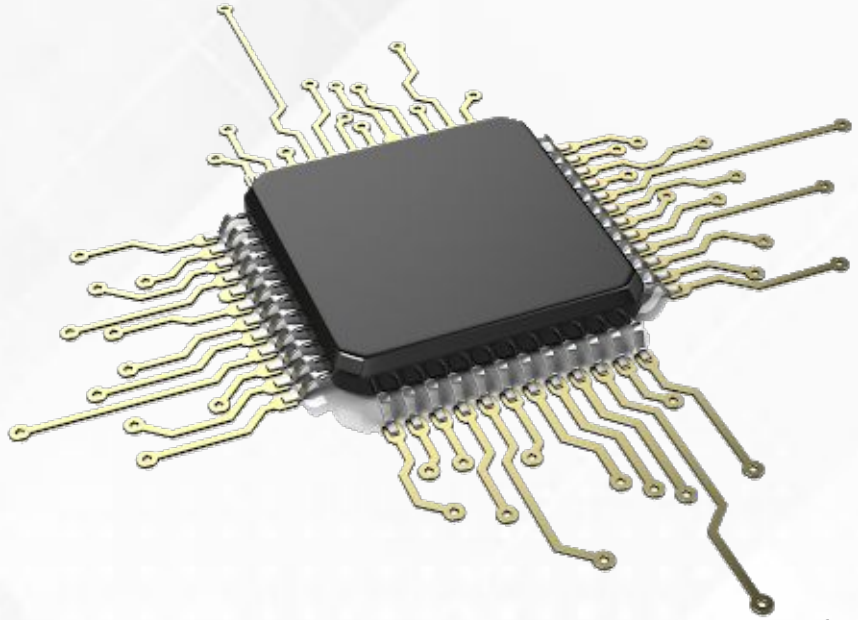
Computer Organization & Architecture
(COA)
GTU # 3140707

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Unit-3

Programming the Basic Computer



Prof. Krunal D. Vyas
Computer Engineering Department
Darshan Institute of Engineering & Technology,
Rajkot
krunal.vyas@darshan.ac.in
9601901005

Outline

- Machine Language
- Assembly Language
- Assembler
- Program loops
- Programming Arithmetic and Logic operations
- Subroutines
- I-O Programming
- Questions asked in GTU exam

Machine Language

Section - 1

Categories of programs

Binary code

- This is a sequence of instructions and operands in binary that list the exact representation of instructions as they appear in computer memory.

Location	Instruction Code			
0	0010	0000	0000	0100
1	0001	0000	0000	0101
10	0011	0000	0000	0110
11	0111	0000	0000	0001
100	0000	0000	0101	0011
101	1111	1111	1110	1001
110	0000	0000	0000	0000

Octal or hexadecimal code

- This is an equivalent translation of the binary code to octal or hexadecimal representation.

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

Categories of programs

- Symbolic code
  - The user employs *symbols* (letters, numerals, or special characters) for the operation part, the address part, and other parts of the instruction code.
  - Each symbolic instruction can be translated into one binary coded instruction by a special program called an assembler and language is referred to as an *assembly language program*.

Location	Instruction	Comment
000	LDA 004	Load first operand into AC
001	ADD 005	Add second operand to AC
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

- High-level programming languages
  - These are special languages developed to reflect the procedures used in the solution of a problem rather than be concerned with the computer hardware behavior. E.g. Fortran, C++, Java, etc.
  - The program is written in a sequence of statements in a form that people prefer to think in when solving a problem.
  - However, each statement must be translated into a sequence of binary instructions before the program can be executed in a computer.

```
INTEGER A, B, C
DATA A, 83 B,-23
C = A + B
END
```

Assembly Language

Section - 2

Pseudo Instruction

- A *pseudo instruction* is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation.

