UNIT : II

Assembler

Date.....
Page.....

* System
It is a collection of many programs
It has two types

1) System software
These program assist general user application programs.

Ex- operating system, assembler etc.

2) Application software
These are the software develop for the specific goals

Ex- Media player, adopt readers, wood processor

* System Programming
It consist of a variety of programs that supports the operations of computer. This software make it possible for the user to focus on an application or the other problems to be solve.

Ex- compiler, loaders, microprocessor, and operating system.

* Components of system programming

1) Assemble
2) loader
3) compiler
4) Microprocessor

source → [translator] → object program
↓
[loader] → [object program ready to execute Memory]

Micro is a single line abbreviation for group of statements.

Ex

MARO - Start of defination

INCK - Micro name

A1 , Data
A2 Data
A3 Data
} Sequence of instruction to be abbreviated.

MEND - END of defination.

* Operating System

- operating system is an interface between the user and hardware. It need resources (memory, processor, Io device) to complete task.

• Function of OS
1) Batch OS
2) Multi programming OS

* Assembler

→ It is a system software which translate assembly language into Machine language.

→ Assembly language is a kind of low level programming language which uses symbolic codes or mnemonics or instructions.

Ex ADD, SUB, LDA.

• Application of assembly language
It use direct hardware manipulation typical uses are device (CD, hard disks) all hardware and embeded system are used by assembly language

* Elements of assembly language

These are three types

1) Mnemonics operation code
The mnemonic operation code for machine instruction are easier to remember than numeric operation code.

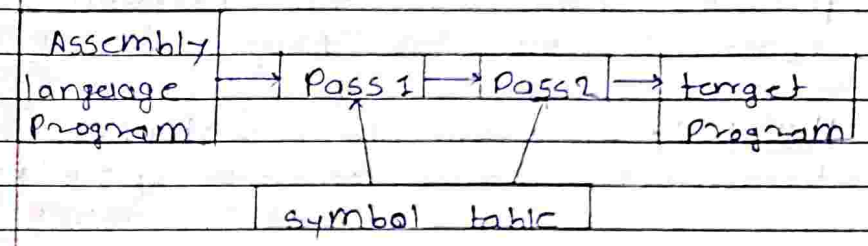
Ex ADD, SUB, MOVE, etc

2) Symbolic operand
A programmer can associate symbolic names with data or instruction used in assembly statements.
Ex - ADD, R1, R2, R3,

2) Data declaration
Data can be declared in a variety of notations including the decimal notation.
Ex:- NUM 1 03 OR
NUM 1 0011 H

* Assembly Process

- It convert the assembly file into object file.



Example of assembly language with assembles directive

- Assembly Program:

label (Prog. Name)	OP-code (Instruction)	operand (address)	LC value (location counters)
John	START	200	
	MOVER	R1, '3'	200
	MOVEM	R1, x	201
L1	MOVER	R2 = '2'	202
	LTORG		20
x	DS	1	204
	END		205

- START :- These instruction start the execution of program from location 200 and label with START provides name for the program (John is name for program).

- MOVER :- It moves the content of literal (='3') in to resistor operand R1.

- MOVEM :- It moves the content of resistor into memory operand (x).

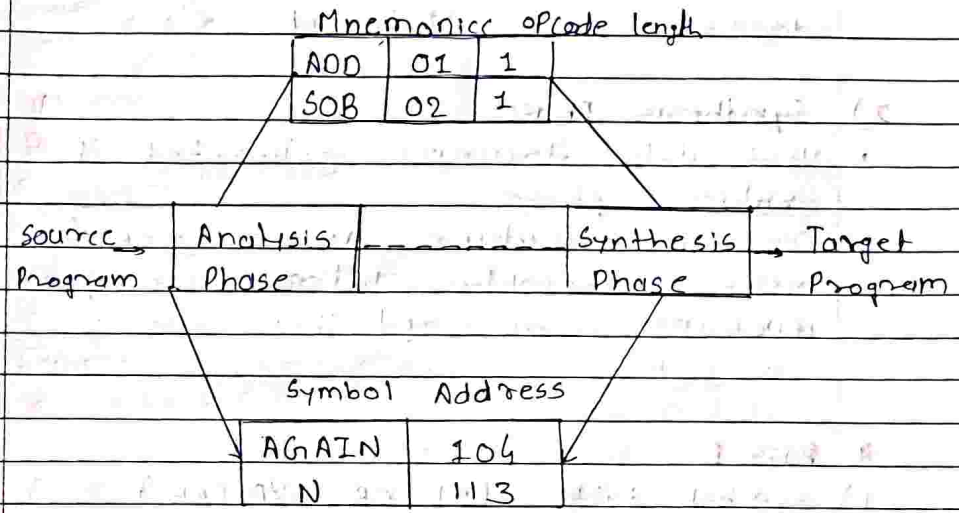
- MOVER :- It again moves the content of literal (='2') into resistor operand R2. and its label is specified as LI.

