Chapter 2
Assemblers

Assembly Language Programming

- Writing a program in assembly lang is more convenient than in machine lang.
- Assembly program is more readable.
- Assembly lang is machine dependent.
- Assembly program is written using symbols(Mnemonics).
- Assembly program is translated into machine code before execution.

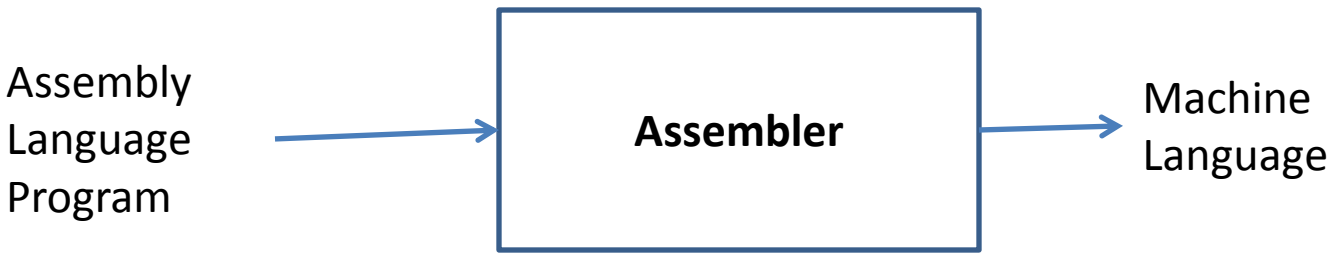


Figure: Assembler

Example:

```
        MOV         AX, X
```

--MOV is a mnemonic opcode.

--AX is a register operand in symbolic form.

--X is a memory operand in symbolic form.

Elements of Assembly Language

1. Mnemonic Operation Code:-

   Eliminates the need to memorize numeric operation code.

2. Symbolic Operands:-

   Symbolic names can be used.

3. Data Declarations:-

   Data can be declared in any form

   Eg: -5, 10.5 etc.

Statement Format

[Label] <Opcode> <operand Spec> [<operand spec>….]

1. Label:- Is optional.
2. Opcode:- Symbolic opcode
3. Operand:- Symbolic name (Register or Memory variable)

Instruction Opcode	Assembly Mnemonic	Remarks
00	STOP	Stop Execution
01	ADD	Op1 ← Op1+ Op2
02	SUB	Op1 ← Op1 – Op2
03	MULT	Op1 ← Op1* Op2
04	MOVER	CPU Reg ← Memory operand
05	MOVEM	Memory ← CPU Reg
06	COMP	Sets Condition Code
07	BC	Branch on Condition
08	DIV	Op1 ← Op1/ Op2
09	READ	Operand 2 ← input Value
10	PRINT	Output ← Operand2

Fig: Mnemonic Operation Codes

Instruction Format

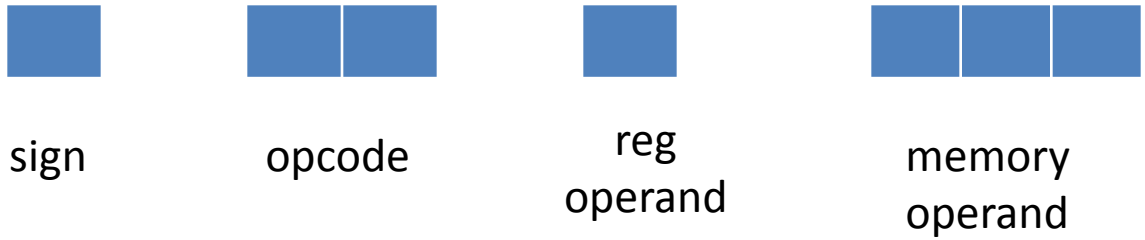


Fig: Instruction Format

Assembly Lang to M/C lang Program

1. Find address of variables and labels.
2. Replace Symbolic addr by numeric addr.
3. Replace Symbolic opcodes by machine opcode.
4. Reserve storage for data.

```
          START     101
          READ      X
          READ      Y
          MOVER     AREG, X
          ADD       AREG, Y
          MOVEM     AREG, RESULT
          PRINT     RESULT
          STOP
     X    DS        1
     Y    DS        1
RESULT    DS        1
          END
```