Symbol	Information for the Assembler
ORG N	Hexadecimal number N is the memory location for the instruction or operand listed in the following line.
END	Denotes the end of symbolic program.
DEC N	Signed decimal number N to be converted to binary.
HEX N	Hexadecimal number N to be converted to binary

Assembler

Section - 3

Assembler

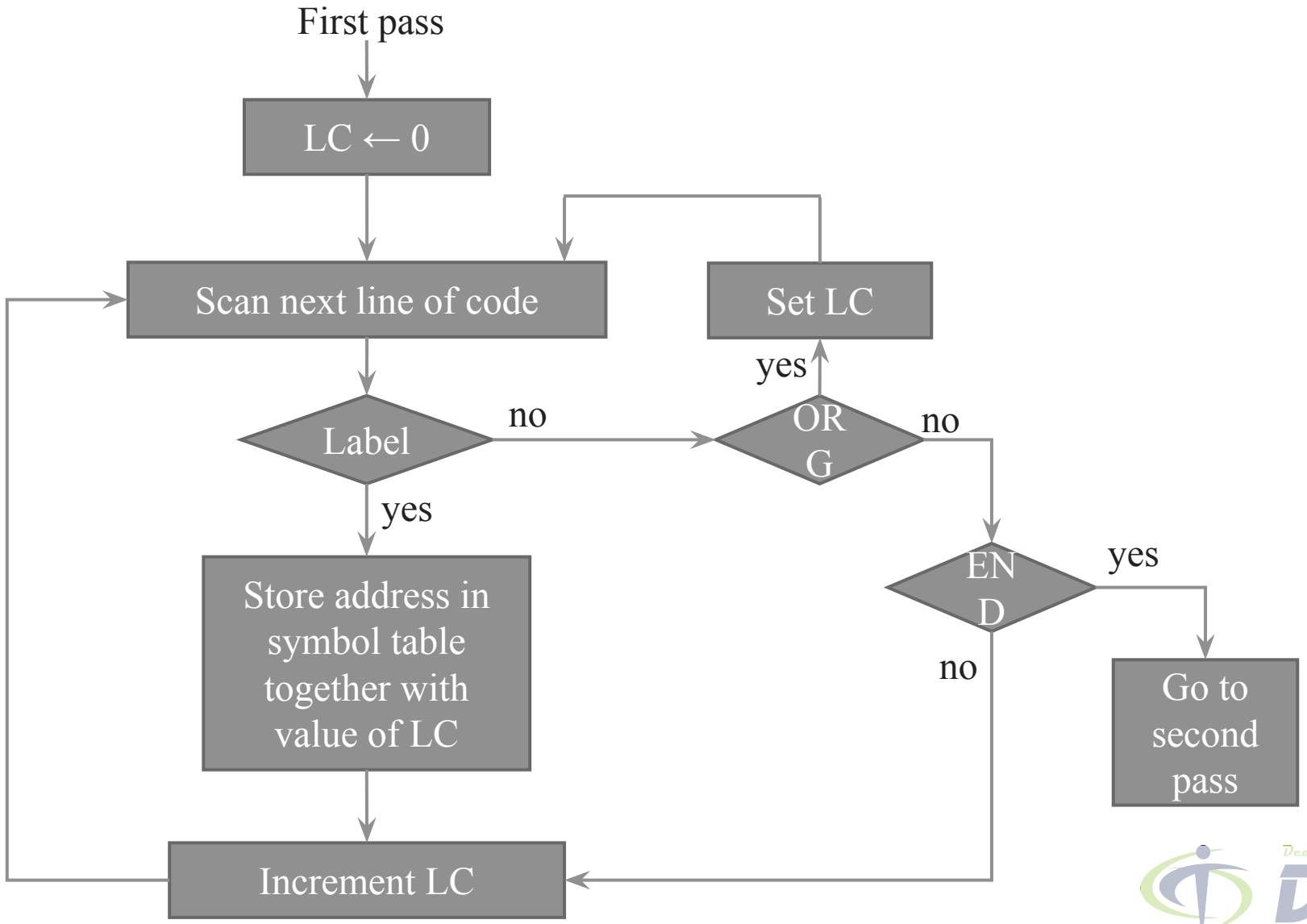
- An *assembler* is a program that accepts a symbolic language program and produces its binary machine language equivalent.
- The input symbolic program is called the source program and the resulting binary program is called the object program.
- The assembler is a program that operates on character strings and produces an equivalent binary interpretation.

