TAMKANG UNIVERSITY
SOFTWARE ENGINEERING GROUP
淡江軟體工程實驗室 http://www.tkse.tku.edu.tw

系統程式

淡江大學資訊工程系 教授
王英宏

2010/2/23

授課方式與成績考核

- 上課方式
  - 板書爲主、投影片相輔
  - 隨堂作業與小考，隨堂作業爲主
  - 隨時自備A4紙
- 上課規定
  - 手機請改設震動或關機
  - 不要私下講話
  - 鼓勵提問
- 教材
  - 教科本
    - System Software Beck
      台北圖書 02-23625376
- 成績考核
  - 出席：15%
  - 助教：20%
  - 作業：30%
  - 期中考試：15%
  - 期末考試：20%

授課方式與成績考核

- 成績評定
  - 出席: 15% 實習課: 20% 作業+程式: 30% 期中考: 15% 期末考: 20%
  - 點名成績係用以區隔同學每週出席與否的成績，故不接受各種形式未出席的原因，但每人可以有三次的緩衝機會，以因應不可抗拒的情況。缺席超過三次者，才開始扣分，反之，缺席少於三次者，可以獲得加分。
  - 每週點名一次，點名超過一次者，即爲加分點名，不在前述100%內累計。正規點名可以接受補點，加分點名則不接受補點
  - 期末考比校訂時間提前一週
  - 指派聆聽演講之出席，每場加一分(採外加計分)
  - 除隨堂作業與上機驗收作業外，其餘作業可接受補交，自繳交截止時間點起，每24小時內補交者扣10分，扣至0分爲止
  - 請助教作業每次發還後隨即公佈登錄資料以供核對，有疑義者須於一週內完成補正，逾期不再受理

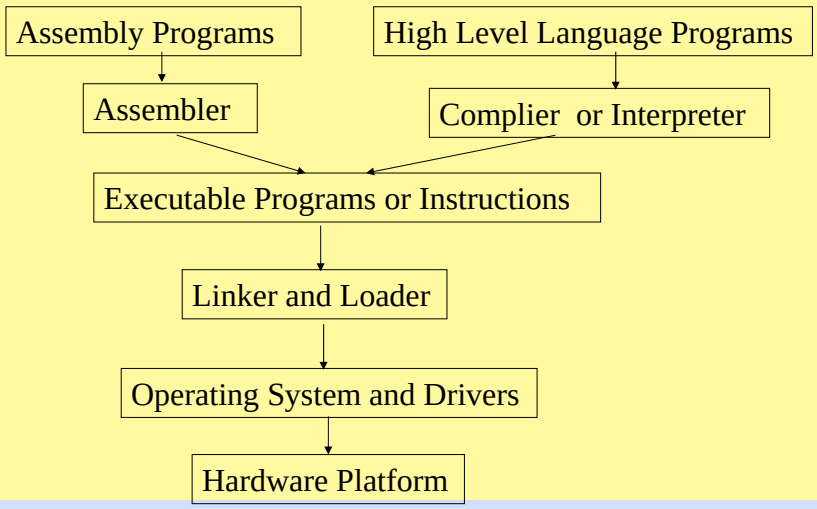
課程內容

- 正課部份
  - Introduction
  - Platform Architecture
  - Assembler
  - Loader and Linker
  - Macro Processor
- 助教部份
  - 作業系統(Operating System)

哈佛大學圖書館自習室座右銘

- 此刻打盹，你將做夢；此刻學習，你將圓夢。
- 學習時的苦痛是暫時的，未學到的痛苦是終生的。
- 現在睡覺流的口水，將成為明天的眼淚。

課程結構



Introduction

- What is System Software(System Programming)
  - Assembler
  - Loader
  - Macro processor
  - Compiler
  - Operating System
  - Data Base Management System
  - User Interface Shell

◎本課程目的在於介紹上述軟體的各型態、類別與設計方法

系統軟體的共同特徵

- What is the difference between System Software and Application Software
  - Machine Dependent
    - Reason
- How to learn those various types of System Software
  - 異中求同

Platform Architecture

- SIC
  - Simple Instructional Computer
    - A standard Model
    - 具有一般電腦大部分的Hardware features and concepts
    - 類似典型的Microcomputer
- SIC/XE
  - eXtra Equipment model of SIC
    - a extended model
    - 由SIC的標準Model加以功能擴充，藉以了解系統軟體在面臨硬體差異時，設計上的調整

SIC的硬體特徵

- Memory Structure
  - 8-bits Byte
  - 3-bytes Word
  - Max. Memory size: 2^{15} = 32,768 bytes
  - Byte address: memory address是以一個byte爲單位，每個byte賦予一個位址編號
  - Word address是以三個連續bytes中最低的byte，亦即位址最小的byte address爲word的address

SIC的硬體特徵(續)

- Register
  - 5個特定用途的暫存器
  - 每個Register容量爲24 bits (3 bytes, one word)
  - 助憶符號、編號、及功能如下：

Mnemonic	Number	Special Use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; used for the JSUB instruction stores the return address
PC	8	Program Counter; contains the address of next instruction
SW	9	Status Word; conatins a variety of information

SIC的硬體特徵(續)

- Data Format
  - Integer: 24-bit binary number
  - 負數：2‘s complement
  - Character: 8-bit ASCII codes
  - No floating -point number
- Instruction Format
  - Only one format: two types of addressing

8	1	15
Opcode	X	Address

X: is used to indicate indexed -addressing mode; 0: direct, 1: indexed

SIC的硬體特徵(續)

- Addressing Mode
  - Direct: TA = address of Instruction Format
  - Indexed: TA = address + (X)
  - TA: Target address
- Instruction Set
  - Register's Load & Store: LDA, STA,, LDX, STX, ….etc.
  - Integer Arithmetic: ADD, SUB, MUL, DIV, … etc.
  - Compare register A with a Word in memory: COMP
  - Conditional Jump: JLT, JEQ, JGT, …etc.
  - Subroutine Call and Return: JSUB and RSUB

SIC的硬體特徵(續)

- Input/Output
  - 只允許一次一個byte的輸入輸出
  - TD: Test Device
  - RD: Read Data from Device
  - WD: Write Data to Device
- Complete Instruction Set of SIC and SIC/XE
  - Appendix A, page 496

SIC/XE的硬體特徵

- Memory Structure
  - Same as SIC
  - Max. memory size is 2^{20} = 1M bytes
- Registers:
  - Five registers are same as SIC
  - Four another register as following:

Mnemonic	Number	Special Use
B	3	Base register; used for addressing
S	4	General data register; no special use
T	5	General data register; no special use
F	6	Floating point accumulator (48 bits)

SIC/XE的硬體特徵(續)

- Data Format
  - Integer data type, Negative representation, Character data type are same as SIC
  - Floating-point data type: 48 bits, format is shown as following:

1	11	36
S	Exponent	Fraction

  - The range is $\pm .F * 2^{(E-1024)}$, where bit S = 0 means positive, else means negative.
  - F is the value of Fraction, E is the value of Exponent

SIC/XE的硬體特徵(續)

- Instruction Formats
  - There are Four types of instruction format of SIC/XE

Format 1: (1 byte)

8
OP

Format 2: (2 bytes)

8	4	4
OP	r1	r2

Format 3: (3 bytes)

6	1	1	1	1	1	1	12
OP	n	i	x	b	p	e	Disp

Format 4: (4 bytes)

6	1	1	1	1	1	1	20
OP	n	i	x	b	p	e	address

Bit e presents format 3 or 4, when e=0 is format 3, e=1 is format 4

SIC/XE的硬體特徵(續)

- Addressing modes
  - There are Four types of addressing mode, they are
    - Relative addressing mode, they can divide into three categories:
      - Index relative: TA = (X) + Disp
      - Base relative: TA = (B) + Disp, ($0 \le$ Disp ≤ 4095)
      - Program counter relative: TA = (PC) + Disp ($-2048 \le$ Disp ≤ 2047)
      - Where Index relative can be combined with Base relative or Program counter relative. However, Base relative and Program counter relative are mutual exclusive
    - Direct addressing mode: TA = address
    - Immediate addressing mode: (TA) = Disp or address
    - Indirect addressing mode: TA = (address)

SIC/XE的硬體特徵(續)

- The relationship between Addressing modes and bits n, i, x, b, p, e
- n = 1, i = 1, for relative or direct addressing modes
  - Index relative: x = 1
  - Base relative: b = 1, and p = 0
  - Program counter relative: b = 0, and p = 1
  - Direct addressing mode: b = 0, and p = 0
  - Immediate addressing mode: n = 0, and i = 1
  - Indirect addressing mode: n = 1, and i = 0
  - Immediate / Indirect addressing mode can combine with relative addressing mode
- n = 0 and i = 0 means this is a instruction belongs to SIC, not SIC/XE

Examples of addressing modes of SIC/XE

Memory Address	Contents
	⋮
3030	003600
	⋮
3600	103000

	⋮
6390	00C303
	⋮
C303	003030

Register	Content
(B)	= 006000
(PC)	= 003000
(X)	= 000090

Machine Instruction Hex	OP	n	i	x	b	p	e	Disp/address	Target Addr.	Value Loaded in reg. A
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	(3030)	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000		000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

SIC/XE的硬體特徵(續)

- Instruction Set of SIC/XE
  - All instructions present in SIC
  - Another new instruction set includes
    - New Register load and store operations: LDB, STS, …etc.
    - Floating-point arithmetic: ADDF, SUBF, DIVF, MULF…
    - Register to Register moving: RMO
    - Register to register arithmetic: ADDR, SUBR, MULR, DIVR
    - Supervisor Call: SVC
  - Input/Output
    - Using byte stream is same as SIC
    - SIC/XE provides I/O instructions for I/O channel, they are SIO: start I/O channel, TIO: Test I/O channel, and HIO: Halt I/O channel

Example

- Shown as Figure 1.5 (a) and (b) present the difference that are written by the instructions of SIC and SIC/XE, respectively.
- Figure 1.5 (b) is shorter and more clear than Figure 1.5 (a).

```
        LDA     ZERO        INITIALIZE INDEX VALUE TO 0
        STA     INDEX
ADDLP   LDX     INDEX       LOAD INDEX VALUE INTO REGISTER X
        LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        LDA     INDEX       ADD 3 TO INDEX VALUE
        ADD     THREE
        STA     INDEX
        COMP    K300        COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX IS LESS THAN 300
        .
        .
        .
INDEX   RESW    1           ONE-WORD VARIABLE FOR INDEX VALUE
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
.                           ONE-WORD CONSTANTS
ZERO    WORD    0
K300    WORD    300
```

(a)

```
        LDS     #3          INITIALIZE REGISTER S TO 3
        LDT     #300        INITIALIZE REGISTER T TO 300
        LDX     #0          INITIALIZE INDEX REGISTER TO 0
ADDLP   LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        ADDR    S,X         ADD 3 TO INDEX VALUE
        COMPR   X,T         COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX VALUE IS LESS THAN 300
        .
        .
        .
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

(b)

Figure 1.5 Sample indexing and looping operations for (a) SIC and (b) SIC/XE.

Hardware Platform 實例探討—Pentium Pro

- Memory Structure
  - Physical Level
    - 8-bit bytes
    - Half Word — 1 byte
    - Full Word — 2 consecutive bytes
    - Double Word — 4 consecutive bytes (dword)
    - 處理dword時，dword address 若能從 4 的倍數開始，處理速度會更快
    - Byte addresses
    - word/dword address是以連續bytes中最低的byte，亦即位址最小的byte address爲word/dword的address
  - Programmer View
    - Memory由數個Segment組成
    - 一個Memory address分成Segment No. + Offset
    - 每個Segment的Size可以不同
    - 一個Segment可以再分成數個Pages
    - 同一個Segment的Pages可以不必同時存在於Memory中，只要正在執行指令所屬的Page在Memory中即可，其他的Page可以暫存於Disk中
    - MMU(Memory Management Unit)負責將Segment/offset的邏輯位址轉換成實際的Physical address.

Hardware Platform 實例探討—Pentium Pro

- Registers
  - 8 general-purpose registers, they are EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, each of them are 32-bit length
    - In general, EAX, EBX, ECX, and EDX are used to store data. The ESI, EDI, EBP and ESP are used to store address, of course, they can store data.
    - All of EAX, EBX, ECX, EDX can be separated into four of one byte, two of one full word, or one dword
    - All of these general-purpose registers can be used as Base register. They can also be used as Index register, ESP is exception
  - EIP use to store the address of next instruction that will be executed. It has 32-bit length
  - FLAGS includes many various flag bits. Its length is 32 bits too.
  - 6 segment registers, they are CS, DS, SS, ES, FS, GS. All of them are 16-bit length
  - There is a Floating-Point Unit (FPU) for floating-point operations. It has eight 80-bit data register and some registers for control and status.

Hardware Platform 實例探討—Pentium Pro

- Data Format
  - 8-, 16-, 32-bits binary integer
  - They can represent as signed or unsigned binary integer
  - They can also use 2's complement to represent negative number
  - FPU can process the operations of 64-bit length integer
  - Little-endian byte ordering: 整數的儲存必須是低有效位數 (Least Significant part)存在同一word或dword最低編號的記憶體位址
  - Integer 亦可用BCD(Binary Coded Decimal)表示，包括Unpacked (或稱Zoned)與Packed兩種

Hardware Platform 實例探討—Pentium Pro

- Floating-point 有下列三種格式表示法：

格式一、單精確度；32bits

1	7	24
S	Exponent	Fraction(Significant)

格式二、雙精確度；64bits

1	10	53
S	Exponent	Fraction (Significant)

格式三、擴增精確度(Extended)；80bits

1	15	64
S	Exponent	Fraction (Significant)

Hardware Platform 實例探討—Pentium Pro

- Character: 8-bit ASCII
- String: 由bits, bytes, words, 或 dwords 組成的 code 均視爲 string

■ Instruction Format

- 如下所示：
- 每個 field的長度都不固定，其中Opcode至少爲 1 個 byte，其他field可以是 0 個byte

Prefix	Opcode	Operand	Address mode

Hardware Platform 實例探討—Pentium Pro

- Addressing Mode
  - Immediate mode
  - Register mode
  - EIP relative mode
  - Direct mode
  - TA = (base) + (index) * scale-factor + disp
    - Where scale-factor may be 1, 2, 4, or 8
    - Disp may be 8-, 16-, or 32-bit binary value

Chap. 2 Assembler

- Basic Function of Assembler
- Machine-Dependent Features of Assembler
- Machine-Independent Features of Assembler
- Optional Features of Assembler

Basic Functions

- This chapter will introduce How to design and Implement an "Assembler"
- Assembler的基本工作
  1. 配置實際的記憶體位址給每一個指令及Programmer所用到的Symbolic label
  2. 將助憶指令(即Assembly instruction)轉換成相對的機器語言指令(Machine Language/Machine Code/Object code)
- 由於Assembly language是Machine dependent，因此Assembler也是Machine dependent的。故首先以SIC與SIC/XE爲範本，了解Assembler的設計與製作，再探討其他Assembler應具有的特性

解讀SIC Assembly 程式

- 首先以SIC的Assembly爲例，因爲SIC只提供Direct與Indexed兩種定址模式，因此在Figure 2.1的組合語言程式範例中，指令中包含有, X的表示使用Indexed定址模式，否則爲Direct定址模式
- Figure 2.1的程式碼中，每一指令列由Line No., Source statement, 及Comments組成。其中Source statement又由下列三部份組成：
  - [LABEL]
  - [OPCODE]
  - [OPERAND]
- 其中[LABEL]與[OPERAND]可以是null，[OPCODE]不爲null

