Computer System Architecture

Ms. Yuzana Hlaing
M.E(IT), Ph.D. Candidate(thesis)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Periods	Lectures
Week-1	Introduction to Computer System Architecture
Week-2	Data Presentations
Week-3	Register Transfer and Microoperations
Week-4	Basic Computer Organization and Design
Week-5	Programming the Basic Computer
Week-6	Microprogrammed Control
Week-7	Central Processing Unit
Week-8	Pipeline and Vector Processing
Week-9	Computer Arithmetic
Week-10	Input-Output Organization
Week-11	Memory Organization
Week-12	Multiprocessors

Lecture-5

Programming the Basic Computer

Lecture-5
Programming the Basic Computer System

- What is Machine Language?
- What is Assembly language?
- The Assembler
- Program Loops
- Programming Arithmetic and Logic Operations
- Input-Output Programming

What is Machine Language?

What is Machine Language?

- A **program** is a list of instructions or statements for directing the computer to perform a required data-processing task.
- There are various types of programming languages written for computers, but the computer can execute programs only when they are internally represented in the binary form.
- Programs written for a computer may be one of the following categories
    1. Binary code
    2. Octal or hexadecimal code
    3. Symbolic code
    4. High-level programming languages
- A **machine language** program is a binary program in which binary code is used.

- A **Binary code** is a sequence of instructions and operands in binary.
- A **Octal or hexadecimal code** is an equivalent translation of the binary code to octal or hexadecimal representation.
- A **Symbolic code** defines that 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.
- This translation is done by a special program called **assembler**.
- **High-level programming languages** are special languages developed to reflect the procedures used in the solution of a problem.
- Fortran, an example of a high-level programming language, employs problem-oriented symbols and formats.
- A program that translates a high-level program to binary is called a **compiler**.

What is Assembly Language?

What is Assembly Language?

- A programming language is defined by a set of rules.
- Every commercial computer may have its own particular assembly language.
- The basic unit of an assembly language program is a line of code.
- The rules of an assembly language are formulated for writing symbolic programs for the basic computer.
- Each line of an assembly language program is arranged in three columns called fields.
- The fields specify the following information.
    1. The **label field** may be empty or it may specify a symbolic address.
    2. The **instruction field** specifies a machine instruction or a pseudo-instruction.
    3. The **comment field** may be empty or it may include a comment.

- A **symbolic address** consists of one, two, or three, but not more than three alphanumeric characters.
- A symbolic address in the label field is terminated by a comma so that it will be recognized as a label by the assembler.
- A **instruction field** in an assembly language program may specify one of the following items.
  1. A memory-reference instruction (MRI)
  2. A register-reference or input-output instruction (non-MRI)
  3. A pseudo-instruction with or without an operand
- A **memory-reference instruction** occupies two or three symbols separated by spaces.
- A **non-MRI** is defined as an instruction that does not have an address part.
- A **pseudo-instruction** is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation.

Translation to Binary

- The translation of the symbolic program into binary is done by a special program called an assembler.
- The translation of the symbolic program into an equivalent binary code may be done by scanning the program and replacing the symbols by their machine code binary equivalent.
- The translation process can be simplified if the entire symbolic program is scanned twice by the assembler.
- The first scan is called the first pass and the second is called the second pass.
- In the first pass, a memory location is assigned for each machine instruction and operand.
- During the second scan, the address value of labels is defined and the translation process is facilitated.

***Write the assembly language program for the subtraction of two numbers and then write the translated program to binary for this program.

Sol:

Assembly language program to subtract two numbers

Labels	Instructions	Comments
	ORG 100	/Origin of program is location 100
	LDA SUB	/Load subtrahend to AC
	CMA	/Complement AC
	INC	/Increment AC
	ADD MIN	/Add minuend to AC
	STA DIF	/Store difference
	HLT	/Halt computer
MIN,	DEC 83	/Minuend
SUB,	DEC -23	/Subtrahend
DIF,	HEX 0	/Difference source here
	END	/End of symbolic program

Translated program to binary for subtraction of two numbers

Location	Content	Symbolic program	
			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 Assembler

The 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**.
- 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.