A.L.P. to subtract 2 numbers

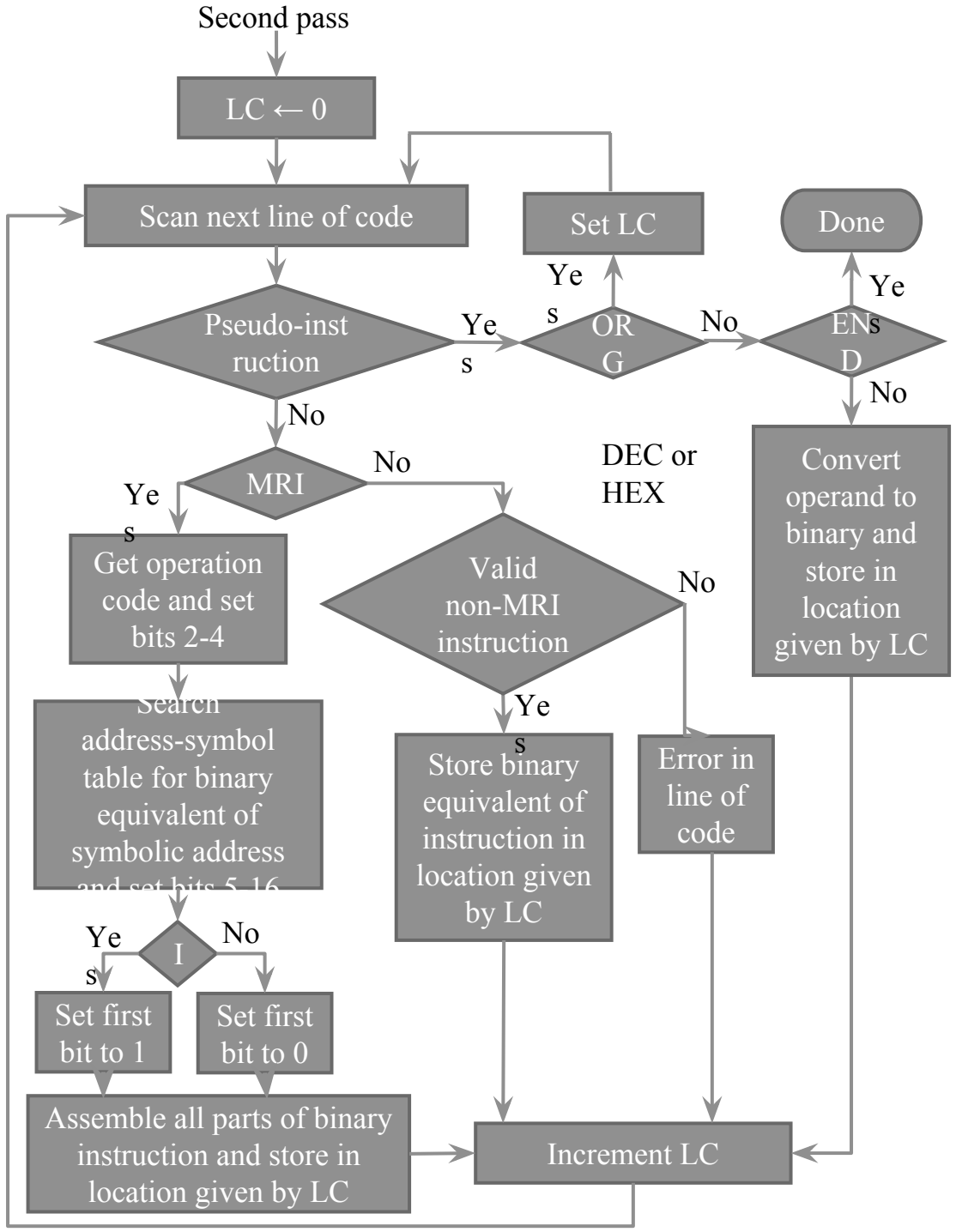
Location		Instruction
		ORG 100
100		LDA SUB
101		CMA
102		INC
103		ADD MIN
104		STA DIF
105		HLT
106	MIN,	DEC 83
107	SUB,	DEC -23
108	DIF,	HEX 0
		END