```
Line        Source statement

 5    COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
10    FIRST     STL     RETADR      SAVE RETURN ADDRESS
15    CLOOP     JSUB    RDREC       READ INPUT RECORD
20              LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
25              COMP    ZERO
30              JEQ     ENDFIL      EXIT IF EOF FOUND
35              JSUB    WRREC       WRITE OUTPUT RECORD
40              J       CLOOP       LOOP
45    ENDFIL    LDA     EOF         INSERT END OF FILE MARKER
50              STA     BUFFER
55              LDA     THREE       SET LENGTH = 3
60              STA     LENGTH
65              JSUB    WRREC       WRITE EOF
70              LDL     RETADR      GET RETURN ADDRESS
75              RSUB                RETURN TO CALLER
80    EOF       BYTE    C'EOF'
85    THREE     WORD    3
90    ZERO      WORD    0
95    RETADR    RESW    1
100   LENGTH    RESW    1           LENGTH OF RECORD
105   BUFFER    RESB    4096        4096-BYTE BUFFER AREA
110     .
115     .               SUBROUTINE TO READ RECORD INTO BUFFER
120     .
125   RDREC     LDX     ZERO        CLEAR LOOP COUNTER
130             LDA     ZERO        CLEAR A TO ZERO
135   RLOOP     TD      INPUT       TEST INPUT DEVICE
140             JEQ     RLOOP       LOOP UNTIL READY
145             RD      INPUT       READ CHARACTER INTO REGISTER A
150             COMP    ZERO        TEST FOR END OF RECORD (X'00')
155             JEQ     EXIT        EXIT LOOP IF EOR
160             STCH    BUFFER,X    STORE CHARACTER IN BUFFER
165             TIX     MAXLEN      LOOP UNLESS MAX LENGTH
170             JLT     RLOOP         HAS BEEN REACHED
175   EXIT      STX     LENGTH      SAVE RECORD LENGTH
180             RSUB                RETURN TO CALLER
185   INPUT     BYTE    X'F1'       CODE FOR INPUT DEVICE
190   MAXLEN    WORD    4096
195     .
200     .               SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
210   WRREC     LDX     ZERO        CLEAR LOOP COUNTER
215   WLOOP     TD      OUTPUT      TEST OUTPUT DEVICE
220             JEQ     WLOOP       LOOP UNTIL READY
225             LDCH    BUFFER,X    GET CHARACTER FROM BUFFER
230             WD      OUTPUT      WRITE CHARACTER
235             TIX     LENGTH      LOOP UNTIL ALL CHARACTERS
240             JLT     WLOOP         HAVE BEEN WRITTEN
245             RSUB                RETURN TO CALLER
250   OUTPUT    BYTE    X'05'       CODE FOR OUTPUT DEVICE
255             END     FIRST
```

Figure 2.1 Example of a SIC assembler language program.

解讀SIC Assembly 程式 (Cont.)

- 此外，由・所引出的指令列爲整列的註解(Comments) ，Assembler不需處理
- 另外，START、END、BYTE、WORD、RESB、RESW等指令只是用來指示Assembler執行一些對應的工作，並沒有實際的機器指令與之對應。故稱爲Assembler directives，或稱爲Pseudo Instructions，或稱爲Virtual Instructions
- Figure 2.1經SIC Assembler處理後，每個指令與Symbolic label的記憶體位址配置及對應的機器碼(Machine code/Object code)應如Figure 2.2所示

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	2039	RDREC	LDX	ZERO	041030
130	203C		LDA	ZERO	001030
135	203F	RLOOP	TD	INPUT	E0205D
140	2042		JEQ	RLOOP	30203F
145	2045		RD	INPUT	D8205D
150	2048		COMP	ZERO	281030
155	204B		JEQ	EXIT	302057
160	204E		STCH	BUFFER,X	549039
165	2051		TIX	MAXLEN	2C205E
170	2054		JLT	RLOOP	38203F
175	2057	EXIT	STX	LENGTH	101036
180	205A		RSUB		4C0000
185	205D	INPUT	BYTE	X'F1'	F1
190	205E	MAXLEN	WORD	4096	001000
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	2061	WRREC	LDX	ZERO	041030
215	2064	WLOOP	TD	OUTPUT	E02079
220	2067		JEQ	WLOOP	302064
225	206A		LDCH	BUFFER,X	509039
230	206D		WD	OUTPUT	DC2079
235	2070		TIX	LENGTH	2C1036
240	2073		JLT	WLOOP	382064
245	2076		RSUB		4C0000
250	2079	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 2.2 Program from Fig. 2.1 with object code.

Basic functions of SIC Assembler

① Convert mnemonic operation codes to their machine language equivalents－e.g., translate STL to 14 (shown as line 10 of Figure 2.2)

② Convert symbolic operands to their equivalent machine addresses －e.g., translate RETADR to 1033 (shown as line 10 of Figure 2.2)

③ Build the machine instructions in the proper format

④ Convert the data constants specified in the source program into their internal machine representations－e.g., translate EOF to 454F46 (shown as line 80 of Figure 2.2)

⑤ Write the object program (Obj file) and the assembly listing (LST file)

Basic functions of SIC Assembler (Cont.)

- 一般而言，爲了處理如Line No. 10 這種延後方定義位址的symbol，Assembler的處理方式都會分成Two Passes (2回合)的處理
  - ①Pass One：執行原始程式的Scan，並賦予每個指令及Label definition的address
  - ②Pass Two：執行主要的Translation工作

Basic functions of SIC Assembler (Cont.)

- Assembler的Output
  - Object File (Object Program)：爲了說明方便，以record的型式表示與說明。SIC Assembler所Output的Object file檔案格式分別由三種Record type組成

① Header Record：Object file的第一個Record，一個Object file僅一個，Record的組成如下：

1. Col. 1 H
2. Col. 2-7 Program Name
3. Col. 8-13 Starting address of object program (with Hexadecimal)
4. Col. 14-19 Length of object program in bytes (with Hexadecimal)

② Text Record：包含程式Object codes的Record，一個Object file含有一個以上的Text record ，Record的組成如下：

Basic functions of SIC Assembler (Cont.)

1. Col. 1 T
2. Col. 2-7 Starting address for object codes in this record(with Hexadecimal)
3. Col. 8-9 Length of object codes in this record in bytes (with Hexadecimal)
4. Col. 10-69 Object codes, represented in Hexadecimal (Two column per byte of object code)

③ End Record：指示Object program結束，一個Object file只有一個End record ，Record的組成如下：

1. Col. 1 E
2. Col. 2-7 Address of first executable instruction in object program(with Hexadecimal)

❖ Example of Object Program corresponding to Figure 2.2 is shown as Figure 2.3

Object Program (Object File)

```
HCOPY  001000001 07A
T0010001E141033482039001036281030301015482061 3C1003 00102A0C103900102D
T00101E150C10364820610810334C0000454F46000003000000
T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
T0020571C1010364C0000F1001000041030E02079302064509039DC20792C1036
T002073073820644C000005
E001000
```

Figure 2.3 Object program corresponding to Fig. 2.2.

Detail Processes of the Two Passes of Assembler

- Pass One (Define Symbols):
  - ① Assign addresses to all statements in the program
  - ② Save the values (addresses) assigned to all labels for use in Pass Two
  - ③ Perform some processing of assembler directives (This includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW, …etc.)
- Pass Two (Assemble instructions and generate object program):
  - ① Assemble instructions (Translating operation codes and looking up addresses)
  - ② Generate data values defined by BYTE, WORD, …etc.
  - ③ Perform processing of assembler directives not done during Pass One
  - ④ Write the object program and the assembly listing

Data Structure that Assembler need to

- **Operation Code Table**：簡稱**OPTAB**
  - Assembler用來對照助憶指令轉換成機器指令
  - 應至少包含下列四欄位
    - 助憶指令
    - 對應的機器指令
    - 指令之Format
    - 指令的Length
  - OPTAB於Pass One之功用，供查詢Source program中的每一指令是否正確
  - OPTAB於Pass Two的功用，用以對照每一助憶指令並轉換成對應的機器指令
  - 應以Hash Table爲其設計結構，並以助憶指令爲Search Key
  - 屬於Static Table

Data Structure that Assembler need to

- **Symbol Table**：簡稱**SYMTAB**
  - Assembler用來對照程式中的Symbol轉換成其值(Value)
  - 應至少包含下列四欄位
    - Symbol的名稱
    - Symbol所對應的值或Address
    - Symbol之Type
    - Symbol的Length
  - SYMTAB於Pass One之功用，填入出現在Source program中位於[LABEL]之symbol，及其對應之LOCCTR的值或其對應之內部表示值
  - SYMTAB於Pass Two的功用，用以對照每一Symbol的值或Address並填入對應的Object code
  - 應以Hash Table爲其設計結構，並以Symbol的名稱爲Search Key
  - Hashing function可以用整個Table length除 Search key的內碼值

Data Structure that Assembler need to

- **LOCCTR**：data type ***unsigned integer***
  - 用來count下一個指令或變數的記憶體位址
- **Starting address**：data type ***unsigned long***
  - 記錄程式被分配的起始位址
- **Program Length**：data type ***unsigned integer***
  - 記錄整個程式的總長度(byte數)
- Source file：
- Intermediate file：包含address, error indicator
- Object file：
- Assembly listing：

Examples of OPTAB and SYMTAB

- **OPTAB**

Name	Opcode	Type	Length
LDA	00	3/4	3/4
ADDF	58	3/4	3/4
COMPR	A0	2	2
FIX	C4	1	1

- **SYMTAB**

Name	Value	Type	Length
BUFFER	1039	R	5 (nibble)
ZERO	1030	R	5 (nibble)
LENGTH	1000	A	6 (nibble)
RDREC	2039	R	6 (nibble)

Algorithm of SIC Assembler

- Pass One shown as Fig.2.4 (a)
- Part one: from 『*read first input line*』 to 『*else initialize LOCCTR to 0*』 that assign the start address of this program and setup the value of LOCCTR
- Part two: from 『*while OPCODE <> 'END' do*』 to 『*end {while}*』 that process the location assignment of assembly instructions and symbolic labels, create SYMTAB contents
- Part three: last two commands create the intermediate file and find the program length
- Pass Two shown as Fig. 2.4 (b)
- Part one: from 『*read first input line*』 to 『*write Header record…*』 that generate the object program and its first record, Header record
- Part two: from 『*initialize first Text record*』 to 『*write last Text record to …*』 that process the object code and Text records generation based on OPTAB and SYMTAB
- Part three: last two commands creates the last record, End record, of object program and the assembly listing

Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
      begin
          save #[OPERAND] as starting address
          initialize LOCCTR to starting address
          write line to intermediate file
          read next input line
      end {if START}
  else
      initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
      begin
          if this is not a comment line then
              begin
                  if there is a symbol in the LABEL field then
                      begin
                          search SYMTAB for LABEL
                          if found then
                              set error flag (duplicate symbol)
                          else
                              insert (LABEL,LOCCTR) into SYMTAB
                      end {if symbol}
                  search OPTAB for OPCODE
                  if found then
                      add 3 {instruction length} to LOCCTR
                  else if OPCODE = 'WORD' then
                      add 3 to LOCCTR
                  else if OPCODE = 'RESW' then
                      add 3 * #[OPERAND] to LOCCTR
                  else if OPCODE = 'RESB' then
                      add #[OPERAND] to LOCCTR
                  else if OPCODE  = 'BYTE' then
                      begin
                          find length of constant in bytes
                          add length to LOCCTR
                      end {if BYTE}
                  else
                      set error flag (invalid operation code)
              end {if not a comment}
          write line to intermediate file
          read next input line
      end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end {Pass 1}
```

Figure 2.4(a) Algorithm for Pass 1 of assembler.

