directly as an immediate operand, or they can used to define symbols or store ASCII bytes in Program Memory (DB directive).
- **Example:**
  - **MOV A,#'m' ; Load A with 06DH (ASCII m)**
  - **DB 'Hello there!' ; Stored in Program Memory**

Comments

- Comments can be placed anywhere, but they must be the last field in any line.
- They are preceded by ';'
- **Example:**

```
        MOV R0, #100 ; Repeat 100 times
LOOP:   DJNZ R0, LOOP
```

- **Beware of useless comments:**

```
        MOV R0, #100 ; Move 100 to R0
LOOP:   DJNZ R0, LOOP ;Decrement and
                      ;jump if no zero
                      ;to LOOP
```

Macros

- A macro is a name assigned to one or more assembly statements or directives. Macros are used to include the same sequence of instructions in several places.
- A macro is a segment of instructions that is enclosed between the directives MAC and ENDMAC. The format of a macro is as follows:

Macros

```
average MAC ; computes the average of three byte arguments
    mov A, %0
    add A, %1
    add A, %2
    mov B, #3
    div AB ; divide B into A and A gets the average
ENDMAC ; terminate the macro definition
```

To invoke the above defined macro, enter the macro name and its parameters. The statement:

```
average (R1 ,R2, R3)
```

will enable the assembler to generate the following instructions, starting from the current location counter:

```
mov A, Rl
add A, R2
add A, R3
mov B, #3
div AB
```

Number representation

- Decimal (default): 2957 or 2957D
- Binary: 101110001101B
- Octal: 5615o or 5615Q
- Hexadecimal: 0B8DH, 0b8dh, 0x0B8D, 0x0b8d

The maximum number that can be input in any radix is 65535 which corresponds to 16 bits.

Addressing Modes

1. Register Inherent
2. Direct
3. Immediate
4. Indirect
5. Indexed
6. Relative
7. Absolute
8. Long
9. Bit Inherent
10. Bit Direct

Addressing Modes: Register Inherent

- The opcode is already associated with the register for speed and efficiency
- Examples:
  - MOV R1, #10 ; opcode: 01111001B + 00001010B
  - INC A ; opcode: 00000100B
  - INC R3 ; opcode: 00001011B

Addressing Modes: Direct

- Used to access the first 128 bytes of internal ram (address 0 to 127) or the SFRs (address 128 to 255).
- Examples:
  - MOV 20H, A ; 11110101B + 00100000B
  - MOV 50H, 25H ; 10000101B + 00100101B + 01010000B
  - MOV 50H, #25H ; 01110101B + 01010000B + 00100101B
  - MOV 80H, A ; 80H>127 therefore is a SFR!
  - MOV P0, A ; SFR P0 is at address 80H

Addressing Modes: Immediate

- Used to initialize a register or memory.
- The number to load MUST be preceded with '#'.
- Example:

```
     MOV R0, #10
L1:  MOV P1, #01H
     NOP ; Does nothing just wastes time!
     MOV P1, #00H
     DJNZ R0, L1 ; Decrement an jump if no zero
```


Addressing Modes: Indirect

- Can be used to access ALL the internal RAM of the microcontroller. It is the only means of accessing the internal RAM from address 128 to 255!
- Only means of accessing XRAM memory using the MOVX instruction.
- EXAMPLES:

```
MOV R0, #80H
MOV A, @R0 ; Copy content of RAM loc. 80H into the Accumulator
MOV A, 80H ; Copy SFR 80H (P0) into the accumulator (See the difference!)
MOV DPTR, #200H
MOVX A, @DPTR ; Copy content of XRAM loc. 200H into the Accumulator
MOV R6, A
INC DPTR
MOVX A, @DPTR ; Copy content of XRAM loc. 201H into the Accumulator
MOV R7, A
```

Addressing Modes: Indexed

- Uses a base register (DPTR) and a offset register (Accumulator)
- Very common in other processors such as the Pentium, but limited in the 8051.
- Examples:

  JMP @a+DPTR

  MOVC a, @a+DPTR

  MOVC a, @a+PC

Addressing Modes: Relative

- Used to jump (conditionally or unconditionally) to other parts of your code.
- The offset is an 8-bit SIGNED number. We can jump in a range -128 to +127 bytes:

```
                    1     $mod52
0000                2     L1:
0000 00             3             nop
0001 80FD           4             sjmp L1
0003 8001           5             sjmp L2
0005 00             6             nop
0006 00             7     L2:     nop
                    8     end
```

Addressing Modes: Relative

- To jump conditionally use any bit variable available:

```
JZ L1
JNZ L2
JC L3
JNC L4
JB P1.3, L5
```


Addressing Modes: Absolute

- Jump or call using a combination of the 11 bits in the destination and 5 upper bits of the program counter.
- It allow to jump or call within the same 2K page you program counter is.
- Two byte instructions!
- No conditional jumping available.
- Only two instructions available:

```
ACALL myroutine
AJMP DONE
```

