21/01/22

CS251 - System programming

- Assemblers
- Loaders
- Linkers
- Microprocessor
- Utility programs

Source program → ASSEMBLY LANGUAGE → MNEUMONIC CODES (ADD A,B ; SUB C,D) → ASSEMBLER → MACHINE LANGUAGE (Object program) → Loader, Linkers

HLL (Printf(...); C=A+B;) → Compiler → MACHINE LANGUAGE

System program ⟷ Application program

Application program: This one has lesser dependence on H/W aspects → hardware

Take ASSEMBLER → 0110 011100 001100 → ADD

In some other ADD may be diff → 001100

The system program is aware of these details.

Dependent on machine architecture → SP have Machine Architecture Dependance.

Since it is dependence on architecture, without architecture we can't study SP.

Suitable architecture → Hypothetical machines

- RISC
- CISC

SIC : Simplified Instructional Computer

SIC/XE : Extra Equipment

We will be focussing on

- Memory
- Registers
- Data formats
- Addressing modes
- Instruction set
- I/O.

Hexadecimal Representation

- Base-16 system
- It is positional numeric system

16 Basic symbols

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Deci	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Human-friendly representation of binary coded decimal

0000 0000 — 4-bits
↳ nibble
↓
1 hexadecimal digit

0 0

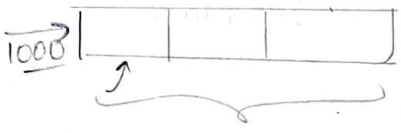
1111 1111
F F

15
↙ ↳
0001 0101

SIC (3 bits) ↳ size of instruction

xxx0
xxx3
} + 3 bits

- Memory :-
  - 8-bit bytes
  - word → 3-consecutive bytes
    ↳ 24-bit
  - Byte addressable
  - Address of a word is given by lowest numbered byte



- Total of 32768 bytes = Size of memory

$\Downarrow$

2^{15}

Any byte can be addressable by 15-bit address.

$2^9 \rightarrow 512$

$2^{10} \rightarrow 1024$

⋮

- Registers:-

- 24-bit length
- 5 registers (standard registers that use in SIC)

Mneumonic	Number	
A	0	(called as Accumulator) Extensively used in Arithmetic operations.

Ex:- LDA (TEN)

where to load is not given
So, it loaded into accumulator. A ← (TEN)

X 1 (INDEX Register)

to index sequence of words

- Sequence of memory

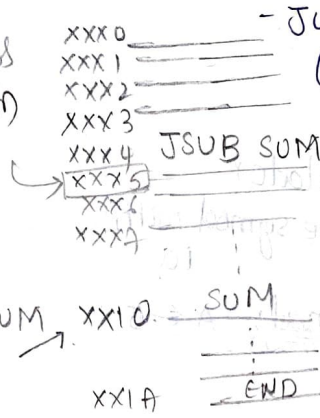
```
i=0
i++; / i--;
Array[i]
```

↓ +[x] X → increment

L 2 (Linkage register)

to store return address of instruction

- Jump to subroutine (JSUB) → return address of JSUB is stored in Linkage register



RSUB
└ Return from Subroutine

PC 8 - (program counter)

SW 9 (Status word)

↳ CC condition compare

CMP P, Q

JLT ....

	LT	EQ	GT	Z	N	
P<Q if LT	1	0	0	0	0	....
P=Q →	0	1	0	0	0	

- Data formats :

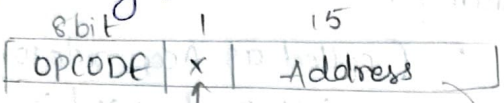
① - Integers (supports only integers)

- 24-bits
- 2's complement is used for -ve (signed integers)

② ASCII codes — SIC is H/W
- 8-bits — it doesn't support floating point

- Instruction formats:

-24-bit length



1 = index addressing
0 = Direct

address of operand

15 is very much sufficient becoz every byte is addressable by 15 bit

- Addressing modes:

In SIC, only 2 addressing modes are supported

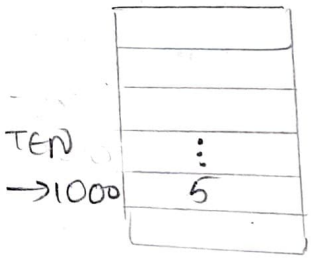
→ X-flag (which addressing mode is choosen is taken care by this x-flag)

- Direct X=0 TA= Address
- Indexed X=1 TA = Address + (x)

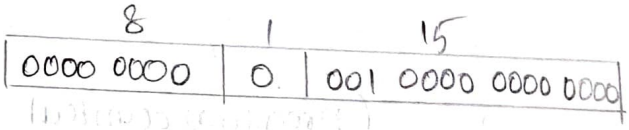
opcode = 0 0

① LDA TEN

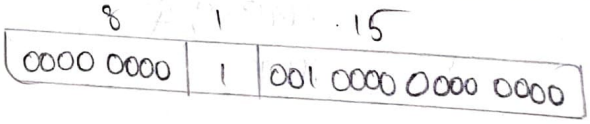
Load the accumulator with the symbol with ID.



Final:- A ← 5



② LDA TEN, X — Indexed addressing is to be used.



EA = 1000 + (x)

Effective address — content of x register if it is 3

③ STCH BUFFER, X → exercise to go through it

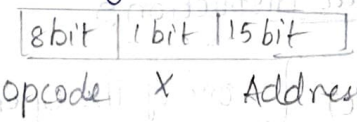
25/1/22

System software - Machine Architecture

- SIC
- SIC/XE
- RISC
- CISC

SIC

- Memory
  - 2^{15} bytes size → 15-bit address
  - Any byte can be addressable
- Registers
  - A, X, L, PC, SW
- Data formats
  - Integers, characters
  - 24 bits = word length
- Instruction formats
  - 24-bit in length



- Addressing modes.
  - Direct Addressing
    - → X = 0
    - EA = Address mentioned in the Addressing field.
  - Indexed Addressing.
    - EA = f(x) + Address field.
    - → X = 1

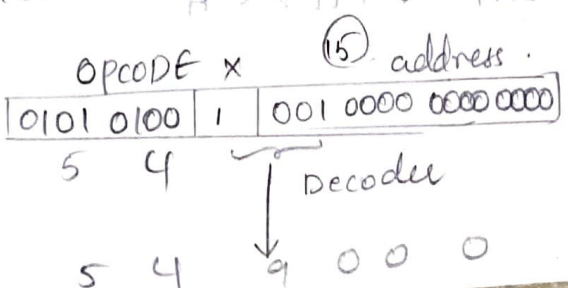
STCH BUFFER, (X) → Indexed address

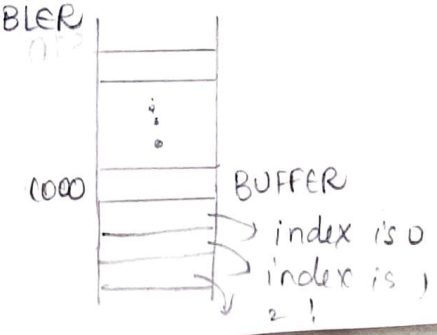
Directive/ information passed to assembler by the programer to use indexed addressing.

MNEUNOMIC instruction → ASSEMBLING by ASSEMBLER

$(54)_{16}$ → opcode

Hexa decimal





0101 0100 1 001 0000 0000 0000

5 4 9 | 0 0 0

↓ STCH ↓ Indexed ↓ Address — store the character

EF = 1000 + [X] → content of X register
indexed register by name X

If assembler did mistake, x flag is not set

0101 0100 | 0 | 001 0000 0000 0000

EF = 1000 (+[X] is not taken) → every time it is written in same address

If programmer did mistake

STCH BUFFER (no,x)

-Instruction set :

⑤ -(Integer) arithmetic instructions.

- ADD
- SUBTRACT
- MUL
- DIV

} one of the operands is A (accumulator)

ADD ALPHA

(A) ← (A) + (ALPHA)

result will go back to A.

DELTA = ALPHA + BETA

ADD DELTA ALPHA BETA → This is Complex (SIC does not support)

1 instruction

① - Load and Store

- LDA, STA (load the accumulator and store the accumulator)
- LDX, STX (" " indexed register)
- LDCH, STCH
- ~~LI~~

LDA BETA → A ← 9

ADD ALPHA → A ← (A+1)

STA DELTA → DELTA ← A

Not 10
10 = A
in Hexadecimal
(Only hexadecimal numbers will store)

② Comparision instruction

- A is one of the operand

CMP ALPHA if (ALPHA > A)
<A
==A

result is stored in condition flag

CC

LT	GT	EQ	. . .

