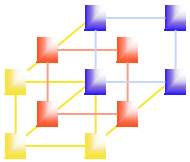
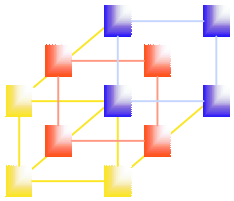
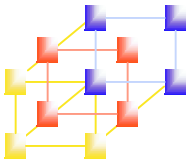


Homework #5

- Goal: Extend your program of hw#3 to support SIC/XE instructions and addressing modes.
  - Implement it as a 2-Pass assembler.
  - The format of the object file must conform with the one shown in Figure 2.20
- Due: 1PM, December 29th.

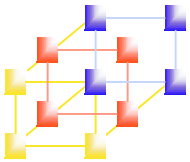


Chapter 3 Loaders and Linkers
-- Machine-Dependent Loader Feature



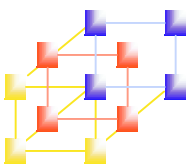
Motivation

- Shortcoming of an absolute loader
  - Programmer needs to specify the actual address at which it will be loaded into memory.
  - It is difficult to run several programs concurrently, sharing memory between them.
  - It is difficult to use subroutine libraries.
- Solution:
  - A more complex loader that provides
    - `Program relocation`
    - `Program linking`



Relocation

- Loaders that allow for program relocation are called *relocating* or *relative* loaders.
- Two methods for specifying relocation as part of the object program
  - Modification records
    - For a small number of relocations required when relative or immediate addressing modes are extensively used
  - Relocation bits
    - For a large number of relocations required when only direct addressing mode can be used in a machine with fixed instruction format (e.g., the standard SIC machine)



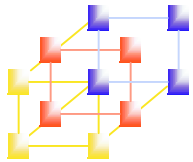
Object program with modification record -- Figure 3.5, pp. 132

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705+COPY
M00001405+COPY
M00002705+COPY
E000000
```

One modification record for each relocation

Modification record

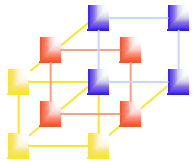
Col 1:	M
Col 2-7:	relocation address
Col 8-9:	length (halfbyte)
Col 10:	flag (+/-)
Col 11-17:	segment name



Relocatable program for SIC -- Figure 3.6, pp. 133

0000	COPY	START	0	
0000	FIRST	STL	RETADR	140033
0003	CLOOP	JSUB	RDREC	481039
0006		LDA	LENGTH	000036
0009		COMP	ZERO	280030
000C		JEQ	ENDFIL	300015
000F		JSUB	WRREC	481061
0012		J	CLOOP	3C0003
0015	ENDFIL	LDA	EOF	00002A
0018		STA	BUFFER	0C0039
001B		LDA	THREE	00002D
001E		STA	LENGTH	0C0036
0021		JSUB	WRREC	481061
0024		LDL	RETADR	080033
0027		RSUB		4C0000
002A	EOF	BYTE	C'EOF'	454F46
002D	THREE	WORD	3	000003
0030	ZERO	WORD	0	000000
0033	RETADR	RESW	1	
0036	LENGTH	RESW	1	
0039	BUFFER	RESB	4096	

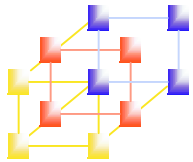
Fixed instruction format
Direct addressing mode



Relocatable program for SIC -- Figure 3.6, pp. 133

```
                .       SUBROUTINE TO READ RECORD INTO BUFFER
                .
1039    RDREC   LDX     ZERO    040030
103C            LDA     ZERO    000030
103F    RLOOP   TD      INPUT   E0105D
1042            JEQ     RLOOP   30103F
1045            RD      INPUT   D8105D
1048            COMP    ZERO    280030
104B            JEQ     EXIT    301057
104E            STCH    BUFFER,X 548039
1051            TIX     MAXLEN  2C105E
1054            JLT     RLOOP   38103F
1057    EXIT    STX     LENGTH  100036
105A            RSUB            4C0000
105D    INPUT   BYTE    X'F1'   F1
105E    MAXLEN  WORD    4096    001000
```

Fixed instruction format
Direct addressing mode

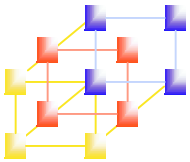


Relocatable program for SIC -- Figure 3.6, pp. 133

