Unit-3
Programming the Basic Computer

Marwadi University

Department of
Computer Engineering

Computer
Organization and
Architecture
01CE1402

Prof. Kishan Makadiya

Outline

- Introduction
- Machine Language
- Assembly Language
- Assembler
- Program loops
- Programming Arithmetic and Logic operations
- Subroutines
- I-O Programming

Introduction

Symbol	Hexa code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to m+1
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC, carry to E
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

Machine Language

- Program list of instructions or statements for directing the computer to perform a required data processing task.
- Various types of programming languages

1. Binary code
2. Octal or hexadecimal code
3. Symbolic code

- Each symbolic instruction can be translated into one binary coded instruction. This translation is done by a special program called an Assembler.

4. High-level Programming language

- The Program that translates a high level language program to binary is called a compiler.

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.
- Example:-

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

Categories of programs

Octal or hexadecimal code

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

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

Categories of programs

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

- Rules of the Language
- 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.

Rules of the language

A Symbolic Address (in LABEL FIELD) consists of one, two or three, but not more than three alphanumeric characters.

- The first character must be a latter.
- Next two may be latter or numerals.
- The symbolic address is terminated by a comma so that it will be recognized by the assembler.

Rules of the language

The INSTRUCTION FIELD in program may specify one of the following items:

- A MRI (Memory Reference Instruction)
- A non-MRI (Register Ref. or I/O instructions)
- A pseudo instruction with or without operand.

- A MRI occupies two or three symbols separate by space.
- Example:

CLA	/non MRI
ADD OPR	/direct-address MRI
ADD PTR I	/Indirect-address MRI

Rules of the language

- The third field in the program is reserved for comments.
- It must be preceded by a slash (i.e. to recognize the beginning of a comment)

Assembly Language

- 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

Assembly Language Program (A.L.P.) to add 2 numbers

Assembly Programming Languages

```
        ORG   0     /Origin of program is 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
```

Assembly Language Program (A.L.P.) to subtract 2 numbers

Label	Instruction	Comment
	ORG 100	/ Origin of program is location 100
	LDA B	/ Load subtrahend to AC
	CMA	/ Complement AC
	INC	/ Increment AC
	ADD A	/ Add minuend to AC
	STA C	/ Store difference
	HLT	/ Halt computer
A,	DEC 83	/ Minuend
B,	DEC 23	/ Subtrahend
C,	HEX 0	/ Difference stored here
	END	/ End of symbolic program

Assembly Language Program (A.L.P.) to subtract 2 numbers

Location	Label	Instruction
		ORG 100
100		LDA B
101		CMA
102		INC
103		ADD A
104		STA C
105		HLT
106	A,	DEC 83
107	B,	DEC 23
108	C,	HEX 0
		END

Symbol	Location
A	106
B	107
C	108

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.
- To keep track of the location of instructions, the assemble uses a memory word called location counter (LC).