- LTORG :- It assign address to literal (current LC value)

- DS :- It assign a data space of 1 to symbol x.

- END :- It finishes the program execution

* Design of working of assembler.



- It is divided into two phase
1) Analysis phase (Pass 1 assembler)

2) Synthesis Phase (Pass 2 Assembler).

1) Analysis Phase
- It build symbol table for synthesis Phase to proceed.
- Determine the address of each symbol called as memory orogution
- Location counter used to hold address of next instruction.
- Isolated label, mnenonics opcode,

Operand, constant etc.

- Analysis of valided meaning and address of each statement.

2) Synthesis Phase

- Used data structure generated by analysis phase
  To build machine instruction for every assembly statement as per mnemonic code and these address assignation.

* Pass I

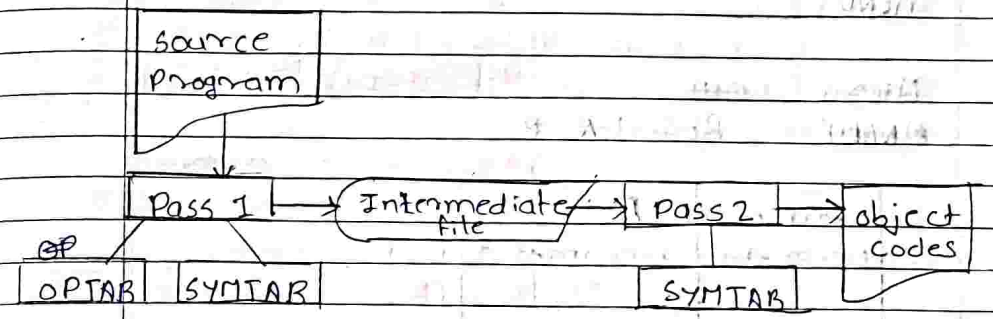
1) Symbol table (ST OR SYMTAB)
   IT store value or address assign to the label.

label	Address
JOHN	200
LI	202
X	204

2) Literal table (LT OR LITAB)
→ It store each literal or constant (='3') with its location.

INDEX	Literal	Address
0	='3'	200
1	='2'	202

* Pass 2



→ In this second pass the instruction are again read and are assembled used in symbol table.

- Basically the assembles goes to the block of diagram and generate machine code for that instruction.
- It convert mnemonic operation code, symbolic table operands with equivalent ~~complete~~ machine code.
- It convert complete program into object file.

Macro defination and

```
MACRO
MNAME    &Par 1 , &Par 2 , &Par 3
MOVER    &Par 1 , &Par 2
ADD      &Par 1 , &Par 3
MOVEM    &Par 1 , &Par 2
MEND
```

~~Micro call~~

~~MNAME , REG, 1, A, B~~

Formal Parameter	Actual Parameter
&Par 1	REG1
&Par 2	A
&Par 3	B

Micro call

MNAME , REG, 1, A, B

Expanded code

```
MOVER    REG 1 , A
ADD      REG 1 , B
MOVEM    REG 1 , A
```

Sigle Pass Assembler

(MOT) - mnemonic opcode Table

OPcode	Mnemonic
01	MOVER
02	MOVEM
03	ADD
04	SUB
05	MULT
06	DIV
07	BC
08	COMP
09	PRINT
10	READ

(POT) Pseudo Opcode Table

OP-code	Mnemonic
01	START
02	END
03	EQU
04	ORIGIN
05	LTORG

DL - Declarative statement

OP-code	Mnemonic
01	DS
02	DC

REGISTERS

Reg NO.	Name
01	AREG
02	BREG
03	CREG
04	DREG

DC - Declarative constant

Source Programming	Target code
START 100	01 - 100
MOVER AREG, A	100) 01 01 - 107
PRINT B	101) 09 - 501
ADD BREG, = '9'	102) 03 02 105
SUB BREG, = D	103) 04 02 503
COMP CREG, = '23'	104) 08 03 106
LTORG	105) - - 009
	106) - - 023 ✓
ADS 3	107) - - 003
LABEL . EQUA	107) NO code generation 003
ORIGIN 500	NO code generation
LI ; MULT CRE	500) 05 03
BDC 10	501) ~~02~~ - 010 ~~008~~
~~END~~ MOVEM CREG, '9'	502) 02 03 504
DD	503) - - - 008
END	504) - - - 007