Representation of Symbolic Program in Memory

- As the starting stage of the assembly process, the symbolic program must be stored in memory.
- A loader program is used to input the characters of the symbolic program into memory.
- The CR code is produced in order to be recognized by an assembler as the end of a line of code.
- A line of code is stored in consecutive memory locations with two-characters in each location.
- For example, the following line of code is stored in seven consecutive memory locations.

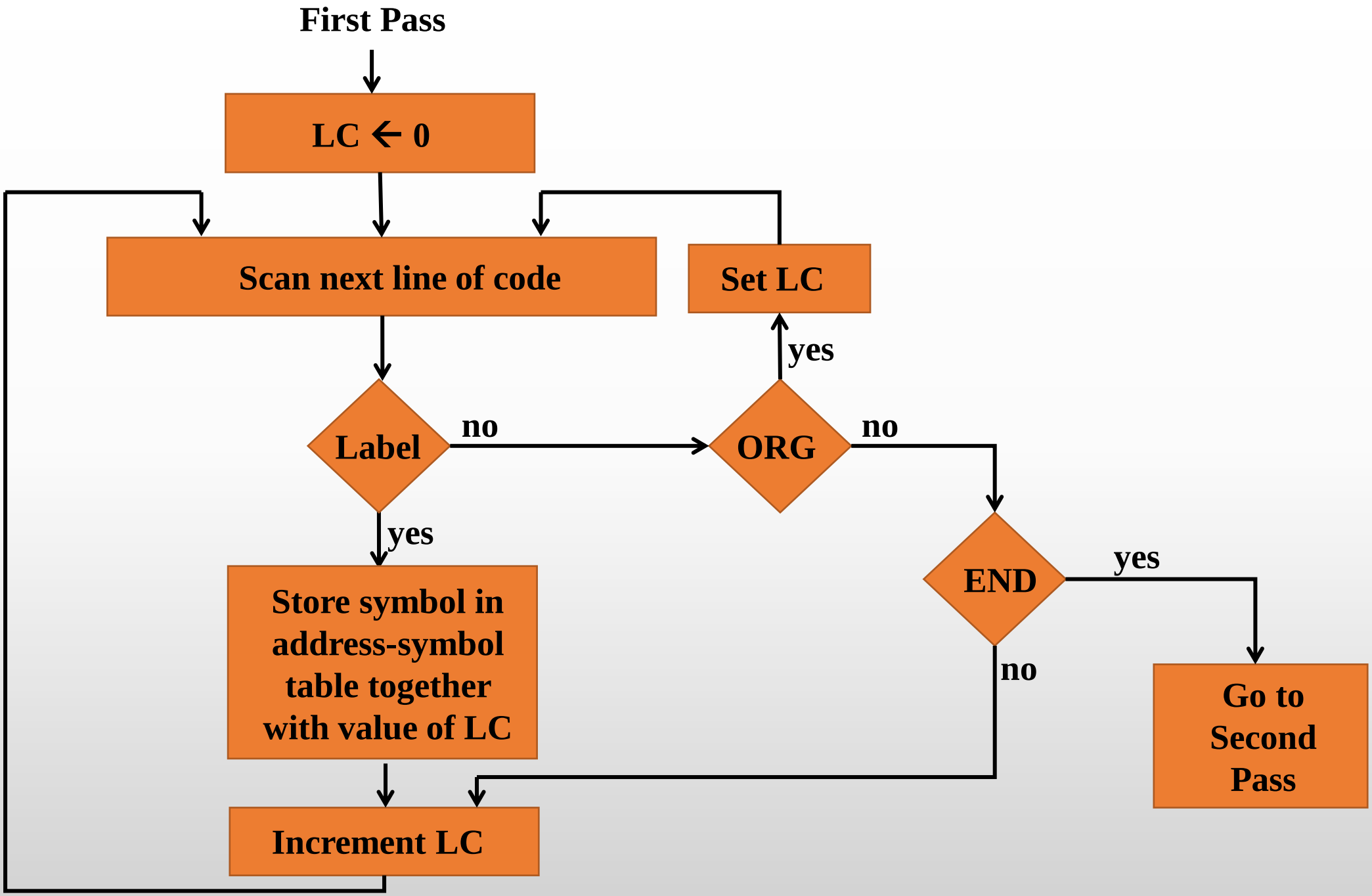
PL3, LDA SUB I

Computer Representation of the Line of Code: PL3 LDA SUB I

Memory word	Symbol	Hexadecimal code	Binary Representation
1	P L	50 4C	0101 0000 0100 1100
2	3 ,	33 2C	0011 0011 0010 1100
3	L D	4C 44	0100 1100 0100 0100
4	A	41 20	0100 0001 0010 0000
5	S U	53 55	0101 0011 0101 0101
6	B	42 20	0100 0010 0010 0000
7	I CR	49 0D	0100 1001 0000 1101

