8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- In the early days of the computer, programmers coded in *machine language*, consisting of 0s and 1s
  - Tedious, slow and prone to error
- *Assembly languages*, which provided mnemonics for the machine code instructions, plus other features, were developed
  - An Assembly language program consist of a series of lines of Assembly language instructions
- Assembly language is referred to as a *low-level language*
  - It deals directly with the internal structure of the CPU

8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- Assembly language instruction includes
  - a mnemonic (abbreviation easy to remember)
    - the commands to the CPU, telling it what those to do with those items
  - optionally followed by one or two operands
    - the data items being manipulated
- A given Assembly language program is a series of statements, or lines
  - Assembly language instructions
    - Tell the CPU what to do
  - Directives (or pseudo-instructions)
    - Give directions to the assembler

- An Assembly language instruction consists of four fields:

```
[label:] Mnemonic [operands] [;comment]
```

```
   ORG   0H              ;start(origin) at location
   0
   MOV   R5, #25H         ;load 25H into R5
   MOV   R7, #34H         ;load 34H in
   MOV   A, #0            ;load 0 into
   ADD   A, R5            ;add content
                          ;now A = A +
   ADD   A, R7            ;add contents of R7 to A
                          ;now A = A + R7
   ADD   A, #12H          ;add to A value 12H
                          ;now A = A + 12H
HERE: SJMP HERE           ;stay in this loop
   END                     ;en
```

Mnemonics produce opcodes

Directives do not generate any machine code and are used only by the assembler

The label field allows the program to refer to a line of code by name

Comments may be at the end of a line or on a line by themselves
The assembler ignores comments

ASSEMBLING AND RUNNING AN 8051 PROGRAM

- The step of Assembly language program are outlines as follows:
  1) First we use an editor to type a program, many excellent editors or word processors are available that can be used to create and/or edit the program
     - Notice that the editor must be able to produce an ASCII file
     - For many assemblers, the file names follow the usual DOS conventions, but the source file has the extension “asm“ or “src”, depending on which assembly you are using

ASSEMBLING AND RUNNING AN 8051 PROGRAM (cont')

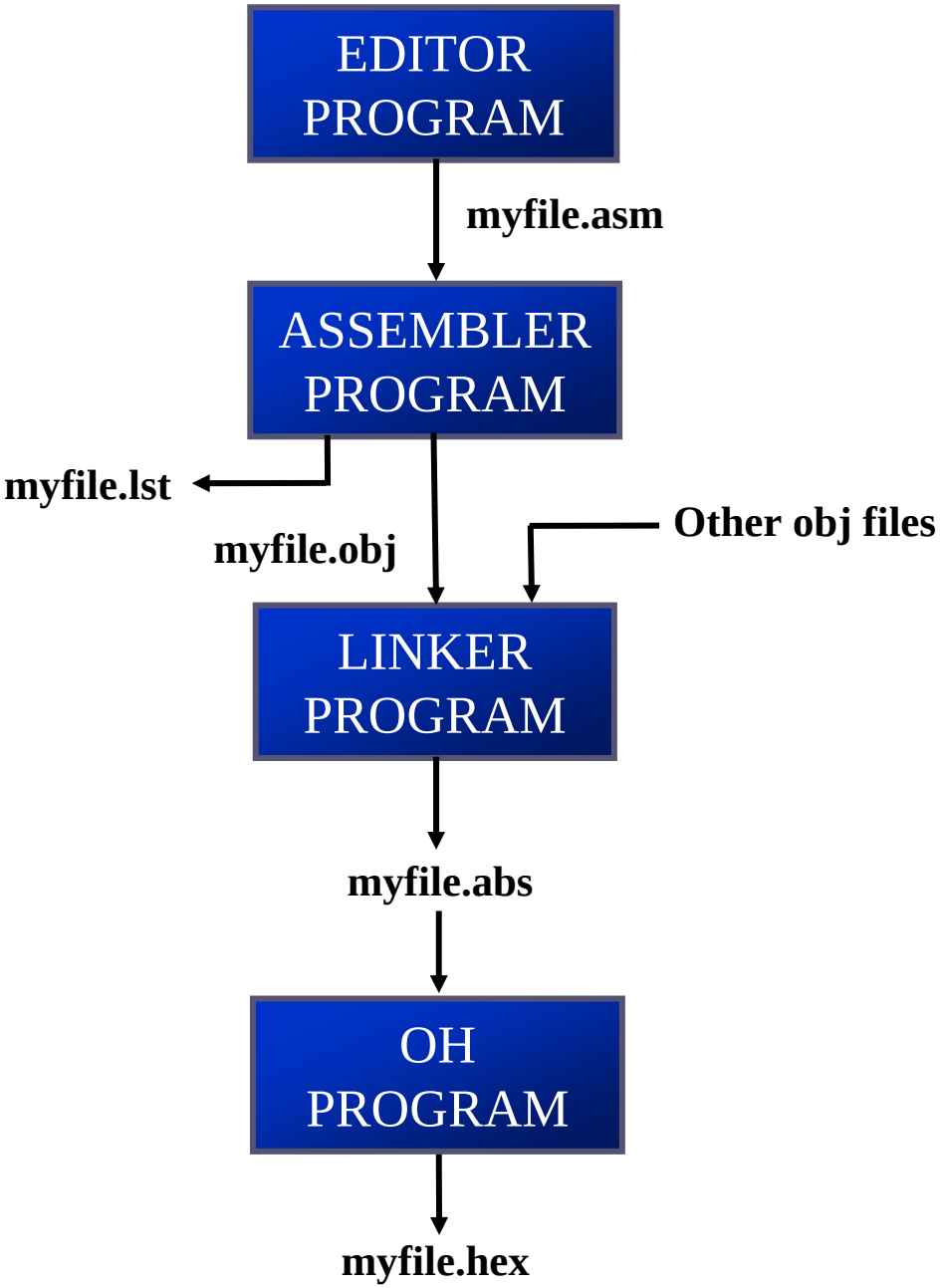
2) The "asm" source file containing the program code created in step 1 is fed to an 8051 assembler
   - The assembler converts the instructions into machine code
   - The assembler will produce an object file and a list file
   - The extension for the object file is "obj" while the extension for the list file is "lst"
3) Assembler require a third step called *linking*
   - The linker program takes one or more object code files and produce an absolute object file with the extension "abs"
   - This abs file is used by 8051 trainers that have a monitor program

