Unit -3
Programmic Basic Computer

* Introduction to Machine language
- It is easier to write high level programs as it is written in high level language. (eg - C, Java, Python)
- But when binary codes were used which included 0, and 1 in this programming it was try to debug and write the program. As we have to memories the patten of 0's and 1's.
- So, Just above that octel and hexadecimel codes were introduce which was binary equivivelent. This code was fixed from before. As it is try to remember each and every code, a new apporch was introduce known as symbolic code.
- Symbolic code used (letters, numbers, special char.) for operation part and address part (instruction code). This language in order to translate we used a speciell program known as assemble.
- Assemble convert assembly language program to machine language program. It is a translator and it is also known as a system software.

-> Symbolic code to add two number

Location	Instruction	Comments
000	LDA 004	Load first operand into AC
001	ADD 005	Add second AC to operand
002	STA 006	Store sum in location 006
003	HLT	Halt Computer
004	0053	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

→ Assembly language Program to add two numbe

```
    ORG 0       / Origin of program in location 0
    LDA A       / Load operand from location A
    ADD B       / Add operand from location B
    STA C       / Store sum in location C
    HLT         / Halt Computer
A,  DEC 83      / Decimal operand
B,  DEC -23     / Decimal operand
C,  DEC 0       / Sum stored in location C
    END         / End of symbolic Program
```

- We can go a step further and replace hexa-decimal addrees by symbolic addrees and each hexa decimal operand by decimal operand. AS it convinient while writing the program
- Assmble Program is divided in three Columns

① Lables ② Instruction ③ Comments

Assembly Language

1 Every commerical computer has it's own particular assembly language. Rules are documented and published in the ~~main~~ manuals.

2 The Specific language is defined by set of rule that specify symbol that may combine to form line of code.

→ Rules of Language

1 As we know assembly language program is arranged in three column there are also called as fields which specify following information

1. Lable field — This field may be empty or it may specify symbolic address
2. Instruction field — It specifies machine Instruction or ~~pseudo~~ pseudo instruction
3. Comment field - This field may be empty or it may content comments.

→ Assembler is a program that take symbolic code/ assembly code and then it gets converted to the equivalent binary ~~co~~ program. Our program is source code and converted program can be object code. To

This object code can be link and then can be executed

pseudo

→ ~~Pseudo~~ Instruction are instruction to Assembler and

Symbol	Information for the Assembler
ORG N	Origin means starting of the program
END	End means ending of the program
DEC N	Decimal number to be converted into binary number
HEX N	Hexadecimal number is converted into binary number

→ Instruction field can contain three types of instruction
1) Memory Reference
2) Register Reference or I/O
3) Pseudo Instruction

Hexadecimal Code

Location	Content	Symbolic program	
~~100~~	~~2107~~		ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC -23
108	0000	DIF,	HEX 0
			END

- The example shown heore is torenslation of our assembly code to binary. (example) Show that our symbolic code after translation is getting converted into hexadecimal code.

- For an instence in second line of the program we have LDA SUB, where the hex code for LDA is two and 107 is the location address of SUB

- This hex code can be called as intermmediate code as it is very to convert from hexadeeimel to binary.

- So his is how torenslation will teke place.

