10/10/17

4. Linker & Loader

- linker - Links
- Loading - Place object program in memory.
- Linking - compile two or more object program
- Relocation - changing the starting add
- linkage Editor - Provides linking without loading

Three types of loader

1. ~~Absolute~~ loade ~~Bootstrap~~		Relocating
Bootstrap	Absolute	
• loads the first pgm usually OS • Stored in ROM • Predefined • Single pass loader is only needed. • It is a special type of absolute loader	• Starting address is already specified • Exact location is known • Simple loader. • H - Verify the pgm name • T - loads object code at specified address • E - goes to starting address to start execution.	

Algorithm for Absolute loader

begin

read Header record

Verify pgm name & length.
Read first Text record
While record type ≠ E do
begin
move obj code to specified loc in mem
read next obj pgm record
end
jump to addr specified in end record.
End

When a computer is first turned on or restarted a special type of absolute loader called a bootstrap loader is executed. This bootstrap loads the first program to be run by the computer usually an operating system. The bootstrap itself begins at address 0 in the memory of the m/c. It loads of the OS starting at address 80. Each byte of object code to be loaded is represented on device F_1 as two hexadecimal digit. The object code from device F_1 is always loaded into consecutive bytes of memory starting at address 80

```
BOOT STRAP  0
        CLEAR   A
        LDX     #128
LOOP    JSUB    GETC
        RMO     A,S
        SHIFTL  S,4
        JSUB    GETC
        ADDR    S,A
        STCH    0,X
        TIXR    X       JLOOP
GETC    TD      INPUT
        JEQ     GETC
        RD      INPUT
        COMP    #4
        JEQ     80
        COMP    #48
        JLT     GETC
        SUB     #48
        COMP    #10
        SUB     #7
RETURN  RSUB
INPUT   BYTE    X'F1'
        END     LOOP
```

Absolute loader

advantages

- Simple loaders
- no memory limitations

disadvantage

- Program has to specify address
- Specifying subroutines is difficult

Machine Dependent features of loader:

→ Relocation ← modificatn record, relocatn bit, h/w
→ Program linking
→ linking load

It is depends on mlc architecture.

Relocation is used bcz the exact locatn is not specified in absolute loader.

Relocation can be done using modificatn record, relocatn bit or h/w changes.

```
0000    COPY    START   0
0000    FIRST   STL     RETADR
  :
0006    CLOOP   +JSUB   RDREC
0007
000A
0013    +JSUB   WRREC
0014
  :
0033    RETADR  RESW    1
```

M 1 M 2-7 start adding

```
H ^     ^
T ^     ^     ^
E ^
M ^ 000007 ^ 05
  1 2 3 4 5 7
M ^ 000014 ^ 05.
```

modificatn record describe each part of obj code that must be change in the above prgm. The lines that are effected by relocatn are 0006 and 0013 all modification record the value of starting address. The modification record for the piece of code is

M ^ 000007 ^ 05
M ^ 000014 ^ 05

Relocation bit:

It is used in SIC mlc because SIC doesn't used relative addressing. Relocation bits are used on mlc that use direct addressing and has a fixed instruction format which mean it'll use relocation bit.

Relocation bit associated with each objcode is used to indicate whether or not this word should be change when the program is relocated.

If relocatn bit of objcode is set to '1' the program starting address is to be added to this code. when the prgm is relocated. A bit of '0' indicates the no modification is necessary.

eg: Bit mask of FFC indicates that all '10' words of objcode in the Text record are to be modified during relocation. 11th and 12th word need not be modify.

19/10/17

Program linning

```
0000  PROGA  START 0
      EXTDEF  LISTA, ENDA
      EXTREF  LISTB, ENDB, LISTC, ENDC
      ⋮
0020  REF 1  LDA   LISTA      032010
0023  REF2  +LDT  LISTB+4     77100004
0027  REF3  LDX   #ENDA-LISTA 050014
0040  LISTA  EQU, *          0054-0040
                             = 0014
      ⋮
0054  ENDA   EQU  *
0054  REF 4  WORD  ENDA-LISTA + LISTC
0057  REF 5  WORD  ENDC-LISTC-10
005A  REF6   WORD  ENDC-LISTC+LISTA
005D  REF7   WORD  ENDA-LISTA-(ENDB-LISTB)
0060  REF8   WORD  LISTB-LISTA
      END    REF1
```

LDA	00
LDT	74
LDX	04

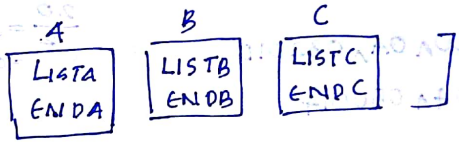
→ 0020 REF1 LDA LISTA.