```
                .       SUBROUTINE TO WRITE RECORD FROM BUFFER
                .
1061        WRREC   LDX     ZERO        040030
1064        WLOOP   TD      OUTPUT      E01079
1067                JEQ     WLOOP       301064
106A                LDCH    BUFFER,X    508039
106D                WD      OUTPUT      DC1079
1070                TIX     LENGTH      2C0036
1073                JLT     LOOP        381064
1076                RSUB                4C0000
1079        OUTPUT  BYTE    X'05'       05
                    END     FIRST
```

Fixed instruction format
Direct addressing mode

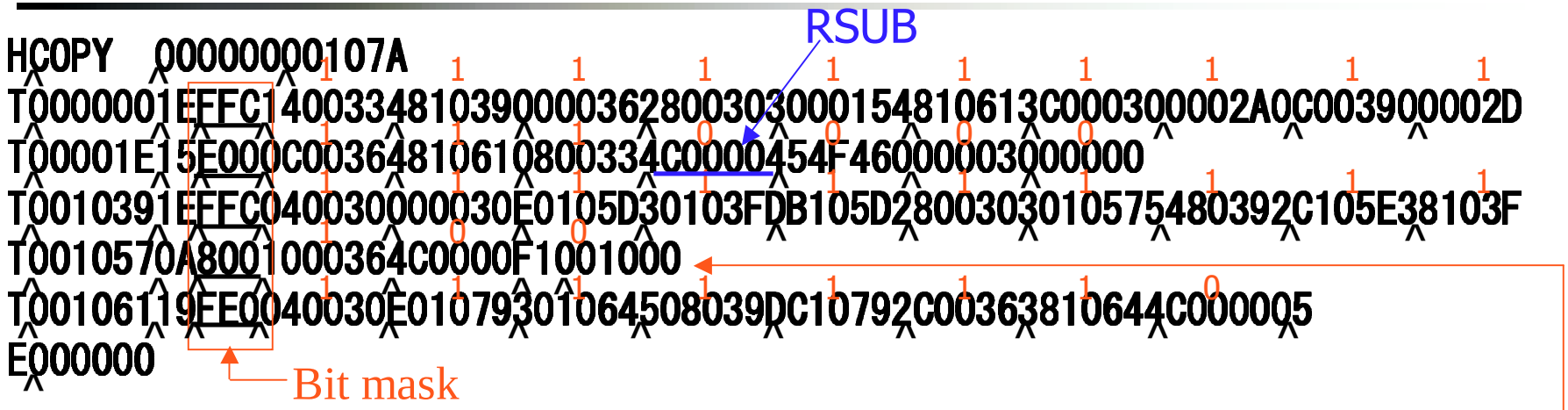
The standard SIC machine does not use relative addressing (*PC-relative*, *Base-relative*)
All instructions expect RSUB in Figure 3.6 need relocation
⇒ too many modification records



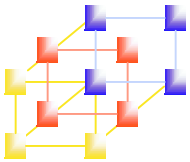
Relocation Bits

- If there are many addresses needed to be modified, it is more efficient to use a relocation bit, instead of a Modification record, to specify every relocation.
  - When the instruction format is fixed