Hexadecimal Character Code

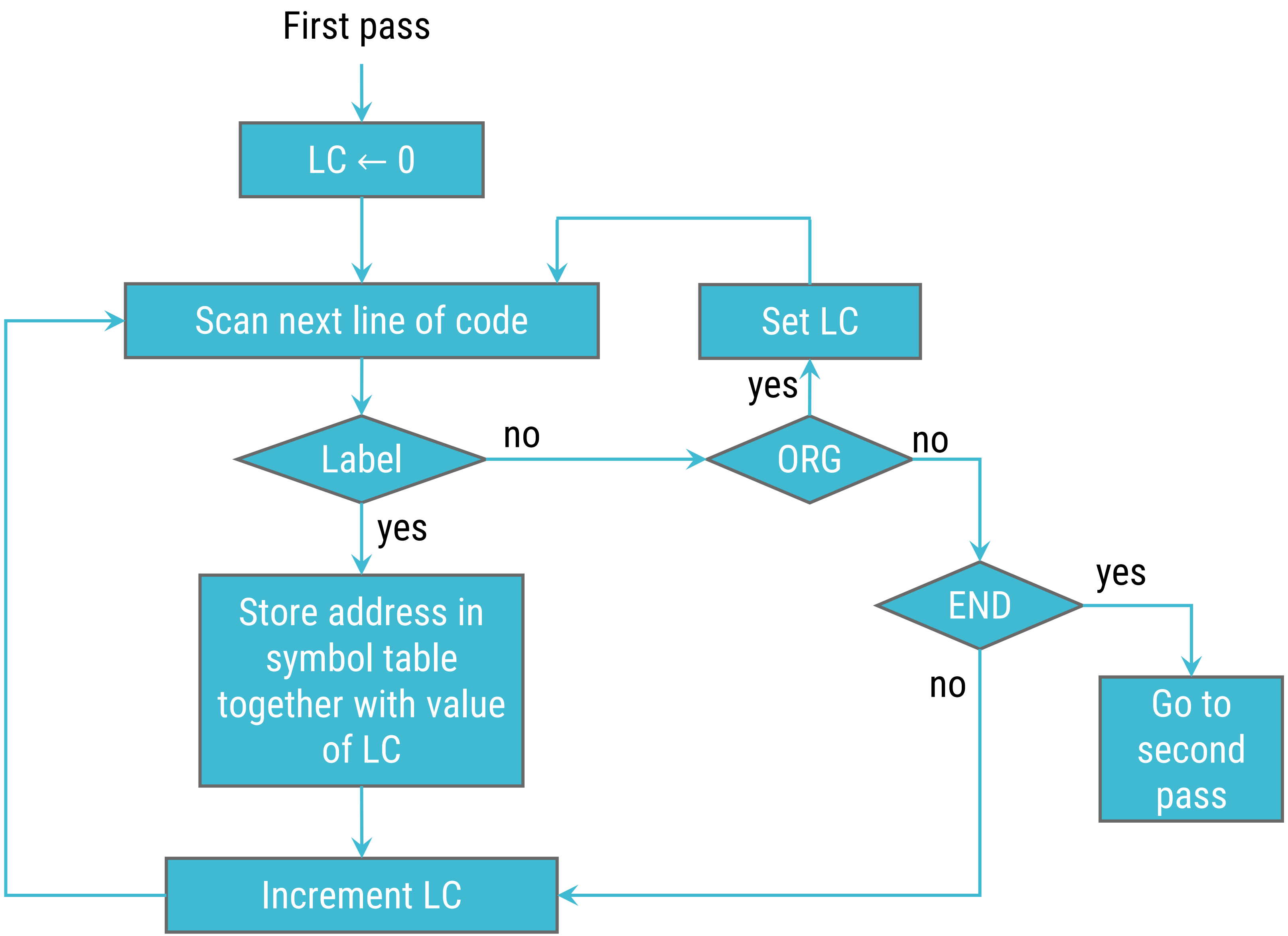
Character	Code	Character	Code	Character	Code
A	41	Q	51	6	36
B	42	R	52	7	37
C	45	S	53	8	38
D	44	T	54	9	39
E	45	U	55	Space	20
F	46	V	56	(	28
G	47	W	57	)	29
H	48	X	58	*	2A
I	49	Y	59	+	2B
J	4A	Z	5A	,	2C
K	4B	0	30	-	2D
L	4C	1	31	.	2E
M	4D	2	32	/	2F
N	4E	3	33	=	3D
O	4F	4	34	CR	0D
P	50	5	35		

