Loader and Linker

Three Working Items

- Loading: loading an object program into memory for execution.
- Relocation: modify the 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 the information needed to allow references between them.
- A loader is a system program that performs the loading function.
- Many loaders also support relocation and linking.
- Some systems have a linker to perform the linking and a separate loader to handle relocation and loading.

Absolute Loader

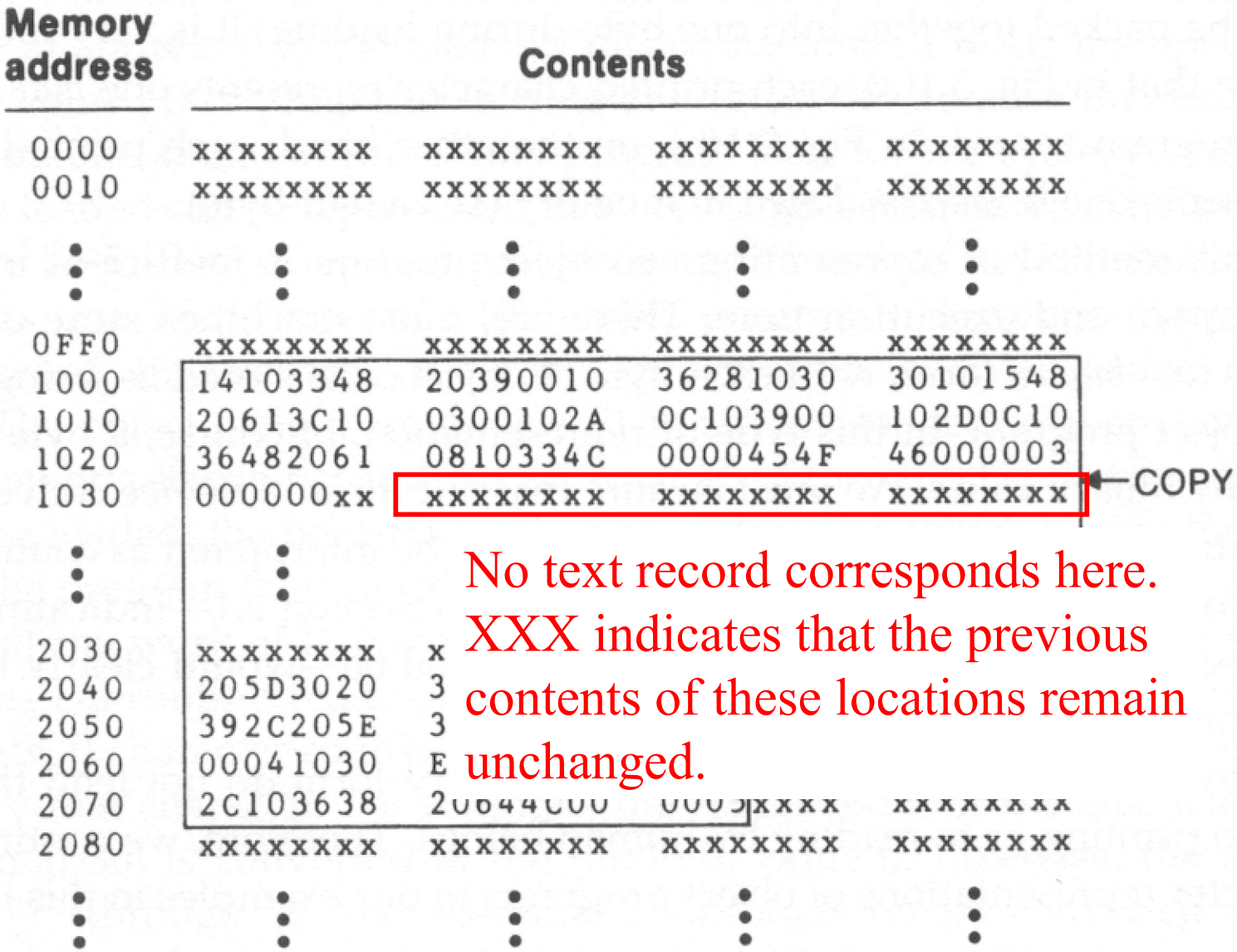
- An object program is loaded at the address specified on the START directive.
- No relocation or linking is needed
- Thus is very simple
- All functions are accomplished in single pass.

```
HCOPY  001000001007A
T001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T002039 1E 041030 001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T002057 1C 101036 4C0000 F1001000 41030E 02079 302064 509039 DC2079 2C1036
T002073 07 3820644C000005
E001000
```

(a) Object program

Absolute Loader

- The Header record of object program is checked to verify that the correct program has been presented for loading.
- As each Text record is read, the object code it contains is moved to indicated address in memory.
- When the End record is encountered the loader jumps to the specified address to begin execution of the loaded program.

