Linkers and Loaders

Loading Relocation Linking

- Loading
  - Bring object program into memory for execution
- Relocation
  - Modifies object program so that it can be loaded at an address different from the location originally specified
- Linking
  - Combines two or more separate object programs and supplies information needed to allow references between them

Loader

- System program that performs Loading
- Some loaders also do relocation and linking
- Some systems have a separate linker or linkage editor

Absolute Loader

- No linking or relocation
- All functions are performed in one pass
- E.g. a Bootstrap Loader

H _ COPY 001000 _ 00107A

T _ 001000 _ 1E1 _ 41033 _ 482039 _ 001036 _ 281030 _ 301015 _ 482061 _ 3C1003 _ 00102A _ 0C1039 _ 00102D

T _ 00101E _ 15 _ 0C1036 _ 482061 _ 081033 _ 4C0000 _ 454F46 _ 000003 _ 000000

T _ 002039 _ 1E 041030 _ 001030 _ E0205D _ 30203F _ D8205D _ 281030 _ 302057 _ 549039 _ 2C205E _ 38203F

T _ 002057 _ 1C _ 101036 _ 4C0000 _ F1 _ 001000 _ 041030 _ E02079 _ 302064 _ 509039 _ DC2079 _ 2C1036

T _ 002073 _ 07 _ 382064 _ 4C000 _ 05

E _ 001000

Object File

Header record:

Col. 1 H

Col. 2-7 Program name

Col. 8-13 Starting address of object program (hexadecimal)

Col. 14-19 Length of object program in bytes (hexadecimal)

Text record:

Col. 1 T

Col. 2-7 Starting address for object code in this record (hexadecimal)

Col. 8-9 Length of object code in this record in bytes (hexadecimal)

Col. 10 – 69 Object code, represented in hexadecimal (2 columns per byte of object code)

End record:

Col. 1 E

Col. 2-7 Address of first executable instruction in object program (hexadecimal)

Absolute Loader

- Check Header file
  - Check if correct program is being loaded, and if it will fit into memory
- Read Text Records
  - For each record move object code to indicated memory address.
- Reach End Record
  - Loader jumps to specified address to begin execution

0000	××××××××	××××××××	××××××××	××××××××
0010	××××××××	××××××××	××××××××	××××××××
⋮	⋮	⋮	⋮	⋮
0*FF*0	××××××××	××××××××	××××××××	××××××××
1000	14103348	20390010	36281030	30101548
1010	20613*C*10	0300102 *A*	0*C*103900	102*D*0*C*10
1020	36482061	0810334*C*	0000454 *F*	46000003
1030	000000 ××	××××××××	××××××××	××××××××
⋮	⋮	⋮	⋮	⋮
2030	××××××××	××××××××	××041030	001030*E*0
2040	205*D*3020	3*FD*8205*D*	28103030	20575490
2050	392*C*205*E*	38203*F*10	10364*C*00	00*F*10010
2060	00041030	*E*0207930	20645090	39*DC*2079
2070	2*C*103638	20644*C*00	0005 ××××	××××××××
2080	××××××××	××××××××	××××××××	××××××××
⋮	⋮	⋮	⋮	⋮

Absolute Loader Algorithm

Begin

read Header record

verify program name and length

read first Text record

While record type ≠ 'E' **do**

begin

{if object code is in character form, convert into internal representation}

move object code to specified location in memory

read next object program record

end

jump to address specified in End record

end

Relocation

- Want to load multiple programs in memory
- Do not know where the programs will be loaded until run-time
- Impossible to specify where to load the program before run time
- Assembler has to provide necessary information so that the loader can determine where to load the program
- A program containing such information is called relocatable

Relocation

- Assembler inserts 0 for the address field of instruction that must be relocated
- Produces modification record
- Starting address of program will be added during loading

M00000705

Modification record:

Col. 1 M

Col. 2-7 Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)

Col. 8-9 Length of field to be modified in half-bytes (hexadecimal)

+Modification to be performed

Modification Records

- Can be problematic for some architectures
- What if only direct addressing is used?
- Every single instruction involving memory address would have to be relocated
- This means a modification record for every memory instruction
- Makes object program twice as long

Relocation Bit

- If direct addressing is rampant
- And instruction format is fixed
- Use bitmask to specify which words are to be relocated and which ones are to be left alone

T0000001EFFC1400334810390000362800303000154810613C000300002A0C003900002D

FFC=111111111100

Control Section

- Part of program that maintains its identity after assembly
- Each control section can be loaded and relocated independently of the others.
- Different control sections are most often used for subroutines and other logical subdivisions of a program.
- Programmer can assemble, load, and manipulate each of these control sections separately

Linking

- When control sections are logically related
- Data or instructions in one control section may be needed by a different control section
- Control sections must be linked to allow these (external) references