  - There is a relocation bit for each word of the object program
  - Relocation bits are put together into a bit mask
  - 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 Bits -- Figure 3.7, pp. 134

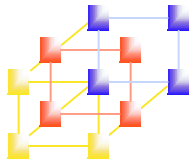


- A bit value of 0 indicates that no modification is necessary or unused words
- A new Text record is created for proper alignment



Program linking

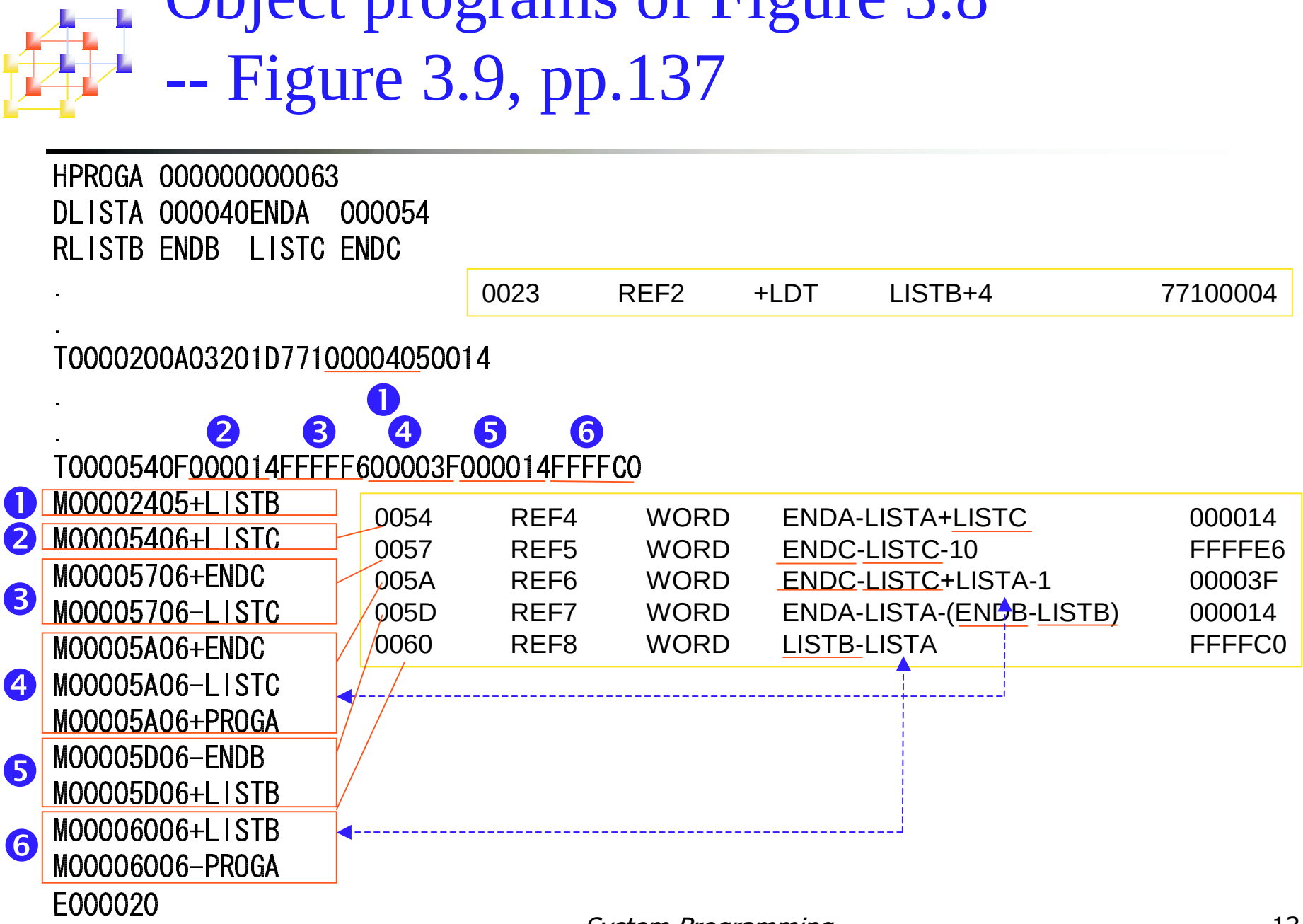
- Goal
  - Resolve the problems with EXTREF and EXTDEF from different control sections (sec 2.3.5)
    - `A program is a logical entity that combines all of the related control sections.`
    - `Control sections could be assembled together, or they could be assembled independently of one another.`
    - `Control sections are to be linked, relocated, and loaded by loaders.`
- Example
  - Program in Figure 3.8 and object code in Figure 3.9
  - Use modification records for both relocation and linking
    - `address constant`
    - `external reference`

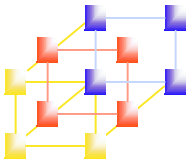


Program for Linking and Relocation -- Figure 3.8, pp.136