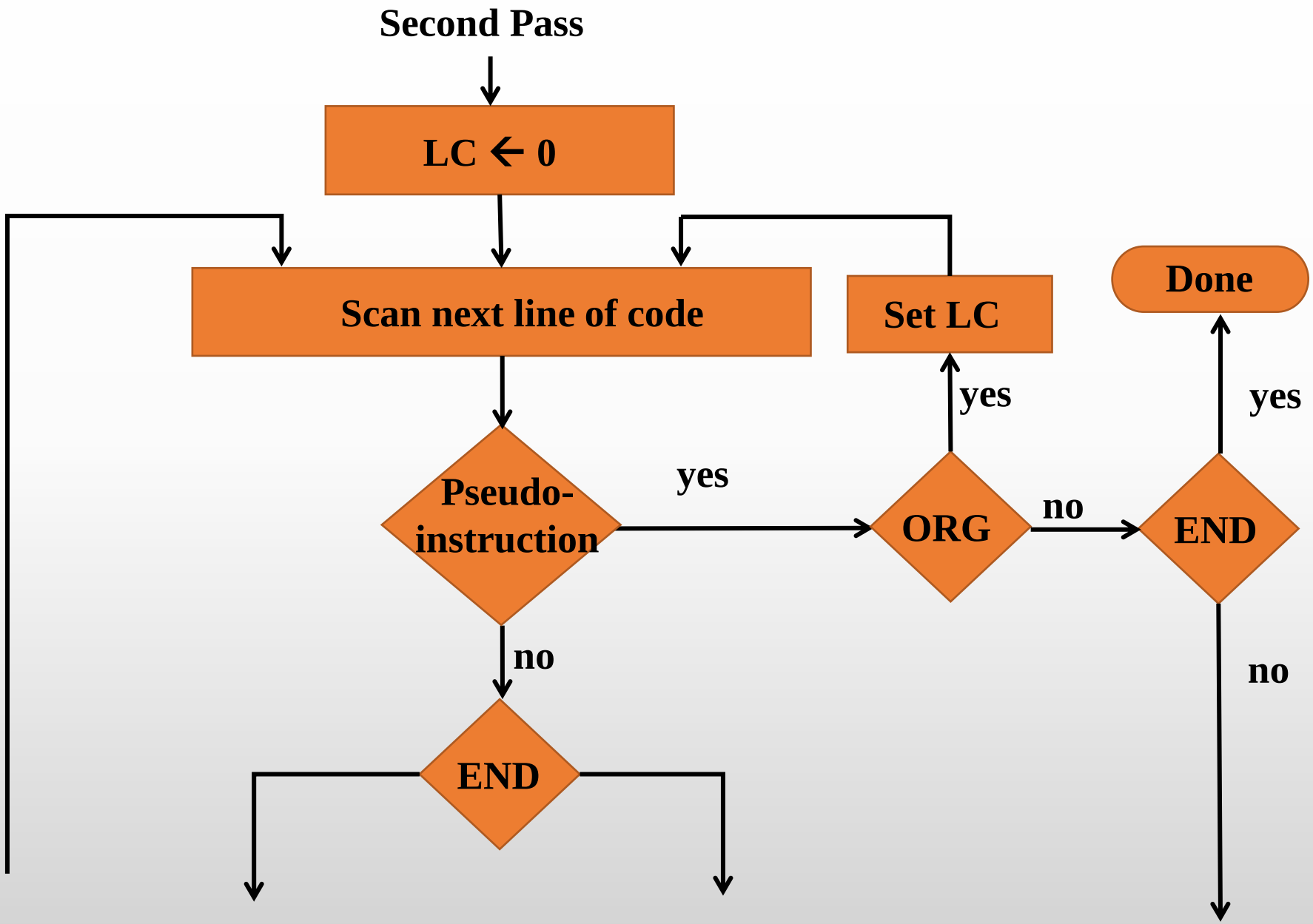
First Pass

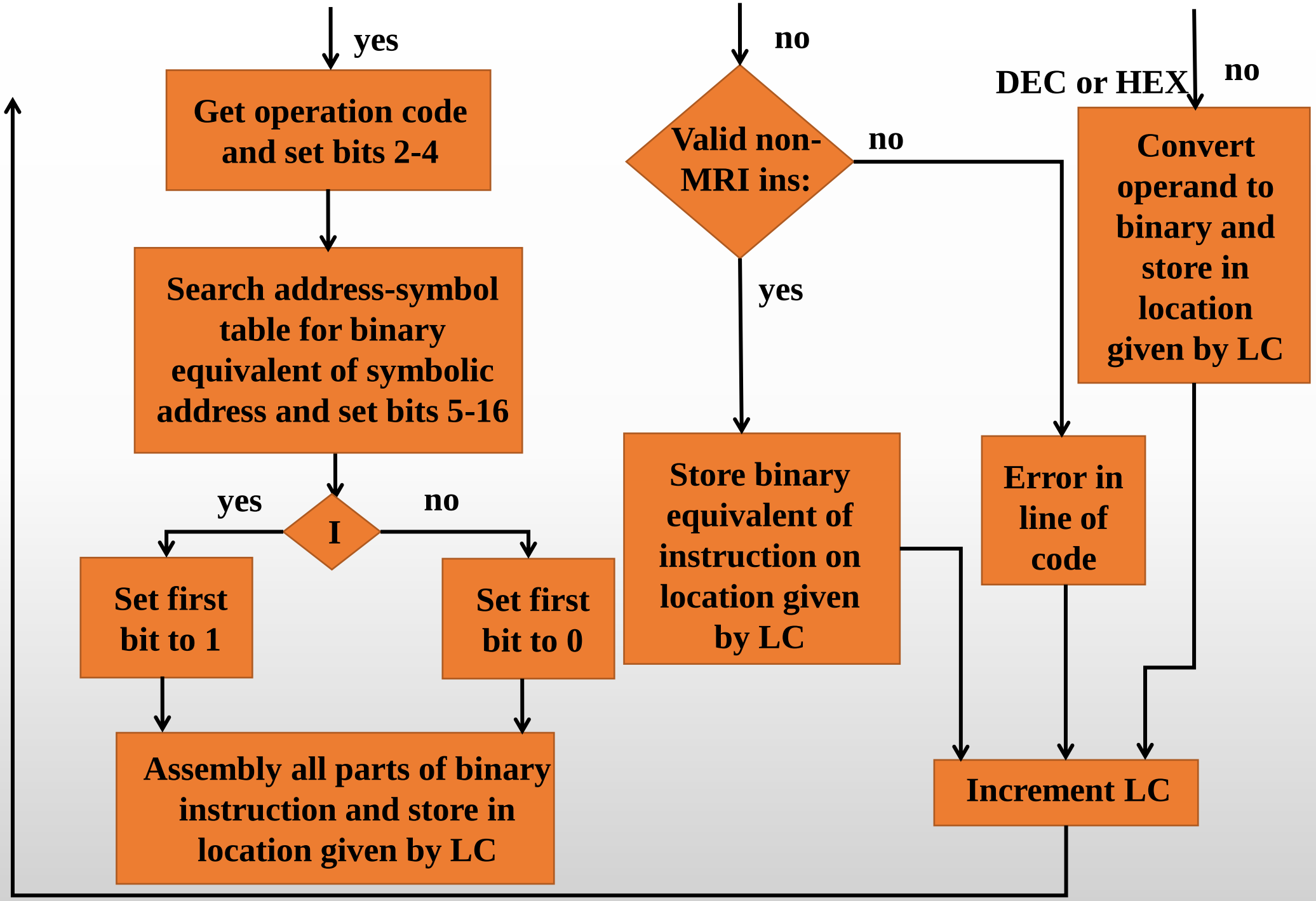
- A two-pass assembler scans the entire symbolic program twice.
- During the first-pass, it generates the table that correlates all user-defined address symbols with their binary equivalent value.
- The assembler uses a memory word called a **location counter (LC)** to keep track of the location of the instructions.
- The content of LC stores the value of the memory location assigned to the instruction or operand presently being processed.
- The ORG pseudo-instruction initializes the LC to the value of the first location.
- The tasks performed by the assembler during the first pass are described in the flowchart.