③ Conditional Jump instruction
Branch

- JLT, JGT, JEQ while ( )
- J (Repeat)

④ Subroutine

- JSUB
- RSUB

- Input and Output :

- It is done when device is ready
  TD (Test Device instruction)
  WD/RD (Write device/ Read device)

DEV1 → device

Read TD DEV1 (Till the device is ready we can't intiate read input)
↳ JEQ Read
if it ready, flag is set to equal
loop (continuously check till it ready)

RD DEV1
↳ A Read into accumulator

WRITE TD DEV2
JEQ write

WD DEV2 once device is ready
intiate write operation

SIC/XE - CISC : (extra equipment) → a bit advanced than SIC

Machine Architecture

- Memory:

  Word length = 24 bit

  Memory size = 2^{20} bytes (1 Mega byte) — 20-bit address

- Registers:

  - 5 registers

A	0
X	1
L	2
PC	8
SW	9

extra/additionally, (4 more registers)

B	3	(Base register) - Relative Addressing
S	4	(General working Registers) They don't carry any special functionality
T	5	
F	6	(Floating point accumulator) - Similar role of integer accumulator {except word length}

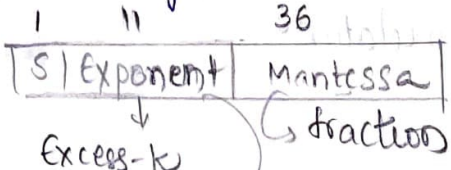
F → 48-bit → word length

- Data formats:

  ① Integers → 24-bit

  ② Characters

  ③ Floating point · (48-bit) ⇒ double of integer bit



biased exponent = Excess-k of true exponent ($2^{k-1}-1$)

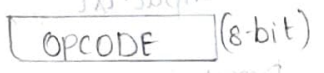
- Instruction formats

  - 4 formats of instruction

27/1/22

- Instruction formats:
  - Different formats
  - CISC → supports
    - ① Relative Addressing
  - 4 formats

① Format 1 : 1 Byte (length)



Ex:- FIX (A) ← (F)

Here A, F both are accumulators

load the accumulator with content of F

(F: Floating point, A- integer)

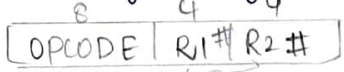
it converts floating type to integer type

FLOAT (F) ← (A)

HIO

(Halt the input/output)

② Format 2 : 2 Byte (length) = 16 bits



Register operands

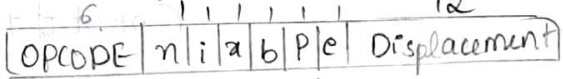
10 Registers

ADDR ← Ex:- COMPR A,S

COMP A, operand
compare accumulator, operand

COMPR A, S
Opcode A0 0 4

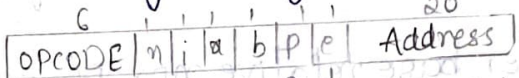
V. Imp ③ Format 3 : 3 byte (length) : 24 bit



6, 1 bit flags

→ It is not address (Address = 20 bit)

④ Format 4 : 4 bytes (length) = 32 bit



Addressing modes

20 bit is sufficient

As Any byte can be addressed by 20 bit address

x : Indexed

e : Extended → F3/F4

b, p : Type of relative addressing

n, i :

0 → Format 3

1 → Format 4 / extended format

+JSUB RSUBROUTINE

- Addressing modes : (2 types)

-SIC/XE supports Relative Addressing

- It is used in Format 3 instructions

/user1/DaS/

↳ sort.c → filename path ⤵

Input.txt /usa1/DS/input.txt

↓

This path is no longer works for this

Reorganizing Directories

/user1/ Fourth /DS

↓

fourth semester

./input.txt is there then that path still works.

/user1/DS/code

/usa1/DS/data

DS

code Data

/user1/DS/data/input.txt

../data/input.txt

Problems on Movement of files to somewhere are not arising because we used relative addressing.

Type	Indication	Target Address	
Base relative	b=1, p=0	TA = [B] + Displacement (Content of base register)	unsigned: 0 ≤ disp ≤ 4095
Program Counter relative (PC relative)	b=0, p=1	TA = [PC] + Displacement (Content of PC)	signed: -2048 ≤ disp ≤ 2047

If neither base relative nor pc relative is used.

Then b=0, p=0 | b=1, p=1

Ex:- for base relative addressing

① STX LENGTH

n i x b=1 e

000100	1	1	0	1	0	0	0000 0000 0000

→ P=0

Suppose [B] ⇔ 0033

x=0 → not index
direct addressing

e=0 → Format 3

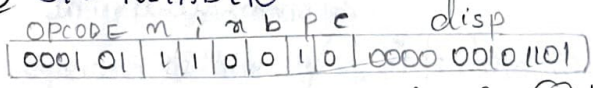
n=1 & i=1 → Simple addressing

b=1, p=0 → Base relative addressing

TA = [B] + Displacement
= 0033 + 000
= 0033

Ex:- PC relative addressing

② STL RETADER



0 2 (13) D

x=0 → Direct addressing

e=0 → Format 3

n=1 & i=1 → simple addressing

b=0, p=1 → Pc reolative addressing

disp = TA - [PC]

TA = [pc] + Displacement

[PC] ⇔ 0003

= 0003 + 02D

0003
002D (Hexadecimal addition)
0030

STL RETADER is stored in address 0030.

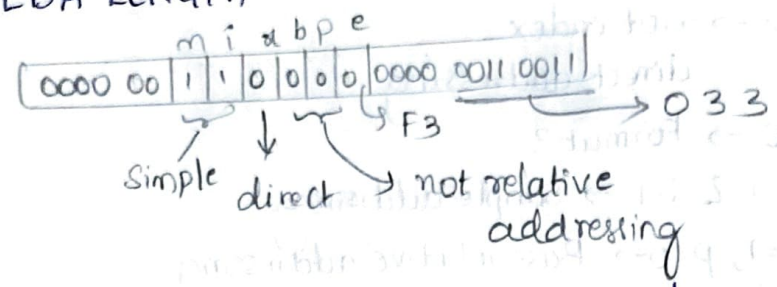
LOOP TD 05
JEQ Loop

Jump fomard → Displacement is +ve
Jump back → Displacement is -ve

Loop COMP N → A,N
JGT EXIT
A0
J LOOP → Jump to loop
EXIT TD

TA > [PC] → disp is +ve

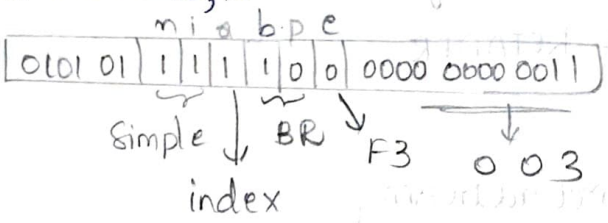
③ LDA LENGTH



When relative addressing is not used, TA is directly given in displacement

TA = 033

④ STCH BUFFER, X



[B] → 0033
[X] → 0

Indexing and Relative together

TA = Disp + [B] + [X]
= 003 + 0033 + 0
= 0036

Content stored in Accumulator (8 bit) will stored under address location 0036.

i=1, n=0 → Immediate addressing.
i=0, n=1 → Indirect addressing

LDA LENGTH ← Assembly language

Language translator Assembler

0000 0011 0000 0000 0011 0011 ← Machine language

This course → how Assembler done that

We have to find disp for that

Disp = TA - [B]
Disp = TA - [PC]

28/1/22

SIC/XE MACHINE ARCHITECTURE

- Instruction formats
  - 4 formats

n, i, x, b, P, e

x : Indexed or Direct

e : Extended → F3|F4

b, P : Type of relative addressing

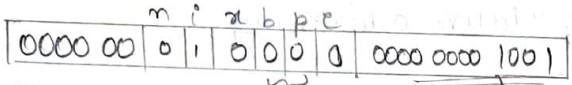
n, i

i=1, n=0 : Immediate addressing

-Operand is directly given in the instruction itself

opcode: 0 0 ← LDA #9

0000 0000



when we write in 6 bit opcode we just ignore last 2bits

no relative addressing concept

Assembler Take this value directly and load the accumulator with 9

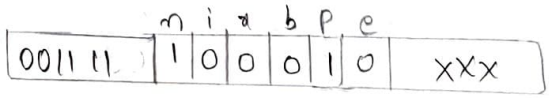
A ← 9

n=1, i=0 : Indirect addressing

J @RETADR

opcode of J is 3C

0011 1100 ↓ ignore



0000 0000 0011

Choosen as PC Relative : b=0, P=1

[PC] ⇒ 002D

RETADR ⇒ ~~0030~~ 0030

Displacement = TA - [PC]

= 0030 16

- 002D

0003

During decoding

It identifies

3C → J

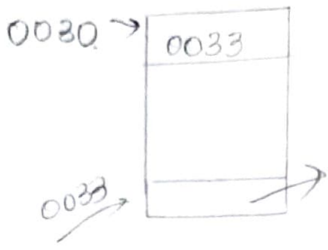
indirect addressing

EA → 002D

0003

0030

As it is indirect addressing,



n=0, i=0 } Simple addressing : Target address is taken as the location of the operand.
n=1, i=1 }

n=0, i=1 : Immediate addressing : No Target address

n=1, i=0 : Indirect addressing. : Target address is taken as the address where the location of operand.

Indexing cannot be used

b=0, P=0 : Direct addressing.
↳ No relative addressing

F3 F4 → b=0, P=0
no need of relative addressing

F3 → Operand address fixed in 12 bit → Displacement → Operand address TA

address is fit in 20 bit.