Pass 2:

```
begin
  read first input line (from intermediate file)
  if OPCODE = 'START' then
      begin
          write listing line
          read next input line
      end {if START}
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
      begin
          if this is not a comment line then
              begin
                  search OPTAB for OPCODE
                  if found then
                      begin
                          if there is a symbol in OPERAND field then
                              begin
                                  search SYMTAB for OPERAND
                                  if found then
                                      store symbol value as operand address
                                  else
                                      begin
                                          store 0 as operand address
                                          set error flag (undefined symbol)
                                      end
                              end {if symbol}
                          else
                              store 0 as operand address
                          assemble the object code instruction
                      end {if opcode found}
                  else if OPCODE = 'BYTE' or 'WORD' then
                      convert constant to object code
                  if object code will not fit into the current Text record then
                      begin
                          write Text record to object program
                          initialize new Text record
                      end
                  add object code to Text record
              end {if not comment}
          write listing line
          read next input line
      end {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
end {Pass 2}
```

Figure 2.4(b) Algorithm for Pass 2 of assembler.

Machine Dependent Assembler Features

- Extends the functionalities of SIC assembler to SIC/XE
- How to represent the addressing mode on the assembly programs
  - @: indirect addressing mode
  - #: immediate addressing mode
  - +: extended format (four bytes of this instruction)
  - BASE: a pseudo instruction, inform the Assembler to understand what value of the Base register
- Example of assembly program running on SIC/XE, Fig. 2.5

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
12		LDB	#LENGTH	ESTABLISH BASE REGISTER
13		BASE	LENGTH	
15	CLOOP	+JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	#0	
30		JEQ	ENDFIL	EXIT IF EOF FOUND
35		+JSUB	WRREC	WRITE OUTPUT RECORD
40		J	CLOOP	LOOP
45	ENDFIL	LDA	EOF	INSERT END OF FILE MARKER
50		STA	BUFFER	
55		LDA	#3	SET LENGTH = 3
60		STA	LENGTH	
65		+JSUB	WRREC	WRITE EOF
70		J	@RETADR	RETURN TO CALLER
80	EOF	BYTE	C'EOF'	
95	RETADR	RESW	1	
100	LENGTH	RESW	1	LENGTH OF RECORD
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA
110	.			
115	.	SUBROUTINE TO READ RECORD INTO BUFFER		
120	.			
125	RDREC	CLEAR	X	CLEAR LOOP COUNTER
130		CLEAR	A	CLEAR A TO ZERO
132		CLEAR	S	CLEAR S TO ZERO
133		+LDT	#4096	
135	RLOOP	TD	INPUT	TEST INPUT DEVICE
140		JEQ	RLOOP	LOOP UNTIL READY
145		RD	INPUT	READ CHARACTER INTO REGISTER A
150		COMPR	A,S	TEST FOR END OF RECORD (X'00')
155		JEQ	EXIT	EXIT LOOP IF EOR
160		STCH	BUFFER,X	STORE CHARACTER IN BUFFER
165		TIXR	T	LOOP UNLESS MAX LENGTH
170		JLT	RLOOP	HAS BEEN REACHED
175	EXIT	STX	LENGTH	SAVE RECORD LENGTH
180		RSUB		RETURN TO CALLER
185	INPUT	BYTE	X'F1'	CODE FOR INPUT DEVICE
195	.			
200	.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205	.			
210	WRREC	CLEAR	X	CLEAR LOOP COUNTER
212		LDT	LENGTH	
215	WLOOP	TD	OUTPUT	TEST OUTPUT DEVICE
220		JEQ	WLOOP	LOOP UNTIL READY
225		LDCH	BUFFER,X	GET CHARACTER FROM BUFFER
230		WD	OUTPUT	WRITE CHARACTER
235		TIXR	T	LOOP UNTIL ALL CHARACTERS
240		JLT	WLOOP	HAVE BEEN WRITTEN
245		RSUB		RETURN TO CALLER
250	OUTPUT	BYTE	X'05'	CODE FOR OUTPUT DEVICE
255		END	FIRST	

Figure 2.5 Example of a SIC/XE program.

Machine Dependent Assembler Features

- Compare the two programs shown as Fig.2.1 with Fig. 2.5
  - Line 150 COMP ZERO ⇒ COMP A, S (r-r)
  - Line 165 TIX MAXLEN ⇒ TIXR T (r-r)
  - Line 25 COMP ZERO ⇒ COMP #0 (immediate)
  - Line 55 LDA THREE ⇒ LDA #3 (immediate)
  - Line 70~75 LDL RETADR, RSUB } ⇒ J @RETADR (indirect)
- The assembly listing is shown as Fig. 2.6

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
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
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	

Figure 2.6 Program from Fig. 2.5 with object code.

What functionality need to be changed and extended from SIC Assembler

1. START 0：it is not really to indicate the beginning address of this program from 0, but means this program is a relocatable
2. Translate r-r instruction：register name need to be transferred to the identified number. In usual, the register name and register number are stored in the SYMTAB
3. Calculate the displacement of general r-m instruction. In usual, they are program-counter relative, a few base relative. That is calculate from TA and content of (PC) or (B) and (X) if it is included
4. Process the immediate, indirect and direct (Format 4) instruction

What functionality need to be changed and extended from SIC Assembler

- The rules of displacement calculating
  - First priority is PC-relative
  - When overflow happened, change to Base-relative
- Some examples:
  - Line 10 0000 FIRST STL RETADR
    - When this instruction being executed, the content of Program Counter (*PC*) is 0003
    - According the SYMTAB, the TA of RETADR is 0030
    - Thus, disp = 0030-0003 = 002D
    - Because it use *PC* to calculate, *p*=1 and *b*=0
    - The Opcode of STL is 14 and combine bits *n* and *i*, thus it is 17

000101	1	1	0	0	1	0	0000	0010	1101

    - 17202D is got.

What functionality need to be changed and extended from SIC Assembler

- Some examples (Continues):
  - Line 40 0017 J CLOOP
    - When this instruction being executed, the content of Program Counter (*PC*) is 001A
    - According the SYMTAB, the TA of CLOOP is 0006
    - Thus, *disp* = 0006-001A = -014_{16} = FEC_{16} (2's complement)
    - -014_{16}= -000000010100_2=111111101100_2
    - Because it use *PC* to calculate, *p*=1 and *b*=0
    - The Opcode of J is 3C and combine bits *n* and *i*, thus it is 3F
    - 3F2FEC is got.
    - It means that go back offset

001111	1	1	0	0	1	0	1111	1110	1100

What functionality need to be changed and extended from SIC Assembler

- Some examples (Continues):
  - Line 160 104E STCH BUFFER, X
    - When this instruction being executed, the content of Program Counter (*PC*) is 1051
    - According the SYMTAB, the TA of BUFFER is 0036
    - Thus, *disp* = 0036-1051 = $-101B_{16}$ < -2048_{10} = $-0FFD_{16}$ (Range of PC-relative)
    - Thus, change to Base-relative. Due to BASE LENGTH, therefore (B) = 0033 and TA = 0036. So disp = 0036 – 0033 = 0003
    - Because it use Base and includes X to calculate, *p*=0, *b*=1, x = 1
    - The Opcode of STCH is54 and combine bits *n* and *i*, thus it is 57
    - 57C003 is got.

010101	1	1	1	1	0	0	0000	0000	0011

What functionality need to be changed and extended from SIC Assembler

-2048(-FFD$_{16}$) < disp < 2047(7FF$_{16}$)

Some examples (Continues):

- Compare Line 20 000A LDA LENGTH
- To Line 175 1056 EXIT STX LENGTH
- According the SYMTAB, both of them the TA of LENGTH is 0033
- However, in the Line 20, the (*PC*) = 000D and the displacement of PC-relative, *disp* = 0033-000D = 0026, thus the object code of Line 20 is

000000	1	1	0	0	1	0	0000	0010	0101

- In the line 175, the (*PC*) = 1059 and the displacement of PC-relative, *disp* = 0033 - 1059 = -1026$_{16}$ < -2048$_{10}$(0FFD$_{16}$).
- Change to BASE-relative, the displacement, *disp* = 0033 – 0033 = 0
- Thus the object code of Line 175 is

000100	1	1	0	1	0	0	0000	0000	0000

disp = 0033 - 1059 = -1026$_{16}$ < -2048$_{10}$ (-0FFD_16進位)

What functionality need to be changed and extended from SIC Assembler

Some examples (Continues):

- Line55 0020 LDA #3
  - Immediate addressing mode, *n*=0, *i*=1, operand = 3, i.e. *disp* = 003

000000	0	1	0	0	0	0	0000	0000	0011

- Line 133 103C +LDT #4096
  - When this instruction being executed, it is immediate addressing mode and extended format
  - So disp = 4096
  - *n* = 0, *i* = 1, *p*=0, *b*=0, *e* = 1
  - The Opcode of LDT is 74 and combine bits *n* and *i*, thus it is 75
  - 75101000 is got.

使用format 4

011101	0	1	0	0	0	1	0000	0001	0000	0000	0000

What functionality need to be changed and extended from SIC Assembler

- Some examples (Continues):
  - Combine immediate + PC-relative addressing mode
  - Line 12 0003 LDB #LENGTH
    - According to the SYMTAB, the TA of LENGTH is 0033 and the (*PC*) = 0006, So *disp* = 0033 – 0006 = 002D. By the way, it indicates immediate addressing mode, therefore,
    - $n = 0$, $i = 1$, $x=0$, $p=1$, $b=0$, $e = 0$
    - The Opcode of LDB is 68 and combine bits *n* and *i*, thus it is 69
    - 69202D is got.

011010	0	1	0	0	1	0	0000	0010	1101

What functionality need to be changed and extended from SIC Assembler

- Some examples (Continues):
  - Another example is Combining indirect + PC-relative addressing mode
  - Line 70 002A J @RETADR
    - According to the SYMTAB, the TA of RETADR is 0030 and the (*PC*) = 002D, So *disp* = 0030 – 002D= 0003. By the way, it indicates indirect addressing mode, therefore,
    - $n = 1$, $i = 0$, $x=0$, $p=1$, $b=0$, $e = 0$
    - The Opcode of J is 3C and combine bits *n* and *i*, thus it is 3E
    - 3E2003 is got.

001111	1	0	0	0	1	0	0000	0000	0011

What functionality need to be changed and extended from SIC Assembler

- Some examples (Continues):
  - Another format examples
  - Line 150 1049 COMPR A, S
    - It is a instruction followed FORMAT 2
    - The Opcode of COMPR is A0, the ***id* no.** of A is 0, of **S** is 4

1010	0000	0000	0400

S的編號是4→ 0100

  - Line165 1051 TIXR T
    - It is a instruction followed FORMAT 2
    - The Opcode of TIXR is B8, the ***id* no.** of T is 5, only one register

1011	1000	0101	0000

Program Relocation

- Absolute Program (Absolute Assembly)之缺點
  - 位置固定
  - 無法共用Resource (如Memory)
  - 調整位置時(Change Starting Address) ，需全部重新組譯 (Re-Assembling)
- 改進之道
  - Relocated Program
  - Assembler不能直接assign memory address
  - Assembler仍可以算出相對位址
  - 並留下Information通知Loader哪一部份需要修改
  - 此Information留在Assembler所Output的Object program

Program Relocation

- Fig. 2.7則指出Fig. 2.5的Object codes被relocate在不同memory address時，object codes的變化範例
- 其中如+JSUB RDREC的指令，其對應的object code一直隨著RDREC所配置的記憶體位址而改變
- Assembler對RREC的位址是依據starting address爲0時所計算出來的相對位址，而+JSUB RDREC則是使用直接定址法(Direct Addressing mode)指定RDREC的位址
- 因此，當程式碼被安排在starting address非爲0時，+JSUB RDREC所對應的object code就會指到不正確的位址！！
- 因此，Assembler必須告訴Loader，當它在loading time時負責載入Object code到memory時，要更改哪些記憶體位址的內容，將其加上真正的starting address以調整爲正確的位址
- 反之，非direct addressing mode的指令，其object code完全不需隨著starting address的不同而有所改變，這就是相對定址法的優點

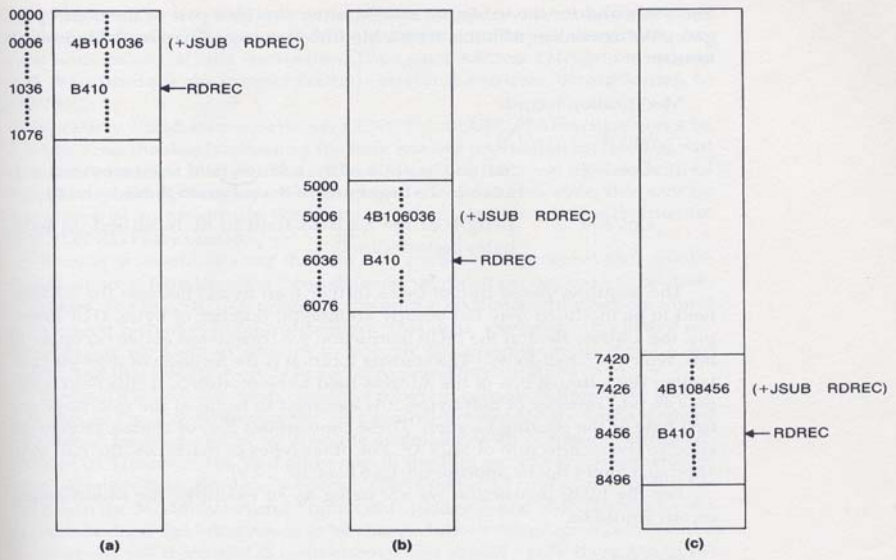


Figure 2.7 Examples of program relocation.