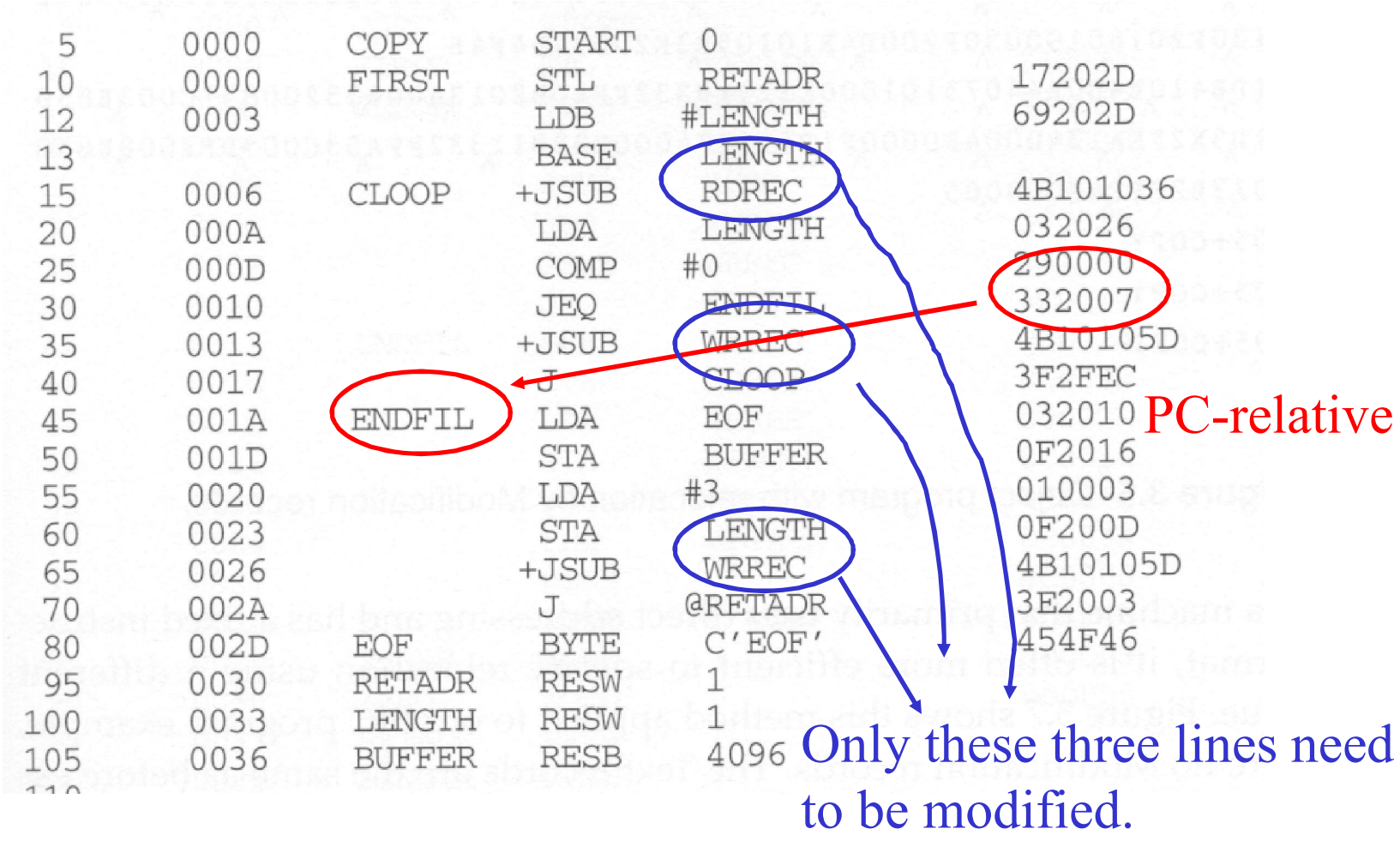


(b) Program loaded in memory

Relocating Loader

- Two methods to describe where in the object program to modify the address (add the program starting address)
  - Use modification records
    - Suitable for a small number of changes
  - Use relocation bit mask
    - Suitable for a large number of changes

Program Written in SIC/XE



```
110                 .
115                 .     SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
125     1036    RDREC     CLEAR   X              B410
130     1038              CLEAR   A              B400
132     103A              CLEAR   S              B440
133     103C              +LDT    #4096          75101000
135     1040    RLOOP     TD      INPUT          E32019
140     1043              JEQ     RLOOP          332FFA
145     1046              RD      INPUT          DB2013
150     1049              COMPR   A,S            A004
155     104B              JEQ     EXIT           332008
160     104E              STCH    BUFFER,X       57C003
165     1051              TIXR    T              B850
170     1053              JLT     RLOOP          3B2FEA
175     1056    EXIT      STX     LENGTH         134000
180     1059              RSUB                   4F0000
185     105C    INPUT     BYTE    X'F1'          F1
```

Base-relative

```
195                 .
200                 .     SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
210     105D    WRREC   CLEAR   X               B410
212     105F            LDT     LENGTH          774000
215     1062    WLOOP   TD      OUTPUT          E32011
220     1065            JEQ     WLOOP           332FFA
225     1068            LDCH    BUFFER,X        53C003
230     106B            WD      OUTPUT          DF2008
235     106E            TIXR    T               B850
240     1070            JLT     WLOOP           3B2FEF
245     1073            RSUB                    4F0000
250     1076    OUTPUT  BYTE    X'05'           05
255                     END     FIRST
```

Base-relative

This program is written in SIC/XE instructions. Program counter-relative and base-relative addressing are extensively used to avoid the need for many address modification records.