TA = Displacement

TA | XXX | ← XXX

b=1, P=0 : Base relative } we can have Indexing with it
b=0, P=1 : PC relative }

	⋮
3030	003600
	⋮
3600	103000
	⋮
6390	000303
	⋮
C303	003030

(B) = 006000
(Pc) = 003000
(X) = 000090

1) 032600

n i x b p e

0000 0011 0010 0101 0000 0000

600

[24 bit, So format-3]

(00) LDA

n=1, i=1 : Simple addressing

x=0 : Direct addressing

b=0, p=1 : PC relative addressing

e=0 : F3

(PC) = 3000

TA = [pc] + Displacement

TA = 3600

3000
600
3600

A ← 103000 (value stored at 3600)

2) 03C300

3) 022030

1/2/22

03C300

n i x b p e

0000 0011 1100 0011 0000 0000

300

00
LDA

n=1, i=1 : Simple addressing

x=1 : Indexed

b=0, p=0 : Base relative

e=0 : F3

TA = [B] + Displacement + [X]

= 006000 + 300 + 000090

6000
300
90
6390

A ← 00C303

022030

ni xbpe

0000 0010 0010 0000 0011 0000

00 — LDA

030

n=1, i=0 : Indirect addressing

x=0 : Direct

b=0, p=1 : PC relative addressing

e=0 : F3.

TA = [PC] + Displacement

= 3000 + 030

3000
030
3030

3030 → 3600

3600 → 103000

As it is indirect addressing,

A ← 103000.

4) 010030

ni xbpe

0000 0001 0000 0000 0011 0000

00 — LDA

030

n=0, i=1 : Immediate addressing

x=0 : Direct

b=0, p=0 : No relative addressing

e=0 : F3

As it is immediate addressing, there will be no TA.

A ← Displacement value

A ← 30

5) 003600

```
          ni xbpe
0000 0000 0010 0110 0000 0000   (24 bit)
<------>          <----------->
   00                  600
  LDA
```

n=0, i=0 : Simple addressing

x=0 : Direct

b=0, p=1 : PC relative addressing

e=0 : F4-3 (But only 24 bit is given)

TA = [PC] + Displacement

= 3000 + 600

= 3600

A ← 003600 (003600)

6) 03100303

```
          ni xbpe
0000 0011 0001 0000 1100 0011 0000 0011   (32 bit)
<------>          <----------->
   LDA                C303
```

Directly we can write

TA = C303

A ← 003030

n=1, i=1 : Simple addressing

x=0 : Direct

b=0, p=0 : No relative addressing

e=1 : F4

- Instruction set

- → LDB, LDT, LDS, STB
  → FP → arithmetic
  → Register-register

  ADDR, SUBR, MULR, DIVR

Format 2 ↓

ADDR r1, r2

r2 ← (r1) + (r2)

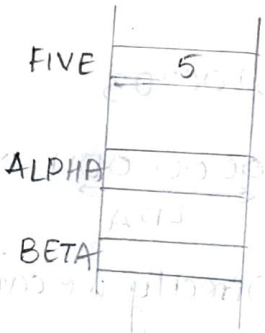
- Input Output

Programming Example : SIC, SIC/XE

① Data Movement

SIC :

→ one memory location → another
→ one word (24 bit)



Store the value of accumulator at ALPHA

```
LDA   FIVE     A ← 5
STA   ALPHA
 .      .
 .      .
 .      .
```

FIVE → is a WORD and Value is 5

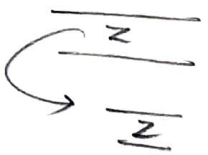
ALPHA → Reserve word RESW (1) word

Symbolic Labels
↳ they refer to specific memory location

WORD → refers to WORD constant
RESW → " " Reserved it is reserved to a word.

↳ one byte (8 bit)

```
LDCH   CHARZ
STCH   C1
```



CHARZ → BYTE constant C'Z'
C1 → RESB (Reserved byte) 1

SIC/XE :

```
LDA   #5
STA   ALPHA
LDA   #90  →  or LDCH
STCH    C1

ALPHA   RESW   1
C1      RESB   1
```

② Arithmetic Operations.

BETA ← (ALPHA + INCR-1) — ①

DELTA ← (GAMMA + INCR-1) — ②

SIC :

```
LDA   ALPHA   }
ADD   INCR    }  ①
SUB   ONE     }
STA   BETA    }
LDA   GAMMA   }
SUB           }
ADD   INCR    }  ②
SUB   ONE     }
STA   DELTA   }
```

```
ONE     WORD   1
ALPHA   RESW   1
BETA
```

```
ALPHA   RESW   1   }
BETA    RESW   1   }
GAMMA   RESW   1   }  1 word
DELTA   RESW   1   }
INCR    RESW   1   }
ONE     WORD   1  → value of word ONE
```

SIC/XE :

```
LDS   INCR    }
LDA   ALPHA   }  ①
ADDR  S,A     }
SUB   #1      }
STA   BETA    }
```

No need to fetch from memory.

```
LDA   GAMMA   }
ADDR  S,A     }  ②
SUB   #1      }
STA   DELTA   }
```

```
ALPHA   RESW   1
BETA    RESW   1
GAMMA   RESW   1
DELTA   RESW   1
INCR    RESW   1
```

③ How to do Indexing : Loop Management -
Based on that how to manage Array :-
I/O. -

4/2/22

SIC, SIC/XE programs/code segments

Arithmetic operation

BETA ← (ALPHA + INR-1)
DELTA ← (GAMMA + INR-1)

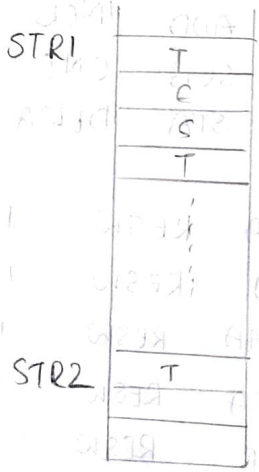
"did in last class"

③ Looping, indexing : String copy
↳ 11-byte string copied from source to destination

SIC :

```
        LDX   ZERO
MOVE  - LDCH  STR1,X
CH      STCH  STR2,X
        TIX   ELEVEN  → X = X+1
        JLT   MOVECH    X < >op
                        X < 11
        :

STR1    BYTE  C'TEST STRING'
STR2    RESB  11
        :
ZERO    WORD  0
ELEVEN  WORD  11
```



SIC/XE :

```
        LDT   #11
        LDX   #0
MOVECH  - LDCH  STR1,X
        STCH  STR2,X
        TIXR  T
        JLT   MOVECH
```

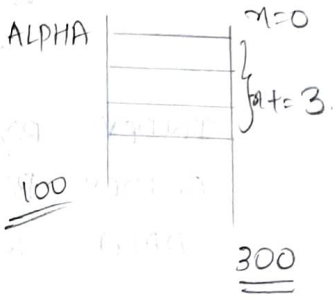
```
        :
STR1    BYTE    C'TEST STRING'
STR2    RESB    11
        :
```

Arrays → ALPHA[100] BETA[100] GAMMA[100]

GAMMA[i] = ALPHA[i] + BETA[i]

SIC :

```
        LDA     ZERO
        STA     INDEX
ADDLP   LDX     INDEX
        LDA     ALPHA,X
        ADD     BETA,X
        STA     GAMMA,X
        LDA     INDEX
        ADD     THREE
        STA     INDEX
        COMP    K300
        JLT     ADDLP
        :
INDEX   RESW    1
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
ZERO    WORD    0
K300    WORD    300
THREE   WORD    3
```



TIX THREE ← +3 length of SIC - 3 bits

TIX only increment by 1 and at the same time it compares.

SIC/XE :

```
        LDS     #3
        LDT     #300
        LDX     #0
ADDLP   LDA     ALPHA,X
        ADD     BETA,X
        STA     GAMMA,X
        ADDR    S,X
        COMPR   X,T
        JLT     ADDLP
```

```
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

I/O

```
INLOOP  TD   INDEV
        JEQ  INLOOP.
        RD   INDEV
        STCH DATA
        :
OUTLP   TD   OUTDEV
        JEQ  OUTLP
        WD
        LDCH DATA     (read the data)
        WD   OUTDEV
```

Hexadecimal

```
INDEV   BYTE   X'F1'     standard INput device
OUTDEV  BYTE   X'05'
 DATA   RESB   1
```

name

character

```
STRI  BYTE  C'TEST STRING'.
```

- Read 100-byte record
  → Memory
- Subroutine

Subroutine name.

```
PC → JSUB READ  ←  JSUB
     ADD  XXX
```

When it is called

① TA is calculated (TA of Subroutine)

② where to return
Linkage registers.
↑ PC

```
READ   LDX   ZERO
RLOOP  TD    INDEV
       JEQ   RLOOP
       RD    INDEV
       STCH  RECORD,X
       TIX   K100
       JLT   RLOOP
       RSUB  ——→ Return from subroutine
```

```
INDEV   BYTE   X'F1'
RECORD  RESB   100      → size
ZERO    WORD   0
K100    WORD   100
```

① SIC program

ALPHA = BETA * GAMMA

```
    LDA  BETA
    MUL  GAMMA
    STA  ALPHA

BETA   RESW  1
GAMMA  RESW  1
ALPHA  RESW  1
```

② SIC/XE program

ALPHA = BE 4 * BETA - 9.