Program Relocation

- Assembler如何通知Loader所需要的information
- Answer:
  - 在Object program中留下*Modification record*
- Modification record的格式如下

  Col. 1 M

  Col. 2~7 Starting location of the address field to be modified, relative to the beginning of the program (hexadecimal)

  Col. 8~9 Length of the address field to be modified, in ***half-bytes*** (hexadecimal)
- Why *half-bytes*？

Program Relocation

- Object program included modification records shown as Fig. 2.8 for source program Fig. 2.5

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```

圖2.8 相對於圖2.6的目的程式

Machine Independent Assembler Features

- 本節將討論一些與machine結構無關，但是在大部分Assembler會共通的特性
  - Literals
  - Symbol Defining statement
  - Expression
  - Program Blocks
  - Control Sections & Program Linking

Literals

- Literals是Assembler提供programmer以constant operand的value作爲指令的一部份，省去爲這些constant定義label，這樣的operand稱爲literal
- Programmer在assembly程式中，必須在operand之前加上'='的符號指明是literal，如:

```
 45    ENDFIL    LDA    =C'EOF'
215    WLOOP     TD     =X'05'
```

其中C表示character，X表十六進位值

Literals

- Literal與immediate operand是不同的:
  - Immediate operand的operand在assembled之後會變成機器指令的一部份，而literal operand則是由assembler自動配予一個address存放這個constant，並將這個constant的memory address作爲指令的target address
  - 比較Fig 2.10的
    - 45 001A ENDFIL LDA =C'EOF' 032010

  及
    - 55 0020 LDA #3 010003
  - 詳細請先參考Fig 2.9及Fig 2.10

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
13		LDB	#LENGTH	ESTABLISH BASE REGISTER
14		BASE	LENGTH	
15	CLOOP	+JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	#0	
30		JEQ	ENDFIL	EXIT IF EOF FOUND
35		+JSUB	WRREC	WRITE OUTPUT RECORD
40		J	CLOOP	LOOP
45	ENDFIL	LDA	=C'EOF'	INSERT END OF FILE MARKER
50		STA	BUFFER	
55		LDA	#3	SET LENGTH = 3
60		STA	LENGTH	
65		+JSUB	WRREC	WRITE EOF
70		J	@RETADR	RETURN TO CALLER
93		LTORG		
95	RETADR	RESW	1	
100	LENGTH	RESW	1	LENGTH OF RECORD
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA
106	BUFEND	EQU	*	
107	MAXLEN	EQU	BUFEND-BUFFER	MAXIMUM RECORD LENGTH
110	.			
115	.	SUBROUTINE TO READ RECORD INTO BUFFER		
120	.			
125	RDREC	CLEAR	X	CLEAR LOOP COUNTER
130		CLEAR	A	CLEAR A TO ZERO
132		CLEAR	S	CLEAR S TO ZERO
133		+LDT	#MAXLEN	
135	RLOOP	TD	INPUT	TEST INPUT DEVICE
140		JEQ	RLOOP	LOOP UNTIL READY
145		RD	INPUT	READ CHARACTER INTO REGISTER A
150		COMPR	A,S	TEST FOR END OF RECORD (X'00')
155		JEQ	EXIT	EXIT LOOP IF EOR
160		STCH	BUFFER,X	STORE CHARACTER IN BUFFER
165		TIXR	T	LOOP UNLESS MAX LENGTH
170		JLT	RLOOP	HAS BEEN REACHED
175	EXIT	STX	LENGTH	SAVE RECORD LENGTH
180		RSUB		RETURN TO CALLER
185	INPUT	BYTE	X'F1'	CODE FOR INPUT DEVICE
195	.			
200	.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205	.			
210	WRREC	CLEAR	X	CLEAR LOOP COUNTER
212		LDT	LENGTH	
215	WLOOP	TD	=X'05'	TEST OUTPUT DEVICE
220		JEQ	WLOOP	LOOP UNTIL READY
225		LDCH	BUFFER,X	GET CHARACTER FROM BUFFER
230		WD	=X'05'	WRITE CHARACTER
235		TIXR	T	LOOP UNTIL ALL CHARACTERS
240		JLT	WLOOP	HAVE BEEN WRITTEN
245		RSUB		RETURN TO CALLER
255		END	FIRST	

Figure 2.9 Program demonstrating additional assembler features.

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
13	0003		LDB	#LENGTH	69202D
14			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	=C'EOF'	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
93			LTORG		
	002D	*	=C'EOF'		454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
106	1036	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	
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	#MAXLEN	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
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	=X'05'	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	=X'05'	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
255			END	FIRST	
	1076	*	=X'05'		05

Figure 2.10 Program from Fig. 2.9 with object code.

Literals

- 通常在程式中所用到的literals會被集中形成一或多個literal pools，而literal pools則置於程式最後面，但如果程式中有宣告LTORG者，則LTORG之前用到的literals則先集中於LTORG之後，如Fig 2.9，2.10
- LTORG的用意是在預防萬一程式太大，前面使用的literals在程式最後分配到的address，會變成large format，因此，LTORG可使literals比較接近用到它的指令.
- 如何辨別duplicate literals:
  - 比較string of literal，例如C'EOF'；X'05'
  - 比較產生的值，如C'EOF'及X'454F46'其實是相同的，可以只存一個
- 另外,這種辨別duplicate literals會產生一種問題爲:
  - 假設我們允許literal使用current value of location counter(記爲 *)，LDB =* , 表示要將目前location counter的值存到Base register.
  - 但是若line 13用到 LDB =* 及line 55用到的 LDA =* ，這時 * 所代表的location counter的value是不同的,分別爲0003及0020，但同樣是literal * 所表示，這樣情況也必須要在literal pool表示出來.

Literals

- 最後，討論assembler如何handle literals
- 首先,assembler必須create一個literal table (LITTAB)
  - LITTAB 的結構爲:
  - literal name
  - operand valve
  - length
  - Address
- LITTAB以hash table的方式組成，以literal name爲 key或以value 爲key
- LITTAB 在Pass1 時產生，Pass1紀錄 ，scan到的literal name及對應的value及length，如果LITTAB內沒有則add進去
- 當Pass1 scan到LTORG或程式最後時，則要產生literal pool，並將address分配給LITTAB中的literal，此時location counter必須跟隨著literal的length改變，以維持address的正確性
- Pass2則是根據LITTAB中literal name對應的address產生適當的機器指令

Symbol Defining Statement

- 允許Programmer自行定義自己的Symbol
- User-defined symbols除了用在label或data area 外,大多的Assembler均會提供programmer自己定義Symbol及指定其value的功能
- 一般的格式如下:

  Symbol EQU value
- EQU常用的情形爲:
  - A. 提高程式的可讀性(readability)

    +LDA #4096 改成

    +LDA #MAXLEN

    MAXLEN EQU 4096

Symbol Defining Statement

- EQU常用的情形爲:

  B.用來以number代替register的助憶名稱

  A EQU 0
  X EQU 1
  L EQU 2

  - 以便讓如RMO這類的指令 ,需要以reg.Number作operand的指令可以用RMO A,X代替RMO 0,1 再由assembler從SYMTAB中找到A,X的Value

Symbol Defining Statement

- 另一種assembler directive是用來reset LOCCTR ,格式如下:

  ORG value

- ORG是origin的意思 , 所以,以下的location改由指定的value開始count
- ORG另一個應用則是用在Table define ,如下例:
  - 假設programmer要定義一個symbol table的structure 如下

Symbol Defining Statement

STAB (100entries)

symbol	valve	flags
6個bytes	1 word	2 byte

- programmer可能會用這種方式來描述

```
STAB      RESB  1100
SYMBOL    EQU   STAB
VALUE     EQU   STAB+6
FLAGS     EQU   STAB+9
```


TAMKANG UNIVERSITY SOFTWARE ENGINEERING GROUP
淡江軟體工程實驗室 http://www.tkse.tku.edu.tw

Symbol Defining Statement

- 之後程式中只要用

  LDA VALUE , X

  即可透過register X 的content indicate table entry而取得 VALUE field
- 上述的例子亦可用ORG完成,方式如下:

```
STAB      RESB  1100
          ORG   STAB
SYMBOL    RESB  6
VALUE     RESW  1
FLAGS     RESB  2
          ORG   STAB+1100
```

Symbol Defining Statement

- 有些assembler允許用ORG代替ORG STAB＋1000 , 這是assembler自動記憶先前的LOCCTR的值
- 另外, EQU right-hand side的Symbol必須是一個已定義的Symbol ,例如:

```
ALPHA   RESW  1
BETA    EQU   ALPHA
```

- 若寫成

```
BETA    EQU   ALPHA
ALPHA   EQU   1
```

則不允許因爲依照Assembler的處理,Pass1 在Scan BETA時, 就必須記住其address,但這種情形,BETA值無法得知

Symbol Defining Statement

- 同樣地,ORG亦不允許這種情形 ,例如:

```
BYTE1   RESB  1
BYTE2   RESB  1
BYTE3   RESB  1
        ORG
ALPHA   RESW  1
```

亦是不允許的

- 更一般性的問題如:

```
ALPHA   EQU   BETA
BETA    EQU   DELTA
DELTA   EQU   1
```

稱爲forward-definition problem或稱forward-reference problem, 是2-pass assembler 所無法容易解決的 , 因此一般的assembler 會加以限制

Expression

- Expression的用法是允許operand的表示不只是Symbol或literal,而是由多個symbol或literal與 + , _ , * , / 等組合而成的 ,例:

  MAXLEN EQU BUFEND-BUFFER
- 其中BUFEND-BUFFER , 就是一個expression
- 由於一般的operand有absolute與relative之分 , 因此 , expression 因其結果也可分成absolute與relative兩種expression , 上例即爲一個absolute expression, 而

  BUFEND+BUFFER

  100-BUFFER

  3*BUFFER
- 則爲relative expression

Expression

- 爲了能夠告知loader哪些Symbol是relative , 哪些是absolute, 必須在SYMTAB中註明
- 以Fig2.10的SYMTBL如下

Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFFEND	R	1036
MAXLEN	A	1000

Program Block

- 前述的Assembler 所處理的 program 包含 subroutines 、 data area 等，均屬於同一的 entity，亦即
- Assembler 產生的 object program 的順序與 source program 的順序是一樣的
- 爲了使 Source & object program 的處理更有彈性，assembler 必須在提供某些 features，如：
  - Some features for 允許產生的 machine instruction 及 data 與 source program 中的順序是不同的 (本書稱爲Program block)
  - Some features for 讓 object program 產生的是一些相互獨立的部分，這些 parts 再由 loader 分開 handle 及 maintain (本書稱爲Control Section)

Program blocks

- Program Block and Control Section
  - Assembler可以提供某些features給Programmer
  - 允許實際上產生的object program(object file)之機器碼可以分成數個Code Segment，且Code Segment的順序與source program之原始指令的順序不同
  - Program Blocks表示Object program內的Object codes會被重新安排在單一object program unit的不同code Segment
  - Control section則是表示Object program內的Object codes會被轉換互相獨立之object program units的Code Segment
  - Control Section於下一節介紹

