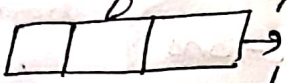
Few more important topics related to PC questions.

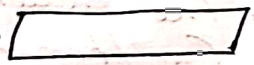
Q. Write a SIC/XE Program for arithematic operations.
OR write SIC Program for arithematic operations.

Notes: Only for understanding → we studied 2 hypothetical computer in system programming.
We also came through few assembler directives. For eg:

START → starting of the program.
END → End of the program
WORD →  → we know 3 consecutive byte form a word → going to store something in it.
int a = 5;

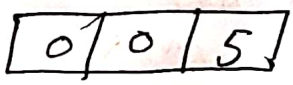
RESW →  → allocating memory but nothing is stored at that instance, reserved for future.
int a;

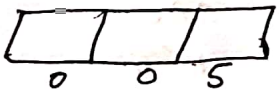
BYTE →  → similar to word, it will have some data.

RESB →  → similar to reserve, memory reserved.

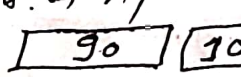
Now consider the following program. Data Movement:

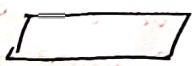
```
LDA   FIVE
STA   ALPHA
LDCH  CHARZ
STCH  C1

ALPHA  RESW 1   ---> going to reserve one word for ALPHA.
FIVE   WORD 5   ---> going to store 5
CHARZ  BYTE C'Z' -> Data part / Declaration part.
C1     RESB 1
```

So, first line, ALPHA RESW 1 → ALPHA 

FIVE 

Now, look first 2 lines of instructions:
Load into A, value of FIVE.
Store into A, ALPHA. So...

Now, Last 2 lines of instructions: → Alphabets are stored in form of 8-bit ASCII codes.
CHARZ BYTE 'C'Z' 

→ CHARZ.
C1 RESB 1 

When you are loading a number, we use LDA
" " " " character, " " LDCH.

So, LDCH CHARZ → load to accumulator character CHARZ.
STCH 1 → store to the accumulator.

So, C1 | 1001 0000 |
Z

Data Movement for SIC/XE.

```
LDA   #5
STA   ALPHA
LDA   #90
STCH  C1.
```

A(A) | 0 | 0 | 5 |

Four instructions

#5 → immediate value, so load directly load to Accumulator.

```
ALPHA  RESW 1
C1     RESB 1
```
→ only 2 declarations.

ALPHA | 0 | 0 | 5 | → reserve word

C1 | Z | → reserve byte.

- Remember, when we are storing value to the accumulator, we ~~dont~~ store ~~convert~~ it ~~into~~ directly. (arithematic value)

LDA stores → Number

LDCH stores → character.

While converting LDA #90 and storing the character, we have to convert it to character.

SIC - Arithematic Operation:
Addition / Subtraction / Multiplication / Division / ----

```
LDA  ALPHA
ADD  INCR
SUB  ONE
STA  BETA
LDA  GAMMA
ADD  INCR
SUB  ONE
STA  DELTA
:
```

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

1→ONE | 0 | 0 | 1 |

→ALPHA | | | |

→BETA | | | |

→GAMMA | | | |

→DELTA | | | |

→INCR | | | |

* All Alpha, Beta, Gamma, Delta & INCR, Reserve word as shown here.
* We have Reserve 5 words in the memory.

P.T. 0/1

Now program starts like this.

1) A ← Alpha
2) A ← Alpha + INCR
3) A ← Alpha + INCR - ONE
4) BETA ← ALPHA + INCR - ONE
5) A ← GAMMA
6) A ← GAMMA + INCR
7) A ← GAMMA + INCR - ONE
8) DELTA ← GAMMA + INCR - ONE //

Imp. SIC/XE Arithmatic Operation.

```
LDS   INCR
LDA   ALPHA
ADDR  S, A
SUB   #1
STA   BETA
LDA   GAMMA
ADDR  S, A
SUB   #1
STA   DELTA
:
:
ALPHA   RESW  1
BETA    RESW  1
GAMMA   RESW  1
DELTA   RESW  1
INCR    RESW  1
```

[Remember, whenever you perform arithmatic operations, value will be placed on accumulator].

1) S ← INCR.
2) A ← ALPHA
3) ADDR A → INCR + ALPHA - 1
4) STA ← BETA. So, BETA = ~~ALPHA +~~ INCR + ALPHA - 1
5) A ← GAMMA
6) ADDR A ← INCR + GAMMA - 1
7) DELTA → INCR + GAMMA - 1.

[ADDR ko value A me store hogi]

Similarly u can try data movement for SIC & SIC/XE. Looping in Book.

Difference between Linking loader and Linkage Editor.

Linking Loader	Linkage Editor.
1. Performs all linking and relocation operations, including library search and loads the linked program into memory for execution.	1. Produces a linked version of a program which is normally written to a file or library for later execution.
2. Suitable when a program is reassembled for nearly every execution.	2) Suitable when a program is executed many times without reassembled.
3. Resolution of external reference and library searching is performed more than once.	3) Here it is performed only once.
4) Linking loaders perform Linking operations at load time	4). Linkage editors perform linking operations before the program is loaded for execution.
5). No need of relocating loader.	5) Relocating loader loads the load module into memory.
6). Loading may requires 2 passes.	6) Here only one pass during loading
7). When program is in development stage then at that time the linking loader can be used.	7). When program is finished or when library is built, then linking editor can be used.