```
LDA  BETA
MUL  #4
SUB  #9
STA  ALPHA
```

```
LDA  ALPHA   BETA
LDS  #4
MULR S, A
SUB  #9
STA  ALPHA
```

SIC :-

```
LDA  BETA
MUL  FOUR
SUB  NINE     } — Memory access
STA  ALPHA
```

③ SIC → Swap ALPHA, BETA

```
    LDA  ALPHA
    STA  TEMP
    LDA  BETA
    STA  ALPHA
    LDA  TEMP
    STA  BETA

ALPHA  RESW  1
BETA   RESW  1
TEMP   RESW  1
```

④ a) SIC program to find sum of elements of an array : ALPHA that has 10 words.

b) SIC/XE.

a) →

```
        LDA  ZERO
        STA  SUM
        STA  INDEX
SUMLOOP LDX  INDEX
        LDA  ALPHA,X
        ADD  SUM
        STA  SUM
        LDA  INDEX
        ADD  THREE
        STA  INDEX
        COMP ARRSIZE
        JLT  SUMLOOP.
        :

INDEX   RESW 1
SUM     RESW 1
ALPHA   RESW 10.

ZERO    WORD 0
THREE   WORD 3
ARRSIZE WORD 30.
```

Try this !! →

ALPHA = BETA * GAMMA

GAMMA = ALPHA * BETA.

↓ ↙ ↓ MUL.

Repeated addition

AL (Source program) → ASSEMBLER (SIC, SIC/XE) → ML (Object program).

b)
```
    LDS #3
    LDT #10
    LDX #0
    LDA #0
    STA SUM
SUMLP. LDA ALPHA,X
    ADD SUM
    STA SUM
    ADDR S,X
    COMPR X,T
    JLT SUMLP
```

ALPHA * BETA = GAMMA.

```
    LDX ZERO
    LDA ALPHA
SUM → ADD GAMMA
    TIX BETA
    JLP SUM
```

```
    LDT BETA
    LDX #0
    LDS GAMMA
    LDA ALPHA
SUM → ADDR A,S
    TIXR T
    JLP SUM
```

```
    LDA #0
    STA GAMMA
```

```
    LDA ZERO
    STA GAMMA
    LDX ZERO
SUM → LDA ALPHA
    ADD GAMMA
    TIX BETA
    JLT SUM
```

```
SUM → LDA ALPHA
    ADD GAMMA
    STA GAMMA
```

7/2/22

Instruction Set -

m → memory address
n → integer btw 1 and 16
r1, r2 → register identifiers.
Paranthesis → denote the contents of register or memory location.

A ← (m, m+2) ⇒ Contents of memory locations m through m+2 are loaded into register A

m, m+2 ← (A) ⇒ Contents of register A are stored in the word that begins at address m

Letters in Notes :-

- P Privileged instruction
- X Instruction available only on (XE extended) version
- F Floating-point instruction
- C Condition code CC set to indicate result of operation (<, =, or >).

Prefix of '+' (F4)

m Mnemonic (+ADD ALPHA)	Format (SIC or SIC/XE)	Opcode	Effect	Notes
ADD m	3/4	18	A ← (A) + (m, m+2)	
ADDF m	3/4	58	F ← (F) + (m, m+5)	XF
ADDR r1, r2	2	90	r2 ← (r2) + (r1)	X
AND m	3/4	40	A ← (A) & (m, m+2)	
CLEAR r1	2	B4	r1 ← 0	X
COMP m	3/4	28	(A) : (m, m+2)	C
COMPF m	3/4	88	(F) : (m, m+5)	XFC
COMPR r1, r2	2	A0	(r1) : (r2)	X C
DIV m	3/4	24	A ← (A) / (m, m+2)	
DIVF m	3/4	64	F ← (F) / (m, m+5)	XF
DIVR r1, r2	2	9C	r2 ← (r2) / (r1)	X

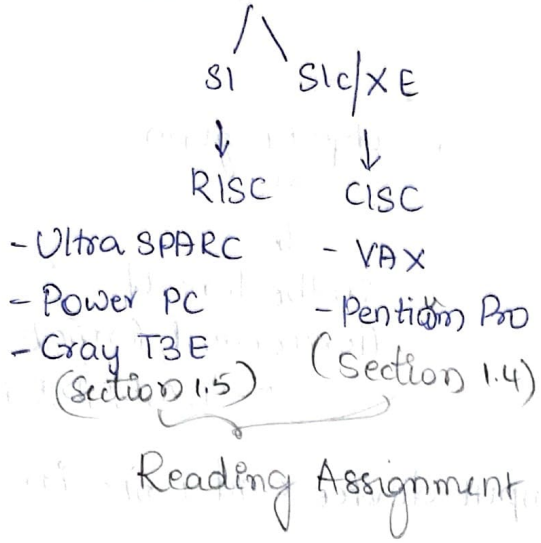
FIX	1	C4	A ← (F) [convert to integer]	XF
FLOAT	1	C0	F ← (A) [" " floating]	XF
HIO	1	F4	Halt I/O channel number (A)	PX
Jm	3/4	3C	PC ← m	
JEQ m	3/4	30	PC ← m iff CC set to =	
JGT m	3/4	34	" >	
JLT m	3/4	38	" <	
JSUB m	3/4	48	L ← (PC); PC ← m	
LDA m	3/4	00	A ← (m.m+2)	
RSUB	3/4	4C	PC ← (L)	
LDB m	"	68	B ← (m.m+2)	X
LDF m	"	70	F ← (m.m+5)	XF
LDL m	"	08	L ← (m.m+2)	
LDS m	"	6C	S ← (")	X
LDT m	"	74	T ← "	X
LDX m	"	04	X ← "	
LDCH m	"	50	A [rightmost byte] ← (m)	
LPS m	"	D0	Load processor status p. from information beginning	
MULm	"	20	A ← (A) * (m.m+2)	F
MULR r1,r2	2	98		
STA m	3/4	0C	m.m+2 ← (A)	
STB m	"	78	m.m+2 ← (B)	X
STCH m	"	54	m ← (A) [rightmost byte]	
STF m	"	80	m.m+5 ← (F)	XF
STI m	"	D4	Interval timer value ← (m.m+2)	PX
STL m	"	14	m.m+2 ← (L)	
STS m	"	7C	" ← (S)	X
STSW m	"	E8	" ← (SW)	P
STT m	"	84	" ← T	X
STX m	"	10	" ← X	
MULF m	3/4	60	F ← (F) * (m.m+5)	XF

JSUB READ
→ LDA ZERO
PC

we all not using → privileged instructions

NORM	1	C8	F ← (F) [normalized]	XF
OR m	3/4	44	A ← (A) | (m. m+2)	
RD m	11	D8	A [rightmost byte] ← data from device specified by (m)	P
RMO r1,r2	2	AC	r2 ← (r1)	X
RSUB	3/4	4C	PC ← (L)	
SHIFTL r1,n	2	A4	r1 ← (r1); left circular shift n bits. [In assembled instruction, r2 = n-1]	X
SHIFTR r1,n	2	A8	r1 ← (r1); right shift n bits, with vacated bit positions set equal to leftmost bit of (r1). [In assembled instruction, r2 = n-1]	X
SIO	1	F0	Start I/O channel no. (A); address of channel program is given by (s)	PX
SSK m	3/4	EC	Protection key for address m ← (A)	PX
SUB m	3/4	IC	A ← (A) - (m. m+2)	
SUBF m	3/4	5C	F ← (F) - (m. m+5)	XF
SUBR r1,r2	2	94	r2 ← (r2) - (r1)	X
SVC n	2	B0	Generate SVC interrupt. [In assembled instruction, r1 = n]	X
TD m	3/4	E0	Test device specified by (m)	PC
TIO	1	F8	Test I/O channel number (A)	PXC
TIX m	3/4	2C	X ← (X) + 1; (X) : (m. m+2)	C
TIXR r1	2	B8	X ← (X) + 1; (X) : (r1)	XC
WD m	3/4	DC	Device specified by (m) ← (A) [rightmost byte]	P

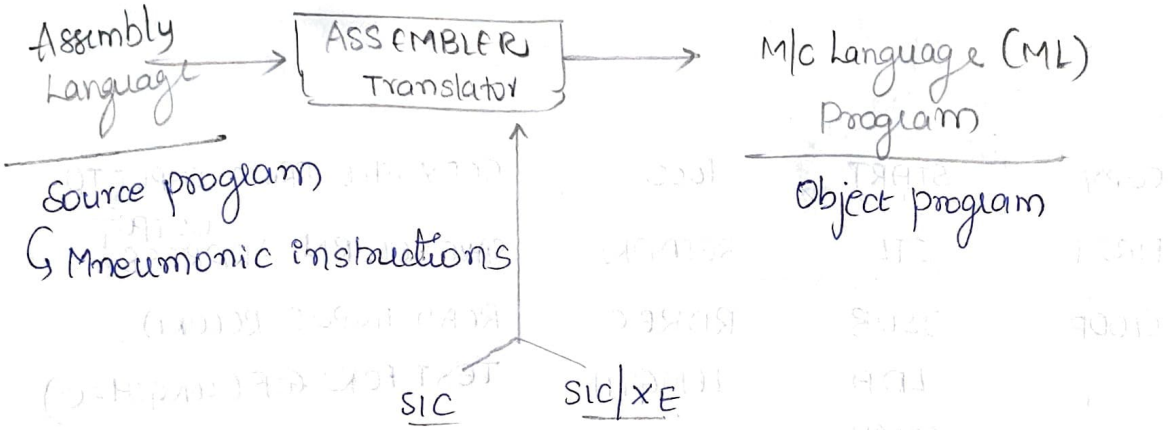
Machine Architectures - SP

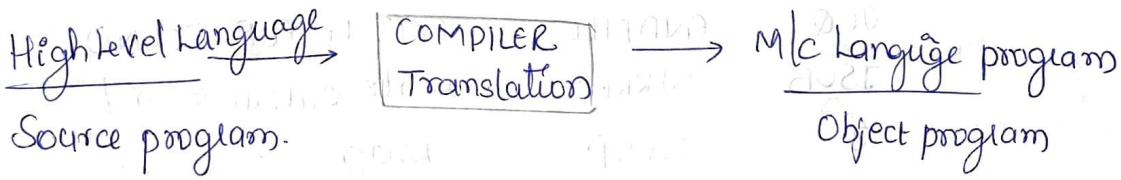


- Memory
- Registers
- Data formats
- Instruction formats
- Addressing modes
- Instruction set
- I/O.

ASSEMBLER

Assembler is a Translator





9/2/22

Assembler

SIC

Source program (AL) → Assemblee (Translator) → object program (ML) ⇓ in the form of Hexdecimal format 0-9, A, B, C, D, E, F

→ Read a record from standard input device and write in standard o/p device

opcode — These are standard not like 'I'

1	2	3	4
labels given by the programmer	ASSEMBLER DIRECTIVES	OPERANDS	: → comment

SIC & START & END are ASSEMBLER DIRECTIVES

it is not executable instruction, just assembler directive

1000	COPY	START	1000	COPY FILE FROM INPUT TO OUTPUT
1000	FIRST	STL	RETADR	SAVE RETURN ADDRESS 141033
1003	CLOOP	JSUB	RDREC	READ INPUT RECORD 482039
1006		LDA	LENGTH	TEST FOR EOF (LENGTH = 0) 00 1036
1009		COMP	ZERO	281030
100C		JEQ	ENDFILL	EXIT IF EOF FOUND 301015
100F		JSUB	WRREC	write output record 482061
1012		J	CLOOP	Loop 3C1003
1015	ENDFILL	LDA	EOF	insert end of file Marker 00102A
1018		STA	BUFFER	0C1039
101B		LDA	THREE	Set length = 3 00102D
101E		STA	LENGTH	0C1036
1021		JSUB	WRREC	write EOF 482061
1024		LDL	RETADR	Get return address 081033
1027		RSUB		return to caller 4C0000
102A	EOF	BYTE	C 'EOF'	454F46
102D	THREE	WORD	3	000003 } constants
1030	ZERO	WORD	0	000000
1033	RETADR	RESW	1	

MAIN

LOC → location