Fig: Sample program to find X+Y

	START	101	*LC*				
	READ	X	101	+	09	0	108
	READ	Y	102	+	09	0	109
	MOVER	AREG, X	103	+	04	1	108
	ADD	AREG, Y	104	+	01	1	109
	MOVEM	AREG, RESULT	105	+	05	0	110
	PRINT	RESULT	106	+	10	0	110
	STOP		107	+	00	0	000
X	DS	1	108				
Y	DS	1	109				
RESULT	DS	1	110				
	END						

Variable	Address
X	108
Y	109
RESULT	110

Figure: After LC Processing

Required M/C Code

LC	Opcode	Register	Address
101	09	0	108
102	09	0	109
103	04	1	108
104	01	1	109
105	05	0	110
106	10	0	110
107	00	0	000
108			
109			
110			
111			

Assembly Language Statement

1. Imperative Statement.
2. Declaration Statement.
3. Assembler Directives.

- Imperative Statements:-
  - Indicates an action to be taken during execution of a program.
  - Eg: MOV, ADD, MULT, etc.

- Declaration Statement:-
  - To reserve memory for variable.

    [Label] DS <constant> eg: X DS 5

    [Label] DC '<value>' eg: X DC 3

- Assembler Directives:-
  - Instructs the assembler to perform ceratin action during assembly of a program.

    START <constant>

    END

Literals & Constants

```
int z=5;
x = x + 5;
```

1. Literal cannot be changed during program execution
2. Literal is more safe and protected than a constant.
3. Literals appear as a part of the instruction.

Advanced Assembler Directives

- ORIGIN
  - ORIGIN <address specification>
- EQU
  - <symbol> EQU <address specification>
- LTORG
  - Allocates address for literals.