Symbol Table

Sy. No	Symbol	Address
01	A	107
02	B	501
03	D	507
04	LABEL	107
05	LI	500

Table of Incomplete Instruction

Lcode	Incomplete Instruction
100	A
101	B
102	= '9'
103	D
104	= '23'
500	= '7'

Literal Table

Lit. NO	Literal	Address
01	= '9'	105
02	= '23'	106
03	= '7'	504

Pool - TAB
01
03

Multi Pass Assembler

① A multi pass compiler compile the source course in multiple passes

② It is not necessary for multi pass assembler for apple more than two pass over the entire program

③ Inclide only the parts of the program envolving references need to be process in multiple passes

e.g.

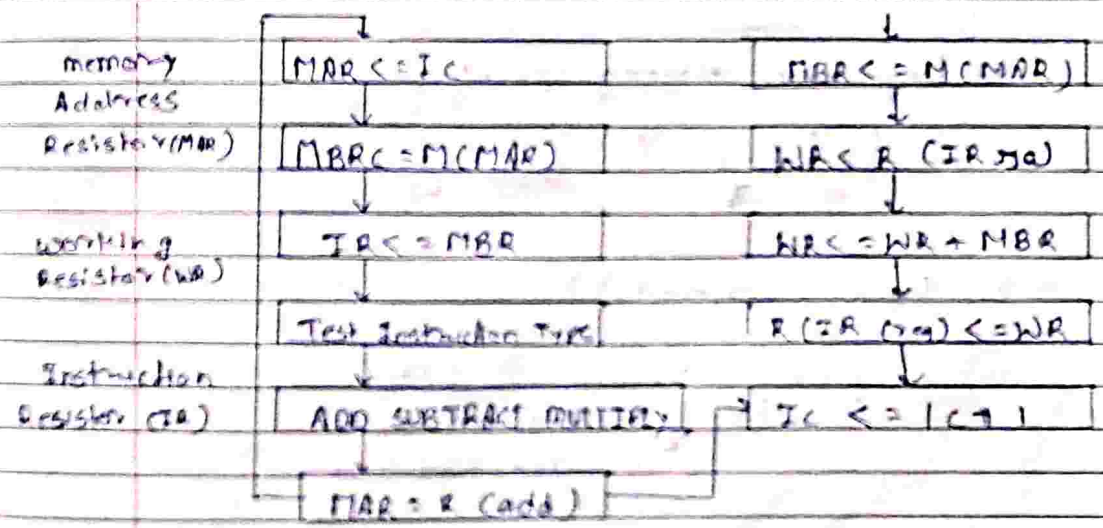
```
HALF SZ    EQU    MAXLEN/2
MAXLEN     EQU    BUFFEND - BUFFER
PREV BT    EQU    BUFFER - 1
  :
  :
BUFFER     RESB   4096
BUFFEND    EQU    *
```

HALF SZ	P 1	MaxLEN /2
MAXLEN	P 2	BUFFEND-BFFER
BUFFEND	*	
BUFFER	*	
PREVENT	P 3	BUFFER - 1

HALF SZ	800	1000 HD of (4096)D
MAXLEN	2034 - 1034 = 1000	4096/2 = (2048)D
BUFFEND	2034	(800)HD
BUFFER	1034	(1034) BUFFER BUFFEND
PREVENT	1033	1034 - 1000 = 2034