Program blocks

- Fig2.11即爲一個以Program block寫成的例子
- 本程式共有3個Program block ,
  - 第一個block爲**unnamed** block 包含可執行指令
  - 第二個block爲**CDATA** block 包含全部的DATA area
  - 第三個block爲**CBLKS** block 包含大的記憶體區塊
- Programmer在編撰本程式時被允許交錯安排各個Block，以求可讀性的便利，讓屬於main routine的data與buffer出現在main routine；RDREC routine所用到的data出現在RDREC routine；WRREC routine亦同
- 但是Assembler則需在產生Object file時，將Object code重新安排，以便讓程式指令的集中在一起、變數資料的集中在一起、暫存區在另外一區。類似Intel 80X系列的code segment、data segment、stack segment與extra segment


```
Line        Source statement
  5   COPY      START   0                 COPY FILE FROM INPUT TO OUTPUT
 10   FIRST     STL     RETADR            SAVE RETURN ADDRESS
 15   CLOOP     JSUB    RDREC             READ INPUT RECORD
 20             LDA     LENGTH            TEST FOR EOF (LENGTH = 0)
 25             COMP    #0
 30             JEQ     ENDFIL            EXIT IF EOF FOUND
 35             JSUB    WRREC             WRITE OUTPUT RECORD
 40             J       CLOOP             LOOP
 45   ENDFIL    LDA     =C'EOF'           INSERT END OF FILE MARKER
 50             STA     BUFFER
 55             LDA     #3                SET LENGTH = 3
 60             STA     LENGTH
 65             JSUB    WRREC             WRITE EOF
 70             J       @RETADR           RETURN TO CALLER
 92             USE     CDATA
 95   RETADR    RESW    1
100   LENGTH    RESW    1                 LENGTH OF RECORD
103             USE     CBLKS
105   BUFFER    RESB    4096              4096-BYTE BUFFER AREA
106   BUFEND    EQU     *                 FIRST LOCATION AFTER BUFFER
107   MAXLEN    EQU     BUFEND-BUFFER     MAXIMUM RECORD LENGTH
110   .
115   .         SUBROUTINE TO READ RECORD INTO BUFFER
120   .
123             USE
125   RDREC     CLEAR   X                 CLEAR LOOP COUNTER
130             CLEAR   A                 CLEAR A TO ZERO
132             CLEAR   S                 CLEAR S TO ZERO
133             +LDT    #MAXLEN
135   RLOOP     TD      INPUT             TEST INPUT DEVICE
140             JEQ     RLOOP             LOOP UNTIL READY
145             RD      INPUT             READ CHARACTER INTO REGISTER A
150             COMPR   A,S               TEST FOR END OF RECORD (X'00')
155             JEQ     EXIT              EXIT LOOP IF EOR
160             STCH    BUFFER,X          STORE CHARACTER IN BUFFER
165             TIXR    T                 LOOP UNLESS MAX LENGTH
170             JLT     RLOOP               HAS BEEN REACHED
175   EXIT      STX     LENGTH            SAVE RECORD LENGTH
180             RSUB                      RETURN TO CALLER
183             USE     CDATA
185   INPUT     BYTE    X'F1'             CODE FOR INPUT DEVICE
195   .
200   .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205   .
208             USE
210   WRREC     CLEAR   X                 CLEAR LOOP COUNTER
212             LDT     LENGTH
215   WLOOP     TD      =X'05'            TEST OUTPUT DEVICE
220             JEQ     WLOOP             LOOP UNTIL READY
225             LDCH    BUFFER,X          GET CHARACTER FROM BUFFER
230             WD      =X'05'            WRITE CHARACTER
235             TIXR    T                 LOOP UNTIL ALL CHARACTERS
240             JLT     WLOOP               HAVE BEEN WRITTEN
245             RSUB                      RETURN TO CALLER
252             USE     CDATA
253             LTORG
255             END     FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

Program blocks

- How to do
  - Assembler應提供一個Assembler directive 供programmer指明block名稱及程式中的歸類
  - 本書的範例是使用**USE**這個Assembler指令
  - **USE**指令可以用來指明與前一個相同名稱的block之連續性
  - 由此可知一個program block在一個程式中可能被分成數不同的Code segment(Program Block)
  - Assembler 必須將這些Code segment重新安排
  - 這些重新安排的工作在Pass1完成logical的部分

TAMKANG UNIVERSITY SOFTWARE ENGINEERING GROUP
淡江軟體工程實驗室 http://www.tkse.tku.edu.tw

Program blocks

- How to do (續)
  - 每一個 Program block給予一個location counter(LOCCTR$_i$)
  - 起始值均set爲0 ，處理不同block時就用各自的counter累計，Block的編號也隨之紀錄到SYMTAB的value中
  - 在Pass1最後，各個location counter的值即爲各block之length , 根據這些資料Pass1可以產生一個Working table如下 ：

Block Name	No.	Start Address	Length
(unnamed)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Program blocks

- How to do (續)
  - 這個Table再供Pass2用來產生Object Code
  - 對一個Symbol，Pass2只需從SYMTAB找到它相對於自己的block的位址
  - 再加上working table 中該block的起始位址
  - 即爲此symbol的TA
  - 再由此TA來計算operand的disp值
  - 例如line 20的displacement係由*LENGTH*在SYMTAB中所找到值 0003, 1，再由*LENGTH* 的 block(No.1)知道 block 的start address 爲 0066，故 TA 爲 0069 (0003 + 0066)
  - 又本指令爲pc-relative ，而 (PC)=0009，故得 disp = 0069-0009，故 disp= 0060，由此產生 object code 爲 032060
  - Fig 2.12 即爲 fig 2.11 的 assembly-listing file

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	
110			.			
115			.	SUBROUTINE TO READ RECORD INTO BUFFER		
120			.			
123	0027	0		USE		
125	0027	0	RDREC	CLEAR	X	B410
130	0029	0		CLEAR	A	B400
132	002B	0		CLEAR	S	B440
133	002D	0		+LDT	#MAXLEN	75101000
135	0031	0	RLOOP	TD	INPUT	E32038
140	0034	0		JEQ	RLOOP	332FFA
145	0037	0		RD	INPUT	DB2032
150	003A	0		COMPR	A,S	A004
155	003C	0		JEQ	EXIT	332008
160	003F	0		STCH	BUFFER,X	57A02F
165	0042	0		TIXR	T	B850
170	0044	0		JLT	RLOOP	3B2FEA
175	0047	0	EXIT	STX	LENGTH	13201F
180	004A	0		RSUB		4F0000
183	0006	1		USE	CDATA	
185	0006	1	INPUT	BYTE	X'F1'	F1
195			.			
200			.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205			.			
208	004D	0		USE		
210	004D	0	WRREC	CLEAR	X	B410
212	004F	0		LDT	LENGTH	772017
215	0052	0	WLOOP	TD	=X'05'	E3201B
220	0055	0		JEQ	WLOOP	332FFA
225	0058	0		LDCH	BUFFER,X	53A016
230	005B	0		WD	=X'05'	DF2012
235	005E	0		TIXR	T	B850
240	0060	0		JLT	WLOOP	3B2FEF
245	0063	0		RSUB		4F0000
252	0007	1		USE	CDATA	
253				LTORG		
	0007	1	*	=C'EOF		454F46
	000A	1	*	=X'05'		05
255				END	FIRST	

Figure 2.12 Program from Fig. 2.11 with object code.

Program blocks

- Program block 的優點：
  - Reduces our addressing problems, because large buffer area is moved to the end of the object program
  - Base register is no longer necessary (no more Base-relative)
  - The problem of placement of literals in the program is easily solved
- 實際上Assembler在產生 Object File 時，並不需要重新安排整個object code的順序爲各個code segment (Program Block)的順序
- 只需逢不同 block 時，write 不同 text record 即可，如 fig 2.13
  - 前兩個 text records 爲 line 5 to line 70的object codes
  - 當新的 use 指令出現後，就準備一個新的 text record 來存對應的起始位址與object code

Object Program for Program Block

```
HCOPY  000000001071
T0000001E1720634B2021032060290000 3320064B203B3F2FEE0320550F2056010003
T00001E090F20484B20293E203F
T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000
T00006C01F1
T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T00006D04454F4605
E000000
```

Figure 2.13 Object program corresponding to Fig. 2.11.

Program blocks

- Fig 2.14 則是說明 source program， object program, and memory 配置的 block 順序
  - 其中 CDATA(1) 與 CBLKS(1) 因為沒有 object code，所以在 object program 中並不存在
  - 但是CDATA(1) 與 CBLKS(1) 在memory中仍保有記憶體區塊
  - 因為它們的原始指令是要求Reserve記憶體空間

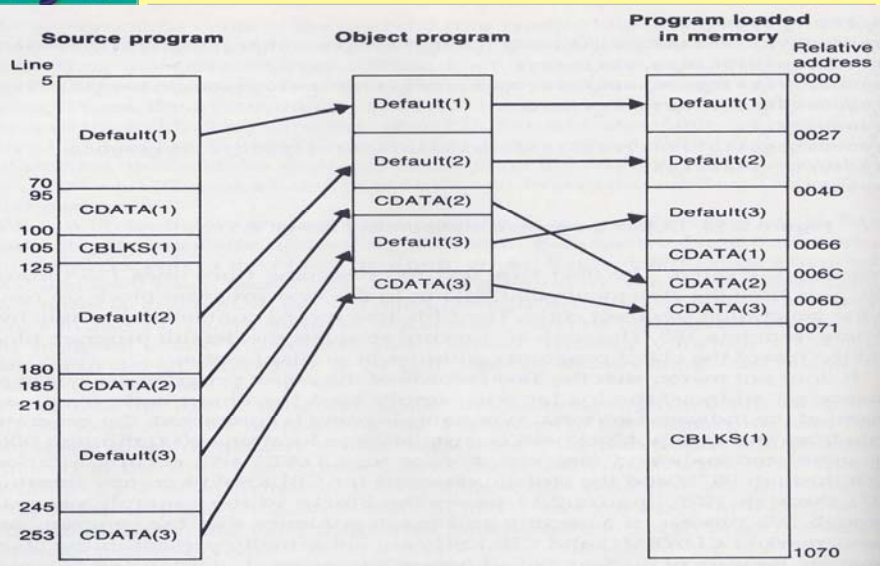


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

Control Section

- 讓 Assembler產生的object program 是一些相互獨立的部分(parts 或 code segments)，這些 code segments再由 loader 分開 handle 及 maintain
- 本書將這種code segments稱爲control sections
- 將 programs 分成多個 control section 可以分別地 load 及relocate，這樣的好處使得 programming 更有彈性
- 既然不同的 control system 有各自的 loading 及 relocating 時間，但這些不同的 sections 又在一個 program 中形成 logical 上相關，這表示實際上需要將它們 linking 在一起

Control Section

- 可是，每個Section 有自己的 loading & relocating time，表示每個 section 在執行時的 location 無法得知
- 那表示Assembler 對一個section 中有 reference 到別的 section 的symbol 就無法取得其value了，這種reference 稱爲 external reference (外部參用)
- 因此，Assembler 必需對每個 external reference 產生一些 information 以便 loader 執行必要的 linking

Control Section

- Fig 2.15 有三個 control sections， 主程式及每個 subroutine 各爲一個 section
- 這三個section可以分別組譯與載入
- 這表示main program 被組譯時，line 15 及 line 35 的 RDREC & WEREC 均是未知
- 對於未在該程式碼中定義卻要直接使用的變數(或 Symbol)，稱爲External Reference(外部參用)


```
Line        Source statement
  5   COPY      START     0                 COPY FILE FROM INPUT TO OUTPUT
  6             EXTDEF    BUFFER,BUFEND,LENGTH
  7             EXTREF    RDREC,WRREC
 10   FIRST     STL       RETADR            SAVE RETURN ADDRESS
 15   CLOOP    +JSUB      RDREC             READ INPUT RECORD
 20             LDA       LENGTH            TEST FOR EOF (LENGTH = 0)
 25             COMP      #0
 30             JEQ       ENDFIL            EXIT IF EOF FOUND
 35            +JSUB      WRREC             WRITE OUTPUT RECORD
 40             J         CLOOP             LOOP
 45   ENDFIL    LDA      =C'EOF'            INSERT END OF FILE MARKER
 50             STA       BUFFER
 55             LDA       #3                SET LENGTH = 3
 60             STA       LENGTH
 65            +JSUB      WRREC             WRITE EOF
 70             J         @RETADR           RETURN TO CALLER
 95   RETADR    RESW      1
100   LENGTH    RESW      1                 LENGTH OF RECORD
103             LTORG
105   BUFFER    RESB      4096              4096-BYTE BUFFER AREA
106   BUFEND    EQU       *
107   MAXLEN    EQU       BUFEND-BUFFER

109   RDREC     CSECT
110   .
115   .         SUBROUTINE TO READ RECORD INTO BUFFER
120   .
122             EXTREF    BUFFER,LENGTH,BUFEND
125             CLEAR     X                 CLEAR LOOP COUNTER
130             CLEAR     A                 CLEAR A TO ZERO
132             CLEAR     S                 CLEAR S TO ZERO
133             LDT       MAXLEN
135   RLOOP     TD        INPUT             TEST INPUT DEVICE
140             JEQ       RLOOP             LOOP UNTIL READY
145             RD        INPUT             READ CHARACTER INTO REGISTER A
150             COMPR     A,S               TEST FOR END OF RECORD (X'00')
155             JEQ       EXIT              EXIT LOOP IF EOR
160            +STCH      BUFFER,X          STORE CHARACTER IN BUFFER
165             TIXR      T                 LOOP UNLESS MAX LENGTH
170             JLT       RLOOP               HAS BEEN REACHED
175   EXIT     +STX       LENGTH            SAVE RECORD LENGTH
180             RSUB                        RETURN TO CALLER
185   INPUT     BYTE      X'F1'             CODE FOR INPUT DEVICE
190   MAXLEN    WORD      BUFEND-BUFFER