```
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
```

Object programs of Figure 3.8 -- Figure 3.9, pp.137





Program for Linking and Relocation -- Figure 3.8, pp.136

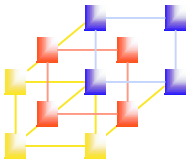
```
0000    PROGB    START    0
                 EXTDEF   LISTB,ENDB
                 EXTREF   LISTA,ENDA,LISTC,ENDC
                 .
                 .
0036    REF1     +LDA     LISTA                  03100000
003A    REF2      LDT     LISTB+4                772027
003D    REF3     +LDX     #ENDA-LISTA            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
```

Object programs of Figure 3.8 -- Figure 3.9, pp.138

```
HPROGB 00000000007F
DLISTB 000060ENDB  000070
RLISTA ENDA  LISTC ENDC
.
.
T0000360B03100000772027051000000
.
.
T0000700F000000FFFFF6FFFFFFFFFFF0000060
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
```

0036	REF1	+LDA	LISTA	03100000
003A	REF2	LDT	LISTB+4	772027
003D	REF3	+LDX	#ENDA-LISTA	05100000

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



Program for Linking and Relocation -- Figure 3.8, pp.137

```
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
```

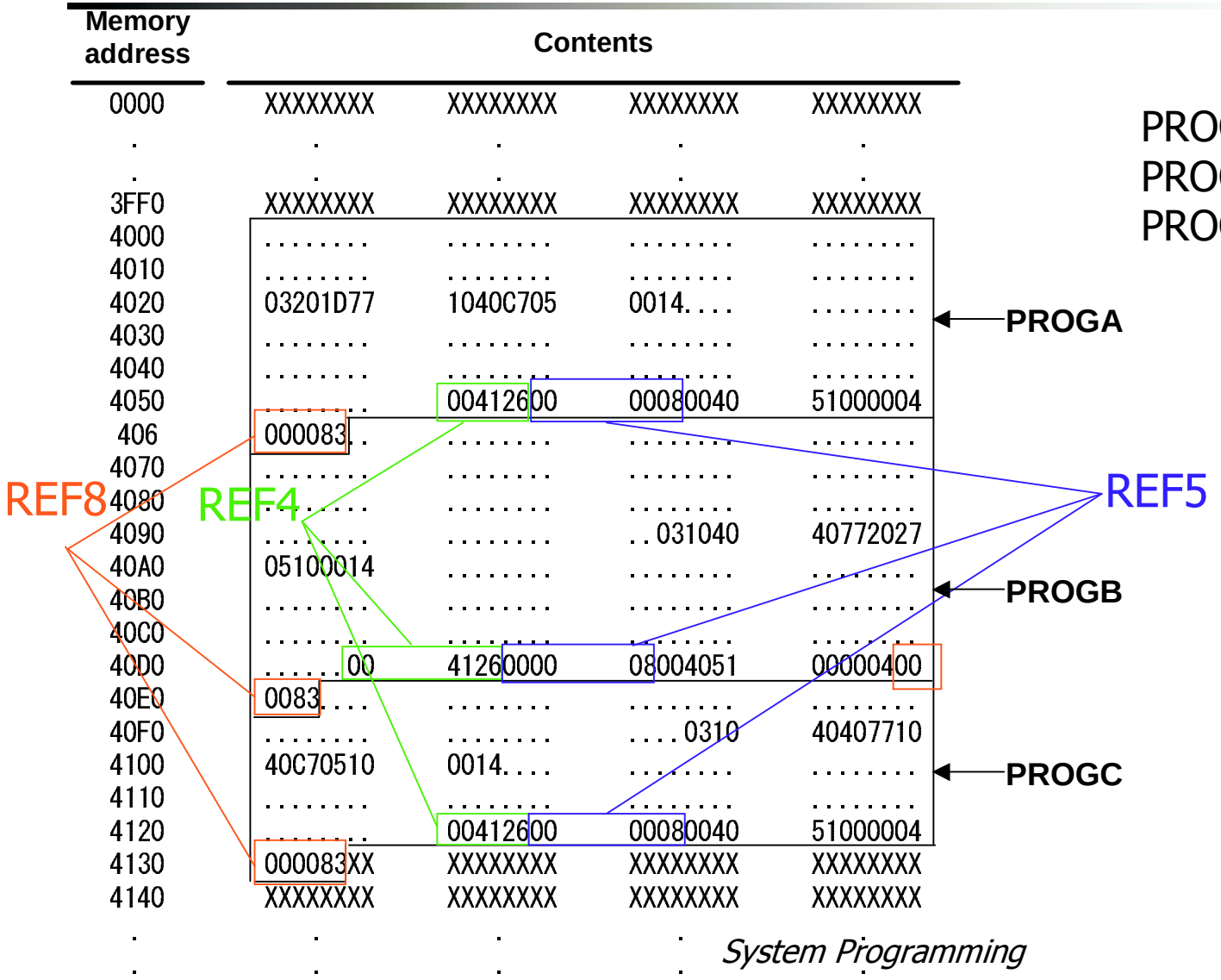
Object programs of Figure 3.8 -- Figure 3.9, pp.138