Symbol	Location
MIN	106
SUB	107
DIF	108

First Pass of an assembler



Second Pass of an assembler



Program loops

Section - 4

Program Loops

- A *program loop* is a sequence of instructions that are executed many times, each time with a different set of data.
- A system program that translates a program written in a high-level programming language to a machine language program is called a *compiler*.

A.L.P. to add 100 numbers

```
1         ORG   100        /Origin of program is HEX
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 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 store here
18        ORG   150        /Origin of operands is HEX
19        DEC   75         /First operand
 •
 •
 •
118       DEC   23         /Last operand
119       END               /End of symbolic program
```

A.L.P. to clear the contents of hex locations 500 to 5FF with 0

```
1        ORG  100      /Origin of program is HEX
2        LDA  ADS      /Load first address of operands
3        STA  PTR      /Store in pointer
4        LDA  NBR      /Load minus 255
5        STA  CTR      /Clear accumulator
6        CLA
7   LOP, STA  PTR      /Store zero to location pointed by PTR
8        ISZ  PTR      /Increment pointer
9        ISZ  CTR      /Increment counter
10       BUN  LOP      /Repeat loop again
11       HLT           /Halt
12  ADS, HEX  500      /First address of operands
13  PTR, HEX  0        /This location reserved for pointer
14  NBR, DEC  -255     /Constant to initialized counter
15  CTR, HEX  0        /This location reserved for a counter
16       END
```