8)

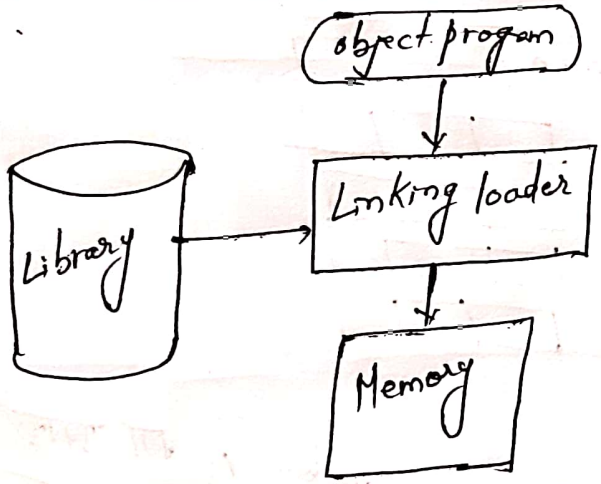


Fig: Linking Loaders.

9)

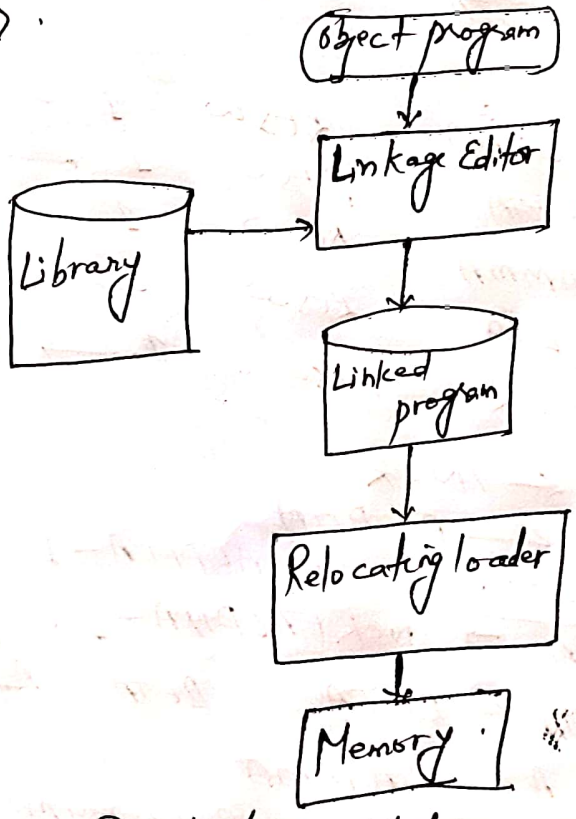


Fig: Linkage Editor.

∴ During class lecture, I had explained the figures but not given it's notes. So, refer this + the notes which I have sent of Unit 3 ||

• Refer Notes: for [Automatic Library Search, Loader Options.] → These are machine independent Loader features.

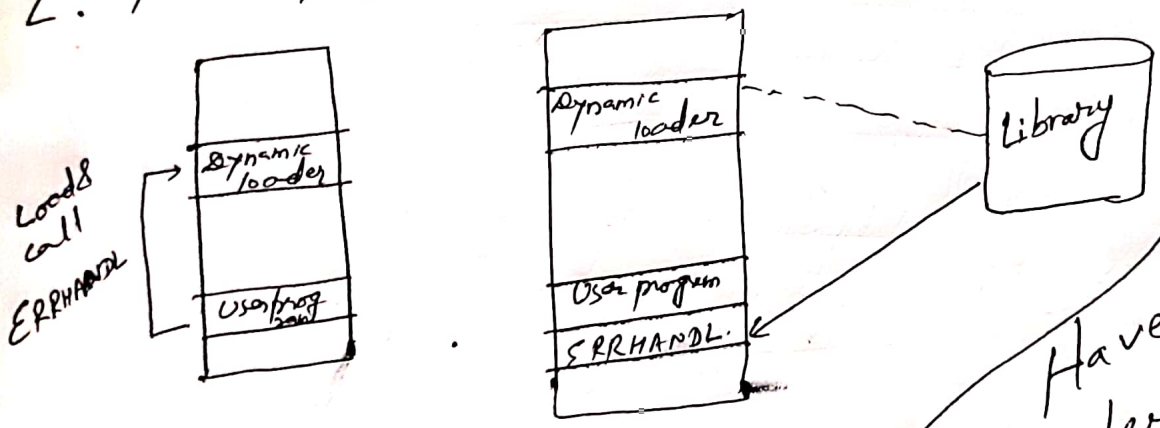
I have explained in class, now make short note by yourself.

Loader Design Options:

1. Linking Loaders.
2. Linkage Editors.
3. Dynamic Linking.

③ Dynamic Linking: It allows several executing programs to share one copy of a subroutine or library. It provides the ability to load the routines only when (and if) they are needed. The subroutine is loaded and linked to the rest of the program when it is first called - usually called dynamic linking.

[Refer to page No: [163, 164, 165] on your Book]



Suppose
① Requires a function
② Ask dynamic loader (O.S)
③ O.S searches in Library & provides the function ERRHANDL

④ [User prog / ERRHANDL] → once completed its operation, the program, returns the control to O.S.

⑤ Now program doesn't go to O.S., it asks for the User program to get the function ERRHANDL.

Have you remembered any thing from class lecture about Dynamic Linking. If not to remember it