PROGRAMMING THE BASIC COMPUTER

Introduction

Machine Language

Assembly Language

Assembler

Program Loops

Programming Arithmetic and Logic Operations

Subroutines

Input-Output Programming

INTRODUCTION

Those concerned with computer architecture should have a knowledge of both hardware and software because the two branches influence each other.

Instruction Set of the *Basic Computer*

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

m: effective address
M: memory word (operand) found at m

MACHINE LANGUAGE

Program

A list of instructions or statements for directing the computer to perform a required data processing task

Various types of programming languages

- Hierarchy of programming languages

- **Machine-language**
  - **- Binary code**
  - **- Octal or hexadecimal code**
- **Assembly-language (Assembler)**
  - **- Symbolic code**
- **High-level language (Compiler)**

COMPARISON OF PROGRAMMING LANGUAGES

• **Binary Program to Add Two Numbers**

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

• **Hexa program**

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

• **Program with Symbolic OP-Code**

Location	Instruction		Comments
000	LDA	004	Load 1st operand into AC
001	ADD	005	Add 2nd operand to AC
002	STA	006	Store sum in location 006
003	HLT		Halt computer
004	0053		1st operand
005	FFE9		2nd operand (negative)
006	0000		Store sum here

• **Assembly-Language Program**

```
      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
```

• **Fortran Program**

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

ASSEMBLY LANGUAGE

Syntax of the BC assembly language

Each line is arranged in three columns called fields

***Label* field**

- May be empty or may specify a symbolic address consists of up to 3 characters

- Terminated by a comma

***Instruction* field**

- Specifies a machine or a pseudo instruction

- May specify one of

*** Memory reference instr. (MRI)**

MRI consists of two or three symbols separated by spaces.

```
ADD  OPR     (direct address MRI)
ADD  PTR  I   (indirect address MRI)
```

*** Register reference or input-output instr.**

Non-MRI does not have an address part

*** Pseudo instr. with or without an operand**

Symbolic address used in the instruction field must be defined somewhere as a label

***Comment* field**

- May be empty or may include a comment

PSEUDO-INSTRUCTIONS

ORG N
Hexadecimal number N is the memory loc.
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 the binary

HEX N
Hexadecimal number N to be converted to the binary

Example: Assembly language program to subtract two numbers

```
                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 stored here
                END            / End of symbolic program
```

TRANSLATION TO BINARY

Hexadecimal Code		*Symbolic Program*	
Location	*Content*		
			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