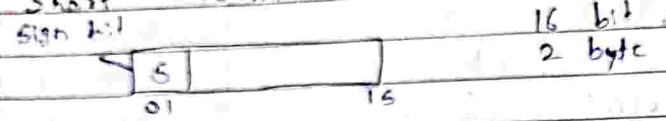
* Design of 2 Pass Assembler

* Micro H flowchart for ADD Instruction :-

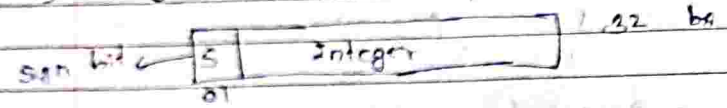


* Data formats for IBM 360 :-

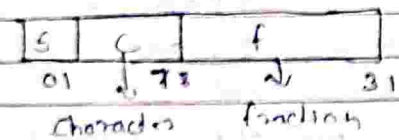
1) Short form fixed point



2) long form fixed point

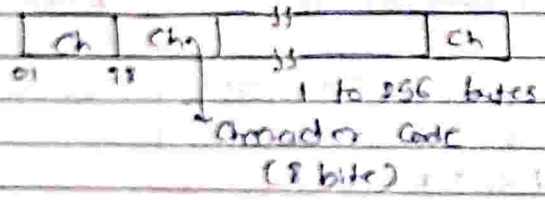


3) short form floating point



4) long form floating point

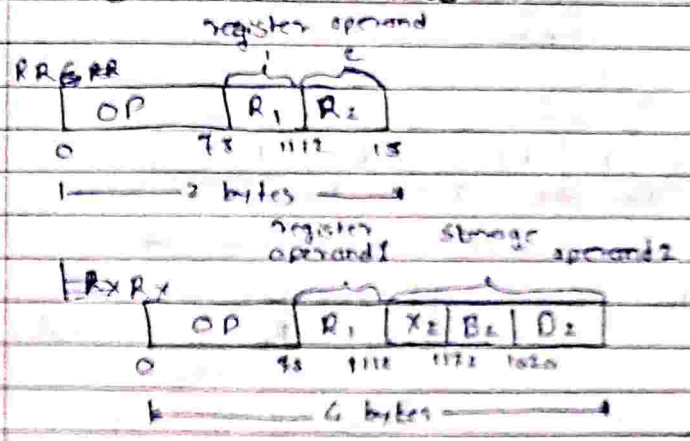
5) logical (character)



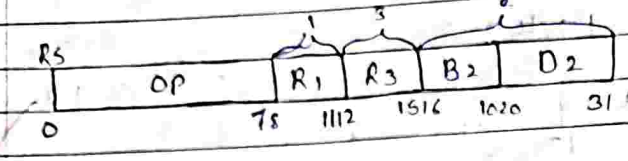
6) Decimal Packed

7) Unpacked Decimal

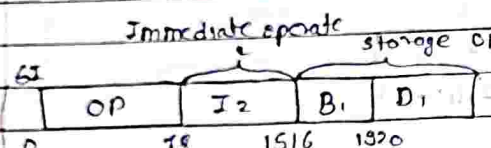
* Instruction Format



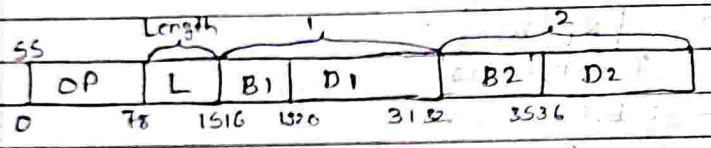
OP → Operation code → 8 bites
R → Register → 4 bit
X → Index Register → 4 bit
B → Base Register → 4 bit
D → Displacement → 12 bit
I → Immediate → 8 bit



e.g LM 2, 900 (3, 1)
index B



e.g MVI 900(1), 64
Base



e.g MVC 900(79, 1), 800(1)
X B B

Date.....
Page.....

(2)nd method

☆ Address modification using index register

Absolute Address	Relative address	Instrn
48	0	SR 4,4
50	2	L 2, 904 (4,1)
54	6	A 2, 900 (0,1)
58	10	ST 2, 904 (4,1)
62	14	A 4, 836 (0,1)
⋮		Branch back of relative location 2, nine times
944	886	4
948	900	99
952	904	number

• Address modification using instruction as data.