The Object Program

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107007 3B2FEF4F000005
M00000705+COPY
M00001405+COPY
M00002705+COPY
E000000
```

Only lines 15, 35, and 65 need to be modified.

The Same Program Written in SIC

```
 5    0000   COPY     START   0
10    0000   FIRST    STL     RETADR          140033
15    0003   CLOOP    JSUB    RDREC           481039
20    0006            LDA     LENGTH          000036
25    0009            COMP    ZERO            280030
30    000C            JEQ     ENDFIL          300015
35    000F            JSUB    WRREC           481061
40    0012            J       CLOOP           3C0003
45    0015   ENDFIL   LDA     EOF             00002A
50    0018            STA     BUFFER          0C0039
55    001B            LDA     THREE           00002D
60    001E            STA     LENGTH          0C0036
65    0021            JSUB    WRREC           481061
70    0024            LDL     RETADR          080033
75    0027            RSUB                    4C0000
80    002A   EOF      BYTE    C'EOF'          454F46
85    002D   THREE    WORD    3               000003
90    0030   ZERO     WORD    0               000000
95    0033   RETADR   RESW    1
100   0036   LENGTH   RESW    1
105   0039   BUFFER   RESB    4096
```

Direct addressing

```
115                 .       SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
125     1039        RDREC   LDX     ZERO            040030
130     103C                LDA     ZERO            000030
135     103F        RLOOP   TD      INPUT           E0105D
140     1042                JEQ     RLOOP           30103F
145     1045                RD      INPUT           D8105D
150     1048                COMP    ZERO            280030
155     104B                JEQ     EXIT            301057
160     104E                STCH    BUFFER,X        548039
165     1051                TIX     MAXLEN          2C105E
170     1054                JLT     RLOOP           38103F
175     1057        EXIT    STX     LENGTH          100036
180     105A                RSUB                    4C0000
185     105D        INPUT   BYTE    X'F1'           F1
190     105E        MAXLEN  WORD    4096            001000
195                 .
```

Direct addressing