Pass structure of assembler

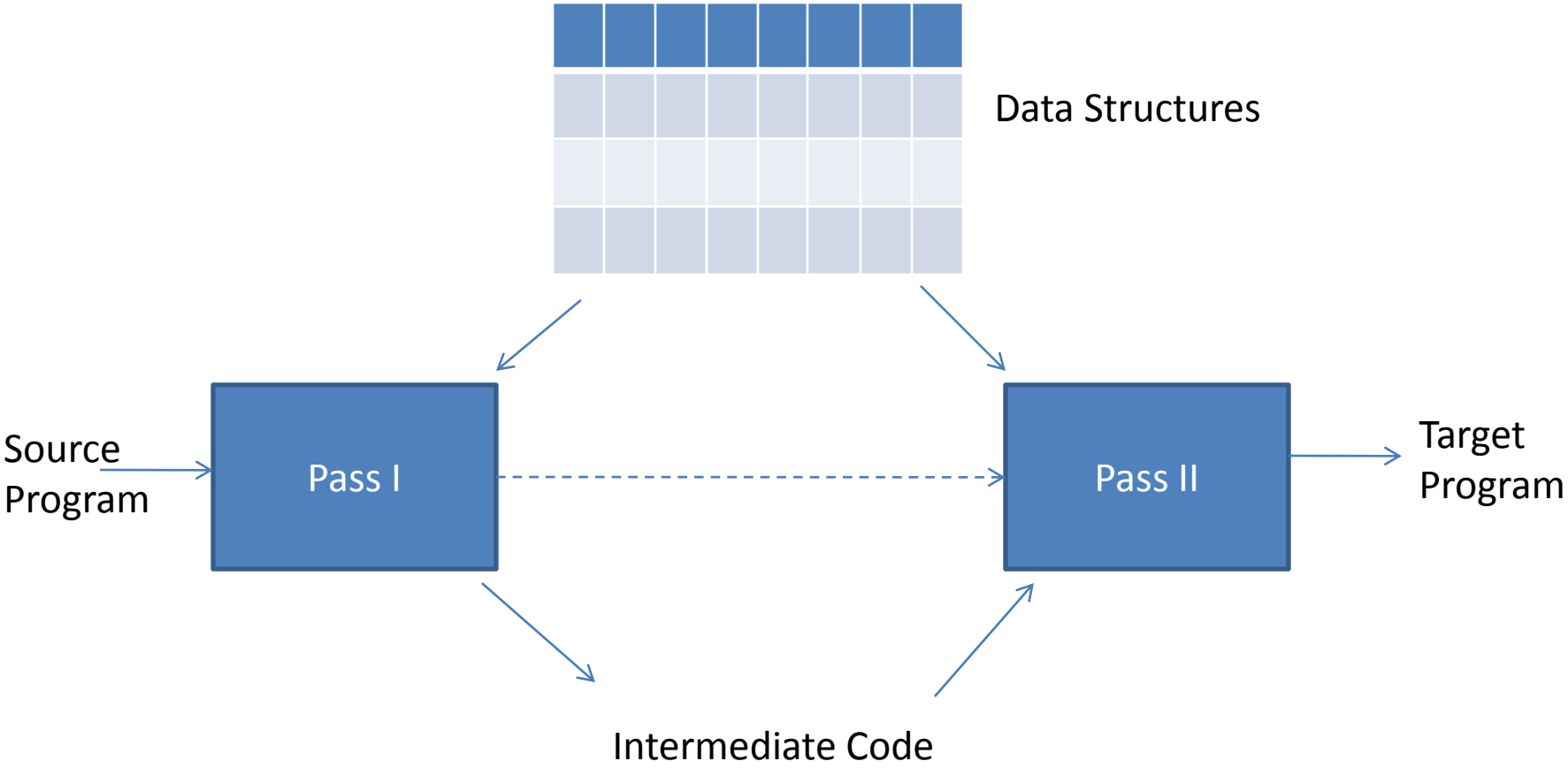


Figure: Overview of Two Pass Assembler

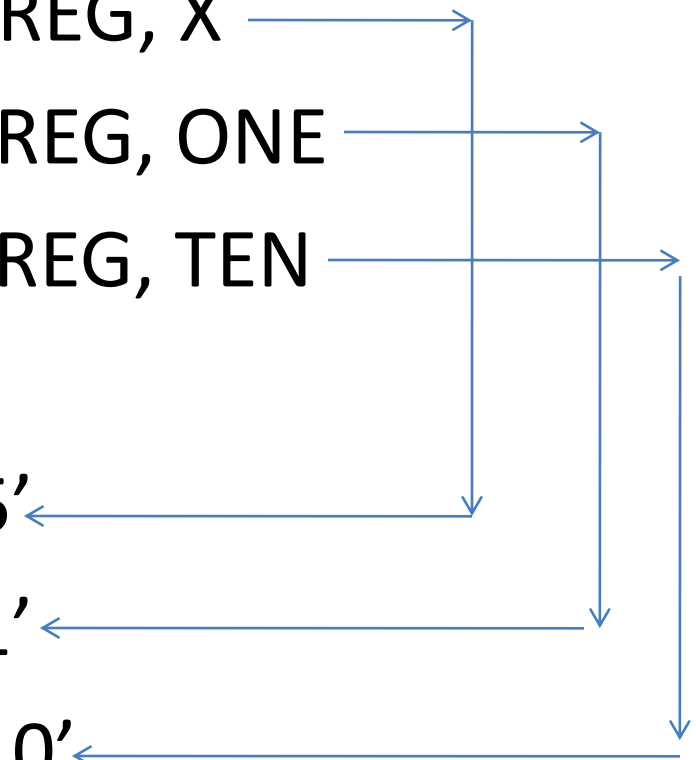
Two Pass Translation

- Handles forward references easily.
- Requires 2 scans of the source program.
- LC processing is performed in the 1st pass and symbols are stored in the symbol table.
- Second pass synthesis Target Program.

