Assembly Programming for 8051

An Elementary Introduction

Dinesh Sharma

Electrical Engineering Department
IIT Bombay

February 1, 2021

Assembly Programming

- We can write a program for a microcomputer (like 8051) in its “assembly language”.
- The assembly language program includes “directives”, which are commands to the assembler program.
- Apart from these directives, the program contains mnemonics for instructions meant for the micro-controller.
- We use a program on a computer (called an assembler) which translates this assembly language program to binary instructions understood by the micro-controller.

Assembly and Download process

- We put down the directives and processor instructions in the assembly program.
- This is a text file with extensions like .asm or .s.
- The assembler program running on a computer translates it to a Hex file, which is a text representation of the binary instructions meant to be executed by the micro-controller.
- Now we run another program (for example, Flip) on the computer to download this hex file to the processor board.
- The processor board has a pre-loaded program which can translate the hex file to a binary format and places the binary instructions at designated addresses in the program memory of the processor.

Assembly Programs for 8051

- Micro-computers of the 8051 family have separate program and data memories.
- When these micro-computers are turned on or are reset, they start fetching instructions from address 0000 onward in the program memory and executing them.
- However, addresses in the range 0003H to 0030H are reserved for instructions required for interrupt processing (which we shall learn later).
- Therefore user programs should begin from an address beyond this range.

Assembly Programs for 8051

- The micro-computer starts executing from address 0000.
- Addresses from 0003H to 0030H are required for interrupts.
- The user program should be placed beyond this. We may choose to place our program at (say) 0100H.
- We place a jump instruction at 0000, asking the micro-controller to jump to the starting address of the user program.
- Now at power on or reset, the first instruction fetched is the jump instruction. Therefore the next instruction is fetched from the destination of the jump – which is the start address of our program.

Example

```
        ORG    0000H    ; This is a directive.
                        ; Sets current address to 0000H
        ljmp   start    ; For skipping the interrupt area
        ORG    0100H    ; Sets current address to 0100H
start:                  ; Start is a label
                        . equated to current address
        ...    ...      ; (Which is 0100H)
```

Note that the assembler keeps track of current address and equates start to 0100H.

So it replaces the first instruction by ljmp 0100H.

ORG, label names etc. are handled by the assembler program and are never seen by the micro-controller.

The HEX file contains only absolute values like 0100H.

Ending a Program

- A processor keeps fetching instructions for ever.
- So how do we end a program?
- – We write an endless loop to end the program.

For example:

```
         ...    ...     User Program
finish:  ljmp   finish  Keep looping here
```

Loading programs in a micro-controller

- Programs for micro-controllers are developed using tools running on PCs.
- This requires cross compilers and cross assemblers which run on PCs, but output code which is suitable for the micro-controller system being developed.
- Once the program has been compiled, it should be loaded into the program memory of the micro-controller.
- This requires hardware to connect the system to the PC and software at the micro-controller to receive and load this program at the desired address.
- This is a chicken and egg problem!
  How do we load the software needed at the micro-controller in the first place?

Solving the Chicken and Egg problem

- Modern micro-controllers contain flash memory (similar to those used in USB thumb drives and SD cards) for storing program and constants.
- The manufacturer pre-loads a program in this flash memory to carry out tasks related to program loading etc.
- This is called a **monitor** program. It is like a miniature operating system.
- At power on, the monitor program starts running. It can receive a file which has all the information required to load multiples blocks of binary data at appropriate addresses.
- Notice that it may be necessary to load a main program, interrupt service routines, constant data blocks etc., all at different addresses.
- Standard file formats have been developed to carry this information, along with error checking etc.

The Intel Hex Format

- Prominent among the standard formats developed to load blocks of binary data into a micro-controller or a ROM is the Intel Hex format.
- Other formats include Motorola S records etc..
- “Hex” refers to a number system with base 16. It uses the standard decimal numerals 0 to 9 and adds the six symbols A, B, C, D, E and F to form a number system where each symbol has a place value of 16.
- The advantage of a “Hex” number system is that it represents 4 bits exactly. Decimal numerals do not represent integral number of bits.
- The other choice is to use an Octal number system, where we only use the numerals 0 to 7. This represents 3 bits exactly. However, because the natural size of quantities in a micro-controller system is a multiple of 4 or 8 bits, the Hex format is preferred.