Line of Code Table

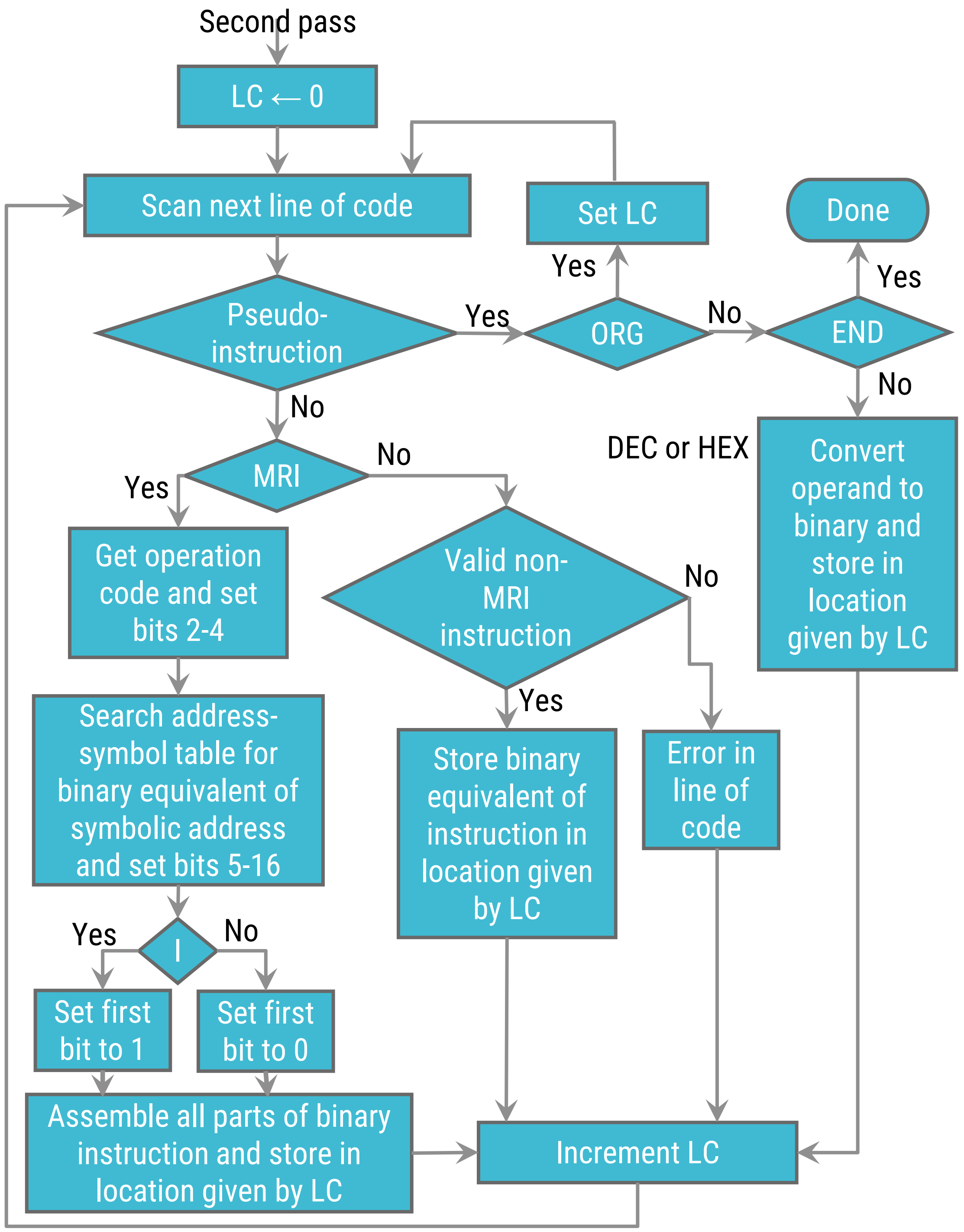
- PL3, LDA SUB I

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 of an assembler



Second Pass of an assembler



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*.

Assembly Language Program to add 100 numbers

```
      ORG 100      /Origin of program is HEX 100
      LDA ADS      /Load first address of operands
      STA PTR      /Store in pointer
      LDA NBR      /Load minus 100
      STA CTR      /Store in counter
      CLA          /Clear accumulator
LOP,  ADD PTR  I   /Add an operand to AC
      ISZ PTR      /Increment pointer
      ISZ CTR      /Increment counter
      BUN LOP      /Repeat loop again
      STA SUM      /Store sum
      HLT          /Halt
```