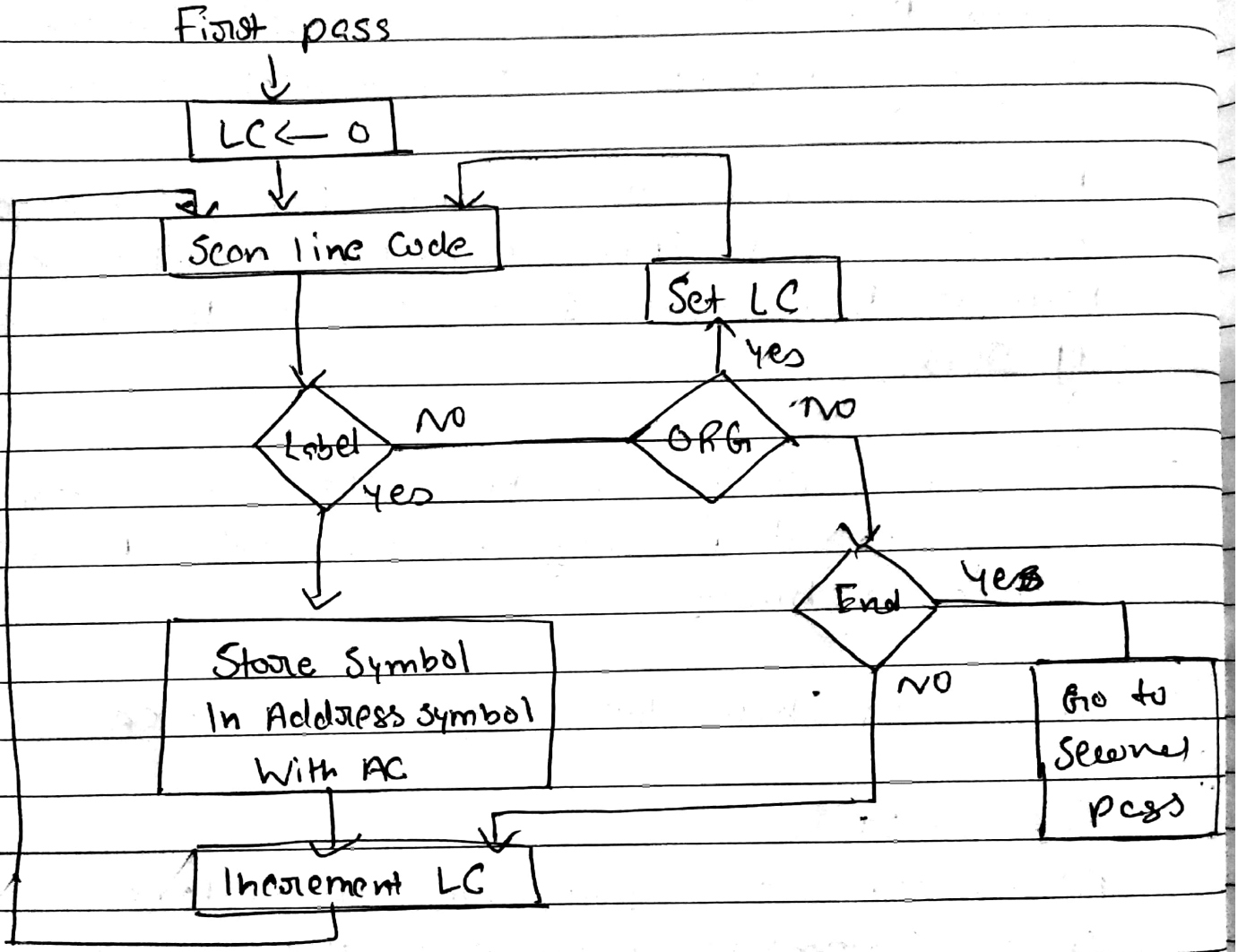
* Assembler

- Assembler is a program that teke Symbolic code/ essrembler code and then it gets converted to the equivalent binary program. Our program is source code and converted program can be object code.

- This object code can be link and then can be executed.