```
1036 LENGTH   RESW   1      Length of record   (it is decimal)
1039 BUFFER   RESB   4096   4096-Byte Buffer Area
                            In Hexadecimal → 1000
+ 1000
      :   SUBROUTINE TO READ RECORD INTO BUFFER
2039 RDREC    LDX    ZERO   Clear loop counter 041030
              LDA    ZERO   Clear A to ZERO    001030
203C RLOOP    TD     INPUT  Test input device  E0205D
203F          JEQ    RLOOP  Loop until Ready   30203F
2042    D8205D RD     INPUT  READ character into register A
2045    281030 COMP   ZERO   Test for end of record (x'00')
2048    302057 JEQ    EXIT   EXIT Loop if EOR.
204B 0101 0100|1|001 0000 0011 100| ← STCH BUFFER,X  Store character in buffer
204E    549039 ← TIX  MAXLEN Loop unless max length
2051    2C205E        JLT    RLOOP  has been reached.
2054    38203F
2057 EXIT 101036 STX  LENGTH Save record length
205A    4C0000 RSUB           Return to caller
205D INPUT    BYTE   X'F1'   Code for input device F1
205E MAXLEN   WORD   4096    (3 bytes) 001000
(input is single byte)  constant, so only 3 is incremented
      :   SUBROUTINE TO WRITE RECORD FROM BUFFER
2061 WRREC 041030 LDX  ZERO   Clear Loop counter
2064 WLOOP E02079 TD   OUTPUT Test output device
2067    302064 JEQ    WLOOP  Loop until ready.
206A    509039 ← LDCH BUFFER,X  Get character from buffer
206D    DC2079 WD     OUTPUT  & WRITE character
2070    2C1036 TIX    LENGTH  Loop until all characters
2073    382064 JLT    WLOOP   Have been written
2076    4C0000 RSUB           Return to caller
2079 OUTPUT   05   BYTE X'05'  Code for output device
              END    FIRST
```

Length is 207A (2079 byte also occupied)

- Instructions
- Assembler directives
  - START, END
    ↓
  - BYTE
    - Generating character - Hexadecimal constant

      BYTE C'N'

      BYTE C'NIT GOA'
  - WORD
    - Generating 1 word constant
  - RESB
    - Reserves indicated no. of bytes for data.
  - RESW
    - Reserves indicated no. of words.

SIC Assembler

- Mnemonic instructions → Object code
- Symbolic operands → Machine address.

  RETADR 1033.
- Constants → values

  EOF → 45 4F 46

  THREE WORD 3 → 000003
- Object program generation

These all are the roles of assembler

Instruction format (SIC)

OPCODE	x	Address
8	1	15

BUFFER, X → X=1

no, X → X=0.

10/2/22

→ Sequential process : Forward reference

↳ opens the file, read it, write (sequentially)

It is not sequential. (pass process).

→ In the first instruction, it can't generate 141033 . becoz retadr address is not known at that time.

- Pass 1
- Pass 2.

Copy is the name of the program

```
HCOPY  001000001077A
T001000(1E)(30 bytes)141033482039001036281036281036301015482061
T00101E(15)0C1036482061081033 4C0000 454F46 3C1003 001 02A0C1039001 02D
                                                    000000 300000
T002039(1E)041030001030E0205D30203FD8205D28103020575490392C
  205E38203F
T002057(1D)1010364C0000F1001000041030E02079302064509039DC207
  92C1036
T002073073820644C000005
```

Object program : Fixed/specific E001000

→ 3-records

① Header record

col 1 : H

(6) col 2-7 : (6 columns) correspond to program name (2 spaces) COPY__

(6 cols) col 8-13 : Starting address of object program 001000 (Hexa decimal)

(6 cols) Col 14-19 : length of object program in bytes. (Hexadecimal) 00107A

1000-2079

2080 207A.

207A
− 1000
107A

② Text records

col 1 : T

{69} col 2-7 : starting address of object code in this record.

{23} 8-9 : Length of object code in this record in bytes

10-69 : object code, represented in hexadecimal

{60} 2 columns per 1 byte

1st T 10 instructions
each is 3 bytes (SIC)
30 bytes (1000)

30 bytes = 1E
16^1 16^0
16 + E = 16 + 14 = 30.

2nd T 7 instructions, 101E
↳ 7 × 3 = 21 bytes
↳ 15.

3rd T. 2039, 10 instructions / 10 records
10 × 3 = 30 bytes
↳ 1E.

4th T 2057, 10 & 9 records
9 × 3 = 27
1 × 1 = 1
28 bytes
↳ 1C

5th T 2073, 3 records
2 × 3 = 6
1 × 1 = 1
7 bytes ↳ 07.

③ End records

col 1 : E

col 2-7 : Address of 1st executable instruction

1000

E001000

11/2/22

SIC Assembler

```
1000   ARRSUM  START  1000
1000           LDA    ZERO      00 13A8
1003           STA    INDEX     0C 1021
1006   ADD LP  LDX    INDEX     04 1021
1009           LDA    ALPHA,X   00 9024
100C           ADD    BETA,X    18 9150
100F           STA    GAMMA,X   0C 927C
1012           LDA    INDEX     00 1021
1015           ADD    THREE     18 13AE
1018           STA    INDEX     0C 1021
101B           COMP   K300      28 13AB
101E           JLT    ADDLP     38 1006