ASSEMBLER - FIRST PASS -

Assembler

Source Program - Symbolic Assembly Language Program

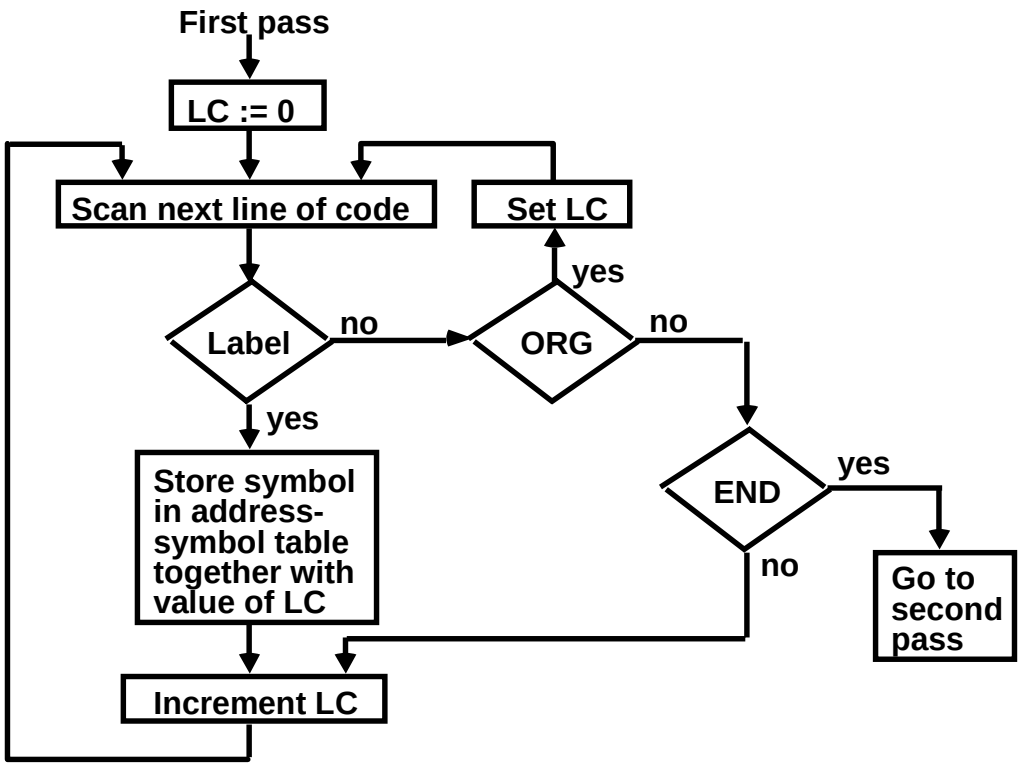
Object Program - Binary Machine Language Program

Two pass assembler

1st pass: generates a table that correlates all user defined (address) symbols with their binary equivalent value