```
200                 .       SUBROUTINE TO WRITE RECORD FROM BUFFE
205                 .
210     1061        WRREC   LDX     ZERO            040030
215     1064        WLOOP   TD      OUTPUT          E01079
220     1067                JEQ     WLOOP           301064
225     106A                LDCH    BUFFER,X        508039
230     106D                WD      OUTPUT          DC1079
235     1070                TIX     LENGTH          2C0036
240     1073                JLT     LOOP            381064
245     1076                RSUB                    4C0000
250     1079        OUTPUT  BYTE    X'05'           05
255                         END     FIRST
```

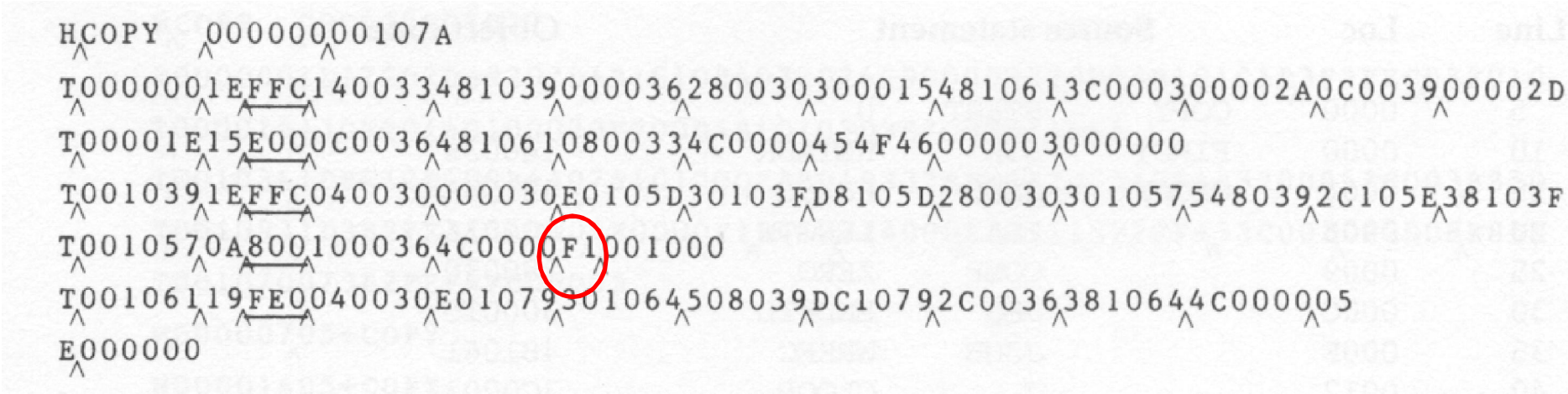
Direct addressing

This program is written in SIC instructions. Only direct addressing can be used. As such, we need many modification records.
This not only makes the object program bigger, it also slows down the loading process.

Relocation Bit Mask

- If an object needs too many modification records, it would be more efficient to use a relocation bit mask to indicate where in the object program should be modified when the object program is loaded.
- A relocation bit is associated with each word of object code. Since all SIC instructions occupy one word, this means that there is one relocation bit for each possible instruction.
- If the relocation bit corresponding to a word of object code is set to 1, the program's starting address will be added to this word when the program is relocated.

Relocation Bit Mask Example



This one-byte “F1” makes the LDX instruction on line 210 begins a new text record. This is because each relocation bit should be associated with a three-byte word. However, this data item occupies only one byte, which violates the Alignment rule.

Program Linking

- A program may be composed of many control sections.
- These control sections may be assembled separately.
- These control sections may be loaded at different addresses in memory.
- External references to symbol defined in other control sections can only be resolved (calculating their addresses in memory) after these control sections are loaded into memory.

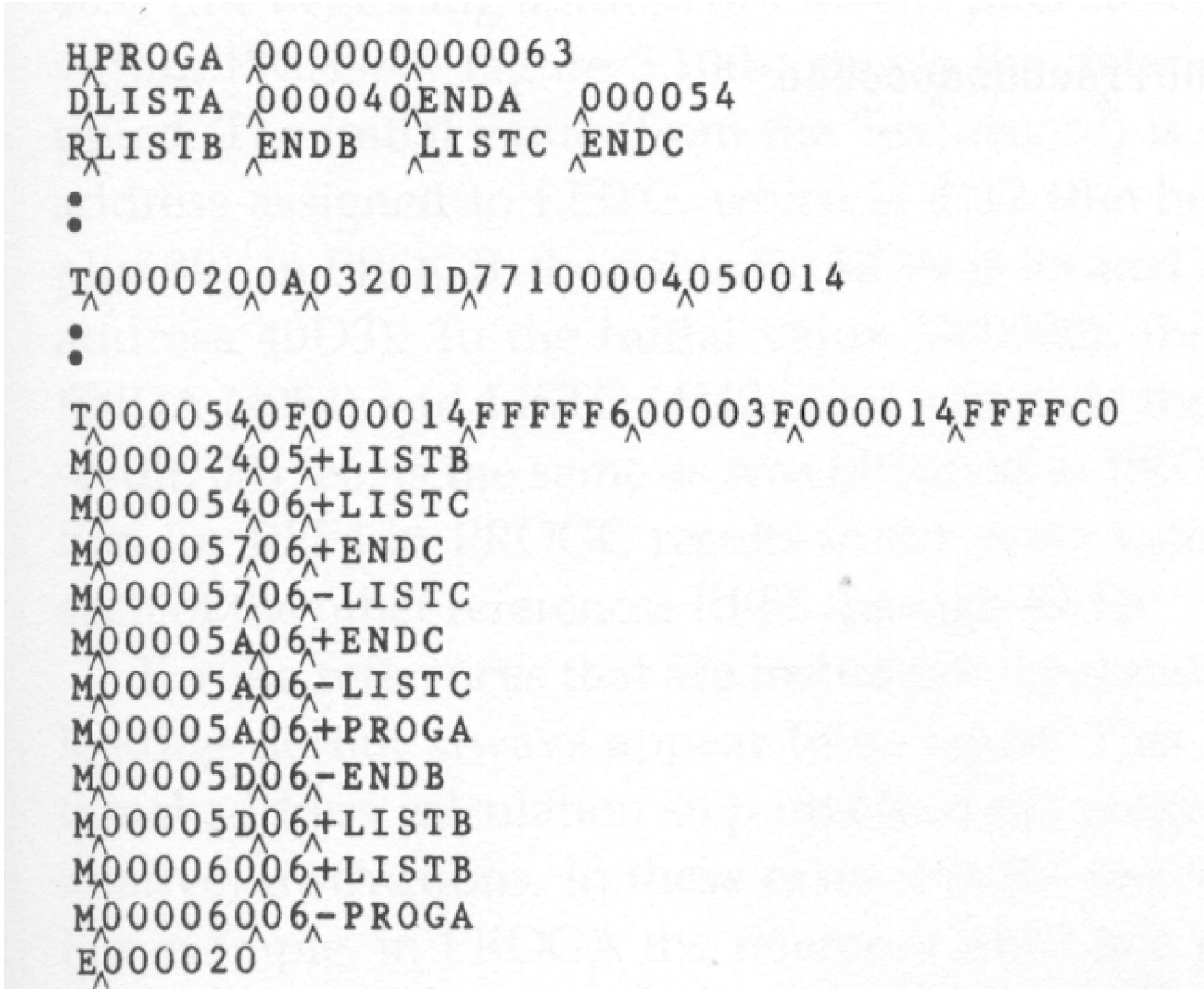
Program Linking Example

Loc		Source statement		Object code
0000	PROGA	START	0	
		EXTDEF	LISTA,ENDA	
		EXTREF	LISTB,ENDB,LISTC,ENDC	
		.		
		.		
		.		
0020	REF1	LDA	LISTA	03201D
0023	REF2	+LDT	LISTB+4	77100004
0027	REF3	LDX	#ENDA-LISTA	050014
		.		
		.		
		.		
0040	LISTA	EQU	*	
		.		
		.		
0054	ENDA	EQU	*	
0054	REF4	WORD	ENDA-LISTA+LISTC	000014
0057	REF5	WORD	ENDC-LISTC-10	FFFFF6
005A	REF6	WORD	ENDC-LISTC+LISTA-1	00003F
005D	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000014
0060	REF8	WORD	LISTB-LISTA	FFFFC0
		END	REF1	

Loc		Source statement		Object code
0000	PROGB	START	0	
		EXTDEF	LISTB,ENDB	
		EXTREF	LISTA,ENDA,LISTC,ENDC	
		.		
		.		
		.		
0036	REF1	+LDA	LISTA	03100000
003A	REF2	LDT	LISTB+4	772027
003D	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0060	LISTB	EQU	*	
		.		
		.		
0070	ENDB	EQU	*	
0070	REF4	WORD	ENDA-LISTA+LISTC	000000
0073	REF5	WORD	ENDC-LISTC-10	FFFFF6
0076	REF6	WORD	ENDC-LISTC+LISTA-1	FFFFFF
0079	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	FFFFF0
007C	REF8	WORD	LISTB-LISTA	000060
		END		

Loc		Source statement		Object code
0000	PROGC	START	0	
		EXTDEF	LISTC,ENDC	
		EXTREF	LISTA,ENDA,LISTB,ENDB	
		.		
		.		
		.		
0018	REF1	+LDA	LISTA	03100000
001C	REF2	+LDT	LISTB+4	77100004
0020	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0030	LISTC	EQU	*	
		.		
		.		
0042	ENDC	EQU	*	
0042	REF4	WORD	ENDA-LISTA+LISTC	000030
0045	REF5	WORD	ENDC-LISTC-10	000008
0048	REF6	WORD	ENDC-LISTC+LISTA-1	000011
004B	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000000
004E	REF8	WORD	LISTB-LISTA	000000
		END		

Object Program Example