Absolute address	Relative address	Instruction
48	0	L 2, 904 (0,1)
52	4	A 2, 900 (0,1)
56	8	ST 2, 904 (0,1)
60	12	L 2, 0 (0,0)
64	16	A 2, 836 (0,1)
68	20	ST 2, 0 (0,1)

72	24	L 2,8 (0,1)
76	28	A 2,896 (0,1)
80	32	ST 2,8 (0,1)
⋮	⋮	Branch to relative
		rel^n 0 nine times
944	896	4
948	900	49
952	904	number

* Classification of Instruction

1) Pseudo OP

i) START :- It is a Pseudo of that tell the assembler where the begining of the program is and allows the user to give a name to the program.

e.g TEST START

ii) END :- It is a Pseudo of that tell the assembler that the end of the program has been reach.

e.g END

iii) USING :- It indicate that to the assembler which general purpose register to use as base register and what its Content will be.

e.g USING *, 15

USING BEGIN, 2, 15

iv) DROP :- which indicate unavailable base register

e.g DROP 15

v) DS :- It is Pseudo of which make the assemblers reserv a space in memory. DS only for define storage.

e.g : BLOCK DS 5S

vi) DC :- It is used to define storage and store a constant
e.g FOUR DC
F '4'

vii) LTORG :- It tells the assembler to place the encountered literals and an earlier location. (use the case where we have long program)

viii) EQU :- Used to define variable with constant value
E.g :- BASE EQU 15

2) Machine OP

i) BALR
Branch and link load the register ~~whi~~ with the next address and branch to the address in the second field
e.g BALR 15, 0

ii) BR :- unconditionally branch to the location whols address in register
e.g BR 14

Date 28/03/23

BCT
Branch and count
This command the given register by one, and if the result is positive, branches to the address in the second field.

Program

```
Test     START
BEGIN    BALR     15, 0
         USING    BEGIN+2, 15
         SR       4, 4
         L        3, TEN
Loop     L        2, DATA (4)
         A        2, FORTY
         ST       2, DATA (4)
         A        4, FOUR
         BCT      3, LOOP

         BR       14
TEN      DC       F '10'
FOUR     DC       F '4'
FOURTY 9 DC       F '49'
DATA     DC       F '1, 3.3, 3.3, 4.5,
                  8.9, 0'.

         END
```

Date .....
Page .....

Ex

```
JOHN    START   0
        USING   *, 15
        L       1, FIVE
        A       1, FOUR
        ST      1, TEMP
FOUR    DC      F'4'
FIVE    DC      F'5'
TEMP    DS      1F
        END
```

Pass 1	Pass 2	Pass 2
0 L 1, -(0,15)	0 L 1, 16(0,15)	
4 A 1, -(0,15)	4 A 1, 12 (0,15)	
8 ST 1, -(0,15)	8 ST 1, 20(0,15)	
12 4	12 4	
16 5	16 5	
20 -	20 -	

UNIT : 3

Date 05-10-23
Page .....

Macro Instruction

The assembly language programmer find it necessary to repeate a block of code many times in the course of the program the block may consist of code to save or exchange sets of register in this situation the programmer will find a macro instruction facility use to macro instruction often called macro are single line ~~prevication~~ abbreviation for group of instruction

```
MACRO                   macro Defination
INCR                    macro name
    A    1, DATA
    A    2, DATA        code
    A    3, DATA
MEND                    macro end options
```

Macro instruction with Arguments

```
MACRO
INCR    & ARG
    A    1, & ARG
    A    2, & ARG
    A    3, & ARG
MEND
```

```
INCR - DATA 1 ---> A 1, DATA
                   A 2, DATA
                   A 3, DATA

INCR - DATA 2 ---> A , 1 DATA 2
                   A , 2 DATA2
                   A , 3 DATA 2

DATA 1  DC  F'<'1
DATA 2  DC  F '10'

DATA I  DATA 2
```

* Nested Macro Calls

~~Macro~~

```
MACRO                          Expanded code level 1      expanded code level 2
ADD 1   & ARG                                             A 1, D1
  A     1, & ARG                                          A 2, D1
  A     2, & ARG                                          A 3, D2
  A     3, & ARG
MEND
MACRO
ADD S   2 ARG 1, & ARG
ADD 1   & ARG 1          ADD 1 D1                         A 1, D2
ADD 2   & ARG 2          ADD 1 D2 ---->                   A 2, D2
ADD 3   & ARG 3          ADD 1 D3                         A 3 D2
MEND
ADDS    D1  D2  D3                                        A 1, D3
D1      DC  F '1'                                         A 2, D3
D2      DC  F '2'                                         A 3, D3
D3      DC  F '3'
```

```
D2      DC  F '2'
D3      DC  F '3'
```