1021   INDEX   RESW   1
1024   ALPHA   RESW   100
1150   BETA    RESW   100
127C   GAMMA   RESW   100
13A8   ZERO    WORD   0    000000
13AB   K300    WORD   300  00012C
13AE   THREE   WORD   3    000003
13B0           END    FIRST.
```

00 1 001 0 24

16 | 300
16 | 18 - 12
1 - 2

12C

1024 → 100 words → 100 × 3 = 300 bytes

1024
12C
1150
12C
127C
12C
13A8
3
B

24 = 16+8

A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

13AE
13AF
13B0

13B0
−1000
3B0

8 bit
1 bit

00 | 1 0 2 4 (15 bit)

0000 0000 | 1 | 001 0000 0010 0100

0000 0000 1001 0000 0010 0100

0 0 9 0 2 4

Object program

H^ARRSUM^001000^0003B0

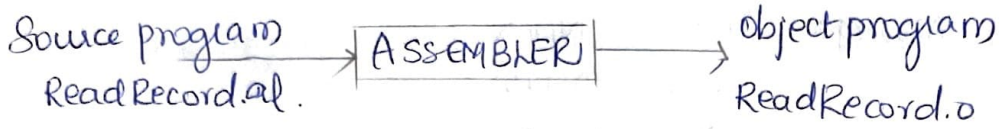
T^001000^1E^00103A8^0C1021^041021^009024^189150^0C927C^
001021^1813AE^0C1021^2813AB

T^00101E^03^381006

T^0013A8^09^000000^00012C^0000013

E^001000.

17/2/22



→ Sequential → Not in one go.

→ 2 passes through the program.

Pass 1 :

→ Assignment of addresses to instructions and symbols.

→ Save the addresses assigned to symbols

→ processing of assembler directives. (START & END)

Pass 2 :

→ Mnemonic instructions → object code

→ BYTE, WORD → represented in ML

→ Completes processing of assembler directives [which is not completed in pass 1 is done here]

→ object program.

Data structures for SIC Assembler

① Location counter [LOCCTR] → assign addresses.

② Operation Code Table [OPTAB]

③ Symbol Table [SYMTAB]

② OPTAB

It will have M/c code corresponding to Mnemonic instruction

M/c code → Mnemonic instruction

Mnemonic instruction	M/c code
LDA	00
STL	14
JSUB	48
⋮	
ADDR	xx

In SIC/XE → 2 → format is also mentioned
3/4

Pass 1 :

- it is used for validating Mnemonic operation code in source program

LDA LENGTH
LEA LENGTH
wrong input → error

in SIC/XE → to find instruction length for incrementing.

Pass 2 :

- Translation LDA LENGTH ⇓ 00

(3) SYMTAB

- Symbol Table
- Name and value for each symbol used.
  ↓ (address)

Symbol	value
COPY	10030
FIRST	
CLOOP	1003
ENDFIL	1015
⋮	

→ complete the table

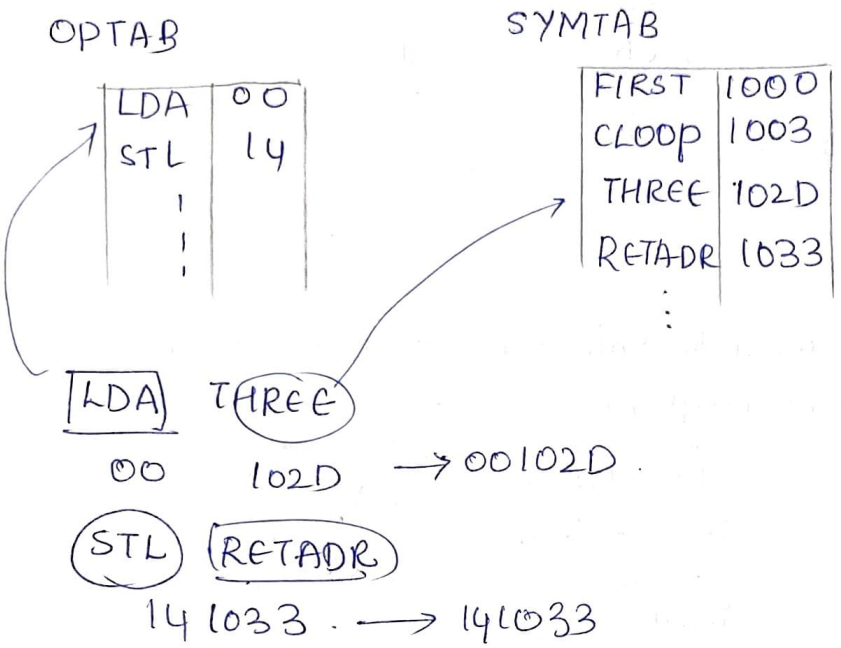
Pass 1 :

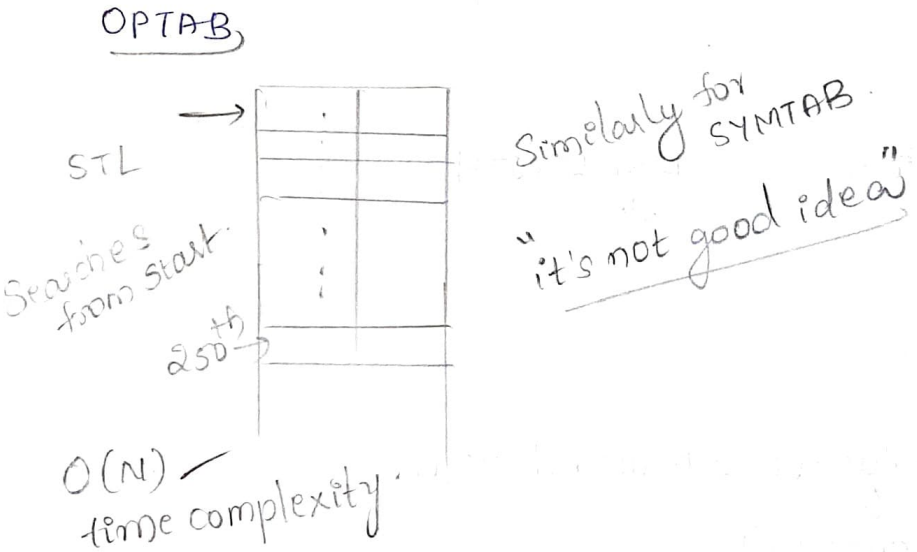
- Name and value, are/is inserted
  Pairs

if 2 CLOOP's are there, before insertion it will search and if it a found CLOOP already inserted the genuate ERROR.

Pass 2 :

- Translation





Hash Table

↳ O(1)

Constant time to access any entry in OPTAB & SYMTAB.

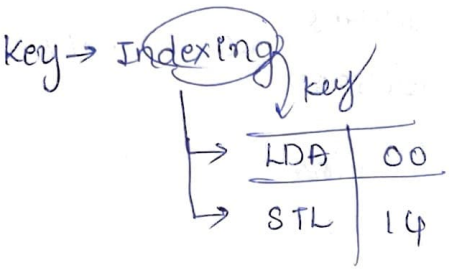
SIC Assembler

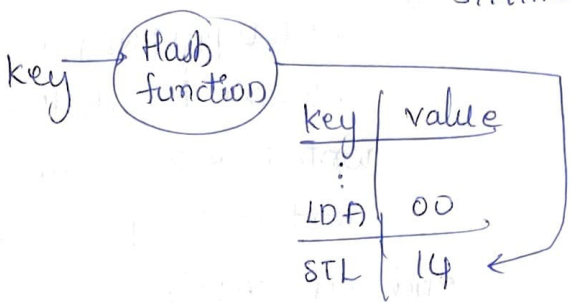
18/2/22

DS for SIC Assembler

- Location counter [LOCCTR]
- Operation code Table [OPTAB]
- Symbol table [SYMTAB]

Hash table [for both OPTAB SYMTAB]





hf (LDA) → index of LDA

→ Hash function

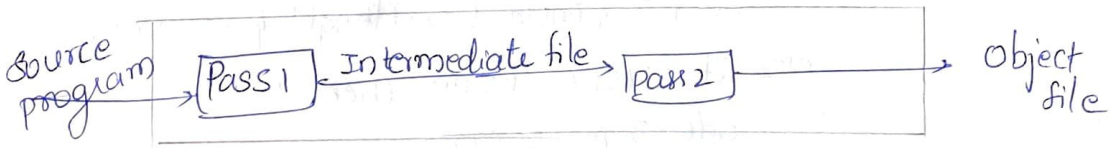
key → its corresponding index

File structures, An object oriented approach with c++ by Michael J. Folk, ...

- collision $hf(k_1)$, $hf(k_2)$ → index

hf (LDA)
hf (LDB)

- Equally complex
- OPTAB
- SYMTAB



- Source program + location counter value error-code

opcode of corresponding mnemonic

LABEL OPCODE OPERAND

Algorithm for pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
     begin
        save #[OPERAND] as starting address        LOCCTR ← 1000
        initialize LOCCTR to starting address
        write line to intermediate file
        read next input line
     end {if START}
  else
     initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              if there is a symbol in the LABEL field then
              begin
                 search SYMTAB for LABEL
                  if found then
                     set error flag (duplicate symbol)
                  else
                     insert (LABEL, LOCCTR) into SYMTAB
              end {if symbol}
           → search OPTAB for OPCODE
           if found then
              add 3 {instruction length} to LOCCTR
           else if OPCODE = 'WORD' then
              add 3 to LOCCTR
           else if OPCODE = 'RESWD' then
              add 3 * #[OPERAND] to LOCCTR
```

```
                else if OPCODE = 'RESB' then
                    add #[OPERAND] to LOCCTR
                else if OPCODE = 'BYTE' then
                    begin    add 1 to LOCCTR
                        find length of constant in bytes
                        and length to LOCCTR
                    end {if BYTE}
                else
                    Set error flag (invalid operation code)
            end {if not a comment}
        Write line to intermediate file
        read next input line
    end {while not END}
    Write last line to intermediate file
    Save (LOCCTR - starting address) as program length
end {pass 1}
```