- Assmebler tekes two passes over the source code. In other words assembler will scan our program twice. (first pass and second pass)

=) First pass of Assembler



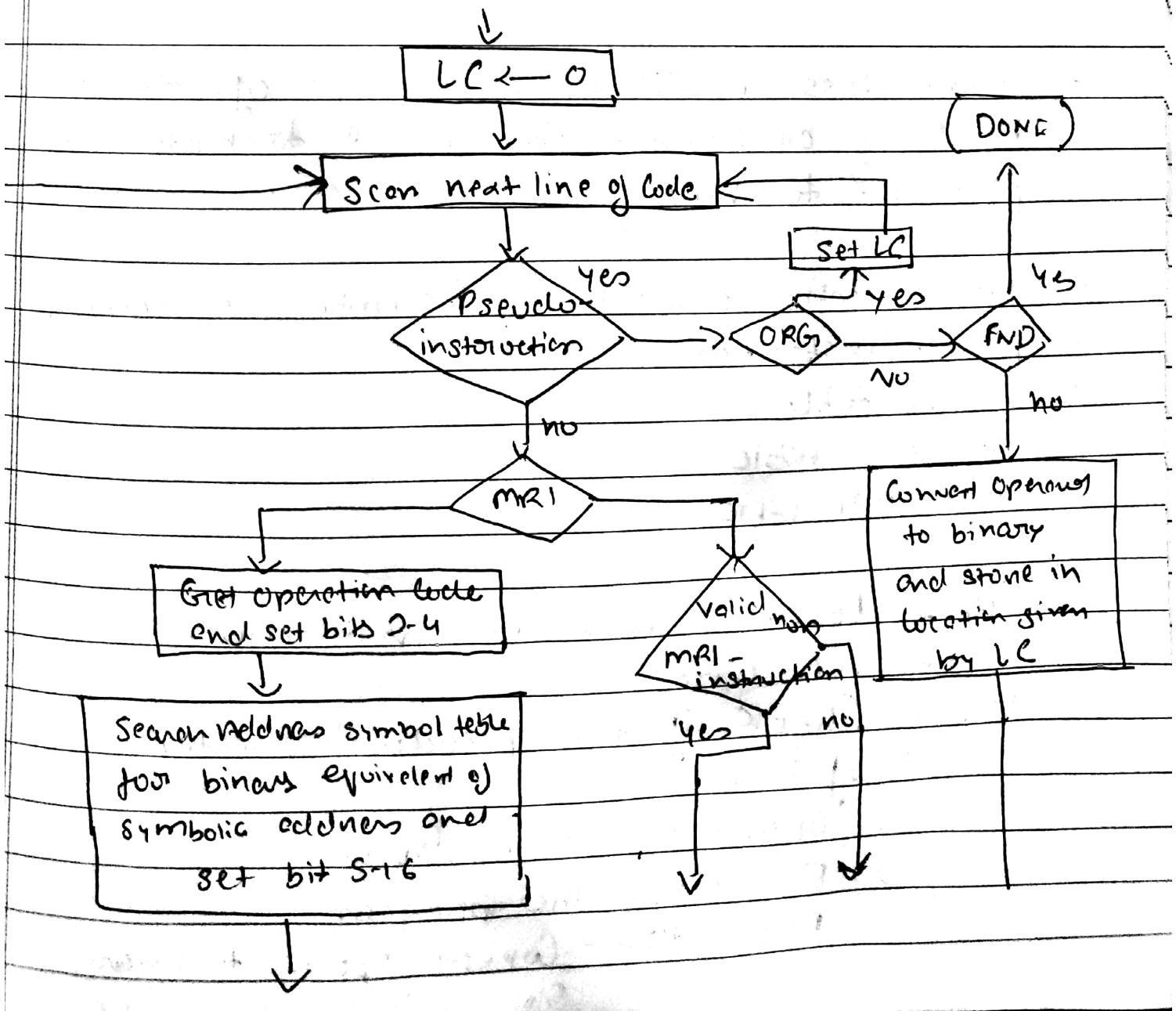
- When assembler scan our program first time it does not do any translation.
- During the first pass it generate table of all symboles with binary equalent value.
- As shown in the flow chart the location counter (LC) will have memory address of our program, it can also be called as the counter of our program intizillization.
- Step 1 : location counter is intilize
  Step 2 : It scan the line check if lable.