* Advance macro facility / Conditional macro facility

1) AIF

AIF (exp) <seq. symbol>

2) AGO

AGO <seq. symbol>

3) Expansion time variable

LCL <& v. name>

<v. name> SET <(exp)>

```
Ex-1) MACRO
      CVAL    & x  & N
      LCL     & M
      & M     SET '0'
MORE  MOVER A MOVER AREG ; AREG , & x + & M
      & M     SET     & M+1
      AIF     (& M  NE  N)  ~~MACRO~~ MORE
      MEND

Ex 2) MACRO
      & Argo  INCR  & count , & argo 1 , & arg 2 , & arg 3
        A 1, & arg 1
      AIF  (& count EQ 1)  FINAL
```

```
A 2, & org 2
AIF (& count EQ 2) FINAL
A 3, & org 3
FINAL MEND
:
Loop1 INCR 3, data 1, data 2, data 3
:
Loop 2 INCR 2, data 3, data 2
:
Loop 3 INCR 1, data 1
:

Data 1   DC '5'
Data 2   DC '6'
Data 3   DC '7'
```

Loop1 → A 1, data1 / A 2, data2 / A 3, data3

Loop2 → A 1, Data3 / A 2, Data2

Loop3 → A 1, Data 1)

* MACRO instruction ~~desig~~ defining Macro.

* MACRO Processure

There are four basic task that a macro processure must ~~perform~~ perform

1) regnize macro defination
2) Save the defination
3) regnize the macro call
4) expand calls and substitute arguments

* Specification of data bases

Pass-I

- Macro defination table (MDT
  - ~~to~~ Store the body of macro defination. (MDT)
- macro defination counter (MDTC)
  - to mark nest available entry MDT
- macro name table (MNT)
  - It use to store the names of macros
- Macro name table counter (MNTC)
  - It is use to indicate next available entry in (MNT)
- Argument ~~list~~ list array (ALA)
  - It is use to substitute index marker for for dummy arguments before storing a macro defination

Pass-2

MDT, created by Pass-I

MNT, created by Pass-I

macro defination counter table

It is use to indicate next line of text to be use during macro expansion

Argument ~~test~~ list Array
It is use to substitute macro ~~calls~~ index argument ~~index~~ markers ~~in the~~ store ~~macro~~ defination

Argument list Array
It is used to substitute index markers in the stored macro if definition

MACRO Defination table (MDT)

Index	card	
15	&LAB INCR	&ARG1, &ARG2, &ARG3
16	#0 A	1, #1
17	A	2, #2
18	A	3, #3
19	MEND	

Macro name Table (MNT)

Index (8 bytes)	Name (4 bytes)	MDT index
13	"INCR bbbb"	15

Argument List ~~test~~ array

index	Argument
0	"bbbbbbb"
1	"~~DATA~~DATA 3 bbb"
2	"DATA 2 bbb"
3	"DATA bbbbbb"

Q) Get MDT, MNT, ALA, and expanded code for the following code fragment.

```
      MACRO
       XYZ        &A
      MACRO
       &A         &B
      MACRO
       &B         &C
       ST         15, A(&C)
       BALR       14, 15
       L          1, =F'3'
       A          1, &C
       ST         1 &C
      MEND
      MEND
      MEND
PROG  START
       BALR
       :
       XYZ        MIT
       MIT        HELLO
HELLO
D1     DC         F'10'
       END
```

1) Macro name table

Index	macro names	MDTI
1	XYZ	1
2	MIT	14
3	HELLO	24

MDT

Index	Macro defination
1	XYZ &A
2	MACRO
3	#1, &B
4	MACRO
5	&B, &C
6	ST 15, A (&C)
7	BALR 14, 15
8	L 1, =F'3'
9	A 1&C
10	ST 1, &C
11	MEND
12	MEND
13	MEND
14	MIT &B
15	MACRO
16	#2 &C
17	ST 15, A (&C)
18	BALR 14, 15
19	L 1, =F'3'
20	A 1, &C
21	ST 1, &C
22	MEND
23	MEND
24	HELLO &C
25	ST 15, A (#3)
26	BLAR 14, 15
27	L 1, =F'3'
28	A 1, &C #3
29	ST 1, &C #3
30	MEND

- ALA

Index	Argument
0	"bbbbbbbb"
1	"XYZbbbb"
2	"MITbbbb"
3	"HELLObbbb"