```
HPROGC 000000000051
DLISTC 000030ENDC  000042
RLISTA ENDA  LISTB ENDB
.
.
T0000180C03100000771000040510000 0
.
.
T0000420F0000300000080000110000000000 0
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
```

0018	REF1	+LDA	LISTA	03100000
001C	REF2	+LDT	LISTB+4	77100004
0020	REF3	+LDX	#ENDA-LISTA	05100000

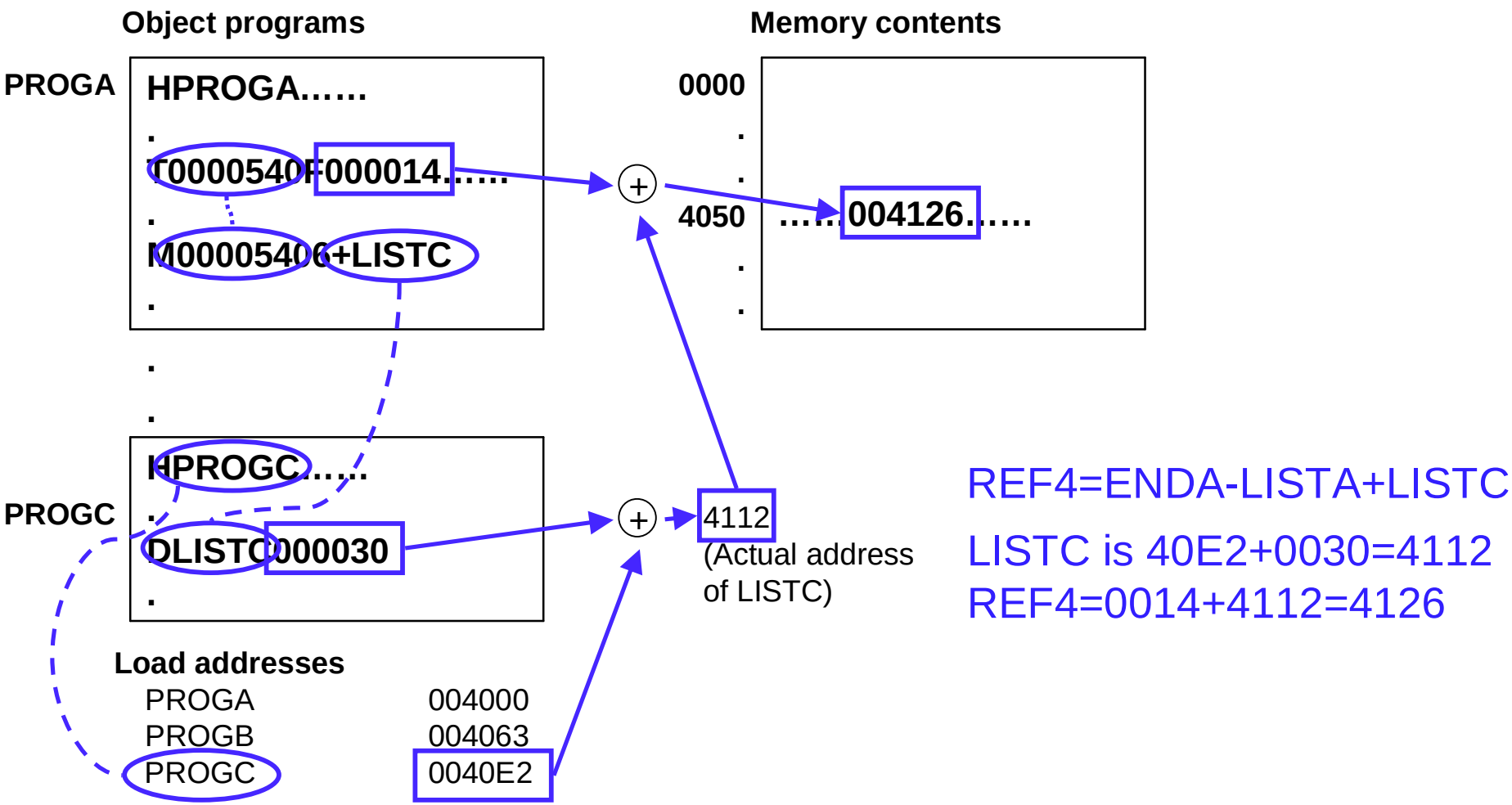
0042	REF4	WORD	ENDA-LISTA+LISTC	000030
0045	REF5	WORD	ENDC-LISTC-10	000008
0048	REF6	WORD	ENDC-LISTC+LISTA-1	000011
004B	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000000
004E	REF8	WORD	LISTB-LISTA	000000

Programs after linking and loading -- Figure 3.10(a), pp.140



PROGA start at 4000
PROGB start at 4063
PROGC start at 40E2

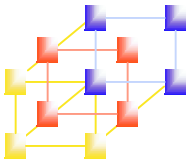
Relocation and linking operations performed on REF4



REF4=ENDA-LISTA+LISTC

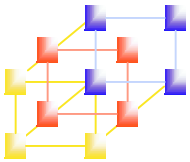
LISTC is 40E2+0030=4112

REF4=0014+4112=4126



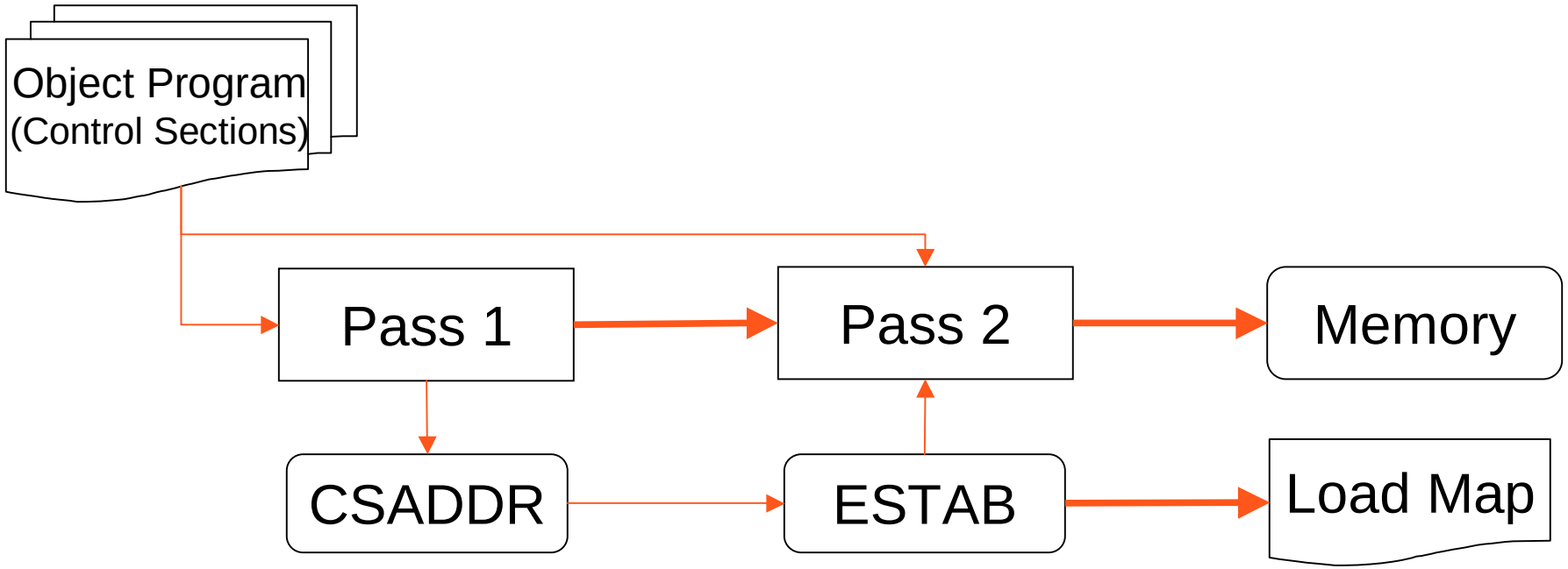
Algorithm and data structure for a linking loader