193   WRREC     CSECT
196   .
200   .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205   .
207             EXTREF    LENGTH,BUFFER
210             CLEAR     X                 CLEAR LOOP COUNTER
212            +LDT       LENGTH
215   WLOOP     TD       =X'05'             TEST OUTPUT DEVICE
220             JEQ       WLOOP             LOOP UNTIL READY
225            +LDCH      BUFFER,X          GET CHARACTER FROM BUFFER
230             WD       =X'05'             WRITE CHARACTER
235             TIXR      T                 LOOP UNTIL ALL CHARACTERS
240             JLT       WLOOP               HAVE BEEN WRITTEN
245             RSUB                        RETURN TO CALLER
255             END       FIRST
```

Figure 2.15 Illustration of control sections and program linking.

Control Section

- 爲了指明那些是 external reference 以便 loader 處理，SIC/XE的Assembler增加了兩個 assembler directives 分別爲 EXTDEF & EXTREF
  - **EXTDEF** (External Definition) 用來指明哪些symbols 在本 section (Control Section) 中被 definition，可供其他 section 使用 (reference)的
  - **EXTREF** (External Reference) 用來指明那些symbols 在本 section 會 reference，但是其他 section 中 definition
- Fig 2.16 就是 fig 2.15 的 Assembly-listing file 其中 line 15， line 160， line 190 均因symbol 未知，故 assembler 暫以0 代入

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
6			EXTDEF	BUFFER,BUFEND,LENGTH	
7			EXTREF	RDREC,WRREC	
10	0000	FIRST	STL	RETADR	172027
15	0003	CLOOP	+JSUB	RDREC	4B100000
20	0007		LDA	LENGTH	032023
25	000A		COMP	#0	290000
30	000D		JEQ	ENDFIL	332007
35	0010		+JSUB	WRREC	4B100000
40	0014		J	CLOOP	3F2FEC
45	0017	ENDFIL	LDA	=C'EOF'	032016
50	001A		STA	BUFFER	0F2016
55	001D		LDA	#3	010003
60	0020		STA	LENGTH	0F200A
65	0023		+JSUB	WRREC	4B100000
70	0027		J	@RETADR	3E2000
95	002A	RETADR	RESW	1	
100	002D	LENGTH	RESW	1	
103			LTORG		
	0030	*	=C'EOF'		454F46
105	0033	BUFFER	RESB	4096	
106	1033	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	
109	0000	RDREC	CSECT		
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
122			EXTREF	BUFFER,LENGTH,BUFEND	
125	0000		CLEAR	X	B410
130	0002		CLEAR	A	B400
132	0004		CLEAR	S	B440
133	0006		LDT	MAXLEN	77201F
135	0009	RLOOP	TD	INPUT	E3201B
140	000C		JEQ	RLOOP	332FFA
145	000F		RD	INPUT	DB2015
150	0012		COMPR	A,S	A004
155	0014		JEQ	EXIT	332009
160	0017		+STCH	BUFFER,X	57900000
165	001B		TIXR	T	B850
170	001D		JLT	RLOOP	3B2FE9
175	0020	EXIT	+STX	LENGTH	13100000
180	0024		RSUB		4F0000
185	0027	INPUT	BYTE	X'F1'	F1
190	0028	MAXLEN	WORD	BUFEND-BUFFER	000000
193	0000	WRREC	CSECT		
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
207			EXTREF	LENGTH,BUFFER	
210	0000		CLEAR	X	B410
212	0002		+LDT	LENGTH	77100000
215	0006	WLOOP	TD	=X'05'	E32012
220	0009		JEQ	WLOOP	332FFA
225	000C		+LDCH	BUFFER,X	53900000
230	0010		WD	=X'05'	DF2008
235	0013		TIXR	T	B850
240	0015		JLT	WLOOP	3B2FEE
245	0018		RSUB		4F0000
255			END	FIRST	
	001B	*	=X'05'		05

Figure 2.16 Program from Fig. 2.15 with object code.

Control Section

- Assembler爲了提供Control Section 這種feature，SIC/XE的Assembler要新增的工作如下
  - Remember 那個Symbol 是由那一個control section所define
  - 一個control section若沒有宣告EXTREF就欲refer to another control section的SYMBOL，則必須Set error flag
  - 若經EXTREF宣告，則Assembler 必須允許相同的Symbol用在不同的Control sections
    - 例如MAXLEN的定義，在fig 2.15的line 107及190不會造成問題

Control Section

- 爲了保留object code 的空間給external reference，也必須在object program 中包含information以便loader再爲external reference填入適當值
- SIC/XE需要增加２種record type如下：
  - 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 others external reference symbols

Control Section

- 並且修改Modification record 爲
  - Modification record (revised):
    - 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 the 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
- 而fig2.15因爲分成3個control sections，因此object Files也分成三個，如fig2.17所示

Object Program for Control Section

```
H^COPY  ^000000^001033
D^BUFFER^000033^BUFEND^001033^LENGTH^00002D
R^RDREC ^WRREC
T^000000^1D^172027^4B^100000^032023^290000^332007^4B^100000^3F2FEC^032016^0F2016
T^000001^0D^010003^0F200A^4B^100000^3E2000
T^000030^03^454F46
M^000004^05^+RDREC
M^000011^05^+WRREC
M^000024^05^+WRREC
E^000000

H^RDREC ^000000^00002B
R^BUFFER^LENGTH^BUFEND
T^000000^1D^B410^B400^B440^77201F^E3201B^332FFA^DB2015^A004^332009^57900000^B850
T^00001D^0E^3B2FE9^131000^004F0000^F1^000000
M^000018^05^+BUFFER
M^000021^05^+LENGTH
M^000028^06^+BUFEND
M^000028^06^-BUFFER
E

H^WRREC ^000000^00001C
R^LENGTH^BUFFER
T^000000^1C^B410^77100000^E32012^332FFA^53900000^DF2008^B850^3B2FEE^4F0000^05
M^000003^05^+LENGTH
M^00000D^05^+BUFFER
E
```

Figure 2.17 Object program corresponding to Fig. 2.15.

Control Section

- 其中main program section的部分，如 line15 +JSUB RDREC
- 該指令本身相對於begin address爲0004，但operand RERECC的address先暫設爲0
- 並在COPY這個Control section設定Modification record

  M00000405+RDREC
- 另外，像line 190的BUFFEND-BUFFER在Assembler的組譯時間已無法計算，故須留在 Loader時才計算，故設定Modification records二筆

  M00002806+BUFFEND

  M00002806-BUFFER

Control Section

- 對於relocation需要配合修改的部分至要留在下一章再討論
- 另外，對於expression 的處理，爲了relocating，必須限制expression中所用的symbol必須在同一section內
  - 如 BUFFEND-BUFFER
  - 但 RDREC-COPY則屬不同section，因爲分屬不同section，則其結果是無法預測的，因爲locating 的地方每次可能都不同

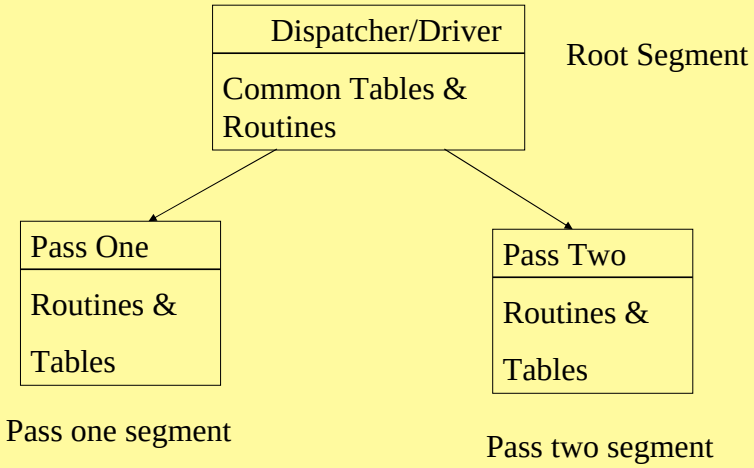
Assembler Design Options

- Two Passes Assembler with Overlay Structure
- One Pass Assembler
- Multi-passes Assembler

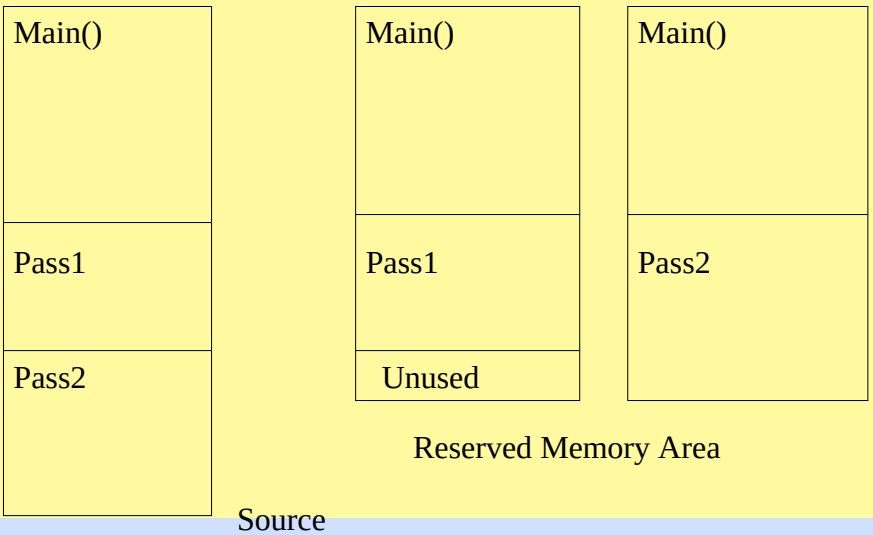
Two Passes Assembler with Overlay Structure

- 爲何要Overlay Structure
  - Internal subroutines & Tables of Pass1 are no longer needed after the pass 1 is completed
  - Routines & tables of pass 1 & pass 2 are never required at same time
  - With overlay structure can reduce memory requirement
- Overlay Structure
  - 整個Two-Pass Assembler structure有 Three segments，以Tree structure 表示
  - 其中root segment 含有一simple driver以便call 另外兩個segment，並包含了共用的tables及routines，如SYMTAB及searching/inserting SYMTAB routines

Overlay Structure of Two Passes Assembler



Overlay Structure of Two Passes Assembler



One-pass assembler

- Two-Passes Assembler 中用了一個pass來處理 symbol definition after symbol using
- 如果規定，symbol definition 必須在symbol被use之前，則可以去掉一個Pass的工作(如fig. 2-18的 Line 1-6)
- 但，這畢竟不是friendly design ，例如jump的方向總不能只能往回跳，副程式一定要先寫

Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9		.			
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48203D
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062
40	2021		J	CLOOP	302012
45	2024	ENDFIL	LDA	EOF	001000
50	2027		STA	BUFFER	0C100F
55	202A		LDA	THREE	001003
60	202D		STA	LENGTH	0C100C
65	2030		JSUB	WRREC	482062
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
121	2039	INPUT	BYTE	X'F1'	F1
122	203A	MAXLEN	WORD	4096	001000
124		.			
125	203D	RDREC	LDX	ZERO	041006
130	2040		LDA	ZERO	001006
135	2043	RLOOP	TD	INPUT	E02039
140	2046		JEQ	RLOOP	302043
145	2049		RD	INPUT	D82039
150	204C		COMP	ZERO	281006
155	204F		JEQ	EXIT	30205B
160	2052		STCH	BUFFER,X	54900F
165	2055		TIX	MAXLEN	2C203A
170	2058		JLT	RLOOP	382043
175	205B	EXIT	STX	LENGTH	10100C
180	205E		RSUB		4C0000
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
206	2061	OUTPUT	BYTE	X'05'	05
207		.			
210	2062	WRREC	LDX	ZERO	041006
215	2065	WLOOP	TD	OUTPUT	E02061
220	2068		JEQ	WLOOP	302065
225	206B		LDCH	BUFFER,X	50900F
230	206E		WD	OUTPUT	DC2061
235	2071		TIX	LENGTH	2C100C
240	2074		JLT	WLOOP	382065
245	2077		RSUB		4C0000
255			END	FIRST	

Figure 2.18 Sample program for a one-pass assembler.

One-pass assembler

- 以下將介紹兩種One-Pass assembler types，並保有Two-passes assembler的提供之functions
- 這兩種One-Pass assembler 的一項共同方法是改變SYMTAB的結構，允許symbol可以在未definition 前(卻先要use時)就先紀錄到SYMTAB，方法如圖例所示
- 第一種：**Load-and-go assembler**：
  - 直接將object code產生並置於memory中，並隨即執行
    - 優：簡單
    - 缺：限於absolute loader，每次執行均需assemble
    - 適用環境：教學練習，absolute addressing
  - 如fig 2-19a-b

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--00100C	28100630	-----48--
2020	--3C2012			
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	* → 2013 0
THREE	1003
ZERO	1006
WRREC	* → 201F 0
EOF	1000
ENDFIL	* → 201C 0
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448--
2020	--3C2012	0010000C	100F0010	030C100C
2030	48----08	10094C00	00F10010	00041006
2040	001006E0	20393020	43D82039	28100630
2050	----5490	0F		
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	* → 201F → 2031 ∅
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	* → 2050 ∅
RLOOP	2043

Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

One-pass assembler

- 第二種：會產生一個object file，以便稍後執行
  - object file 較two-passes assembler所產生的大，其他功效與Two-pass assembler相同
  - 方式與第一種類似，SYMTAB的結構也相同。但將產生的object code寫到Text record
  - 但因爲是寫到file去，所以不能像第一種方法一樣，隨時回頭去改記憶體內容
  - 因此，遇到forward defined Symbol時，operand的部分先以0000代入，並記錄此位置及Symbol name
  - 當Symbol被defined之後，就Symbol被use的位置，長度及Symbol的值來產生新的Text record，因此Text record數會變多。因此, Fig2.18以第二種方法產生的之object file 如下

```
H^COPY^^00100^000107A
T^00100^09^454F46^000003^000000
T^00200^F^15^141009^48000^000100^C2810^06^300000^48000^3C2012
T^00201^C^02^2024
T^00202^19^0010000^C100F^0010030^C100C^4800000^810094^C0000F^1001000
T^00201^3^02^203D
T^00203^D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^00205^0^02^205B
T^00205B^07^10100C^4C000005
T^00201F^02^2062
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

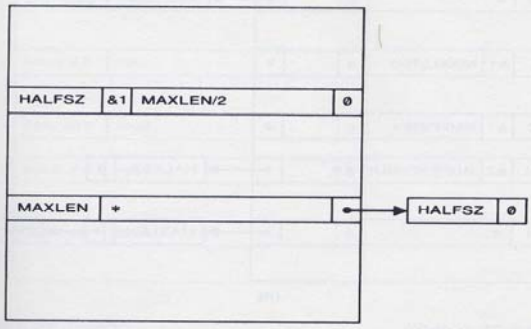
Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

Multi-Passes Assembler

- 使用Multi-Passes Assembler可以解決forward-reference problem (如圖2-21a)
- 但是隨著forward的級數就需要相對的Passes數，很難給于一個定額的passes個數
- 所以在此介紹一種用Two-pass assembler處理forward-reference problems的設計
- 方法是對SYMTAB的結構加以設計(如圖2-21b)，在Two Passes Assembler的Pass One允許Used Symbol without Definition可以先存到SYMTAB中，並紀錄對應的Reference的位置

```
1   HALFSZ    EQU    MAXLEN/2
2   MAXLEN    EQU    BUFEND-BUFFER
3   PREVBT    EQU    BUFFER-1
                     .
                     .
                     .
4   BUFFER    RESB   4096
5   BUFEND    EQU    *
```

(a)



(b)

Figure 2.21 Example of multi-pass assembler operation.