Q Construct MOT, ST, LT, BTRL generate machine instruction for following code.

```
        START
Test    START   15,0
BEGIN   BALR    BEGIN+2,15
        USING   4,4
        SR      3=F'10'
        L       2,DATA(4)
LOOP    L       2,=F'49'
        ST      2,DATA(4)
        A       4=F'4'
        BCT     3,LOOP
        BR      14
DATA    LTORG   ,F'1,3,3,3,3,4,5,9,6
        DC
        END
```

- Symbol table

Index	Symbol	LC value
1	TEST	0
2	BEGIN	0
3	Loop	8
4	Data	44

- Litral Table

Index	Litral ~~Symbo~~	LC value
1	F'10'	4
2	F'49'	12
3	F'4'	20

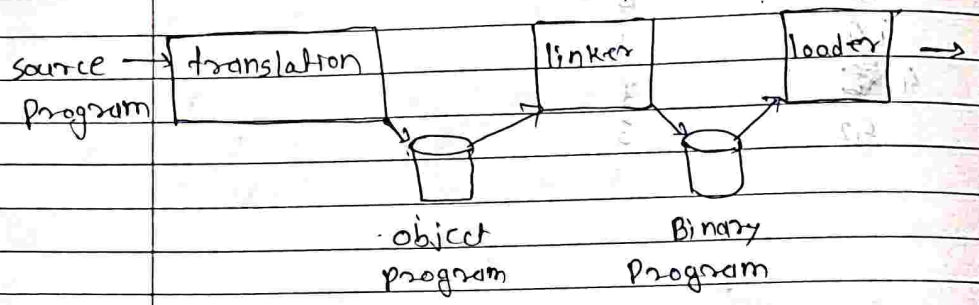
- BT Table

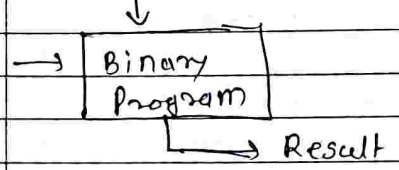
Register No	Content
15	2

- Machine Code

Relative Location	M/C code
0	BALR 15,0
2	SR 4,4
4	L 3,30 (0,15)
8	L 2,42 (4,15)
12	A 4,34 (0,15)
16	ST 2,42 (4,15)
20	A,4,38 (0,15)
24	BCT 3,6 (0,15)
28	BR 15,14
32	10
36	49
40	4
44	1
48	3
52	3

- Linker : The linker links all the funct^n and finds required by the object code and converts object code to executable code. the linker gives errors if the file or funct^n that has to be link does not exits





- Terminology use for Program entity

1) translated origin
Address by the origin ~~by~~ assume by translator.

2) link origin - address of the origin assign by the ~~origin~~ linker by processing. Producing a binary program.

3) Load origin - address of the origin assign by the loader while loading program for execution.

* Program relocation

	Standard	Address	Code
	START 500		
	ENTRY TOTAL		
	EXTRN MAX, ALPHA		
	READ A	500)	+09 0 540
Loop	⋮	501)	
	MOVER AREG, ALPHA.	518)	+04 1 000
	BC ANY, MAX	519)	+06 6 000
	⋮	⋮	
	BC LT, LOOP	535)	+06 1 501
	STOP	540)	+00 0 000
A	DS 1	541)	
Total	DS 1		
	END.		

Consider

1) AA - set of absolute addresses may be instruction address or data address

2) $AA \neq \phi$
means instruction or data occupy memory words with specific address

3) Address sensitive program contain
i) Address sensitive instruction
2) Address constant

- defination
Address sensitive program can execute correctly only if the start address of the memory area alocated it is the same as its translated origin

Start address == translator address

1) Thus, to execute correctly from any memory area, address use in each address sensitive instruction of program must be corrected.

The process of modifine the addresses use in the address sensitive instruction of a program such that Program can execute correctly from the resignated

1) if linked origin is not equal to translated origin then relocation is performe by linker.
2) if load origin is not equal to link origin then relocation is perform by loader.

~~We can to~~
We can classify the Program depending on the manner in which a program can modify or modify itself to execute from a given load origin

1) Non Relocatable Program
2) Relocatable Program
3) self relocating program

1) Non relocatable Program
- relocatable information is not available
- In non relocatable program translation origin is equal to load prime origin so we can't ~~recolate~~ relocate. it use
- In non-~~relocate~~ absolute loader.

ex non relocatable program i.e .com files on dos.