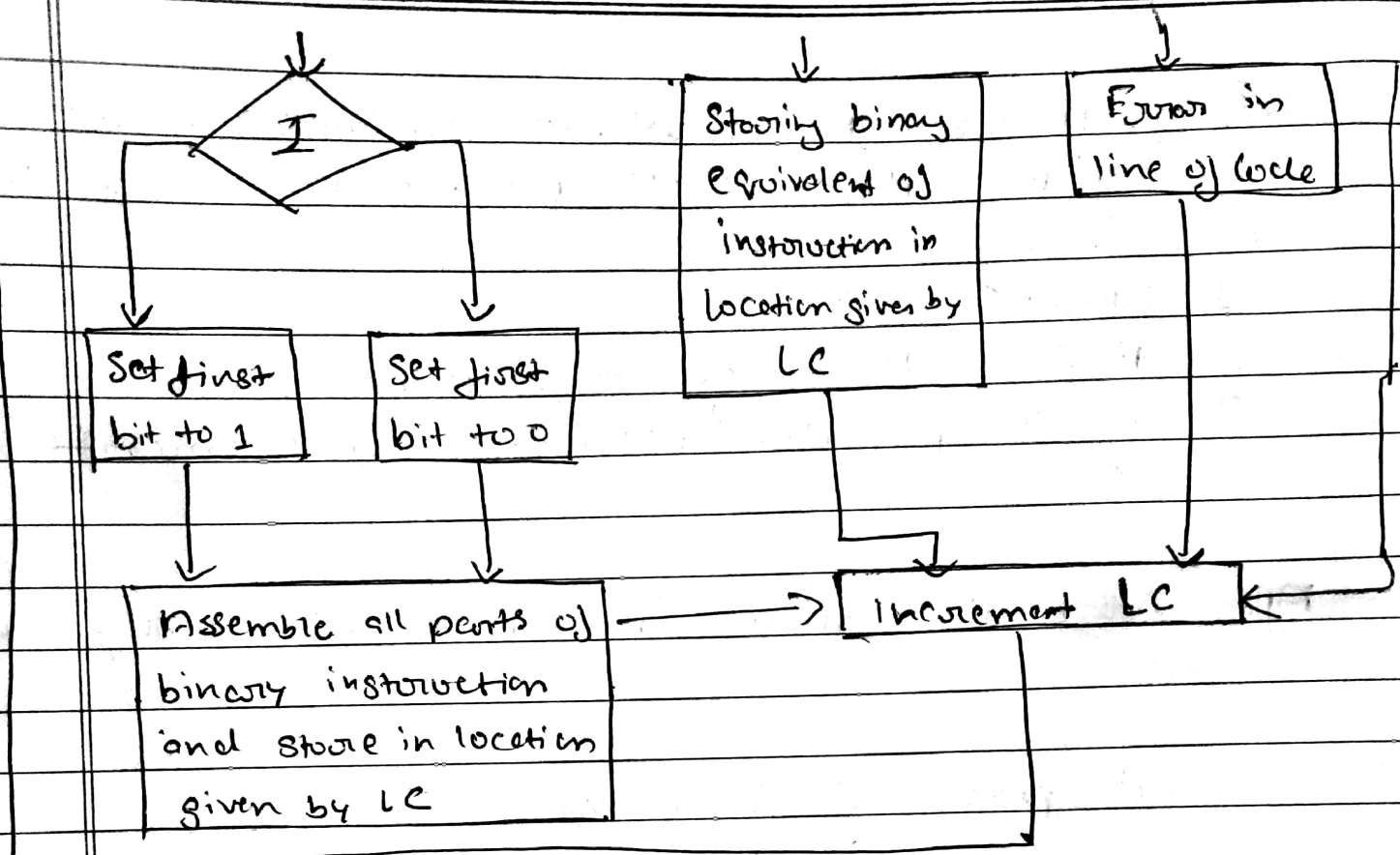
Step 3: If label found store symbole in symbole teble if not then check whether it is pseudo instruction.