Single Pass Translation

- The problem of forward reference can be handled using a technique called as back patching.

```
        START   100
        MOVER   AREG, X
L1      ADD     BREG, ONE
        ADD     CREG, TEN
        STOP
X       DC      '5'
ONE     DC      '1'
TEN     DC      '10'
        END
```



```
        START   100
        MOVER   AREG, X
L1      ADD     BREG, ONE
        ADD     CREG, TEN
        STOP
X       DC      '5'
ONE     DC      '1'
TEN     DC      '10'
        END
```

```
100     04  1   _ _ _
101     01  2   _ _ _
102     06  3   _ _ _
103     00  0   000
104
105
106
```

Instruction Address	Symbol Making a forward reference
100	X
101	ONE
102	TEN

Figure : TII

Machine Instruction After Backpatching

04	**1**	**104**
01	2	105
06	2	106
00	0	000

Design of a Two Pass Assembler

- Pass I:-
  1. Separate the symbol, mnemonic, opcode and operand.
  2. Build Symbol Table.
  3. Perform LC Processing.
  4. Construct Intermediate Representation.
- Pass II:-
  1. Process IR to synthesize the target program.

Pass I

- Pass I uses the following data structures
  1. Machine Opcode table (OPTAB)
  2. Symbol Table (ST)
  3. Literal Table (LT)
  4. Pool Table (PT)

1. OPTAB contains opcode, class and opcode length.
2. SYMTAB contains symbol and address.
3. LITTAB contains literal and address.
4. POOLTAB contains starting literal number of each pool.

```
        START   200             LC
        MOVER   AREG, ='5'      200
        MOVEM   AREG, X         201
L1      MOVER   BREG, ='2'      202
        ORIGIN  L1+3
        LTORG                   205
                                206
NEXT    ADD     AREG,='1'       207
        SUB     BREG,='2'       208
        BC      LT, BACK        209
        LTORG                   210
                                211
BACK    EQU     L1              212
        ORIGIN  NEXT+5
        MULT    CREG,='4'       212
        STOP                    213
X       DS      1               214
        END
```

START 200

Symbol	Address

Literal	Address

Pool Table
0

MOVER AREG,=‘5’ 200

Symbol	Address

Literal	Address
=‘5’	---

Pool Table
0

MOVEM AREG,X 201

Symbol	Address
X	----

Literal	Address
='5'	---

Pool Table
0

L1 MOVER BREG,='2' 202

Symbol	Address
X	----
L1	202

Literal	Address
='5'	---
='2'	---

Pool Table
0

ORIGIN L1+3 203

Symbol	Address
X	----
L1	202

Literal	Address
=‘5’	---
=‘2’	---

Pool Table
0

LTORG 205
206

Symbol	Address
X	----
L1	202

Literal	Address
=‘5’	205
=‘2’	206

Pool Table
0
2

NEXT ADD AREG, =‘1’ 207

Symbol	Address
X	----
L1	202
NEXT	207

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	----

Pool Table
0
2

SUB BREG,=‘2’ 208

Symbol	Address
X	----
L1	202
NEXT	207

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	----
=‘2’	-----

Pool Table
0
2

BC LT, BACK 209

Symbol	Address
X	----
L1	202
NEXT	207
BACK	----

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	----
=‘2’	----

Pool Table
0
2

LTORG 210
211

Symbol	Address
X	----
L1	202
NEXT	207

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	210
=‘2’	211

Pool Table
0
2
4

BACK EQU L1 212

Symbol	Address
X	----
L1	202
NEXT	207
BACK	202

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	210
=‘2’	211

Pool Table
0
2
4

ORIGIN NEXT+5 213

Symbol	Address
X	----
L1	202
NEXT	207
BACK	202

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	210
=‘2’	211

Pool Table
0
2
4

MULT CREG,='4' 212

Symbol	Address
X	----
L1	202
NEXT	207
BACK	202

Literal	Address
='5'	205
='2'	206
='1'	210
='2'	211
='4'	----

Pool Table
0
2
4

STOP 213

Symbol	Address
X	----
L1	202
NEXT	207
BACK	202

Literal	Address
='5'	205
='2'	206
='1'	210
='2'	211
='4'	----

Pool Table
0
2
4

X DS 1 214

Symbol	Address
X	214
L1	202
NEXT	207
BACK	202

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	210
=‘2’	211
=‘4’	----

Pool Table
0
2
4

END

Symbol	Address
X	214
L1	202
NEXT	207
BACK	202

Literal	Address
=‘5’	205
=‘2’	206
=‘1’	210
=‘2’	211
=‘4’	215

Pool Table
0
2
4
5