② Relocatable Program

- At relocation program the relocation part is added to add part of the add sensitive instruct^n this process is generally done by op. system and output which program relocation is called the linkage editors.
- the

③ Self relocating Program

A program which perform relocation by its own.
relocation logic address is always specified & execution add of any program. Using relocation logic and all information related to address sensitive instruction Perform relocation by itself.
It mostly use in multi user system.

* Overlay structure Program

1) It is a part of program or software which has same load origin as same to other part of program.

2) It is used to reduce requirement of main memory a program which uses overlay such program called as overlay structure Program. It to

3) It consist of
1) A permenantly resident portion called root.
2) set of overlay

4) To execute an overlay structure program the root is loaded to main memory and given a control for execution

5) When an overlay loaded to main memory over write the previosly loaded overlay with the same load origin.

e.g

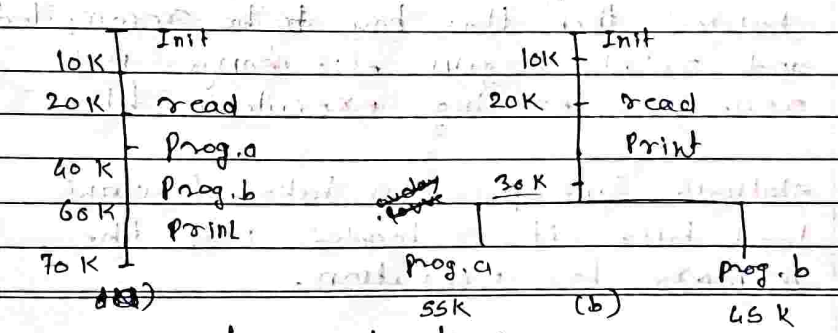


fig : overlay tree

Date.....
Page.....

* Static Linking

1) Process of copies all library module use in the program into final executable image this is perform by linker it is done at the last step of compilation process.

The linker combined library routine with the program code to resolve external and to generate and executable image suitable for loading into memory.

2) It is perform by program called linker as last step of compiling a program (linker are also called link editor)

3) Statically link file large is size because external program are build into executable file.

4) If any of the external program has changed then they have to be recompiled and relink again else changes not reflect in existing executable file.

5) statically link program takes constant load time it is loaded into the memory for execution.

Date.....
Page.....

It is faster than dynamic linking

* dynamic linking

1) Here names of the external library are placed in the final executable file while actual linking take place and run time when both executable file and libraries are placed in memory.

2) It is perform at run time by operating system.

3) Only one copy of shared library kept in memory this reduce in redability of executable program.

4) In dynamic linking the changes not reflect in existing file executable file here individual module can be updated and recompile.

5) Load time might be reduce if shared library code is already placed.

6) It is slower than static linking.

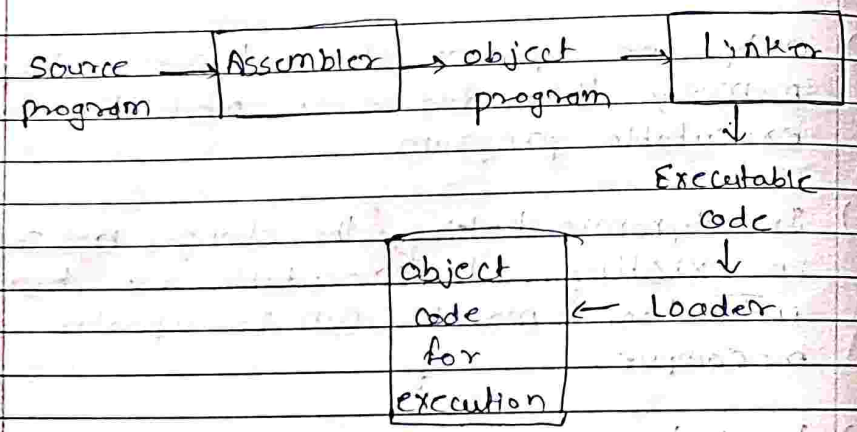
Date.....
Page.....

* Loader

1) A loader is a system software program that perform the loading function.

2) Loading is the process of placing the secondary memory file or program in main memory

3) Loader is responsible for initiating the execution of the process



function of Loader

1) alocation :- alocate memory space of program

2) Linking :- link object program code with necessary library routing & generate executable program code.

Date.....
Page.....

3) Relocation :- It change memory address of program when it loaded into the memory

4) Loading :- finally executable file ~~contain~~ content (machine instruction and data) will be place into physically memory for execution