Labels

Addressing Modes: Long

- Jumps or calls to any destination in the 16-bit range of the microcontroller.
- Three byte instructions (takes one more byte that the absolute instructions)
- No conditional jumping available.
- Only two instructions available:

  LCALL myroutine

  LJMP DONE

Relative vs. Absolute vs. Long

```
                        1    $mod52
0000 8007               2           sjmp L1
0002 0109               3           ajmp L1
0004 020009             4           ljmp L1
0007 4000               5           jc L1
0009 00                 6    L1:    nop
                        7    end
```

The encoding of the AJMP/ACALL instructions is a bit weird. For example for ACALL:

A_{10}	A_9	A_8	1	0	0	0	1	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Addressing Modes: Bit Inherent

- Works only with the carry flag.
- Uses these three instructions:

```
SETB C
CLR C
CPL C
```


Addressing Modes: Bit Direct

- Very few processors have this mode.
- The first 128 bits are mapped in internal ram from bytes 20H to 2FH.
- The second 128 bits are mapped to SFR if the SFR address is divisible by 8.
- Examples:

```
CLR P0.0 ; Clear bit 0 of port 0 (Addr. 80H)
SETB P0.1 ; Set bit 1 of port 0 (Addr. 81H)
SETB 00H ;  This is bit 0 of byte 20 in internal RAM
```

EXAMPLE: Blinky

```
; Blinky.asm: blinks LEDR0 of the CV-8052 each second.
$MODDE0CV

org 0000H
        ljmp myprogram

;For a 33.333333MHz clock, one machine cycle takes 30ns
WaitHalfSec:
        mov R2, #90
L3:     mov R1, #250
L2:     mov R0, #250
L1:     djnz R0, L1 ; 3 machine cycles-> 3*30ns*250=22.5us
        djnz R1, L2 ; 22.5us*250=5.625ms
        djnz R2, L3 ; 5.625ms*90=0.506s (approximately)
        ret
myprogram:
        mov SP, #7FH ; Set the beginning of the stack (more on this later)
        mov LEDRA, #0 ; Turn off all unused LEDs (Too bright!)
        mov LEDRB, #0
M0:
        cpl LEDRA.0
        lcall WaitHalfSec
        sjmp M0
END
```


EXAMPLE: Blinky

- To run “Blinky” in the CV-8052 we need to complete the following steps:
  1. Setup the CV-8052 processor in the Altera DE0-CV board.
  2. Edit and Compile “Blinky.asm”.
  3. Load and run “Blinky.hex” into the CV-8052 processor.

Setup the CV-8052 processor

- Download and Install Quartus from the Altera/Intel website into your computer. Also, follow Altera's instructions on how to install the USB-Blaster driver. Of course you can skip this step if you have installed Quartus already! Execute these instructions only once:
  - Download and open the project CV-8052 from the course web page.
  - Connect the USB cable to the DE0-CV board and the computer. Toggle switch SW10 from "RUN" to "PROG". Turn the DE0-CV board on.
  - Click Tools->Programmer->Start. Wait for the program to finish.
  - Toggle SW10 back to "RUN".

Compile "Blinky.asm" and flash "Blinky.hex".

- Download and install Crosside. See slide 4.
- Start Crosside and create a new assembly file. Copy and paste the code from slide 39. Save as "Blinky.asm".
- Click Build->ASM51. Click the "Browse" button beside "Complete path of assembler" and select a51.exe from the "Call51\bin" folder where you installed Crosside. Then press "Ok".

Flash and run into the CV-8052

- While pressing button “KEY0”, press and release button “FPGA_RESET”. When “boot” shows up on the 7-segment display, release “KEY0”.
- In Crosside, click “fLash”->”Quartus SignalTap II”. Make sure the file “Blinky.hex” is selected. Also make sure that the “quartus_STP.exe” and “Load_script.tcl” fields point to valid files. Click the “Flash” button and wait until it finishes.
- Press and release “FPGA_RESET”. The program starts running!

Exercises

- Explain the differences between these two assembly instructions:

```
mov a, #10H
mov a, 10H
```

- There are two ways to access the internal memory of the 8051/8052 microcontroller: directly and indirectly. Explain why the combination of these two instructions

```
mov R0, #0A0H
mov a, @R0
```

and this supposedly equivalent instruction

```
mov a, 0A0H
```

Result with different values in the accumulator.

Exercises

- In the CV-8052 the seven segments displays HEX0 to HEX5 are mapped to Special Function Registers with the same name. For instance, to turn on all segments of display HEX5 in assembly language we use "mov HEX5, #00H". Write an assembly program that counts from '0' to '9' and displays the number in HEX0.
- Write an assembly program that displays the least significant digits of your UBC student number (6 decimal digits) into the seven segment displays of the CV-8052.
- Write a macro that decrements the data pointer.