Pc-relative

displacement = TA − PC

= 0040 − 0023 = 01D

[There are three ~~control section~~ programs in a control section




```
LOC     Source Statmx
0000    PROGA  START 0
               EXTDEF  LISTA, ENDA
               EXTREF  LISTB, ENDB, LISTC, ENDC
               ⋮
0020    REF1   LDA   LISTA
0023    REF2   +LDT  LISTB+4
0       REF3   LDX   #ENDA-LISTA
               ⋮
0040    LISTA  EQU   *
               ⋮
0054    ENDA   EQU   *
0054    REF4   WORD  ENDA-LISTA+LISTC
0057    REF5
```

→ Modification ~~Record~~:

H^PROGA^0000^0063 (progm, length) 0060 + 3 bytes (word)

D^LISTA^0040^ENDA^0054.

R^LISTB, ENDB, LISTC, ENDC.

OBJECT RECORD.

:

T^000020^0A^03201D (length of object code)

$\frac{20}{2}$ = 10 Hex → = 0A

77100004^050014

Modificatn record is required in forward reference

In the above prgm, 00054 .. (+LIST C) is ~~to be no~~ forward reference.

M^000054^05^+^LISTC (to specify ... word)

- 0023 .....

M^000024^05^+^LISTB (1 byte for opcode)

- 0054 REF4 WOR ...

M^000054^06^+^ENDC

- 0057 REF5 .....

M^000057^06^-^LISTC.

- 0054 REF6 ....

M^00005A^06^+^ENDC.

- 005A REF6 ....

M^00005A^06^-^LISTC.

- 005D REF7 ....

M^00005D^06^-^ENDB

M^00005D^06^+^LISTB

M^000060^06^+^LISTB

```
0000 COPY   START  0
            EXTDEF BUFFER, BUFFEND, LENGTH
            EXTREF RDREC, WRREC
0000 FIRST  STL    RETADR       172027
0003 CLOOP  +JSUB  RDREC        4B100000
0007        LDA    LENGTH       032023
000A        COMP   #0           290000
000D        JEQ    ENDFIL       332007
:
0017 ENDFIL LDA    = C'EOF'     032016
:
0030        *      = C'EOF'     454F46
002A RETADR RESW
002D LENGTH RESW
0000 RDREC  CSECT
            EXTREF BUFFER, LENGTH, BUFFEND.
0000        CLEAR  X            B410
0002        CLEAR  A            B400
```

MNEUMONIC	OPCODE	ASCII CHARA	HEX CODE	Regi
STL	14	A	41	A
LDA	00	E	45	X
COMP	28	F	46	L
JSUB	48	O	4F	B
CLEAR	B4	X	58	S
JEQ	30	Z	5A	T

Very IMP

PASS ONE OF LINKING LOADER

```
begin
get PROGADDR from operating system
Set CSADDR to PROGADDR
While not end of input do
begin
    Read next input record
    Set CSLTH to control section length.
    Search ESTAB for control section name.
    If found then
        Set error flag
    else
        Enter control section name into ESTAB with value
    while record type ≠ 'E' do.
    begin.
    Read next input record
    If record type = 'D' then  Define record.
    For each symbol in the record do
    begin
        Search ESTAB for symbol
        If found then.
            Set error flag (duplicate external symbol)
        else
            Enter symbol into ESTAB with value
            (CSADDR + indicated address)
    End (for);
    End {while ≠ 'E'}
    add CSLTH to CSADDR (starting address for next
                         control section)
End (while not EOF)
End (pass 1)
```

Data structures of Linking Loader.

1) PROGADDR
2) CSADDR
3) ESTAB

} Variables.

ESTAB is a table similar to SYMBTAB.
- To store external symbols

PROGADDR - to store starting address of the programme.

CSADDR - to store the control section.

Linking loader:

Linking loader is a loader that consists of a set of object programs that are to be linked together. First pass assigns address to all external symbols. Second pass performs actual loading relocating and linking. There are 3 data structures used.

PROGADDR: It is the begin address where link prgm is to be loaded. This value is supplied by OS.

Control section Address: CSADDR

It is the starting address of control section currently being ~~loaded~~ scan by the loader.

This value is added in all relative address with in the control section to convert them to the actual address

External symbol table ESTAB

Store ~~external~~ name and address of each external symbol using a hash organisation.

Algorithm for Pass 2 of a Linking loader.

```
begin
    CSADDR to PROGADDR
    EXECADDR to PROGADDR
while not end of input do
    begin
        read next input record (Header Record)
        set CSLTH to control section length
        while record type ≠ 'E' do
            begin
            read next input record
            if record type = 'T' then
            begin
            {if object code is in character form
            convert into internal representation}
            move object code from record to
            location [CSADDR + specified address]
            end {if 'T'}
            else if record type = 'M' then
            begin
            search ESTAB for modifying symbol name
                If found then
                    add or subtract symbol value at
                    location [CSADDR + specified addr]
                else
                    set error flag [undefined external
                                    symbol]
```

End {if 'M'}
End {while ≠ 'E'}
if an address is specified {in End record} then
set EXECADDR to (CSADDR + specified address)
add CSLTH to CSADDR
End {while not EOF}
to location given by EXECADDR {to start execut" of loaded program}
{pass 2}

11/17 Pass 1
Address of forward reference.

