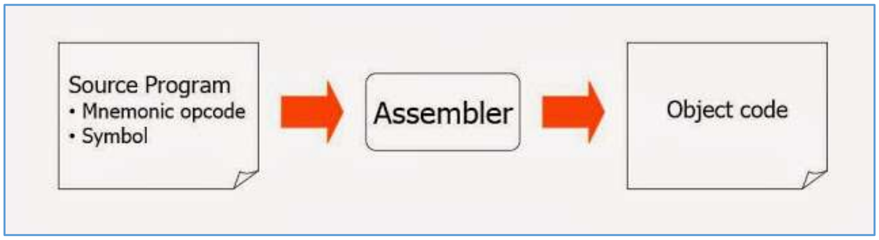
Software Systems-1

THE ASSEMBLER- TWO PASS ASSEMBLER

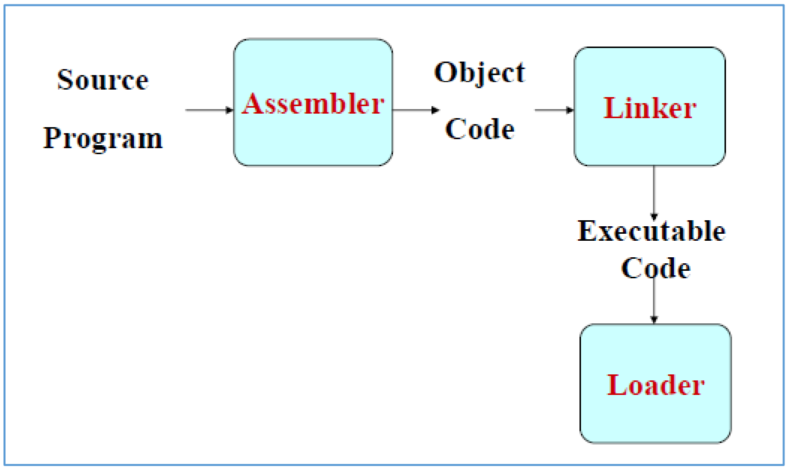
Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

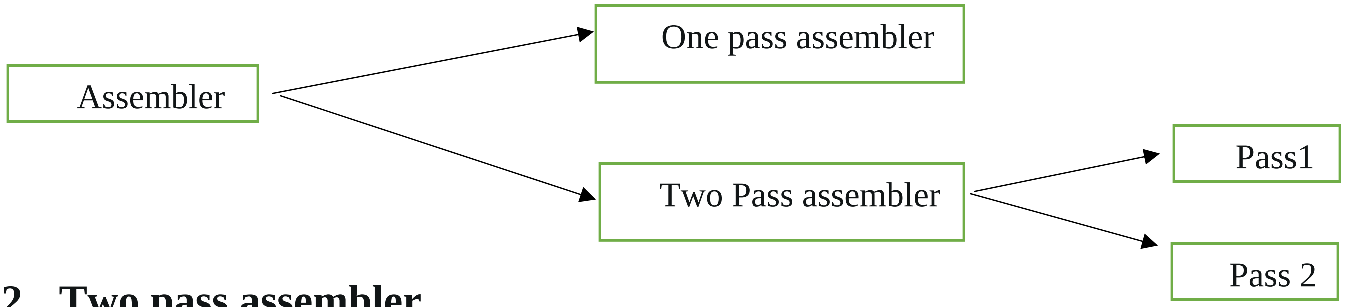
1 Introduction

An assembler is a program that turns assembly language into machine code that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. Some people call these instructions assembler language and others use the term assembly language.



The programmer can write a program using a sequence of these assembler instructions. This sequence of assembler instructions, known as the source code or source program, is then specified to the assembler program when that program is started. The assembler program takes each program statement in the source program and generates a corresponding bit stream or pattern (a series of 0's and 1's of a given length). The output of the assembler program is called the ***object code*** or ***object program*** relative to the input source program. The sequence of 0's and 1's that constitute the object program is sometimes called ***machine code***. The object program can then be run (or executed) whenever desired.

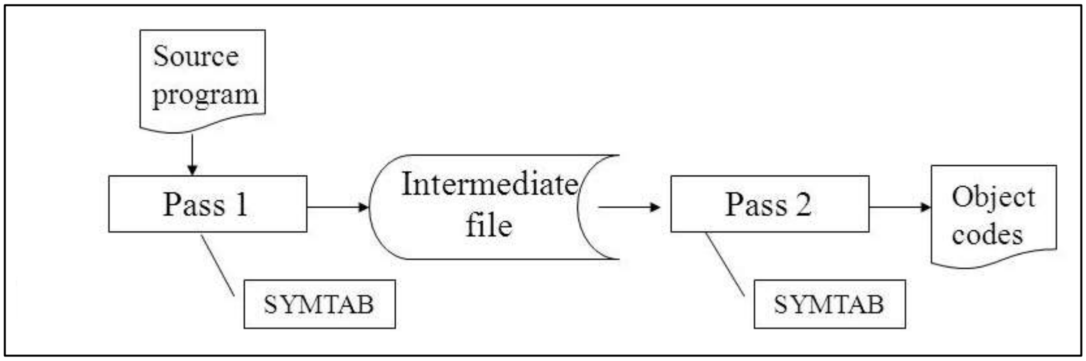




2 Two pass assembler

A two-pass assembler takes a first pass through the assembly program to construct a symbol table (**SYMTAB**) that contains a list of all labels and their associated location counter values and intermediate file. It then takes a second pass to translate the assembly program into object code.