First Pass
LC ← 0
Scan next line of code
Set LC
yes
Label
no
ORG
no
yes
Store symbol in address-symbol table together with value of LC
END
yes
no
Go to Second Pass
Increment LC

Second Pass

- Machine instructions are translated during the second pass by means of table-lookup procedure.
- A **table-lookup procedure** is a search of table entries to determine whether a specific item matches one of the items stored in the table.
- There are four tables that the assembler uses.
  1. Pseudo-instruction table
  2. MRI table
  3. Non-MRI table
  4. Address symbol table
- Any symbol that is encountered in the program must be available as an entry in one of these tables, otherwise, the symbols cannot be interpreted.


Second Pass
LC ← 0
Scan next line of code
Pseudo-instruction
yes
ORG
no
END
yes
Set LC
yes
Done
no
END
no


yes
Get operation code and set bits 2-4
Search address-symbol table for binary equivalent of symbolic address and set bits 5-16
I
yes
no
Set first bit to 1
Set first bit to 0
Assembly all parts of binary instruction and store in location given by LC
no
Valid non-MRI ins:
no
yes
Store binary equivalent of instruction on location given by LC
DEC or HEX
no
Convert operand to binary and store in location given by LC
Error in line of code
Increment LC

Program Loops

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 compiler is a more complicated program than an assembler and requires knowledge of systems programming to fully understand its operation.
- A compiler may use an assembly language as an intermediate step in the translation or may translate the program directly to binary.

***Write a Fortran program for the addition of 100 integer numbers and then convert assembly language using compiler.

Sol:

Fortran Program for 100 integers addition,

```
DIMENSION A(100)
INTEGER SUM,  A
SUM = 0
DO    J = 1, 100
SUM = SUM +A (J)
```