Programming Arithmetic and Logic operations

Section - 5

A.L.P. to Add Two Double-Precision Numbers

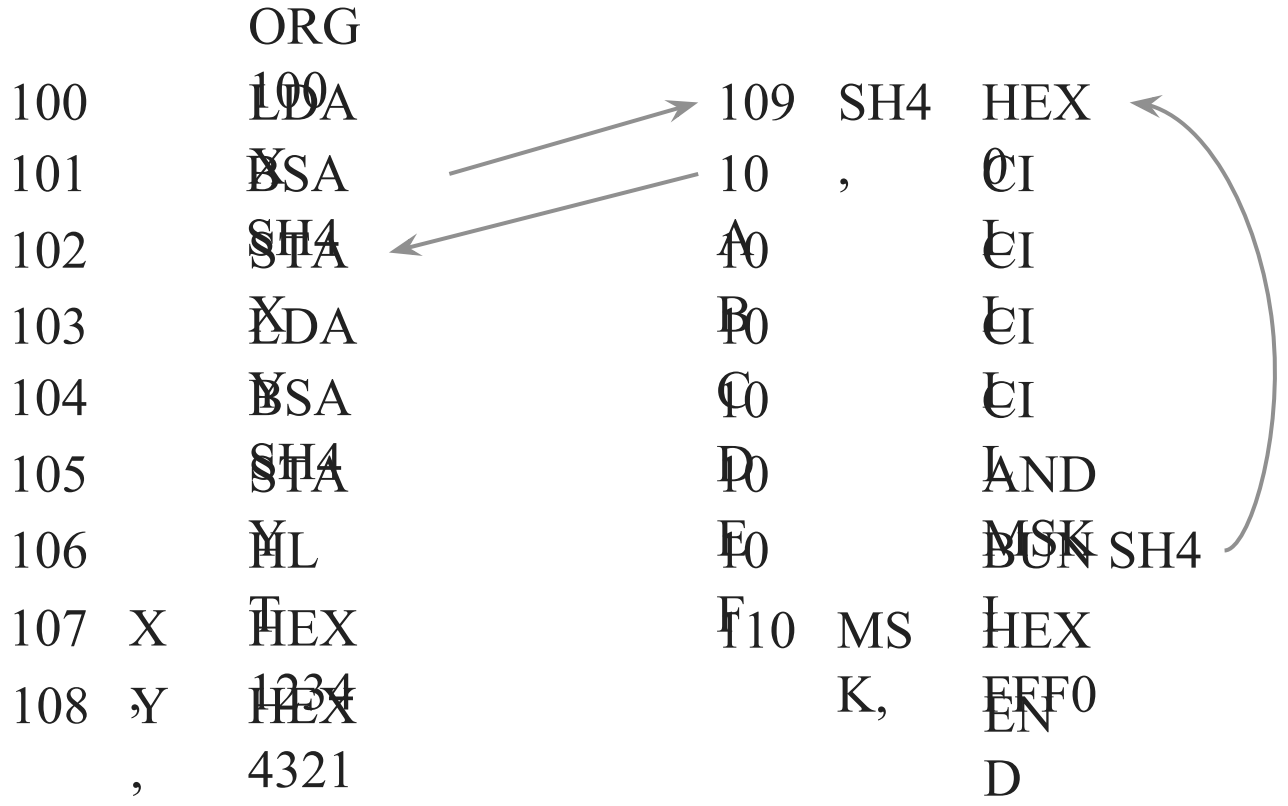
```
1     ORG 100          /Origin of program is HEX 100
2     LDA AL           /Load A low
3     ADD BL           /Add B low, carry in E
4     STA CL           /Store in C low
5     CLE              /Clear E
6     CIR              /Circulate to bring carry into AC(16)
7     ADD AH           /Add A high and carry
8     ADD BH           /Add B high
9     STA CH           /Store in C high
10    HLT              /Halt
```

Subroutines

Section - 6

Subroutine with example

- A set of common instructions that can be used in a program many times is called a *subroutine*.
- Each time that a subroutine is used in the main part of the program, a branch is executed to the beginning of the subroutine.
- After the subroutine has been executed, a branch is made back to the main program.
- A subroutine consists of a self contained sequence of instructions that carries a given task.



I-O Programming

Section - 7

A.L.P. to input one character & output one character

Input Program

```
1         ORG  100        /Origin of program is HEX
2  CIF,   SKI  INF        /Check input flag
3         BUN  CIF        /Flag = 0, branch to check again
4         INP             /Flag = 1, input character
5         OUT             /Print character
6         STA  CHR        /Store character
7         HLT
8  CHR,   DEC  0          /Store character here
9         END
```

Output Program

```
1         ORG  100        /Origin of program is HEX
2         LDA  CHR        /Load character into AC
3  COF,   SKO             /Check output flag
4         BUN  COF        /Flag = 0, branch to check again
5         OUT             /Flag = 1, output character
6         HLT
7  CHR,   HEX  0057       /Character is "W"
8         END
```

Questions asked in GTU exam

Section - 8

Questions asked in GTU exam

1. What is an Assembler? With clear flowcharts for first and second pass, explain its working.
2. Write an assembly language program to add 10 numbers from memory.
3. Write a brief note on: Subroutine call and return.
4. Write an ALP for multiplying 3 integers stored in register stack.
5. Write an assembly program to multiply two positive numbers.
6. What is machine language? How it differs from assembly language?
7. Define pseudo-instruction.
8. For the following C language code, write assembly language program:

```
int a, b, c;
a = 83;  //plus 83
b = -23; //minus 23
c = a + b;
```