2nd pass: binary translation

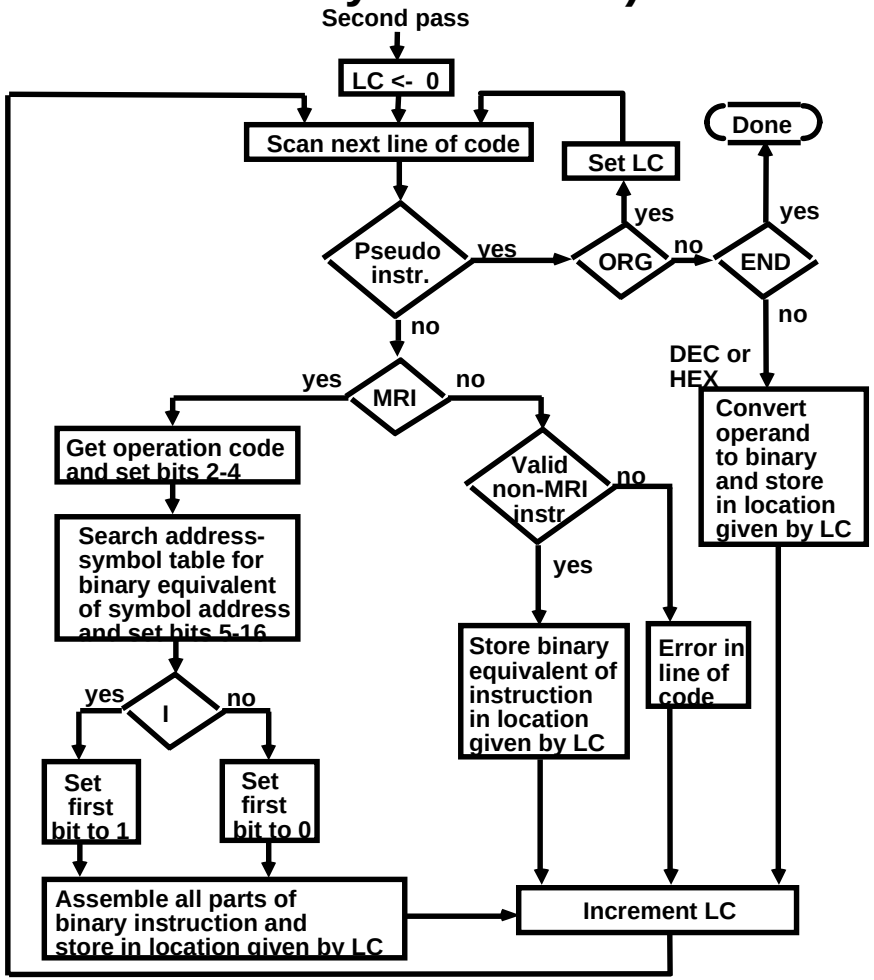
First pass



ASSEMBLER - SECOND PASS -

Second Pass

Machine instructions are translated by means of table-lookup procedures;
(1. Pseudo-Instruction Table, 2. MRI Table, 3. Non-MRI Table
4. Address Symbol Table)



PROGRAM LOOPS

Loop: A sequence of instructions that are executed many times, each with a different set of data

Fortran program to add 100 numbers:

```
      DIMENSION  A(100)
      INTEGER  SUM,  A
      SUM = 0
      DO  3  J = 1,  100
3     SUM = SUM + A(J)
```

Assembly-language program to add 100 numbers:

```
              ORG  100          / Origin of program is HEX 100
              LDA  ADS          / Load first address of operand
              STA  PTR          / Store in pointer
              LDA  NBR          / Load -100
              STA  CTR          / Store in counter
              CLA               / Clear AC
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            / Reserved for a pointer
NBR,          DEC  -100         / Initial value for a counter
CTR,          HEX  0            / Reserved for a counter
SUM,          HEX  0            / Sum is stored here
              ORG  150          / Origin of operands is HEX 150
              DEC  75           / First operand
                 :
                 :
                 .
              DEC  23           / Last operand
              END               / End of symbolic program
```

PROGRAMMING ARITHMETIC AND LOGIC OPERATIONS

Implementation of Arithmetic and Logic Operations

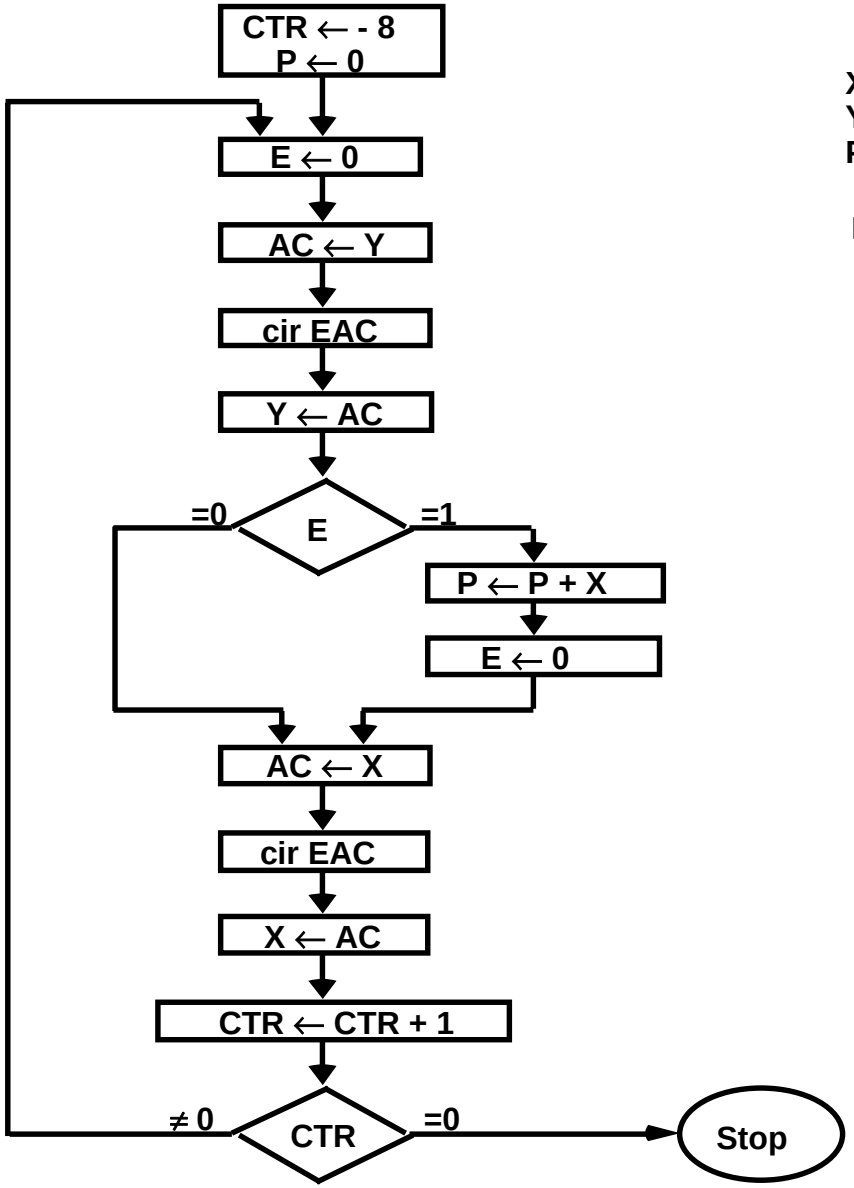
- **Software Implementation**
  - **Implementation of an operation with a program using machine instruction set**
  - **Usually when the operation is not included in the instruction set**