Symbolic Program to Add 100 Integers

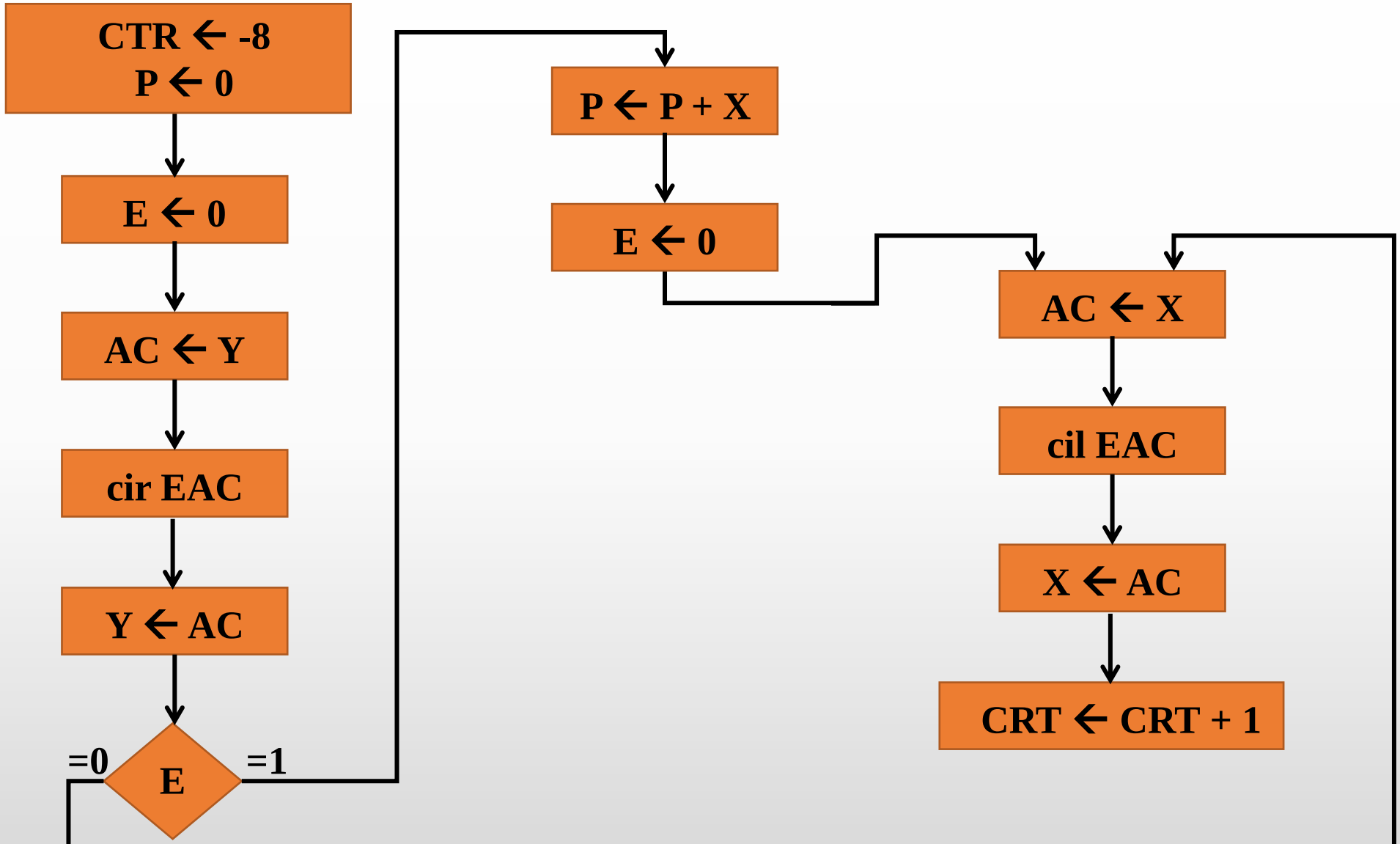
Line			
1		ORG 100	/Origin of program is HEX 100
2		LDA ADS	/Load first address of operands
3		STA PTR	/Store in pointer
4		LDA NBR	/Load minus 100
5		STA CTR	/Store in counter
6		CLA	/Clear accumulator
7	LOP,	ADD PTR I	/Add an operand to AC
8		ISZ PTR	/Increment pointer
9		ISZ CTR	/Increment counter
10		BUN LOP	/Repeat loop again
11		STA SUM	/Store sum
12		HLT	/Halt

Line			
13	ADS,	HEX 150	/First address of operands
14	PTR,	HEX 0	/This location reserved for a pointer
15	NBR,	DEC -100	/Constant to initialize counter
16	CTR,	HEX 0	/This location reserved for a counter
17	SUM,	HEX 0	/Sum is stored here
18		ORG 150	/Origin of operands is HEX 150
19		DEC 75	/First operand
…			
…			
…			
118		DEC 23	/Last operand
119		END	/End of symbolic program

Program Arithmetic and Logic Operations

Multiplication Program

- To consider the multiplication process traditionally, it consists of checking the bits of multiplier Y and adding the multiplicand X as many times as there are 1's in Y, provided that the value of X is shifted left from one line to the next.
- In computer, the intermediate sums are called partial products that are stored in a memory location called P because they hold a partial product until all numbers are added.
- The final value in P forms the product.
- The final product may be at most twice of the number of significant bits of multiplier.
- The step-by-step procedure for programming the multiplication operation is described in the flowchart.