Pass 2 :

```
begin
    read first input line {from intermediate file}
    if OPCODE = 'START' then
        begin
            write listing line
            read next input line
        end {if START}
    write Header record to object program
    initialize first Text record
    while OPCODE ≠ 'END' do
        begin
            if this is not a comment line then
                begin
                    seach OPTAB for OPCODE
                    if found then
```

```
begin
    if there is a symbol in OPERAND field then
        begin
            search SYMTAB for OPERAND
            if found then
                store symbol value as operand address
            else
                begin
                    store 0 as operand address
                    set error flag (undefined symbol)
                end
        end {if symbol}
    else
        store 0 as operand address
    assemble the object code instruction
end {if opcode found}
else if OPCODE = 'BYTE' or 'WORD' then
    convert constant to object code
if object code will not fit into the current Text record then
    begin
        write text record to object program
        initialize new Text record
    end
    add object code to Text record
end {if not comment}
write listing line
read next input line
end {while not END}
write last Text record to object program
write end record to object program
Write last listing line
end {pass 2}
```

21/2/22 Exercise:

4000	SUM	START	4000		
4000	FIRST	LDX	ZERO		045788
4003		LDA	ZERO		005788
4006	LOOP	ADD	TABLE,X		18C015
4009		TIX	COUNT		2C5785
400C		JLT	LOOP		384006
400F		STA	TOTAL		0C578B
4012		RSUB			4C0000
4015	TABLE	RESW	2000.		
5785	COUNT	RESW	1	16	6000
5788	ZERO	WORD	0	16	375-0 000000
578B		RESW	1	16	23-7
		END			1-2

578D

4015
8000
C015

1770
4015
5785

SYMTAB

FIRST	4000
LOOP	4006
TABLE	4015
COUNT	5785
ZERO	5788

4015
$2^3\ 2^2\ 2^1\ 2^0$
0100
8+4=12

Program length = 578E − 4000 = 178E+1

H^SUM---^004000^00178E

T^004000^15^045788^005788^18C015^2C5785^384006^0C578B^4C0000

T^005788^03^000000

E^004000

SIC/XE Assemblee

Source Statement

COPY START 0 COPY FILE FROM INPUT TO OUTPUT

Differences

- Starting address
- F4 instructions +JUSB
- #B → immediate addressing
  #LENGTH
- @RETADR → Indirect addressing
- CLEAR → F2
- B,S → registers are used
- COMPR → F2 instructions.

F2 → +2
F3 → +3
F4 → +4

→ Register-to-Register
→ Immediate Address
} Faster process (No memory access)

A	0
X	1
L	2
B	3
S	4
T	5
F	6
PC	8
SW	9

```
0000  COPY   START  0
0000  FIRST  STL-14  RETADR
0003          LDB-68  #LENGTH
              BASE    LENGTH
0006  CLOOP   +JSUB   RDREC
000A          LDA     LENGTH
000D          COMP-28 #0
0010          JEQ-30  ENDFIL
0013          +JSUB   WRREC
0017          J-3C    CLOOP
001A  ENDFIL  LDA     EOF
001D          STA-0C  BUFFER
0020          LDA     #3
0023          STA     LENGTH
0026          +JSUB   WRREC
002A          J       @RETADR
002D  EOF     BYTE    C'EOF'
0030  RETADR  RESW    1
0033  LENGTH  RESW    1
0036  BUFFER  RESB    4096
1036
1000

: SUBroutine to read record into Buffer

1036  RDREC   CLEAR   X
1038          CLEAR   A
103A          CLEAR   S
103C          +LDT    #4096
1040  RLOOP   TD-E0   INPUT
1043          JEQ-30  RLOOP
1046          RD-D8   INPUT
1049          COMPR   A,S
104B          JEQ-30  EXIT
104E          STCH    BUFFER,X
1051          TIXR-B8 T
1053          JLT-38  RLOOP
1056  EXIT    STX-10  LENGTH
1059          RSUB-4C
```

TA: 0033 − 0003 = 0030

0033 − 0006 = 002D

0001 01 110010 0000 0010 1101

1 + 202D

6 [10010010] 02D

69202D

4B101036

0 [00110010] 026 → 032026

2 [10 01 0000] 000 → 290000

3 [00 11 0010] 007 → 332007

4B10105D

3 [11 110010] FEC

0 [00 11 0010] → 3F2FEC

0 [11 11 0010] 010 → 032010

0 [00 01 0000] 016 → 0F2016

0 [11 11 0010] 003 → 010003

0 [11 11 0010] 00D → 0F200D

4B10105D

3 [11 10 0010] 003 → 3E2003

454F46 → Constant value

B410

B400

B440

75101000

E [00 11 0010] 019 → E32019

3 [00110010] FFA → 332FFA

D [10 11 0010] 013 → DB2013

A004

3 [00 11 0010] 008 = 332008

57C003

B850

3 [10 11 0010] FEA = 3B2FEA

1 [00 11 0100] 000 = 134000

4 [11 11 0000] 000 = 4F0000

```
+3  105C   INPUT   BYTE  X'F1'                F1
+1  BYTE:1  : SUBROutine to write record from buffer
    105D   WRREC   CLEAR  X                    B410
+2  105F           LDT:74 Length               → [01 11 0100] 000 : 774000
+3  1062   WLOOP   TD:E0 = X'05' OUTPUT        E [00 11 0010] 011 : E32011
+3  1065           JEQ:30 WLOOP                3 [00 11 0010] FFA : 332FFA
+3  1068           LDCH:50 BUFFEX,X            5 [00 11 1100] 003 : 53C003
+3  106B           WD:DC = X'05' OUTPUT        D [11 11 0010] 008 : DF2008
+3  106E           TIXR   T                    B850
+2  1070           JLT:38 WLOOP                3 [10 11 0010] FEF : 3B2FEF
+3  1073           RSUB:4C                     4F0000
                   END    FIRST
   1076  OUTPUT    = X'05' BYTE X'05'          05
                    END    FIRST
```

Format 2:

OPCODE	R1	R2
8	4	4

Example: CLEAR X, A → B410

B4 1 0 A

CLEAR A → B400

COMPR A,S → B404

A 0

Format 4 :- 32 bit

opcode	n	i	x	b	p	e	Address
6	1	1	1	1	1	1	20

n: =0, =1

CLOOP +JSUB RDREC

48 1036

0100 10 1 1 0001 0000 0001 0000 0011 0110

4 B 1 0 1 0 3 6

+JSUB WRREC

↓ ↓

48 105D

0100 10 11 0 0 0 1 0000 0001 0000 0101 1101

4 B 1 0 1 0 5 D

+LDT #4096

↓

74

hexa decimal

01 000

0111 01 0 1 0 0 0 1 0000 0001 0000 0000 0000

7 5 1 0 1 0 0 0

n i x b p e

Format 3:

opcode	n	i	x	b	p	e	Displacement
6	1	1	1	1	1	1	12

Reference

PC B

↓ ↓

PC-relative Base-relative

b=0, p=1 b=1, p=0.

EA/TA = (PC) + disp

disp = EA - (PC)

STL RETADR

↓ ↓

14

TA = RETADR address = 0030

PC = next instruction address = 0003

$$\begin{array}{r} 0030_{16} \\ -0003 \\ \hline 002D \end{array}$$

0001 01 1 1 0 0 1 0 0000 0010 1101

n i x b p e

1 7 2 0 2 D

n, i = 0,0 / 1,1 → Simple

0 1 → immediate

1 0 → indirect

x → indexed

e → format

b, p → base & pc

b=0, p=1 → PC

b=1, p=0 → base

23/2/22

→ Relative addressing

- Address – 12 bits
- 20-bit ⅋ Displacement < PC / B

→ Case 1: Is the instruction → to F4.
or
whether
transformed

④ JSUB ....

← STL RETADR

→ Case 2:

Tries to first use

→ PC Relative addressing

Disp = TA/EA – (PC)

↳ 12 bit field.

→ Case 3: → Base Relative

→ BASE → X

(B)

Disp = TA – (B)

12 bits

⇓

Error.

PC/B relative if not possible, assembler throws an error not take F4.

(Disp) → > 12 bits

Ex:

STCH BUFFER, X.

↓ ↓

54 0036.

PC → 1051

PC relative:

Displacement = TA – PC.

= 0036 – 1051

PC relative addressing is not possible

```
    4
  10 3 6 17
- 0 0 3 6
= 1 0 1 B
```

```
  F F F F
- 1 0 1 B
  E F E 4
    + 1
  ⑤ F E 5
```

15-11 = 4

16's complement

length > 12

so pc not possible

Base relative:

Disp = TA - (B)

Disp = 0036 − 0033 = 0003.

LDB #LENGTH

if normal LENGTH → calculate displacement then value of length is taken.