Multi-Passes Assembler

- 如圖2.21b-f的Symbol table structure
- 其中,&?是表示define此Symbol中的Expression裡， 尚有幾個Symbol仍是undefined
- Value欄位則先填入已知的Expression
- Link則是連結到受此Symbol name影響而undefined的Symbol，以便未來此Symbol被define以後，可以沿著此Link去修訂一連串未被defined的symbol
- 過程如圖2.21b到2.21f

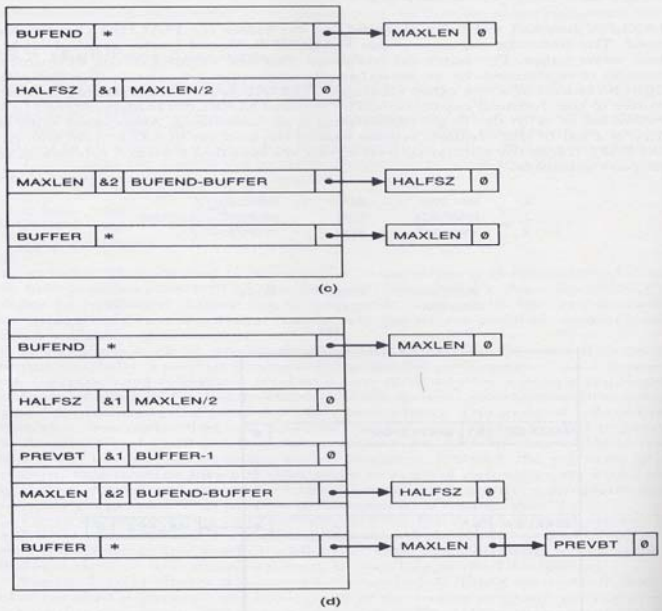


Figure 2.21 (*cont'd*)

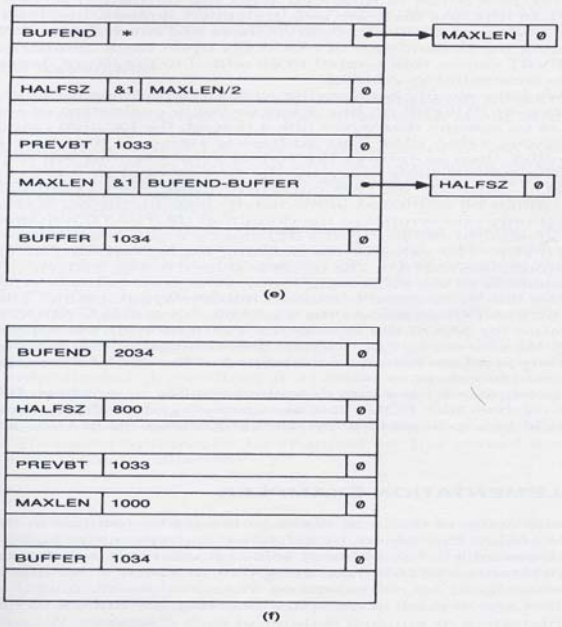


Figure 2.21 (*con'd*)

Chap. 3 Loader & Linker

- Basic Loader Functions
- Machine-Dependent Loader Features
- Machine-Independent Loader Features
- Loader Design Options

Loader & Linker

- 本章介紹重點是3個Processes
  - Loading
  - Relocation
  - Linking
- Loader :基本上是執行Loading function但大多數的Loader包含有Relocation及Linking的functions
- Linker :單純執行linking的工作，此時的 Loader則執行Loading & relocation
- Loader最基本的功能: Loading一個Object Program到memory，並開始執行該程式

Loader & Linker

- 本章介紹的Loader & Linker主要有下列幾種
  - Absolute loader: only with Loading function
  - Relocating loader: with Loading & Relocation function
  - Linking loader: with Loading, Relation, and Linking
  - Linkage editor: it is a Linker, only has Linking function
  - Dynamic linking loader: it is a Linking Loader that run the loading, relation, and linking functions in execution time of loaded program
- Basic Functions of Loader
  - Bringing an object program into memory
  - Starting its execution

Design an absolute loader

- Absolute loader可以用one pass完成所有功能
- 請先參閱Fig. 3-1a與Fig. 3-1b
- Figure3.1(b)是表示Figure3.1(a)的Object program放到memory的結果
- 其中 xxx，屬於Reserved Area，Object Program未提供Object code，故loader不改變原memory的內容
- Figure3.2則是一個Absolute Loader的Algorithm
  - 其中SIC假設Object file中每個十六進位的Digit (理論上只有4個Bits)都是一個Character來呈現
  - 而實際上在Object file的一對Character(佔2 bytes)，放到memory時要轉成一個Byte

```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^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^002073^07^382064^4C0000^05
E^001000
```

(a) Object program

Memory address	Contents			
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
0FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
1000	14103348	20390010	36281030	30101548
1010	20613C10	0300102A	0C103900	102D0C10
1020	36482061	0810334C	0000454F	46000003
1030	000000xx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
2030	xxxxxxxx	xxxxxxxx	xx041030	001030E0
2040	205D3020	3FD8205D	28103030	20575490
2050	392C205E	38203F10	10364C00	00F10010
2060	00041030	E0207930	20645090	39DC2079
2070	2C103638	20644C00	0005xxxx	xxxxxxxx
2080	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮

← COPY

(b) Program loaded in memory

Figure 3.1 Loading of an absolute program.

Algorithm of Absolute Loader

```
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
```

Figure 3.2 Algorithm for an absolute loader.

Design an absolute loader

- 因此，loader必須要注意轉換工作(Algorithm的if object code is in character form…)
- 更一般的做法是Assembler的output就以binary表示
- 但，這種做法也須注意另一件事；Object file中的byte是否和某些device的control code相同，若是，可能會在file的r/w時，造成系統其他行爲，而object file本身卻不完整

A Simple Bootstrap Loader

- 一般電腦上有一支更簡單，具特定用途的absolute Loader
  - 稱爲Bootstrap Loader
  - Bootstrap Loader是一部Computer起始執行的第一個program
  - 其的工作負責load O.S. 到memory & run
  - 或load某一特定程式，當computer是一特定用途之電腦
- Fig3.3是SIC/XE Bootstrap Loader之Source Code
  - 其中Bootstrap自己在address 0 開始
  - 而operating system自80位址開始，O.S.的object file則是從device”F1” read in

```
BOOT     START    0        BOOTSTRAP LOADER FOR SIC/XE
.
. THIS BOOTSTRAP READS OBJECT CODE FROM DEVICE F1 AND ENTERS IT
. INTO MEMORY STARTING AT ADDRESS 80 (HEXADECIMAL). AFTER ALL OF
. THE CODE FROM DEVF1 HAS BEEN SEEN ENTERED INTO MEMORY, THE
. BOOTSTRAP EXECUTES A JUMP TO ADDRESS 80 TO BEGIN EXECUTION OF
. THE PROGRAM JUST LOADED.  REGISTER X CONTAINS THE NEXT ADDRESS
. TO BE LOADED.
.
         CLEAR    A        CLEAR REGISTER A TO ZERO
         LDX      #128     INITIALIZE REGISTER X TO HEX 80
LOOP     JSUB     GETC     READ HEX DIGIT FROM PROGRAM BEING LOADED
         RMO      A,S      SAVE IN REGISTER S
         SHIFTL   S,4      MOVE TO HIGH-ORDER 4 BITS OF BYTE
         JSUB     GETC     GET NEXT HEX DIGIT
         ADDR     S,A      COMBINE DIGITS TO FORM ONE BYTE
         STCH     0,X      STORE AT ADDRESS IN REGISTER X
         TIXR     X,X      ADD 1 TO MEMORY ADDRESS BEING LOADED
         J        LOOP     LOOP UNTIL END OF INPUT IS REACHED
.
. SUBROUTINE TO READ ONE CHARACTER FROM INPUT DEVICE AND
. CONVERT IT FROM ASCII CODE TO HEXADECIMAL DIGIT VALUE. THE
. CONVERTED DIGIT VALUE IS RETURNED IN REGISTER A. WHEN AN
. END-OF-FILE IS READ, CONTROL IS TRANSFERRED TO THE STARTING
. ADDRESS (HEX 80).
.
GETC     TD       INPUT    TEST INPUT DEVICE
         JEQ      GETC     LOOP UNTIL READY
         RD       INPUT    READ CHARACTER
         COMP     #4       IF CHARACTER IS HEX 04 (END OF FILE),
         JEQ      80           JUMP TO START OF PROGRAM JUST LOADED
         COMP     #48      COMPARE TO HEX 30 (CHARACTER '0')
         JLT      GETC     SKIP CHARACTERS LESS THAN '0'
         SUB      #48      SUBTRACT HEX 30 FROM ASCII CODE
         COMP     #10      IF RESULT IS LESS THAN 10, CONVERSION IS
         JLT      RETURN       COMPLETE. OTHERWISE, SUBTRACT 7 MORE
         SUB      #7           (FOR HEX DIGITS 'A' THROUGH 'F')
RETURN   RSUB                 RETURN TO CALLER
INPUT    BYTE     X'F1'    CODE FOR INPUT DEVICE
         END      LOOP
```

Figure 3.3 Bootstrap loader for SIC/XE.

Machine Dependent Loader Features

- Absolute loader 的缺點
  - 在object file放到記憶體之前，programmer必須指定好起始位址
  - Absolute program難以有效地使用subroutine libraries
- 本 section則探討一個more complex loader的design & implementation，此 loader provides for program relocation and linking

Relocation

- Relocating loader/Relative loader
- Loader 允許program relocation者，稱之
- 兩種方法來完成program relocation
- 均需憑藉Assembler 在output object file時留下必要資訊的幫助

Relocation

- 利用Modification record
  - 在modification record中每一個值在relocation時，Load均須依據真正的起始位址改變其實際值，如Figure3.4之program及Figure3.5之object file
  - 但是，若是program是一個SIC program或用了許多direct addressing mode時，那modification record將會相當多，這時，必須用第二種方法解決。
- 利用Modification Bits
  - 每個Text record 最多可容納10個word，每個word用一個relocation bit來標明是否要調整(relocation)
  - 因此，每個record加上3個nibble (half-byte)來指示下列word中，哪些要relocation
  - 如Figure3.6是一個standard SIC program，若要提供relocation function則assembler output必須如Figure3.7所示，loader才能完成relocating function。

```
Line   Loc    Source statement                    Object code

 5     0000   COPY     START   0
10     0000   FIRST    STL     RETADR            17202D
12     0003            LDB     #LENGTH           69202D
13                     BASE    LENGTH
15     0006   CLOOP   +JSUB    RDREC             4B101036
20     000A            LDA     LENGTH            032026
25     000D            COMP    #0                290000
30     0010            JEQ     ENDFIL            332007
35     0013           +JSUB    WRREC             4B10105D
40     0017            J       CLOOP             3F2FEC
45     001A   ENDFIL   LDA     EOF               032010
50     001D            STA     BUFFER            0F2016
55     0020            LDA     #3                010003
60     0023            STA     LENGTH            0F200D
65     0026           +JSUB    WRREC             4B10105D
70     002A            J       @RETADR           3E2003
80     002D   EOF      BYTE    C'EOF'            454F46
95     0030   RETADR   RESW    1
100    0033   LENGTH   RESW    1
105    0036   BUFFER   RESB    4096
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
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
```

Figure 3.4 Example of a SIC/XE program (from Fig. 2.6).

利用Modification Record

```
HCOPY  000000001077
T0000001D17202D69202D4B101036032026290000332007 4B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A004332008 57C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705+COPY
M00001405+COPY
M00002705+COPY
E000000
```

Figure 3.5 Object program with relocation by Modification records.

Line	Loc	Source statement			Object code
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	
110		.			
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		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
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	

Figure 3.6 Relocatable program for a standard SIC machine.

利用Modification Bits

```
H^COPY  ^00000^00107A
T^000000^1E^FFC^140033^481039^000036^280030^300015^481061^3C0003^00002A^0C0039^00002D
T^00001E^15^E000C^003648^10610^800334^C0000^454F46^000003^000000
T^001039^1E^FFC^040030^000030^E0105D^30103F^D8105D^280030^301057^548039^2C105E^38103F
T^001057^0A^8001^000364^C0000^F1^001000
T^00106119^FE^0040030^E01079^301064^508039^DC1079^2C0036^381064^4C0000^05
E^000000
```

Figure 3.7 Object program with relocation by bit mask.

Program Linking

- Programmer自然地將相關的control sections合併在一個Source Program
- 但就loader觀點而言，只要知道control sections要被linked，relocated及loaded
- 而不會(也不需要)知道那些control section在同一時間內要被組合在一起
- 以Figure3.8的3個control sections爲例
  - 3個control section功能相似
  - 指令不同
  - 每個section各擁有一個list及同性質的REF1~REF3，REF4~REF8
- 在此要強調的是relocation及linking處理的關係
- Figure3.8之object program如Figure3.9所示


```
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
```

Figure 3.8 Sample programs illustrating linking and relocation.