1. ORG - Set LC
2. IF END then go to second pass
3. IF NO pseudo instruction increment the LC

After store the symbol increment LC and go to step 2.

→ Second pass of Assembler





- Second Pass works on a concept of table lookup by using all available tables it will translate our program to binary.

→ List of tables available to Assembler in second pass:
- Pseudo Instruction
- Symbol table
- MRI table
- Non-MRI table

- Step 1: Location Counter is initialized.
Step 2: Scan line
Step 3: Check if it is pseudo instruction
→ If yes
1. ORG - Set location Counter
2. END - End the program
3. DEC or HEX - Convert operand to binary and Store in location given by LC

→ If not

1. Check for MRI

If yes

1. Get OPCode
2. Get Address from Symbol table
3. Check if I is direct or indirect
4. Assemble everything
5. Increment LC and go to Step 2.

If

2. Check for valid MRI instruction

1. Translate
2.
3. If not MRI it will generate a error code
4. Increment LC and go to Step 2.

- Whenver the HLT instruction is encounter the program is stop.

* Program Loops

A program loop is a sequence of instruction that are executed many times, each time with a different set of data.

Eg –

```
For (i=0 ; i<=5 ; i++)
{
    printf ("Enter no");
    Scanf ("%d", &a[i]);
}
```

⇒ Symbolic Program to Add 100 Numbers

```
Line
 1          ORG 100       / origin of program is HEX 100
 2          LDA ADS       / Load first address of operands
 3          STA PTR       / Store in pointer
 4          LDA NBR       / Load minus 100
 5          STA CTR       / Store in counter
 6          CLA           / Clear Accumulator
 7  LOP,    ADD PTR I     / Add an operand to AC
 8          ISZ PTR       / Increment pointer
 9          ISZ CTR       / Increment counter
10          BUN LOP       / Repeat loop again
11          STA SUM       / Store Sum
12          HLT           / Halt
13  ADS,    HEX 150       / First address of operands
14  PTR,    HEX 0         / This location reserved for a pointer
15  NBR,    DEC -100      / Constant to initialized counter
16  CTR,    HEX 0         / This location reserved for a counter
17  SUM,    HEX 0         / Sum is stored here
```

✱ Subroutines

```
Locations
                ORG 100     / Main Program
100             LDA X       / Load X
101             BSA SH4     / Branch to Subroutine
102             STA X       / Store Shifted Number
103             LDA Y       / Load Y
104             BSA SH4     / Branch to Subroutine again
105             STA Y       / Store Shifted number
106             HLT
107     X,      HEX 1234
```

```
108   Y,   HEX 4321
                    / Subroutine to shift left 4 times
109   SH4, HEX 0    / Store return address here
10A        CIL      / Circulate left once
10B        CIL
10C        CIL
10D        CIL      / Circulate left fourth time
10E        AND MSIC / Set AC (13-16) to zero
10F        BUN SH4I / Return to main program
110   MSIC HEX FFF0 / Mask operand
           END
```

Subroutines means the same peace of code / line of code written over again in the program at different parts. ~~In Asp set~~ Instead repeating the code everytime it is needed, there is an ~~avd~~ advantage if it is return only 1's. The set of common instruction can be used many time in a program.

If we compare program loop and subroutine then program loop is set of instruction repeated many of times over different input. Program loop is within the program and will not alter / change the ~~g~~ program flow.

Why Subroutine is a set of instruction that can be repeated many time in a program (function call) but the body of subroutine is outside the main program.

In Subroutine the concept of branch (Conditional or

unconditional) is used. which will alter the flow of program.

- Each time that a subroutine is used in main part of a program the branch instruction is executed that the being and after the execution the branch is returned ~~the~~ to the main program.
- After subroutine execution when the address is available the same subroutine may make many branches to the same subroutine.
- Therefore the saving of return address is a common operation for subroutine. BSA and BUN - (Branch and Save return address & and Branch unconditionally)

* Arithmetic and logic Operation.