- **Hardware Implementation**
  - **Implementation of an operation in a computer with one machine instruction**

Software Implementation example:

* **Multiplication**
- **For simplicity, unsigned positive numbers**
- **8-bit numbers -> 16-bit product**

FLOWCHART OF A PROGRAM - Multiplication -



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

Example with four significant digits

		P
X =	0000 1111	
Y =	0000 1011	0000 0000
	0000 1111	0000 1111
	0001 1110	0010 1101
	0000 0000	0010 1101
	0111 1000	1010 0101
	1010 0101	

ASSEMBLY LANGUAGE PROGRAM - Multiplication -

```
        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
```

ASSEMBLY LANGUAGE PROGRAM
- Double Precision 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 bring carry into AC(16)
ADD   AH      / Add A high and carry
ADD   BH      / Add B high
STA   CH      / Store in C high
HLT
```

ASSEMBLY LANGUAGE PROGRAM
- Logic and Shift Operations -

- **Logic operations**

  - **BC instructions : AND, CMA, CLA**
  - **Program for OR operation**

```
LDA   A      / Load 1st operand
CMA          / Complement to get A'
STA   TMP    / Store in a temporary location
LDA   B      / Load 2nd operand B
CMA          / Complement to get B'
AND   TMP    / AND with A' to get A' AND B'
CMA          / Complement again to get A OR B
```

- **Shift operations - BC has *Circular Shift* only**

  - **Logical shift-right operation**

```
CLE
CIR
```

  - **Logical shift-left operation**

```
CLE
CIL
```

  - **Arithmetic right-shift operation**

```
CLE     / Clear E to 0
SPA     / Skip if AC is positive
CME     / AC is negative
CIR     / Circulate E and AC
```

SUBROUTINES

Subroutine

- A set of common instructions that can be used in a program many times.
- Subroutine *linkage* : a procedure for branching to a subroutine and returning to the main program

Example

```
Loc.
100         ORG  100        / Main program
101         LDA  X          / Load X
102         BSA  SH4        / Branch to subroutine
103         STA  X          / Store shifted number
104         LDA  Y          / Load Y
105         BSA  SH4        / Branch to subroutine again
106         STA  Y          / Store shifted number
107         HLT
108   X,    HEX  1234
109   Y,    HEX  4321
                            / Subroutine to shift left 4 times
10A   SH4,  HEX  0          / Store return address here
10B         CIL             / Circulate left once
10C         CIL
10D         CIL
10E         CIL             / Circulate left fourth time
10F         AND  MSK        / Set AC(13-16) to zero
110         BUN  SH4  I     / Return to main program
            HEX  FFF0       / Mask operand
            END
```

SUBROUTINE PARAMETERS AND DATA LINKAGE

Linkage of Parameters and Data between the Main Program and a Subroutine

- **via Registers**
- **via Memory locations**
- **….**

Example: Subroutine performing *LOGICAL OR operation*; Need two parameters