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		

Figure 3.8 (*cont'd*)

Object Programs of Fig. 3.8

```
H^PROGA ^000000^000063
D^LISTA ^000040^ENDA  ^000054
R^LISTB ^ENDB  ^LISTC ^ENDC
:
T^000020^0A^03201D^77100004^050014
:
T^000054^0F^000014^FFFFF6^00003F^000014^FFFFC0
M^000024^05^+LISTB
M^000054^06^+LISTC
M^000057^06^+ENDC
M^000057^06^-LISTC
M^00005A^06^+ENDC
M^00005A^06^-LISTC
M^00005A^06^+PROGA
M^00005D^06^-ENDB
M^00005D^06^+LISTB
M^000060^06^+LISTB
M^000060^06^-PROGA
E^000020
```

Figure 3.9 Object programs corresponding to Fig. 3.8.

```
H^PROGB ^000000^00007F
D^LISTB ^000060^ENDB  ^000070
R^LISTA ^ENDA  ^LISTC ^ENDC
:
T^000036^0B^03100000^772027^05100000
:
T^000070^0F^000000^FFFFF6^FFFFFF^FFFFF0^000060
M^000037^05^+LISTA
M^00003E^05^+ENDA
M^00003E^05^-LISTA
M^000070^06^+ENDA
M^000070^06^-LISTA
M^000070^06^+LISTC
M^000073^06^+ENDC
M^000073^06^-LISTC
M^000076^06^+ENDC
M^000076^06^-LISTC
M^000076^06^+LISTA
M^000079^06^+ENDA
M^000079^06^-LISTA
M^00007C^06^+PROGB
M^00007C^06^-LISTA
E

H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^LISTA ^ENDA  ^LISTB ^ENDB
:
T^000018^0C^03100000^771000^05100000
:
T^000042^0F^000030^000008^000011^000000^000000
M^000019^05^+LISTA
M^00001D^05^+LISTB
M^000021^05^+ENDA
M^000021^05^-LISTA
M^000042^06^+ENDA
M^000042^06^-LISTA
M^000042^06^+PROGC
M^000048^06^+LISTA
M^00004B^06^+ENDA
M^00004B^06^-LISTA
M^00004B^06^-ENDB
M^00004B^06^+LISTB
M^00004E^06^+LISTB
M^00004E^06^-LISTA
E
```

Figure 3.9 (*cont'd*)

Program Linking

- Figure3.10(a)則表示Figure3.9的object program被放置到memory location 4000的地方
- 以及REF1~REF8相關的contents被計算出來後的結果
- Figure3.10(b)則是說明PROGA中REF4被relocation及link的執行過程

Relocation and Link of Fig 3.9

Memory address	Contents				
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	
3FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4000	--------	--------	--------	--------	
4010	--------	--------	--------	--------	
4020	03201D77	1040C705	0014----	--------	← PROGA
4030	--------	--------	--------	--------	
4040	--------	--------	--------	--------	
4050	--------	00412600	00080040	51000004	
4060	000083--	--------	--------	--------	
4070	--------	--------	--------	--------	
4080	--------	--------	--------	--------	
4090	--------	--------	--031040	40772027	← PROGB
40A0	05100014	--------	--------	--------	
40B0	--------	--------	--------	--------	
40C0	--------	--------	--------	--------	
40D0	------00	41260000	08004051	00000400	
40E0	0083----	--------	--------	--------	
40F0	--------	--------	----0310	40407710	
4100	40C70510	0014----	--------	--------	← PROGC
4110	--------	--------	--------	--------	
4120	--------	00412600	00080040	51000004	
4130	000083xx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4140	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	

Figure 3.10(a) Programs from Fig. 3.8 after linking and loading.

Memory Map of Fig 3.9

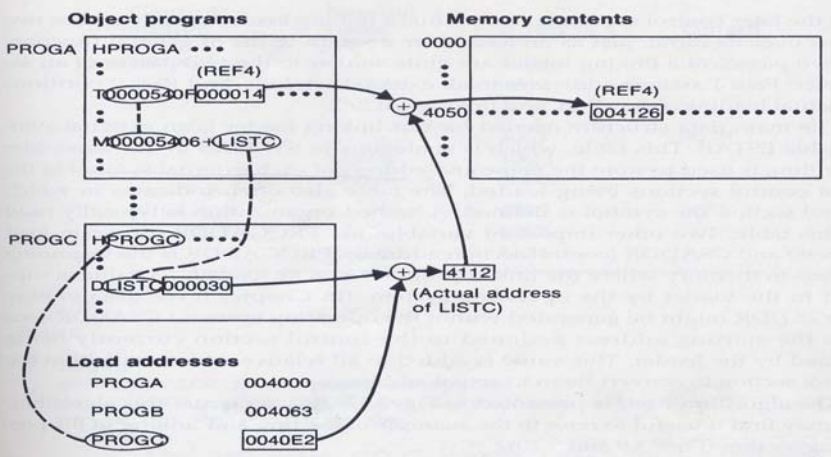


Figure 3.10(b) Relocation and linking operations performed on REF4 from PROGA.

Tables & Logic for Linking Loader

- Linking loader 分成2個Passes:
  - Pass1 : Assigns address to all external symbol
  - Pass2 : Performs the actual loading , relocation and linking
- 主要的Data Structure 有：
  - ESTAB : External Symbol TABle包含：
    - Symbol name
    - Address in its control section
    - Control section defined the symbol
    - 可以使用Hash方法組織此table
  - PROGADDR : 紀錄program load address
  - CSADDR : 紀錄目前處理之control section 所assign的起始位址
  - CSLTH : 紀錄目前處理之control section 的長度

Tables & Logic for Linking Loader

- Linking loader 的演算法如下：
  - Pass1的工作：
    - Allocate Memory
    - 建立ESTAB
    - Allocate 每個control section 在memory 的起始位址
    - 如Fig 3.11(a)
  - Pass2 的工作：
    - 依ESTAB及object file 內容將object code 放到memory 的適合位置
    - 如Fig 3.11(b)
  - Pass1 所concern 的是
    - Header 及 Define Record
    - 並且把control section name 也建入ESTAB內，並存入相對應之起始位址
  - Pass2所concern 的是
    - Text record 及 Modification record
    - 是實際執行object file 的loading ,relocation 及linking 的工作

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	

Pass 1:

```
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
                           (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)
```

Figure 3.11(a) Algorithm for Pass 1 of a linking loader.

Pass 2:

```
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 EXECADDR (to start execution of loaded program)
end (Pass 2)
```

Figure 3.11(b) Algorithm for Pass 2 of a linking loader.

Tables & Logic for Linking Loader

- 有許多Loader 會再由ESTAB衍生出一個Load Map
  - 專門記錄每個Control section 的起始位址及每個Control section 中define 的External Definition symbol 之位址
  - 如下表，即爲Fig3.9及3.10的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	

Tables & Logic for Linking Loader

- 若要再提高此演算法效能
  - 可以將object file 中 External Reference symbol 改用 Reference number 來代替
  - 如此一來，在loading 一個control section 時
  - 相同的symbol 就可以在ESTAB上避免multiple search
  - 如Fig3.8的object file 即可改用reference number 改成Fig. 3.12

External Reference Number for Fig. 3.8

```
HPROGA 000000000063
DLISTA 000040ENDA  000054
R02LISTB 03ENDB  04LISTC 05ENDC
:
T000020 0A 03201D 771000 04 050014
:
T000054 0F 000014 FFFFF6 00003F 000014 FFFFC0
M000024 05 +02
M000054 06 +04
M000057 06 +05
M000057 06 -04
M00005A 06 +05
M00005A 06 -04
M00005A 06 +01
M00005D 06 -03
M00005D 06 +02
M000060 06 +02
M000060 06 -01
E000020
```

Figure 3.12 Object programs corresponding to Fig. 3.8 using reference numbers for code modification. (Reference numbers are underlined for easier reading.)


```
HPROGB 000000000007F
DLISTB 000060ENDB  000070
R02LISTA 03ENDA  04LISTC 05ENDC
:
T000036 0B 031000007720 27 05100000
:
T000070 0E 000000 FFFFF6 FFFFFF FFFFF0 000060
M000037 05 +02
M00003E 05 +03
M00003E 05 -02
M000070 06 +03
M000070 06 -02
M000070 06 +04
M000073 06 +05
M000073 06 -04
M000076 06 +05
M000076 06 -04
M000076 06 +02
M000079 06 +03
M000079 06 -02
M00007C 06 +01
M00007C 06 -02
E

HPROGC 000000000051
DLISTC 000030ENDC  000042
R02LISTA 03ENDA  04LISTB 05ENDB
:
T000018 0C 031000007710 00 04 05100000
:
T000042 0E 000030 000008 000011 000000 000000
M000019 05 +02
M00001D 05 +04
M000021 05 +03
M000021 05 -02
M000042 06 +03
M000042 06 -02
M000042 06 +01
M000048 06 +02
M00004B 06 +03
M00004B 06 -02
M00004B 06 -05
M00004B 06 +04
M00004E 06 +04
M00004E 06 -02
E
```

Figure 3.12 (*cont'd*)

Machine-Independent Loader Features

- Automatic Library Search
  - 配合Linking Loader的鏈結功能，如果Programmer在程式中用到System Library所提供的函式與變數
  - 原始程式不必宣告System Library所定義的Symbols
  - 此時，Loader雖然無法自原程式找到這些Symbols的定義與位址，但是尚不會立即判定有Undefined Symbol
  - 而是開啓System Library，搜尋尚未被定義的Symbols，並Assign屬於System Library所指定的Symbols與對應的值
  - 只有在原程式與System Library都找不到Symbol Definition時，才判定Undefinied symbol的Error
  - 通常Loader搜尋External Symbol的定義時是優先搜尋原程式各Control Sections，找不到才搜尋System Library
  - 這樣的搜尋順序才能允許Programmer definition function override System library的定義

Machine-Independent Loader Features

- Loader Options
  - 本功能允許Loader再執行載入程式時，接受使用者的要求，經由參數指定載入的內容與變化
  - 範例如下：

```
INCLUDE  READ(UTLIB)
INCLUDE  WRITE(UTLIB)
DELETE   RDREC, WRREC
CHANGE  RDREC, READ
CHANGE  WRREC, WRITE
LIBRARY  MYLIB
NOCALL  STDDEV, PLOT, CORREL
```

Loader Design Options

- Linkage Editor
  - It is a Linker
  - It only has linking function
  - The output of Linkage Editor is a Linked Program, often called ***load module*** or ***executable image***
  - 與Linking Loader的比較

TAMKANG UNIVERSITY SOFTWARE ENGINEERING GROUP
淡江軟體工程實驗室 http://www.tkse.tku.edu.tw

Loader Design Options

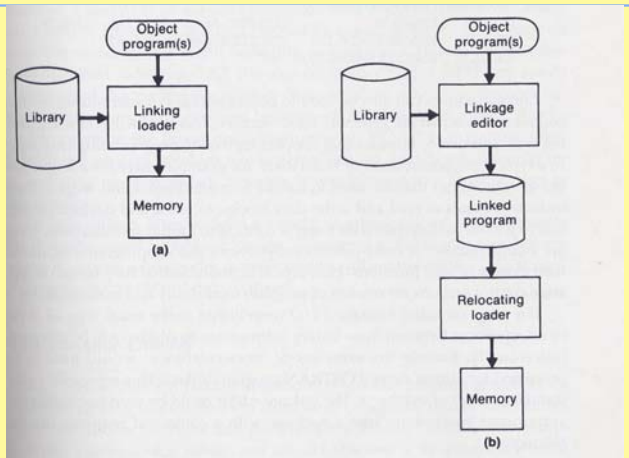


Figure 3.13 Processing of an object program using (a) linking loader and (b) linkage editor.

Loader Design Options

- Dynamic Linking
  - 又稱爲Dynamic loading、Load on call
  - 提供Control Section (或稱爲Program Module)在被呼叫時，才執行Linking、Relocation and Loading工作
  - 優點是需要時才被載入，不佔記憶體空間
  - 缺點是執行效率慢
  - 執行方式如圖

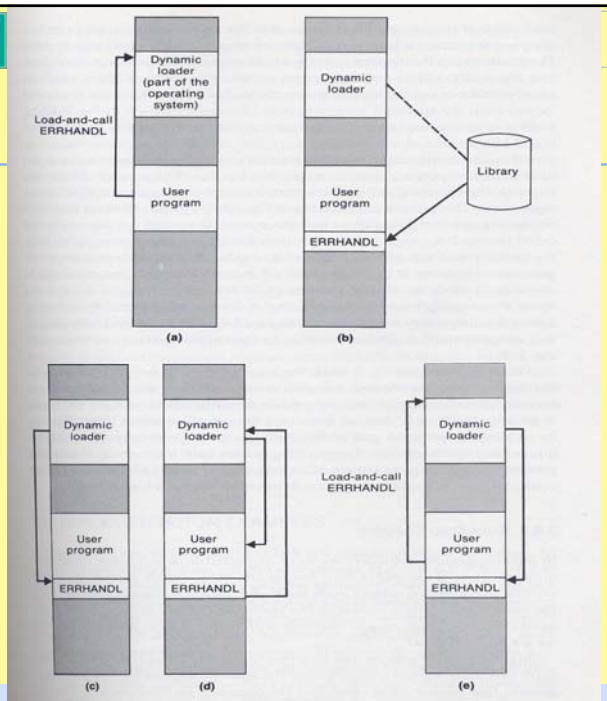


Figure 3.14 Loading and calling of a subroutine using dynamic linking.

Loader Design Options

- Dynamic Linking
  - 比較幾種Loader的Relocation、Linking的時間點
    - Absolute Loader 無須Relocate與Link，在Assemble Time已決定一切
    - Relocating Loader 無須Link，在Loading Time決定Relocation
    - Linking Loader 在Loading Time決定Relocation 與 Linking
    - Linkage Editor 在Assemble Time之後，Loading Time之前決定Linking與部分的Relocation
    - Dynamic Linking 在Execution Time才決定Relocation、Linking以及Loading

Macro Processors