```
HPROGA 000000000063
DLISTA 000040ENDA  000054
RLISTB ENDB  LISTC ENDC
.
.
T0000200A03201D771000040500l4
.
.
T0000540F000014FFFFF600003F000014FFFFC0
M00002405+LISTB
M00005406+LISTC
M00005706+ENDC
M00005706-LISTC
M00005A06+ENDC
M00005A06-LISTC
M00005A06+PROGA
M00005D06-ENDB
M00005D06+LISTB
M00006006+LISTB
M00006006-PROGA
E000020
```

```
HPROGB 000000000007F
DLISTB 000060ENDB  000070
RLISTA ENDA  LISTC ENDC
:
:
T0000360B0310000077202705100000
:
:
T0000700F000000FFFFF6FFFFFFFFFF0000060
M00003705+LISTA
M00003E05+ENDA
M00003E05-LISTA
M00007006+ENDA
M00007006-LISTA
M00007006+LISTC
M00007306+ENDC
M00007306-LISTC
M00007606+ENDC
M00007606-LISTC
M00007606+LISTA
M00007906+ENDA
M00007906-LISTA
M00007C06+PROGB
M00007C06-LISTA
E
```

```
HPROGC 000000000051
DLISTC 000030ENDC  000042
RLISTA ENDA  LISTB ENDB
:

T0000180C0310000077100004051000000
:

T0000420F000030000008000011000000000000
M00001905+LISTA
M00001D05+LISTB
M00002105+ENDA
M00002105-LISTA
M00004206+ENDA
M00004206-LISTA
M00004206+PROGC
M00004806+LISTA
M00004B06+ENDA
M00004B06-LISTA
M00004B06-ENDB
M00004B06+LISTB
M00004E06+LISTB
M00004E06-LISTA
E
```

Program Linking Example

- Notice that program A defines LISTA and ENDA, program B defines LISTB and ENDB, and program defines LISTC and ENDC.
- Notice that the definitions of REF1, REF2, .., to REF7 in all of these three control sections are the same.
- Therefore, after these three control sections are loaded, no matter where they are loaded, the values of REF1 to REF7 in all of these programs should be the same.

REF1

- Program A
  - LISTA is defined in its own program and its address is immediately available. Therefore, we can simply use program counter-relative addressing
- Program B
  - Because LISTA is an external reference, its address is not available now. Therefore an extended-format instruction with address field set to 00000 is used. A modification record in inserted into the object code so that once LISTA's address is known, it can be added to this field.
- Program C
  - The same as that processed in Program B.

REF2

- Program A
  - Because LISTB is an external reference, its address is not available now. Therefore an extended-format instruction with address field set to 00004 is used. A modification record is inserted into the object code so that once LISTB's address is available, it can be added to this field.
- Program B
  - LISTB is defined in its own program and its address is immediately available. Therefore, we can simply use program counter-relative addressing
- Program C
  - The same as that processed in Program A.

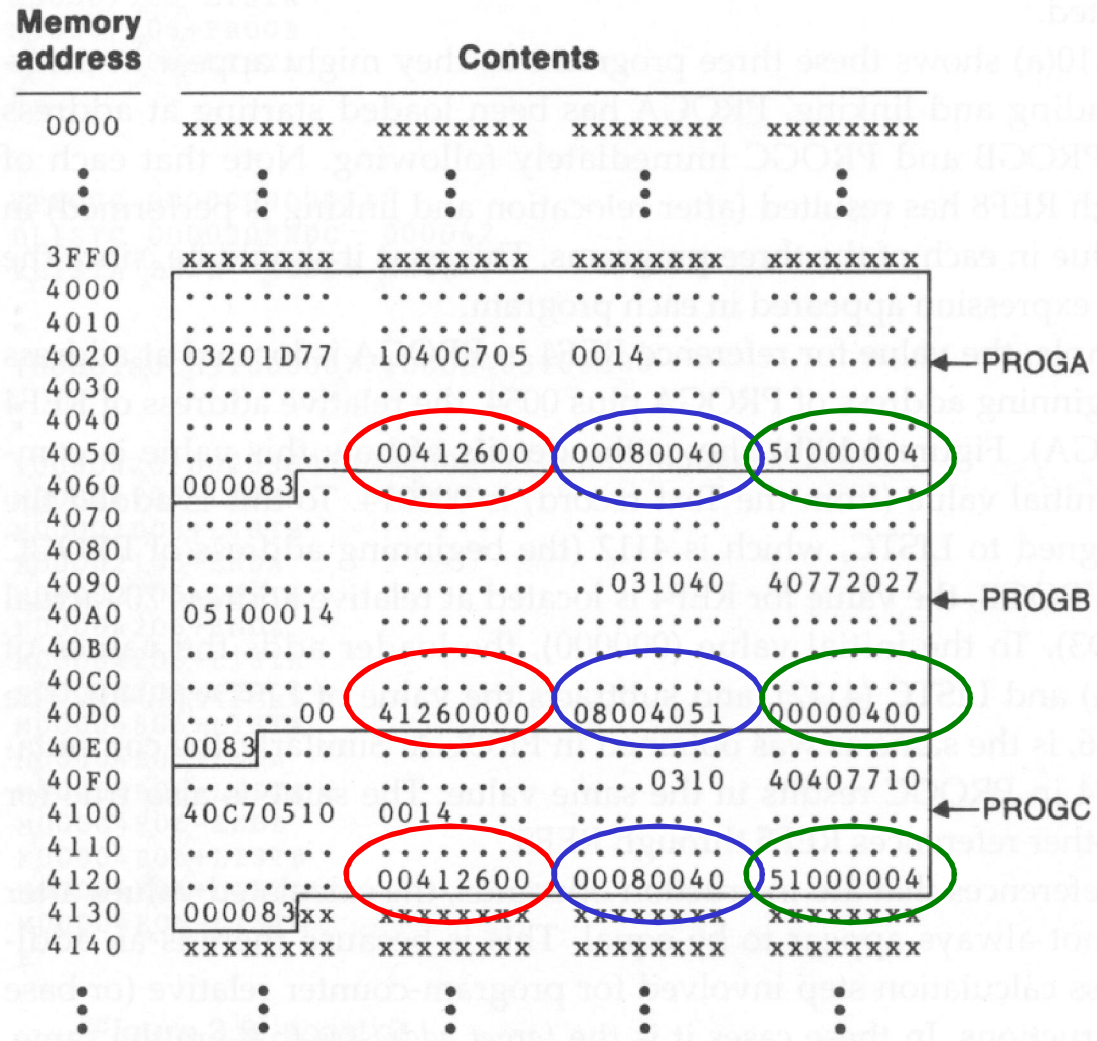
REF3

- Program A
  - The difference between ENDA and LISTA (14) is immediately available during assembly.
- Program B
  - Because the values of ENDA and LISTA are unknown during assembly, we need to use an extended-format instruction with its address field set to 0.
  - Two modification records are inserted to the object program – one for +ENDA and the other for –LISTA.
- Program C
  - The same as that processed in Program B.

REF4

- Program A
  - The difference between ENDA and LISTA can be known now. Only the value of LISTC is unknown. Therefore, an initial value of 000014 is stored with one modification record for LISTC.
- Program B
  - Because none of ENDA, LISTA, and LISTC's values can be known now, an initial value of 000000 is stored with three modification records for all of them.
- Program C
  - The value of LISTC is known now. However, the values for ENDA and LISTA are unknown. An initial value of 000030 is stored with two modification records for ENDA and LISTA.

After Loading into Memory

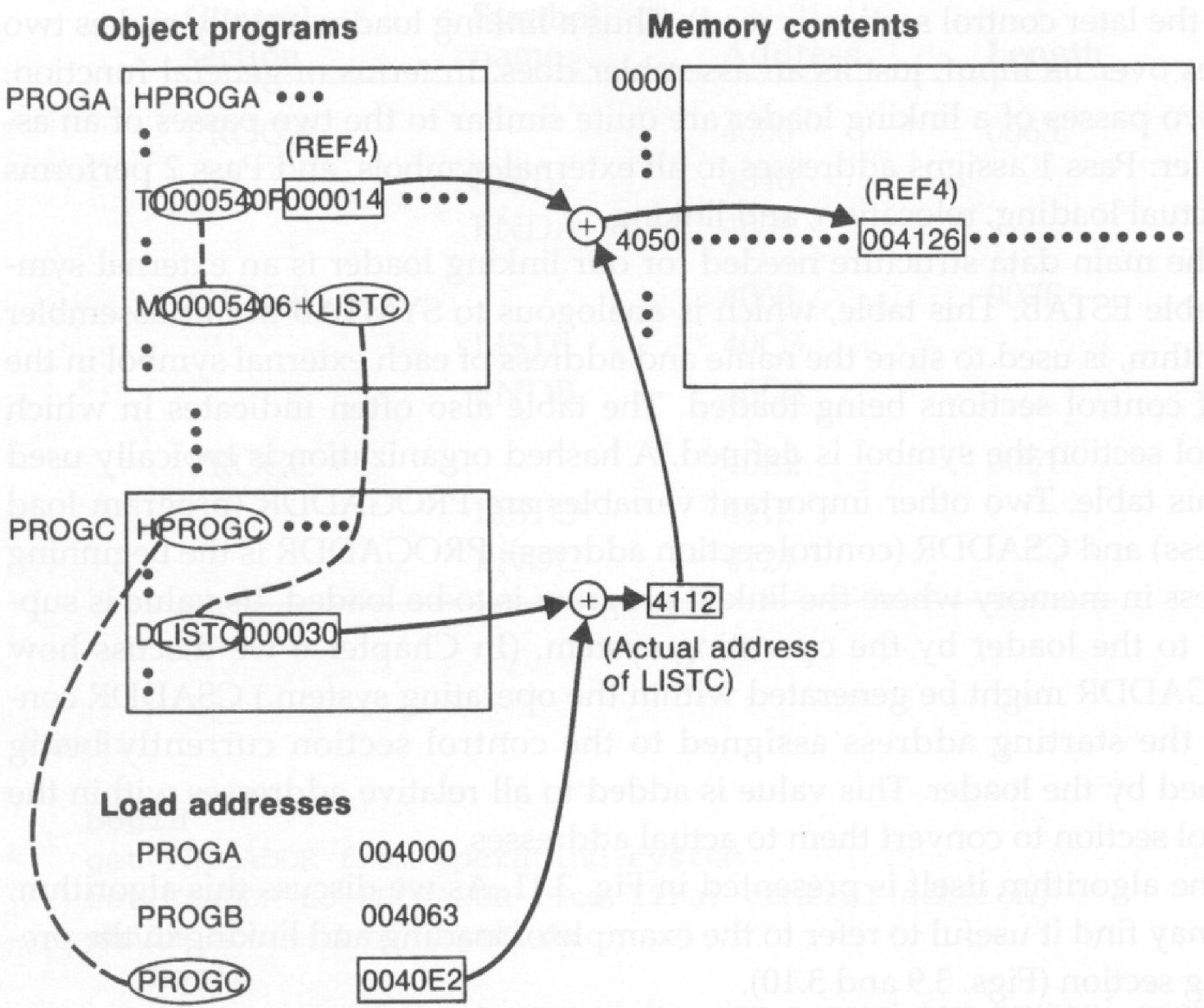


Suppose that program A is loaded at 004000, program B at 004063, and program C at 0040E2.

Notice that REF4, REF5, REF6, and REF7 in all of these three programs have the same values.

REF4 after Linking

- Program A
  - The address of REF4 is 4054 (4000 + 54) because program A is loaded at 4000 and the relative address of REF4 within program A is 54.
  - The value of REF4 is 004126 because
    - The address of LISTC is 0040E2 (the loaded address of program C) + 000030 (the relative address of LISTC in program C)
    - 0040E2 + 000014 (constant already calculated) = 004126.


Object programs
Memory contents
PROGA
HPROGA
(REF4)
T 000054 0F 000014
M 00005406 +LISTC
0000
(REF4)
4050
004126
PROGC
HPROGC
D LISTC 000030
4112
(Actual address
of LISTC)
Load addresses
PROGA 004000
PROGB 004063
PROGC 0040E2

REF4 after Linking

- Program B
  - The address of REF4 is 40D3 (4063 + 70) because program B is loaded at 4063 and the relative address of REF4 within program A is 70.
  - The value of REF4 is 004126 because
    - The address of LISTC is 004112
    - The address of ENDA is 004054