```
Loc.        ORG  200
200         LDA  X        / Load 1st operand into AC
201         BSA  OR       / Branch to subroutine OR
202         HEX  3AF6     / 2nd operand stored here
203         STA  Y        / Subroutine returns here
204         HLT
205   X,    HEX  7B95     / 1st operand stored here
206   Y,    HEX  0        / Result stored here
207   OR,   HEX  0        / Subroutine OR
208         CMA           / Complement 1st operand
209         STA  TMP      / Store in temporary location
20A         LDA  OR I     / Load 2nd operand
20B         CMA           / Complement 2nd operand
20C         AND  TMP      / AND complemented 1st operand
20D         CMA           / Complement again to get OR
20E         ISZ  OR       / Increment return address
20F         BUN  OR I     / Return to main program
210   TMP,  HEX  0        / Temporary storage
            END
```

SUBROUTINE - Moving a Block of Data -

```
                        / Main program
        BSA  MVE        / Branch to subroutine
        HEX  100        / 1st address of source data
        HEX  200        / 1st address of destination data
        DEC  -16        / Number of items to move
        HLT
MVE,    HEX  0          / Subroutine MVE
        LDA  MVE  I     / Bring address of source
        STA  PT1        / Store in 1st pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring address of destination
        STA  PT2        / Store in 2nd pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring number of items
        STA  CTR        / Store in counter
        ISZ  MVE        / Increment return address
LOP,    LDA  PT1  I     / Load source item
        STA  PT2  I     / Store in destination
        ISZ  PT1        / Increment source pointer
        ISZ  PT2        / Increment destination pointer
        ISZ  CTR        / Increment counter
        BUN  LOP        / Repeat 16 times
        BUN  MVE  I     / Return to main program
PT1,    --
PT2,    --
CTR,    --
```

- **Fortran subroutine**

```
      SUBROUTINE  MVE (SOURCE, DEST, N)
      DIMENSION  SOURCE(N), DEST(N)
      DO  20  I = 1, N
20    DEST(I) = SOURCE(I)
      RETURN
      END
```

INPUT OUTPUT PROGRAM

Program to Input one Character(Byte)

```
CIF,    SKI          / Check input flag
        BUN  CIF     / Flag=0, branch to check again
        INP          / Flag=1, input character
        OUT          / Display to ensure correctness
        STA  CHR     / Store character
        HLT
CHR,    --           / Store character here
```

Program to Output a Character

```
        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

Subroutine to Input 2 Characters and pack into a word

```
IN2,    --              / Subroutine entry
FST,    SKI
        BUN  FST
        INP             / Input 1st character
        OUT
        BSA  SH4        / Logical Shift left 4 bits
        BSA  SH4        / 4 more bits
SCD,    SKI
        BUN  SCD
        INP             / Input 2nd character
        OUT
        BUN  IN2  I     / Return
```

PROGRAM INTERRUPT

Tasks of Interrupt Service Routine

- Save the Status of CPU
Contents of processor registers and Flags

- Identify the source of Interrupt
Check which flag is set

- Service the device whose flag is set
(Input Output Subroutine)

- Restore contents of processor registers and flags

- Turn the interrupt facility on

- Return to the running program
Load PC of the interrupted program

INTERRUPT SERVICE ROUTINE

```
Loc.
  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
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
                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
        SAC,    -           / AC is stored here
        SE,     -           / E is stored here
        PT1,    -           / Pointer of input buffer
        PT2,    -           / Pointer of output buffer
```