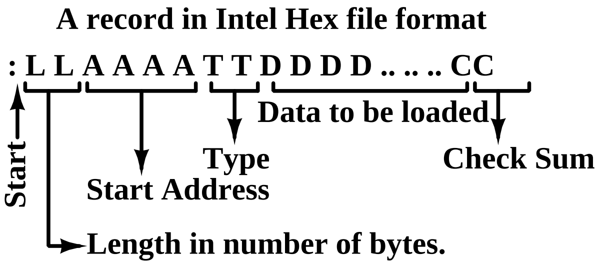
The Intel Hex Format

- The file used in Intel Hex format is a text file, so it is human readable.
- Binary data is represented 4 bits at a time using text characters ‘0’, ‘1’, . . . , ‘9’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ and ‘F’.
- The file contains a number of **records**, each record represents a data block to be loaded at a specified address.
- Each record begins with the character **‘:’**, followed by the length of the binary data in this record, expressed as two hex characters.
- The length field encodes the number of bytes (represented by two hex characters) in the record which carry actual data to be loaded in memory.
- The length field is followed by 4 hex characters representing the 16 bit address where this record is to be loaded.

The Intel Hex Format

- The address is followed by two hex characters which specify the type of the record.
- Type '00' represents a data record, while type '01' represents an end of file record.
- Other types are used for extending the address size beyond 16 bits and for carrying specific information relevant to other micro-controllers.
- The type field is followed by actual data, each byte being represented by two hex characters.
- The last two characters represent the check sum for the record. Each pair of hex characters after ':' is added, ignoring any overflows. The 2's complement of the sum of all these bytes is the check sum.
- The monitor can add all pairs of hex characters (interpreted as 8 bit bytes) following ":" (inclusive of the check sum), and should always get zero if there has been no error in sending the file.

A record example in Intel Hex format


A record in Intel Hex file format
: L L A A A A T T D D D D .. .. .. C C
Start
Start Address
Type
Data to be loaded
Check Sum
Length in number of bytes.

A record is ended by the end of line character(s) in the text file.

An example

Consider the following assembly program:

```
        MOV A, #100
LA:     DJNZ ACC, LA
LB:     SJMP LB
```

Here we have initialized the accumulator with the decimal constant 100 (64 in Hex) and then decremented it in a loop till it becomes zero. After that, we just loop in the last statement.

When assembled, it gives the following listing:

```
 000000   74 64             MOV A, #100
 000002   D5 E0 FD   LA:    DJNZ ACC, LA
 000005   80 FE      LB:    SJMP LB
```

The left column is the address, the next column gives the binary code, the third column has labels while the last column has instructions.

An example

```
000000   74 64             MOV A, #100
000002   D5 E0 FD   LA:    DJNZ ACC, LA
000005   80 FE      LB:    SJMP LB
```

To load this program, we want to load
74, 64 at addresses 0000 and 0001.
D5, E0, FD at addresses 0002, 0003 and 0004.
80, FE at addresses 0005 and 0006.

This will be done by the Intel hex file:

```
:070000007464D5E0FD80FEF1
:00000001FF
```

Let us interpret these records.

An example

The Intel Hex file was
:070000007464D5E0FD80FEF1
:00000001FF

The fields are :LLAAAATTDDDD…CC
In the first record, Length is 07, so 7 bytes are to be loaded in the memory, starting from the address 0000. The type is 00, so this is a data record. The 7 bytes 74, 64, D5, E0, FD, 80 and FE are loaded at addresses 0000 to 0006.

Finally, the last byte is the check sum, F1.
we can check that the sum of all bytes (ignoring overflows) is 00.

The second record has zero bytes to load, the address 0000 is therefore irrelevant, and the type is 01, so this ends the loading of bytes.
As in the previous record, the checksum FF ensures that the sum of all bytes in this record is 00.