ASSEMBLING AND RUNNING AN 8051 PROGRAM (cont')

4) Next the "abs" file is fed into a program called "OH" (object to hex converter) which creates a file with extension "hex" that is ready to burn into ROM
   - This program comes with all 8051 assemblers
   - Recent Windows-based assemblers combine step 2 through 4 into one step

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to Create a Program



ASSEMBLING AND RUNNING AN 8051 PROGRAM

lst File

- The lst (list) file, which is optional, is very useful to the programmer
  - It lists all the opcodes and addresses as well as errors that the assembler detected
  - The programmer uses the lst file to find the syntax errors or debug

```
1 0000          ORG 0H          ;start (origin) at 0
2 0000  7D25    MOV R5,#25H     ;load 25H into R5
3 0002  7F34    MOV R7,#34H     ;load 34H into R7
4 0004  7400    MOV A,#0        ;load 0 into A
5 0006  2D      ADD A,R5        ;add contents of R5 to A
                                ;now A = A + R5
6 0007  2F      ADD A,R7        ;add contents of R7 to A
                                ;now A = A + R7
7 0008  2412    ADD A,#12H      ;add to A value 12H
                                ;now A = A + 12H
8 000A  80EF HERE: SJMP HERE;stay in this loop
9 000C          END             ;end of asm source file
```

address

PROGRAM COUNTER AND ROM SPACE

Program Counter

- The program counter points to the address of the next instruction to be executed
  - As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction
- The program counter is 16 bits wide
  - This means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

PROGRAM COUNTER AND ROM SPACE

Power up

- All 8051 members start at memory address 0000 when they're powered up
  - Program Counter has the value of 0000
  - The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted
  - We achieve this by the ORG statement in the source program

PROGRAM COUNTER AND ROM SPACE

Placing Code in ROM

- Examine the list file and how the code is placed in ROM

```
1 0000             ORG 0H              ;start (origin) at 0
2 0000  7D25       MOV R5,#25H         ;load 25H into R5
3 0002  7F34       MOV R7,#34H         ;load 34H into R7
4 0004  7400       MOV A,#0            ;load 0 into A
5 0006  2D         ADD A,R5            ;add contents of R5 to A
                                       ;now A = A + R5
6 0007  2F         ADD A,R7            ;add contents of R7 to A
                                       ;now A = A + R7
7 0008  2412       ADD A,#12H          ;add to A value 12H
                                       ;now A = A + 12H
8 000A  80EF       HERE: SJMP HERE     ;stay in this loop
9 000C             END                 ;end of asm source file
```

ROM Address	Machine Language	Assembly Language
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

PROGRAM COUNTER AND ROM SPACE

Placing Code in ROM (cont')

- After the program is burned into ROM, the opcode and operand are placed in ROM memory location starting at 0000

ROM contents

Address	Code
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

PROGRAM COUNTER AND ROM SPACE

Executing Program

- A step-by-step description of the action of the 8051 upon applying power on it
  1. When 8051 is powered up, the PC has 0000 and starts to fetch the first opcode from location 0000 of program ROM
     - Upon executing the opcode 7D, the CPU fetches the value 25 and places it in R5
     - Now one instruction is finished, and then the PC is incremented to point to 0002, containing opcode 7F
  2. Upon executing the opcode 7F, the value 34H is moved into R7
     - The PC is incremented to 0004

PROGRAM COUNTER AND ROM SPACE

Executing Program (cont')

- (cont')
  3. The instruction at location 0004 is executed and now PC = 0006
  4. After the execution of the 1-byte instruction at location 0006, PC = 0007
  5. Upon execution of this 1-byte instruction at 0007, PC is incremented to 0008
     - This process goes on until all the instructions are fetched and executed
     - The fact that program counter points at the next instruction to be executed explains some microprocessors call it the *instruction pointer*

PROGRAM COUNTER AND ROM SPACE

ROM Memory Map in 8051 Family

- No member of 8051 family can access more than 64K bytes of opcode
  - The program counter is a 16-bit register