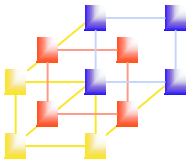
- Section 3.2.3 considers the following conditions:
  - Most instructions use *relative addressing*; no relocation is necessary
  - *Modification records* are used in this type of machine
- A linking loader usually makes two passes over its input:
  - Because some external symbols are processed before read



Two passes linking loader

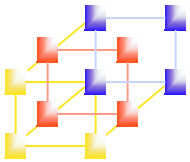
- Two Passes Logic
  - Pass 1: assign addresses to all external symbols
  - Pass 2: perform the actual loading, relocation, and linking





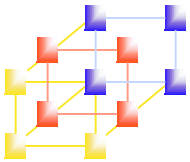
Linking loader -- Pass 1

- Assign address to all external symbols
  - Only processes *Header Record* and *Define Record*
  - Builds an *external symbol table* (*ESTTAB*)
    - its name
    - its address
    - in which control section the symbol is defined
  - Program Load Address (PROGADDR)
    - The beginning address in memory where the linked program is to be loaded (supplied by OS).
  - Control Section Address (CSADDR)
    - The starting address assigned to the control section currently being scanned by the loader.
    - CSADDR is added to all relative addresses within the control section



Linking loader -- Pass 1 (Cond.)

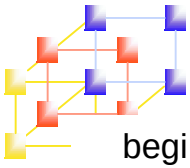
- Add symbol to ESTAB
  - Control section name: (name, CSADDR) → ESTAB
    - `Get control section name from H record`
    - `If the first control section`
      - `CSADDR = PROGADDR`
    - `When E record is encountered, read the next control section`
      - `CSADDR = CSADDR + CSLTH (known from H record)`
  - EXTDEF: (name, CSADDR+value in the record) → ESTAB
    - `Get EXTDEF from D record`



Linking loader -- Pass 1 (Cond.)

- Print the load map if necessary (optional)

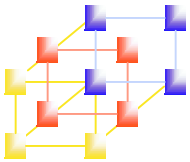
Control section	Symbol name	Address	Length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	



Linking loader -- Pass 1 algorithm

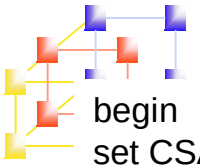
(Only processes *Header Record* and *Define Record*)