Control Section 1

5	0000	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
6			**EXTDEF**	**BUFFER, BUFEND, LENGTH**	
7			**EXTREF**	**RDREC, WRREC**	
10	0000	FIRST	STL	RETADR	
15	0003	CLOOP	+JSUB	RDREC	
20	0007		LDA	LENGTH	
25	000A		COMP	#0	
30	000D		JEQ	ENDFIL	
35	0010		+JSUB	WRREC	
40	0014	J	CLOOP		
45	0017	ENDFIL	LDA	=C’EOF’	
50	001A		STA	BUFFER	
55	001D		LDA	=C’EOF’	
60	0020		STA	LENGTH	
65	0023		+JSUB	WRREC	
70	0027		J	@RETADR	
95	002A		RETARD	RESW	1
100	**002D**		**LENGTH**	**RESW**	**1**
103				LTORG	
	0030	*	=C’EOF’		
•	0033	**BUFFER**	**RESB**	**4096**	
•	1033	**BUFEND**	**EQU**	*****	
105	1000	MAXLEN	EQU	BUFEND-BUFFER	

Control Section 2

```
109   0000        RDREC   CSECT
122               EXTREF  BUFFER, LENGTH, BUFEND
125   0000        CLEAR   X
130   0002        CLEAR   A
132   0004        CLEAR   S
133   0006        LDT     MAXLEN
135   0009  RLOOP  TD      INPUT
140   000C        JEQ     RLOOP
145   000F        RD      INPUT
150   0012        COMPR   A,S
155   0014        JEQ     EXIT
160   0017        +STCH   BUFFER,X
165   001B        TIXR    T
170   001D        JLT     RLOOP
175   0020  EXIT   +STX    LENGTH
180   0024        RSUB
185   0027  INPUT  BYTE    X'F1'
186   0028  MAXLEN WORD    BUFEND-BUFFER
```

Control Section 3

```
193   0000  WRREC    CSECT
195
                     EXTREF    LENGTH,BUFFER
212   0000            CLEAR     X
215   0002            +LDT      LENGTH
220   0006  WLOOP     TD        =X'05'
225   0009            JEQ       WLOOP
230   000C            +LDCH     BUFFER,X
235   0010            WD        =X'05'
240   0013            TXR       T
245   0015            JLT       WLOOP
•     0018            RSUB
255                   END       FIRST
      001B  *         =X'05'
```

Linking

- CSECT Defines "Control SECTion"
- SYMTAB keeps track of which control section the symbol belongs to
- EXTDEF specifies symbols declared locally
- EXTREF specifies symbols declared elsewhere

```
CLOOP       +JSUB       RDREC
```

- Is an external reference

Linking

- Assembler doesn’t know where RDREC (Read process Data RECord) control section will be loaded
- Cannot assemble address of the instruction
- Set address to zero, pass information to loader
  - Causes proper address to be inserted at load time.

New Records in Object Program

Define record:

Col. 1 D

Col. 2-7 Name of external symbol defined in this control section

Col. 8-13 Relative address of symbol within this control section (hexadecimal)

Col. 14-73 Repeat information in Col. 2-13 for other external symbols

Refer record:

Col. 1 R

Col. 2-7 Name of external symbol referred to in this control section

Col. 8-73 Names of other external reference symbols

New Records

Modification record:

Col. 1 M

Col. 2-7 Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)

Col. 8-9 Length of field to be modified in half-bytes (hexadecimal)

Col 10 Modification flag (+ or -)

Col. 11 – 16 External symbol whose value is to be added to or subtracted from the indicated field.

Linking

```
5     0000 COPY     START    0 COPY FILE FROM INPUT TO OUTPUT
6                   EXTDEF   BUFFER, BUFEND, LENGTH
7                   EXTREF   RDREC, WRREC
10    0000 FIRST    STL      RETADR
15    0003 CLOOP    +JSUB    RDREC
```

M 000004 05 + RDREC

Linking

190 0028 MAXLEN WORD BUFEND-BUFFER

M 000028 06 + BUFEND

M 000028 06 - BUFFER

Pass 1

- Assign address to Externals
- Variables
  - PROGADDR
    - takes address from OS
  - CSADDR (Control Section Address)
  - CSLTH (Control Section Lenght)
  - ESTAB (External Symbol Table)

Pass 2

- Load
- Relocation linking
- Modify Record using ESTAB
- END Eecord
  - Transfer Address

- Efficiency
  - Reference Numbers to each EXT Symbol

Linking Loader - Pass1

(Assigns addresses to all external symbols)

```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {for first control section}
while not end of input do
   begin
      read next input record {Header record for control section}
      set CSLTH to control section length
      search ESTAB for control section name
      if found then  set error flag {duplicate external symbol}
      else        enter control section name into ESTAB with value CSADDR
      while record type != 'E' do
                  begin
                     read next input symbol
                     if record type = 'D' then
                        for each symbol in the record do
                                    begin
                                       search ESTAB for symbol name
                                       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}
```

ESTAB = External symbol table
PROGADDR = Program load address
CSADDR = Control section address

```
Begin
Set CSADDR to PROGADDR
Set 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 address)
          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}
Jump to location given by EXECDDR {to start execution of loaded program}
```

Pass2

(Perform actual loading, relocation, and linking)