(B) ← 0033

Length → *

0101	01	1	1	1	1	0	0	0000 0000 0011
	n	i	x	b	p	e		
5	7		C				0 0 3	

TA = (B) + Displacement + (X)

0033
0003
0036
+ (X)

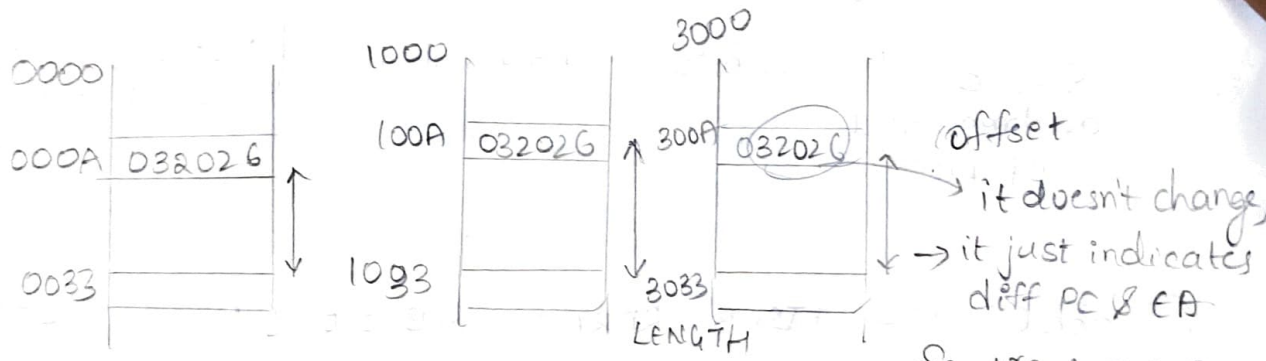
"object code"

In SIC, address fields changes wrt starting address, No scope of program relocation.

24/2/22

Program Realocation

000A LDA LENGTH 032026.



So, program can relocate to anywhere.

Program is relocatable, anywhere.

not like SIC.

→ But there is an issue in this,

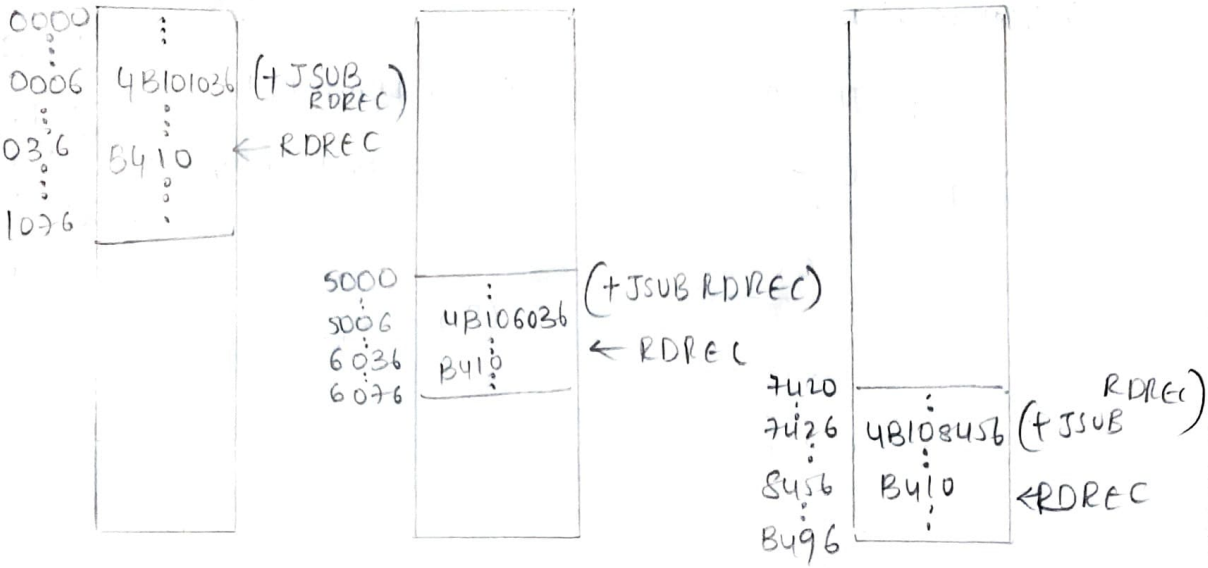
Issue with Format 4.

CLOOP +JSUB RDREC

1036 → wrt 0 as starting address.

like SIC

if it changes, then object code will also change.



Some kind of modification for address field of format 4.

↳ (during loading only we can do)

⇓ Modification record.

↓ Means through object program

COL 1 : M

COL 2-7 : Starting location of the address field to be modified (6) relative to be beginning of the program

col 8-9 : Length of the address field to be modified in half bytes. (4) (2)

Size of address field of F4. ← 20 bits : 5 half bytes

[Not possible to write in bytes (8)]

① 0006 CLOP +JSUB RDREC 4B101036 ← M00080705.

0006 + 1 = 000007

it covers some portion of address field

So it is taken as starting location of address field to be modified.

(2) 0013 +JSUB WRREC 4B10105D → M00001305

Object program :

H^COPY ^000000^001077

T^000000^1D^172027^692027^4B10105E^032026^290000^
332007^4B10105D^3F2FEC^032010

T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^
454F46

T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^
DB2013^A004^332008^57C003^B850

T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^
E32011^332FFA^53C003^DF2008^B850

T^001070^07^3B2FEF4^4F0000^05

M^000007^05
M^000014^05
M^000027^05
E^000000

→ This program itself is not machine independent

25/2/22

Purpose of program Relocation -
To support loading multiple programs at a time.

Machine Independent Assembler feature → These extra features make the program machine independent.

ENDFIL LDA =C'EOF'

BUFFEND EQU *
MAXLEN EQU BUFFEND-BUFFER

=X'05'

① LITERAL

- Constant - use
- Fixed value.

Pseudocode EQU is evaluated during first pass.

```
001A  ENDFIL  LDA   EOF
      :
002D  EOF     BYTE  C'EOF'
```

Adding constants as literal will avoid seperately making a label.

```
001A  ENDFIL  LDA   =C'EOF'
      :                 ↓ literal → placed in Literal pool
002D  EOF*    =C'EOF'
```

How to process the literals ?

- we had already used placing contant values directly
  ↓
  #0, #3 } immediate addressing

LITTAB: Hash table

Literal Name	operand value	length

LTORG : Directive.

Literal → convenient way value

every literal is considered as symbol only.

Pass 1

- whenever literal is encountered, =C'EOF' it is pushed onto LITTAB.
- Whenever LTORG is encountered then it is End of program.
  ↓
  Scan the LITTAB.
  ↳ for each entry in LITTAB, assign the address. as well as increment the location counter accordingly.

Pass 2

when the object code for an instruction is generated.

```
TD   =X'05'
 :
WD   =X'05'   → duplicate literal.
              but it can't enter multiple times.
```

② Symbol Defining statements

```
1036  BUFEND  EQU   *          → present value of location counter
1000  MAXLEN  EQU   BUFEND-BUFFER   → Expression
```

these are not addresses. Values assigned to them from the expressions

Two ways of creating a symbol.

① - Symbol (or label) given to an instruction [CLOOP, - -]

② Data.

Creating symbol using Assemble directive called EQU

↓ Stands for Equate.

```
SYMBOL  EQU  VALUE
```

MAXLEN = outcome of expression BUFEND-BUFFER.

Usage of symbol defining statement

```
A      EQU  0
X      EQU  1
 :
INDEX  EQU  R1   → R1 become index register
```

③ Expressions → reduce no. of variables & labels.

- literals, operands
- -, +, *, / ↳ Evaluate

↑ special expression (not multiplication)
- Present value of location counter.

LOCCTR — Location Counter

2/3/22

Machine Independent Assembler Features

- Literals
- Symbol defining symbols statements
- Expression

(4). Program blocks.

- Entire program is considered as single entity. or treated.
- Instructions and data → different order.
  order is diff from how they appear in source Program.
  - Rearranged.
    ↳ object program } Memory loaded version of the program.

→ Assemble Directive : USE

- 3 program blocks.
  ① unnamed block. : which comprises of all Executable instructions
  USE

  ② CDATA
  → All data areas of that are correspond to a few words or less in length data

  ③ CBLKS
  - Data areas consisting of larger blocks of memory.

Pass 2

when the object code for an instruction is generated.

```
TD    =X'05'
 :
WD    =X'05'   → duplicate literal.
                 but it can't enter multiple times.
```

② Symbol Defining statements

present value of location counter

```
1036  BUFEND  EQU   *
1000  MAXLEN  EQU   BUFEND-BUFFER   → Expression
                                      - it is an another feature
```

these are not addresses. Values assigned to them from the expressions

Two ways of creating a symbol.

① – Symbol (or label) given to an instruction [CLOOP, - -]

② Data.

Creating symbol using Assemble directive called EQU

Stands for Equate.

```
SYMBOL  EQU  VALUE
```

MAXLEN = outcome of expression BUFEND-BUFFER.

Usage of symbol defining statement

```
A      EQU  0
X      EQU  1
 :
INDEX  EQU  R1   → R1 become index register
```

③ Expressions → reduce no. of variables & labels.

- literals, operands
- −, +, *, / & Evaluate

special expression (not multiplication)
- Present value of location counter.