CTR ← -8
P ← 0
E ← 0
AC ← Y
cir EAC
Y ← AC
=0
E
=1
P ← P + X
E ← 0
AC ← X
cil EAC
X ← AC
CRT ← CRT + 1

***Write an assembly language program for the multiplication of two positive 8-bit numbers.

Sol:

Assembly Program for Multiplication

Labels	Instructions	Comments
	ORG 100	/Origin of program is location 100
LOP,	CLE	/Clear E
	LDA Y	/Load multiplier
	CIR	/Transfer multiplier bit to E
	STA Y	/Store shifted multiplier
	SZE	/Check if bit is zero
	BUN ONE	/Bit is one; go to ONE
	BUN ZERO	/Bit is zero; go to ZRO
ONE,	LDA X	/Load multiplicand
	ADD P	/Add to partial product

Labels	Instructions	Comments
	STA P	/Store partial product
	CLE	/Clear E
ZRO,	LDA X	/Load multiplicand
	CIL	/Shift left
	STA X	/Store shifted multiplicand
	ISZ CTR	/Increment counter
	BUN LOP	/Counter not zero; repeat loop
	HLT	/Counter is zero; halt
CTR,	DEC -8	/This location serves as a counter
X,	HEX 000F	/Multiplicand stored here
Y,	HEX 000B	/Multiplier stored here
P,	HEX 0	/Product formed here
	END	

Double-Precision Addition

- A number stored in two memory words is said to have **double precision**.
- The following program is for two double-precision numbers addition.

```
        LDA AL      /Load A low
        ADD BL      /Add B low, carry in E
        STA CL      /Store in C low
        CLA         /Clear AC
        CIL         /Circulate to binary carry into AC (16)
        ADD AH      /Add A high and carry
        ADD BH      /Add B high
        STA CH      /Store in C high
        HLT
AL,     __          /Location of operands
AH,     __
BL,     __
BH,     __
CL,     __
CH,     __
```

Logic Operations

- There are three machine instructions in basic computer that perform logic operation: AND, CMA, and CLA.
- All 16 logic operations can be implemented by software means because any logic function can be implemented using the AND and complement operations.
- The OR operation can be performed by using the following program.

Instructions	Comments
LDA A	/Load first operand A
CMA	/Complement to get A’
STA TMP	/Store in a temporary location
LDA B	/Load second operand B
CMA	/Complement to get B’
AND TMP	/AND with A’ to get A’^ B’
CMA	/Complement again to get A V B

Shift Operations

- The circular-shift operations are machine operations in the basic computer.
- The two instructions necessary for a logical shift-right operation are
  CLE
  CIR
- The two instructions required for a logical shift-left operation are
  CLE
  CIL
- The program for the arithmetic shift-right is described in the following.

Instructions	Comments
CLE	/Clear E to 0
SPA	/Skip if AC is positive; E remains 0
CME	/AC is negative; set E to 1
CIR	/Circulate E and AC

Input-Output Programming

Input-Output Programming

- The symbols are strings of characters and each character is assigned an 8-bit code so that it can be stored in computer memory.
- A binary-coded character enters the computer when an INP (input) instruction is executed.
- A binary-coded character is transferred to the output device when an OUT (output) instruction is executed.
- The SKI instruction checks the input flag to see if a character is available for transfer.
- The required instructions for input and output one character are described in the following.

Program to Input One Character

Labels	Instructions	Comments
(a) Input a character		
CIF,	SKI	/Check input flag
	BUN CIF	/Flag=0, branch to check again
	INP	/Flag=1, input character
	OUT	/Print character
	STA CHR	/Store character
	HLT	
CHR,	__	/Store character here

Program to Output One Character

Labels	Instructions	Comments
(a) Output one character		
CIF,	LDA CHR	/Load character into AC
COF,	SKO	/Check output flag
	BUN COF	/Flag=0, branch to check again
	OUT	/Flag=1, output character
	HLT	
CHR,	HEX 0057	/Character is “W”