But a one-pass assembler on the other hand combines both operations in a single pass, and resolves forward references on the fly



Basic Assembler Functions are:

- Assembler directives (pseudo-instructions) ORG, DB, DW, EQU, END.
- These statements are not translated into machine instructions. Instead, they provide instructions to the assembler itself.

Pseudo-operations, are commands given to an assembler "directing it to perform operations other than assembling instructions. Directives affect how the assembler operates and "may affect the object code, the symbol table, the listing file, and the values of internal assembler parameters.

Below example shows how the data being saved into memory.

```
            ORG  100h
    N1      dB    -1
    N2      dW    3, -6
    Name    dB   'Adnan'
    Len     EQU   33h
    N3      dB     5,3
    N4      dW     3 dup (?)
    SS      dB     2 dup (-1)
    DD      dB     3,?, $
            end
```

N1→	100	FF
N2→	101	03
	102	00
	103	FA
	104	FF
Name	105	'A'
	106	'd'
	107	'n'
	108	'a'
	109	'n'
N3→	10A	05
	10B	03
N4→	10C	00
	10D	00
	10E	00
	10F	00
	110	00
	111	00
SS→	112	FF
	113	FF
DD→	114	03
	115	00
	116	'$'

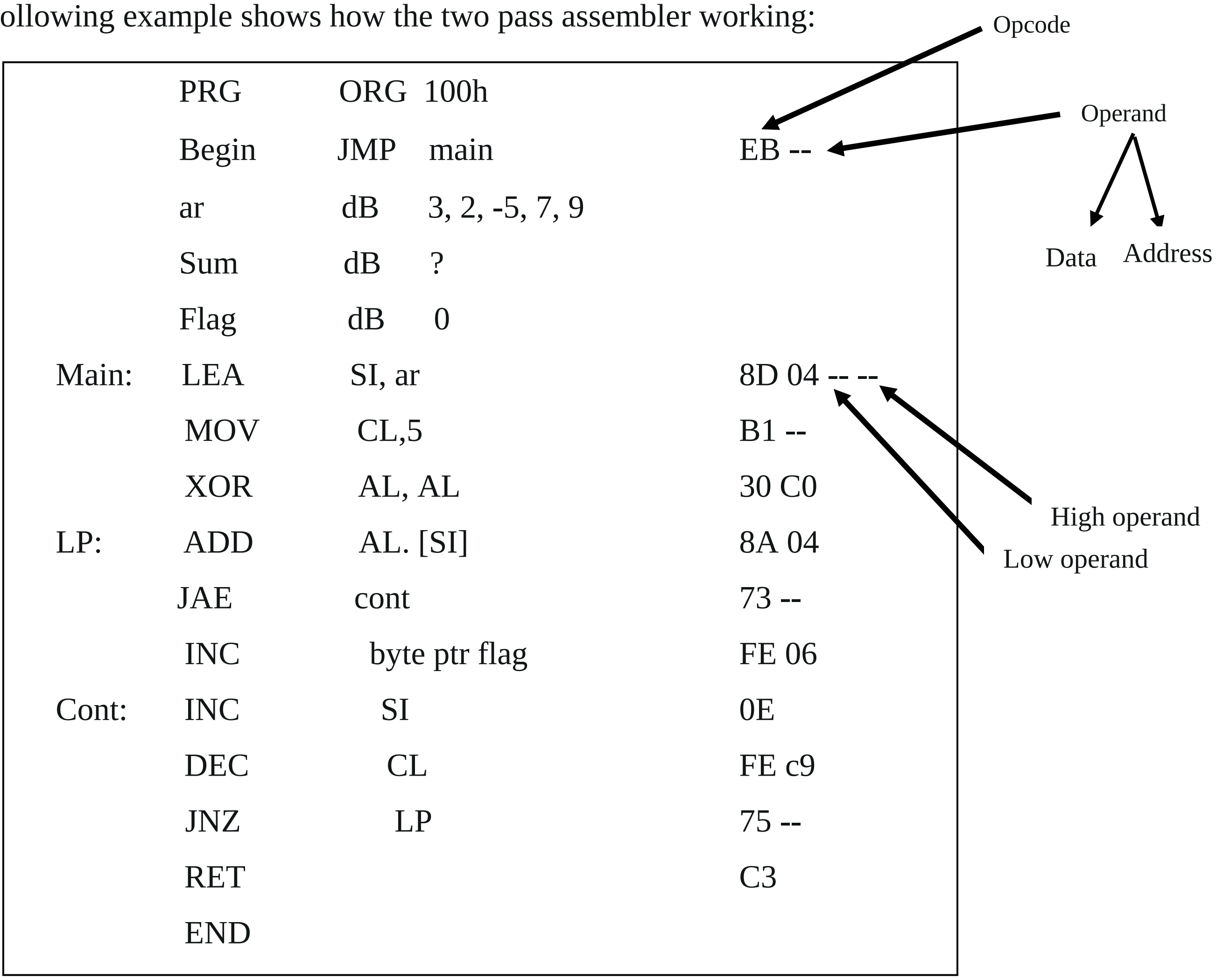
Some useful info.