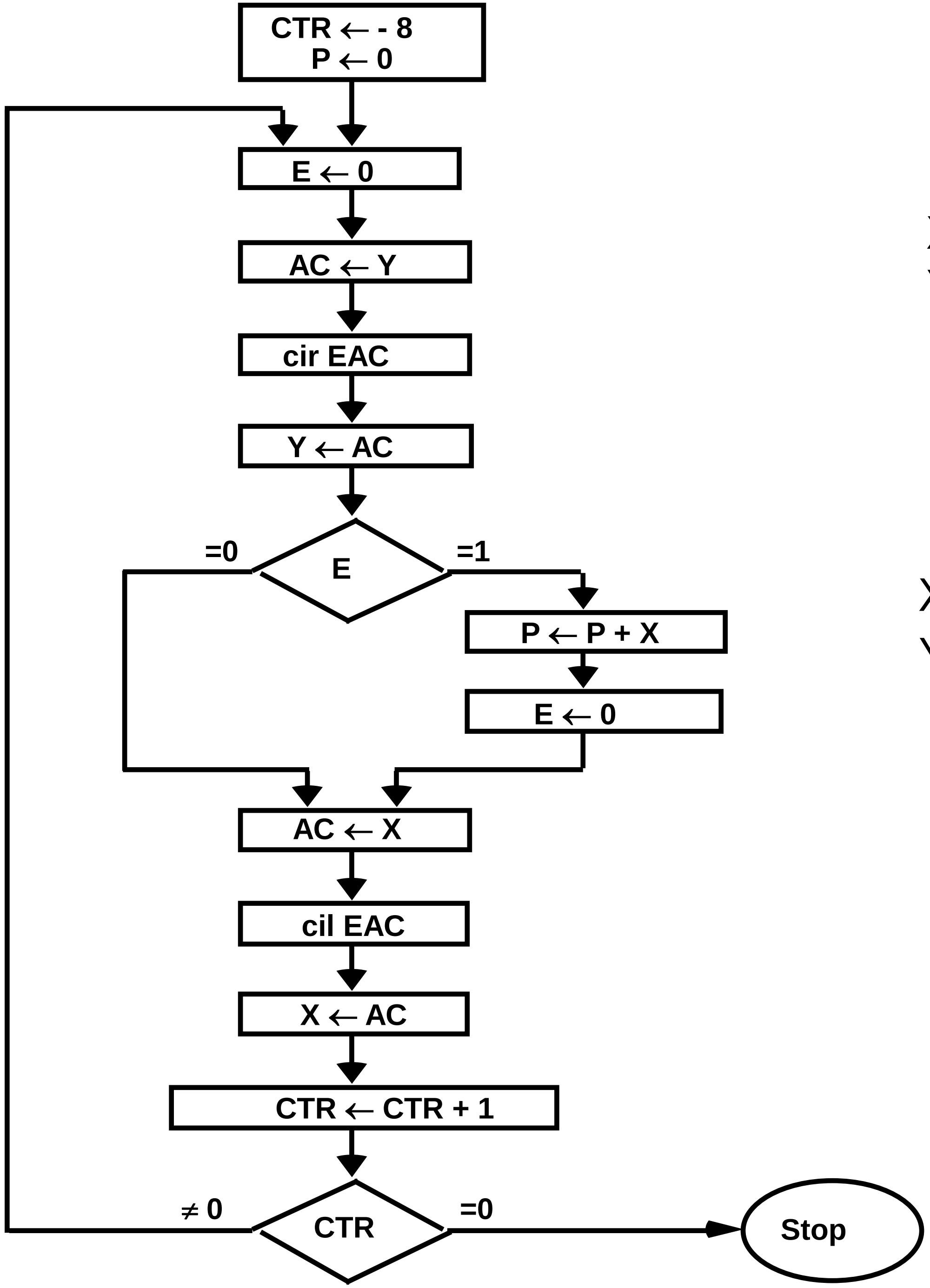
```
ADS,  HEX 150      /First address of operands
PTR,  HEX 0        /This location reserved for pointer
NBR,  DEC -100     /Constant to initialized counter
CTR,  HEX 0        /This location reserved for a counte
SUM,  HEX 0        /Sum is store here
      ORG 150      /Origin of operands is HEX 150
      DEC 75       /First operand
        :
      DEC 23       /Last operand
      END          /End of symbolic program
```

Programming Arithmetic & logic Operations

Multiplication Program

- Multiplying two numbers
- For simplify, neglect the sign bit and assume positive numbers
- Assume that two binary numbers have no more than eight significant bits so their product can not exceed the word capacity of 16 bits.
- It is possible to modify to take care of signs or use 16-bit numbers.
- *Adding the multiplicand X as many times as there are 1's in multiplier Y, provided that the value of X is shifted left from one line to the next.*
- Since, the computer can add only two numbers at a time, we reserve a memory location P, to store intermediate sums (called partial product).