    - The address of LISTA is 004040
    - 004054 + 004112 – 004040 = 004126

Instruction Operands

- For references that are instruction operands, the calculated values after loading do no always appear to be equal.
- This is because there is an additional address calculation step involved for program-counter (base) relative instructions.
- In such cases, it is the target addresses that are the same.
- For example, in program A, the reference REF1 is a program-counter relative instruction with displacement 1D. When this instruction is executed, the PC contains the value 4023. Therefore the resulting address is 4040. In program B, because direct addressing is used, 4040 (4000 + 40) is stored in the loaded program for REF1.

The Implementation of a Linking Loader

- A linking loader makes two passes over its input
  - In pass 1: assign addresses to external references
  - In pass 2: perform the actually loading, relocation, and linking
- Very similar to what a two-pass assembler does.

Data Structures

- External symbol tables (ESTAB)
  - Like SYMTAB, store the name and address of each external symbol in the set of control sections being loaded.
  - It needs to indicate in which control section the symbol is defined.
- PROGADDR
  - The beginning address in memory where the linked program is to be loaded. (given by the OS)
- CSADDR
  - It contains the starting address assigned to the control section currently being scanned by the loader.
  - This value is added to all relative addresses within the control sections.

Algorithm

- During pass 1, the loader is concerned only with HEADER and DEFINE record types in the control sections to build ESTAB.
- PROGADDR is obtained from OS.
- This becomes the starting address (CSADDR) for the first control section.
- The control section name from the header record is entered into ESTAB, with value given by CSADDR.

Algorithm (Cont'd)

- All external symbols appearing in the DEFINE records for the current control section are also entered into ESTAB.
- Their addresses are obtained by adding the value (offset) specified in the DEFINE to CSADDR.
- At the end, ESTAB contains all external symbols defined in the set of control sections together with the addresses assigned to each.
- A Load Map can be generated to show these symbols and their addresses.

A Load Map

Control section	Symbol name	Address	Length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	

Algorithm (Cont'd)

- During pass 2, the loader performs the actual loading, relocation, and linking.
- CSADDR is used in the same way as it was used in pass 1
  - It always contains the actual starting address of the control section being loaded.
- As each text record is read, the object code is moved to the specified address (plus CSADDR)
- When a modification record is encountered, the symbol whose value is to be used for modification is looked up in ESTAB.
- This value is then added to or subtracted from the indicated location in memory.

Machine Independent Features

Loader Options

- Many loaders allow the user to specify options that modify the standard processing.
- For example:
  - Include program-name (library name)
    - Direct the loader to read the designated object program from a library
  - Delete csect-name
    - Instruct the loader to delete the named control sections from the set of programs being loaded
  - Change name1, name2
    - Cause the external symbol name1 to be changed to name2 wherever it appears in the program

Loader Options Application

- In the COPY program, we write two subroutines RDREC and WRREC to perform read records and write records.
- Suppose that the computer system provides READ and WRITE subroutines which has similar but advanced functions.
- Without modifying the source program and reassembling it, we can use the following loader options to make the COPY object program use READ rather than RDREC and WRITE rather than WRREC.