1h→ $(0000\ 0001)_b$ → 2's complement→ $(1111\ 1111)_h$ → FF

6h→ $(0000\ 0000\ 0000\ 0110)_b$ → 2's complement→1111 1111 1111 1010b → $(FF\ FA)_h$

N2 dW 3→ 00 03

The following example shows how the two pass assembler working:



```
        PRG     ORG  100h
        Begin   JMP   main                  EB --
        ar      dB    3, 2, -5, 7, 9
        Sum     dB     ?
        Flag    dB     0
Main:   LEA      SI, ar                     8D 04 -- --
        MOV      CL,5                       B1 --
        XOR      AL, AL                     30 C0
LP:     ADD      AL. [SI]                   8A 04
        JAE     cont                        73 --
        INC       byte ptr flag             FE 06
Cont:   INC         SI                      0E
        DEC          CL                     FE c9
        JNZ           LP                    75 --
        RET                                 C3
        END
```

Some useful information

LEA (Load effective address)

The LEA instruction places the address specified by its second operand into the register specified by its first operand. Note, the contents of the memory location are not loaded, only the **effective address** is computed and placed into the **register**. This is useful for obtaining a pointer into a memory region.

Pass1

Intermediate File:

Address	Mnemonic field	Error flag
0000	PRG ORG 100h	0
0100	Begin JMP main	0
0102	ar dB 3,2,-5,7,9	0
0107	Sum dB ?	0
0108	Flag dB 0	0
0109	Main: LEA SI, ar	0
010D	MOV CL,5	0
010F	XOR AL, AL	0
0111	LP: ADD AL, [SI]	0
0113	JAE cont	0
0115	INC byte ptr flag	0
0117	Cont: INC SI	0
0118	DEC CL	0
011A	JNZ LP	0
011C	RET	0
011D	END	0

SYMTAB

Symbol	Value	Error Flag
PRG	0000	0
Begin	0100	0
ar	0102	0
Sum	0107	0
Flag:	0108	0
Main:	0109	0
LP:	0111	0
cont	0117	0

Pass 1 Analysis:

This part scans the program looking for symbols, labels, variables, etc. and organize them in tables

- Passes through the instruction in sequences, looking for symbols addresses.
- Create a symbol and literal table
- Keep track of the location counter
- Error checking.

Pass2

Listing files

Line no	Address	Mnemonic filed	Machin code	Error flag
1	0000	PRG ORG 100h		0
2	0100	Begin JMP main	EB 07	0
3	0102	ar dB 3,2,-5,7,9	03,02,FB,07,09	0
4	0107	Sum dB ?	00	0
5	0108	Flag dB 0	00	0
6	0109	Main: LEA SI, ar	8D 04 02 01	0
7	010D	MOV CL,5	B1 05	0
8	010F	XOR AL, AL	30 C0	0
9	0111	LP: ADD AL. [SI]	8A 04	0
10	0113	JAE Cont	73 02	0
11	0115	INC byte ptr flag	FE 06	0
12	0117	Cont: INC SI	0E	0
13	0118	DEC CL	FE C9	0
14	011A	JNZ LP	75 F5	0
15	011C	RET	C3	0
16	011D	End		

Note: ***Displacement = Destination address – Source address – instruction length***

0109 - 0100-2 =07 → main OR 0109 -0102 =07

0117 - 0113-2 =2 → Cont OR 0117 -0115=02

010F - 011A = $\frac{\begin{array}{r}-011A\\+010F\end{array}}{-000B}$ ➔ - 0000 1011 ➔(2’s complement) 1111 0101 ➔F5

Object program

Header → H prg … 0100h(starting address) 001Dh (Program Length)

Text → T 0100 10 EB 07 03 02 FB 07 09 00 00 8D 04 02 01 B1 05 30

T 0110 0D C0 8A 04 73 02 FE 06 0E FE C9 75 F5 C3

End → E 0100

Pass2: If no errors are found in pass 1 then the second pass assembles the code into object code.