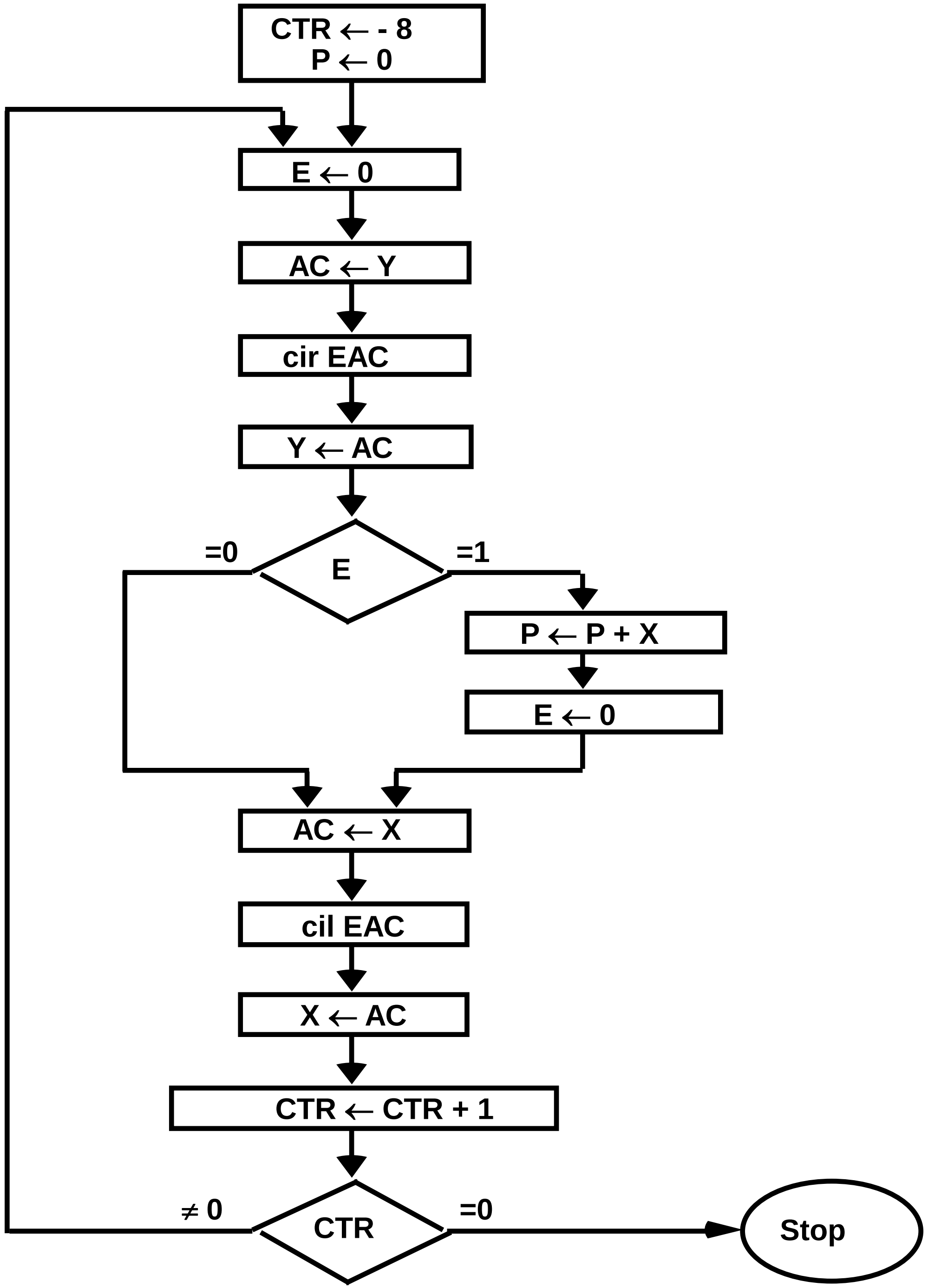
Programming Arithmetic & logic Operations