```
Include READ (Util)
Include WRITE (Util)
Delete RDREC, WRREC
Change RDREC, READ
Change WRREC, WRITE
```

Loader Design Options

Linkage Editor

- The difference between a linkage editor and a linking loader:
  - A linking loader performs all linking and relocation operations, including automatic library search, and loads the linked program into memory for execution.
  - A linkage editor produces a linked version of the program, which is normally written to a file for later execution.

Linkage Editor

- When the user is ready to run the linked program, a simple relocating loader can be used to load the program into memory.
- The only object code modification necessary is the addition of an actual address to relative values within the program.
- The linkage editor performs relocation of all control sections relative to the start of the linked program.

Linkage Editor

- All items that need to be modified at load time have values that are relative to the start of the linked program.
- This means that the loading can be accomplished in one pass with no external symbol table required.
- Thus, if a program is to be executed many times without being reassembled, the use of a linkage editor can substantially reduces the overhead required.
  - Resolution of external references and library searching are only performed once.

Dynamic Linking

- Linkage editors perform linking before the program is loaded for execution.
- Linking loaders perform these same operations at load time.
- Dynamic linking postpones the linking function until execution time.
  - A subroutine is loaded and linked to the test of the program when it is first called.

Dynamic Linking Application

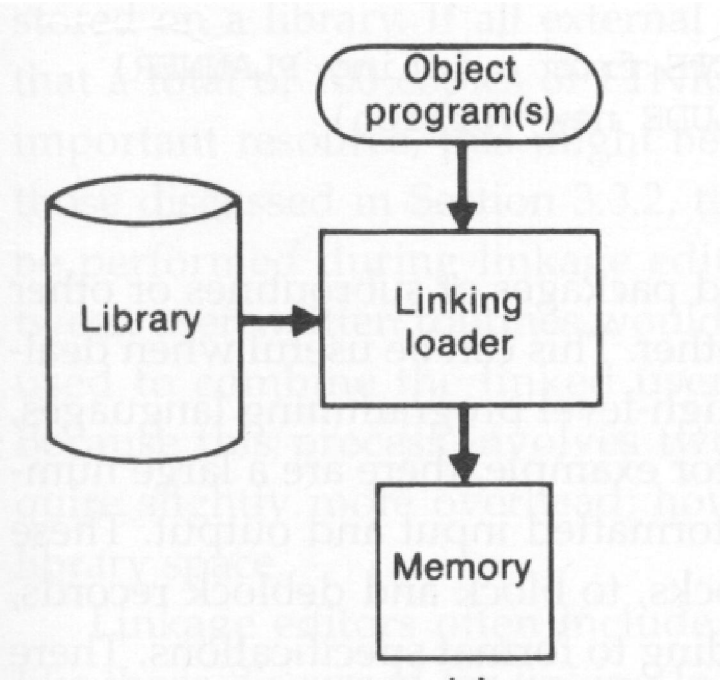
- Dynamic linking is often used to allow several executing programs to share one copy of a subroutine or library.
- For example, a single copy of the standard C library can be loaded into memory.
- All C programs currently in execution can be linked to this one copy, instead of linking a separate copy into each object program.

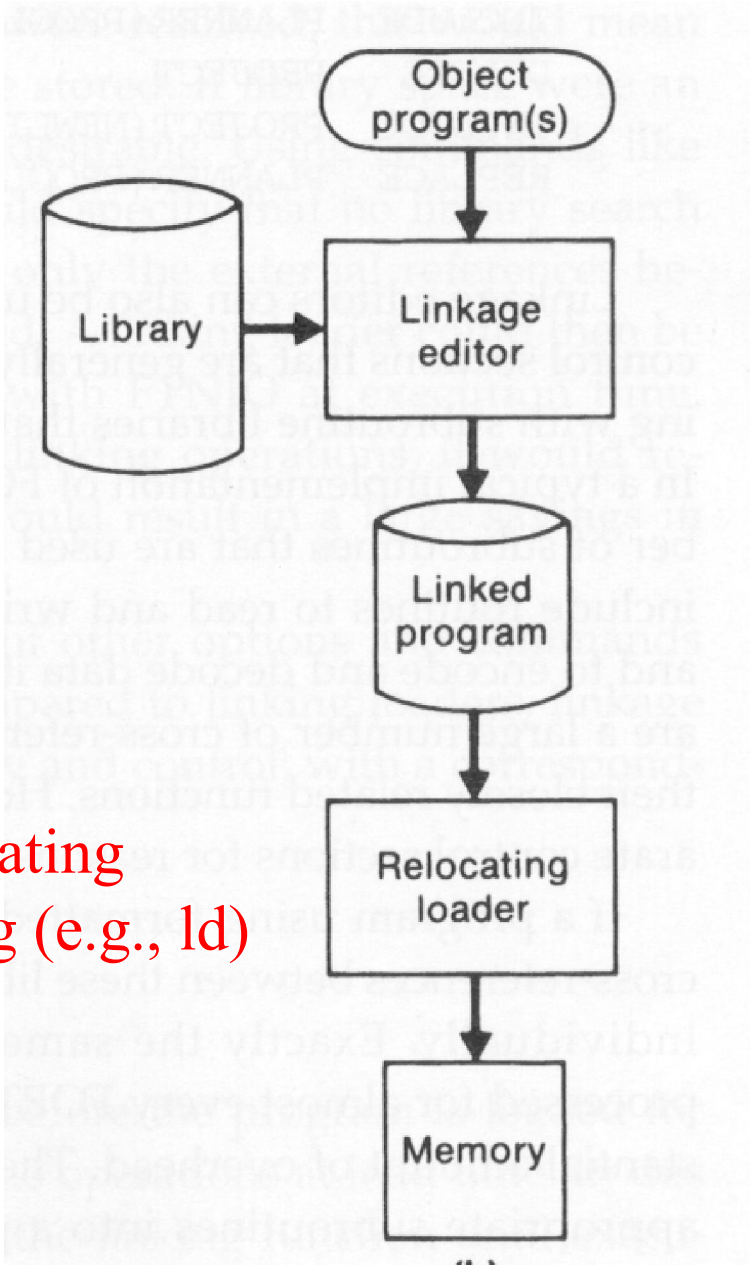
Dynamic Linking Application

- In an object-oriented system, dynamic linking is often used for references to software object.
- This allows the implementation of the object and its method to be determined at the time the program is run. (e.g., C++)
- The implementation can be changed at any time, without affecting the program that makes use of the object.

Dynamic Linking Advantage

- The subroutines that diagnose errors may never need to be called at all.
- However, without using dynamic linking, these subroutines must be loaded and linked every time the program is run.
- Using dynamic linking can save both space for storing the object program on disk and in memory, and time for loading the bigger object program.





On PC Windows or UNIX operating systems, normally you are using (e.g., ld) a linkage editor to generate an executable program.