Character Manipulation

- The binary-coded characters that represent symbols can be manipulated by computer instructions to achieve various data-processing tasks.
- One of these tasks may be to pack two characters in one word.
- The process of input and pack two characters is written in the following program.

```
IN2,    __          /Subroutine entry
FST,    SKI
        BUN FST
        INP         /Input first character
        OUT
        BSA SH4     /Shift left four times
        BSA SH4     /Shift left four more times
SCD,    SKI
        BUN SCD
        INP         /Input second character
        OUT
        BUN IN2 I   /Return
```

***Write a symbolic program for storing input characters in a buffer.

Sol:

The following program can be used to input characters (symbolic program) from the keyboard, pack two characters in one word, and store them in the buffer.

Labels	**Instructions**	**Comments**
	LDA ADS	/Load first address of buffer
	STA PTR	/Initialize pointer
LOP,	BSA IN2	/Go to subroutine IN2
	STA PTR I	/Store double character word in buffer
	ISZ PTR	/Increment pointer
	BUN LOP	/Branch to input more characters
	HLT	
ADS,	HEX 500	/First address of buffer
PTR,	HEX 0	/Location for pointer

Program Interrupt

- The running time of input and output program is made up of the time spent by the computer in waiting for the external device to set its flag.
- The waiting time of external device can be eliminated if the interrupt facility is used to notify the computer when a flag is set.
- The advantage of using the interrupt is that the information transfer is initiated upon request from the external device.
- The running program must include an ION instruction to turn the interrupt on.
- If the interrupt facility is not used, the program must include an IOF instruction to turn it off.
- The interrupt facility allows the running program to proceed until the input or output device sets its ready flag.

Interrupt Service Routine

- The service routine required for the input or output transfer can be stored anywhere in memory provided a branch to the start of the routine.
- The service routine must have instructions to perform the following tasks:
  1. Save contents of processor registers.
  2. Check which flag is set.
  3. Service the device whose flag is set.
  4. Restore contents of processor registers.
  5. Turn the interrupt facility on.
  6. Return to the running program.
- The service routine must turn the interrupt on before the return to the running program.

***Write a symbolic program to compare two words required for interrupt routine.
Sol:

Program to Compare Two Words

Labels	Instructions	Comments
	LDA WD1	/Load first WORD
	CMA	
	INC	/Form 2's complement
	ADD WD2	/Add second word
	SZA	/Skip if AC is zero
	BUN UEQ	/Branch to "unequal" routine
	BUN EQL	/Branch to "equal" routine
WD1,	__	
WD2,	__	

***Write a symbolic program to service an interrupt routine.

Sol:

Program to Service an Interrupt

Location			
0	ZRO,	__	/Return address stored here
1		BUN SRV	/Branch to service routine
100		CLA	/Portion of running program
101		ION	/Turn on interrupt facility
102		LDA X	
103		ADD Y	/Interrupt occurs here
104		STA Z	/Program returns here after interrupt
…		…	
…		…	
…			/Interrupt service routine

Program to Service an Interrupt (Cont.)

Location			
200	SRV,	STA SAC	/Store content of AC
		CIR	/Move E into AC(1)
		STA SE	/Store content of E
		SKI	/Check input flag
		BUN NXT	/Flag is off; check next flag
		INP	/Flag is on; input character
		OUT	/Print character
		STA PT1 I	/Store it in input buffer
		ISZ PT1	/Increment input pointer
	NXT,	SKO	/Check output flag
		BUN EXT	/Flag is off, exit
		LDA PT2 I	/Load character from output buffer
		OUT	/Output character

Program to Service an Interrupt (Cont.)

Location			
		ISZ PT2	/Increment output pointer
	EXT,	LDA SE	/Restore value of AC (1)
		CIL	/Shift it to E
		LDA SAC	/Restore content of AC
		ION	/Turn interrupt on
		BUN ZRO I	/Return to running program
		__	/AC is stored here
		__	/E is stored here
		__	/Pointer of input buffer
		__	/Pointer of output buffer

Thank You

Lecture for Next Week

- **Microprogrammed Control**
  - **Control Memory**
  - **Address Sequencing**
  - **Microprogram Example**
  - **Design of Control Unit**