```
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 record
               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
                                    (CSADR + indicated address)
                        end {for}
            end {while != 'E'}
      add CSLTH to CSADDR {starting address for next control section}
   end {while not EOF}
end {Pass 1}
```



Linking loader -- Pass 2

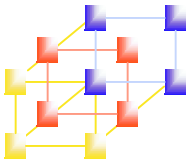
- Perform the actual loading, relocation, and linking
  - Only processes *Text Record* and *Modification Record*
  - Get address of *external symbol* from ESTAB
  - When read T record
    - `Moving object code to the specified address`
  - When read M record
    - `(+/-) EXTREF in M records are handled`
- Last step: transfer control to the address in E
  - If more than one transfer address: use the last one
  - If no transfer: transfer control to the first instruction (PROGADDR)



Linking loader -- Pass 2 algorithm

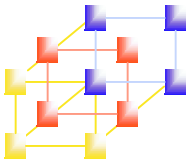
(Only processes *Text Record* and *Modification Record*)

```
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 = '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 EXECADDR {to start execution of loaded program}
end {Pass 2}
```



Linking loader -- Improve efficiency

- We can make the linking loader algorithm more efficient by
  - Assigning a *reference number* to each external symbol referred to in a control section
    - `01: control section name`
    - `02~: external reference symbols`
  - Using this reference number (instead of the symbol name) in Modification records
  - *Avoiding multiple searches* of ESTAB for *the same symbol* during the loading of a control section.
    - `Search of ESTAB for each external symbol can be performed once and the result is stored in a table indexed by the reference number.`
    - `The values for code modification can then be obtained by simply indexing into the table.`



Examples of Using Reference Numbers -- Figure 3.12, pp. 145

PROGA

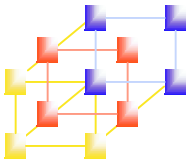
Ref No.	Symbol	Address
1	PROGA	4000
2	LISTB	40C3
3	ENDB	40D3
4	LISTC	4112
5	ENDC	4124

PROGB

Ref No.	Symbol	Address
1	PROGB	4063
2	LISTA	4040
3	ENDA	4054
4	LISTC	4112
5	ENDC	4124

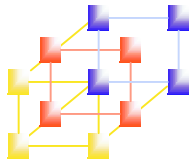
PROGC

Ref No.	Symbol	Address
1	PROGC	4063
2	LISTA	4040
3	ENDA	4054
4	LISTB	40C3
5	ENDB	40D3



Examples of Using Reference Numbers -- Figure 3.12, pp. 145 (Cond.)

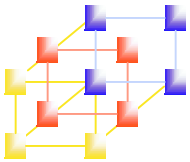
```
HPROGA 000000000063
DLISTA 000040ENDA  000054
R02LISTB 03ENDB  04LISTC 05ENDC
.
.
T0000200A03201D77100004050014
.
.
T0000540F000014FFFFF600003F000014FFFFC0
M00002405+02
M00005406+04
M00005706+05
M00005706-04
M00005A06+05
M00005A06-04
M00005A06+01
M00005D06-03
M00005D06+02
M00006006+02
M00006006-01
E000020
```



Examples of Using Reference Numbers -- Figure 3.12, pp. 145 (Cond.)

```
HPROGB 00000000007F
DLISTB 000060ENDB  000070
R02LISTA 03ENDA  04LISTC 05ENDC
.
.
T0000360B03100000772027051000000
.
.
T0000700F000000FFFFF6FFFFFFFFFFF0000060
M00003705+02
M00003E05+03
M00003E05-02
M00007006+03
M00007006-02
M00007006+04
M00007306+05
M00007306-04
M00007606+05
M00007606-04
M00007606+02
M00007906+03
M00007906-02
M00007C06+01
M00007C06-02
E
```

```
HPROGC 000000000051
DLISTC 000030ENDC  000042
R02LISTA 03ENDA  04LISTB 05ENDB
.
.
T0000180C03100000771000040510000 0
.
.
T0000420F000030000008000011000000000000
M00001905+02
M00001D05+04
M00002105+03
M00002105-02
M00004206+03
M00004206-02
M00004206+01
M00004806+02
M00004B06+03
M00004B06-02
M00004B06-05
M00004B06+04
M00004E06+04
M00004E06-02
E
```



Mid-Term Exam

- Date: December 30th, Friday
- Time: 09:10-12:00

- Scope:
  - Section 2.1 – Section 2.4
  - Section 3.1 – Section 3.2

- (Open book)