X holds the multiplicand
Y holds the multiplier
P holds the product

Example with four significant digits

		P
X =	1111	
Y =	1011	0000 0000
	1111	0000 1111
	1 1110	0010 1101
	00 0000	0010 1101
	111 1000	1010 0101
	1010 0101	




```
        ORG  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; goto ONE
        BUN  ZRO     / Bit is zero; goto ZRO
ONE,    LDA  X       / Load multiplicand
        ADD  P       / Add to partial product
        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
```

Programming Arithmetic & logic Operations

- **Double-Precision Addition**
- When two 16-bit unsigned numbers are multiplied, the result is a 32-bit product that must be stored in two memory words.
- A number stored in two memory words is said to have double precision.
- When a partial product is computed, it is necessary that a double precision number be added to the shifted multiplicand, which is also a double precision number.

Assembly Language Program (A.L.P.) to Add Two Double-Precision Numbers

```
     ORG 100      /Origin of program is HEX 100
     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 bring carry into AC(16)
     ADD AH       /Add A high and carry
     ADD BH       /Add B high
     STA CH       /Store in C high
     HLT          /Halt
AL,  -
AH,  -
BL,  -
BH,  -
CL,  -
CH,  -
```

Logic and Shift Operations

Logic Operations

- The basic computer has three machine instructions that perform logic operations: AND, CMA and CLA
- The LDA instruction may be considered as a logic operation that transfers a logic operand into the *AC*.
- We know 16 different logic operations. All 16 operations can be implemented using AND and Complement operations.
- Program for OR operation - x + y = (x' y')'

```
        ORG 100
        LDA  X          / Load 1st operand X
        CMA             / Complement to get X'
        STA  TMP        / Store in a temporary location
        LDA  Y          / Load 2nd operand Y
        CMA             / Complement to get Y'
        AND  TMP        / AND with X' to get X' AND Y'
        CMA             / Complement again to get X OR Y
X,      -
Y,      -
TMP,    -
        END
```

Logic and Shift Operations

Shift Operations

- The basic computer has circular-shift operations as machine instructions.
- The other shifts of interest are the logical shifts and arithmetic shifts, which can be programmed with a small number of instructions.
- Logical shift-right // zero be added to left most position
  - CLE
  - CIR
- Logical shift-left //zero be added to right most position
  - CLE
  - CIL

Logic and Shift Operations

Shift Operations (Cont..)

- Arithmetic right-shift

  CLE / Clear E to 0

  SPA / Skip if AC is positive

  CME / AC is negative; set E to 1

  CIR / Circulate E and AC

- Arithmetic left-shift
  - Added bit in the least significant position be 0.

    CLE / Clear E to 0

    CIL / Circulate *E* and *AC*
  - The sign bit must not change during the shift.
  - After shifting, compare *E* and *AC(15)*
  - If equal, correct shift
  - If not equal, an overflow occurs.

Subroutines

- 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.

```
            ORG 100
100         LDA X
101         BSA SH4
102         STA X
103         LDA Y
104         BSA SH4
105         STA Y
106         HLT
107   X,    HEX 1234
108   Y,    HEX 4321
109   SH4,  HEX 0
10A         CIL
10B         CIL
10C         CIL
10D         CIL
10E         AND MSK
10F         BUN SH4 I
110   MSK,  HEX FFF0
            END
```

I-O Programming - A.L.P. to input one character

```
        ORG 100     /Origin of program is HEX 100
CIF,    SKI         /Check input flag
        BUN CIF
        INP         /Flag = 1, input character
        OUT         /Print character
        STA CHR     /Store character
        HLT
CHR,    -           /Store character here
        END
```

I-O Programming - A.L.P. to output one character

```
        ORG 100     /Origin of program is HEX 100
        LDA CHR     /Load character into AC
COF,    SKO         /Check output flag
        BUN COF
        OUT         /Flag = 1, output character
        HLT
CHR,    HEX 0057    /Character is "W"
        END
```

Thank You !!!