1. Resolve External reference.
2. Store its address to ESTAB
3. H, refer record, define record.

Pass -2

1. Actual linking, loading, relocation
2. Execution starts from EXECADDR.
3. Text record, Modification, End record.
   - Modification record also relocates depending on + or − symbol.

- Efficiency of Pass-2 can be increased by replacing External Symbol with ref no

Machine Independent Loader features

1. ALS
2. Loader options

LOADER DESIGN OPTIONS

- Linkage Editor
- Dynamic Linking
- Bootstrap Loader.

Automatic Library Search / call

ALS allows pgms to use skds subroutines without explicitly including them in the pgm to be loaded the routines are automatically retrieved from a library as they are needed during linking

eg: Sqrt already in library, if progrmer calls the Sqrt. Assembly ~~cafe~~ directive present before the sqrt is EXTREF. Then ESTAB is created Eventhough address is not found after pass-1. Then library automatilly enters its address

ESTAB

SQRT	← unresolved forward reference.

Step1: Enter symbols from each R record into External symbol table. When D is encountered address is assigned at the end of pass-1 Symbols in EXSTAB that remain undefined ~~~~ are called unresolved forward reference loader searches the library that contain definitions of these symbols and processes the ~~sy~~ subroutine found.

A special file structure is used for library. that gives name of each subroutine and Pointer to its address

9 Write code to replace RDREC, WRREC with READ & WRITE fns from utility lib.

~~M/c Independent Loader features~~

6/11/17

Loader option

INCLUDE READ(UTLIB) } INCLUDE Prg-name Lib-name
INCLUDE WRITE(UTLIB) } utility library

DELETE RDREC, WRREC → DELETE name.

CHANGE RDREC, READ } CHANGE symbol 1, symbol 2
CHANGE WRREC, WRITE }

LOADER DESIGN OPTIONS:

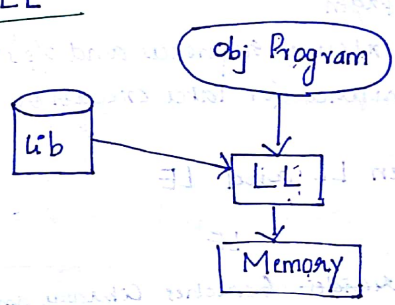
Linking → b4 load time (linkage editor)
→ during ~~after~~ load time (linking loader)
→ during execution (dynamic linking)

Linking loader: Linking & loader during loade.

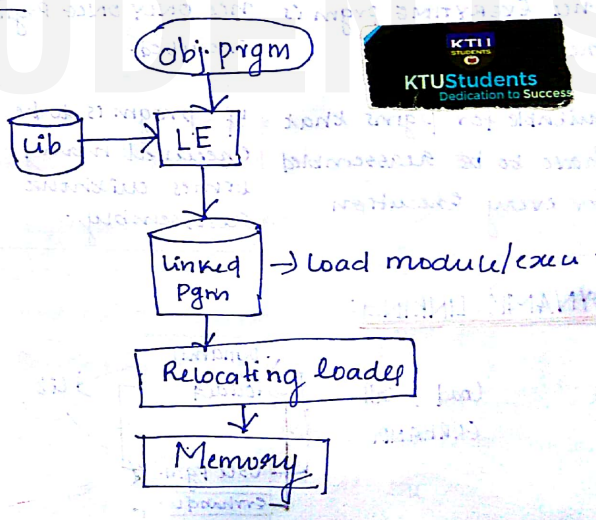
Linkage Editor: first link then stores in library at required it takes from library.

9/10 Sure.

LL



LE



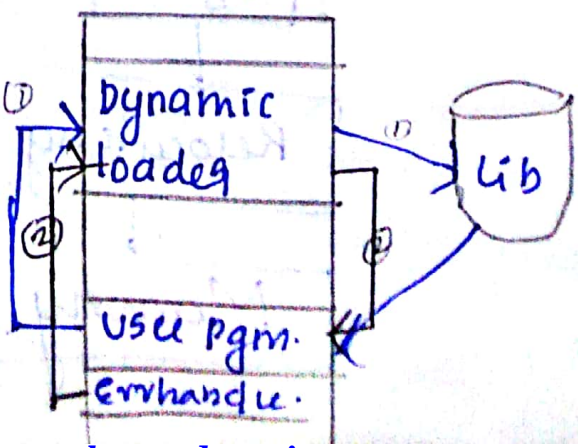
LE produces linked version of progm (load module/executable image) which is written to a library for later use. LE performs

- Relocation of all control sections relative to start of the link pgm
- It resolves all external references and o/p's a relocatable module for later execution

→ Difference between LL and LE

LL	LE
* Searches library & resolves external reference EVERYTIME prgm is executed	Searches library and resolves external reference ONLY ONCE Prgm is executed.
* Suitable for pgms that have to be reassembled for every execution	If progm is to be executed many times without reassembly.

DYNAMIC LINKING:



fig*

Load & call ERRHANDL