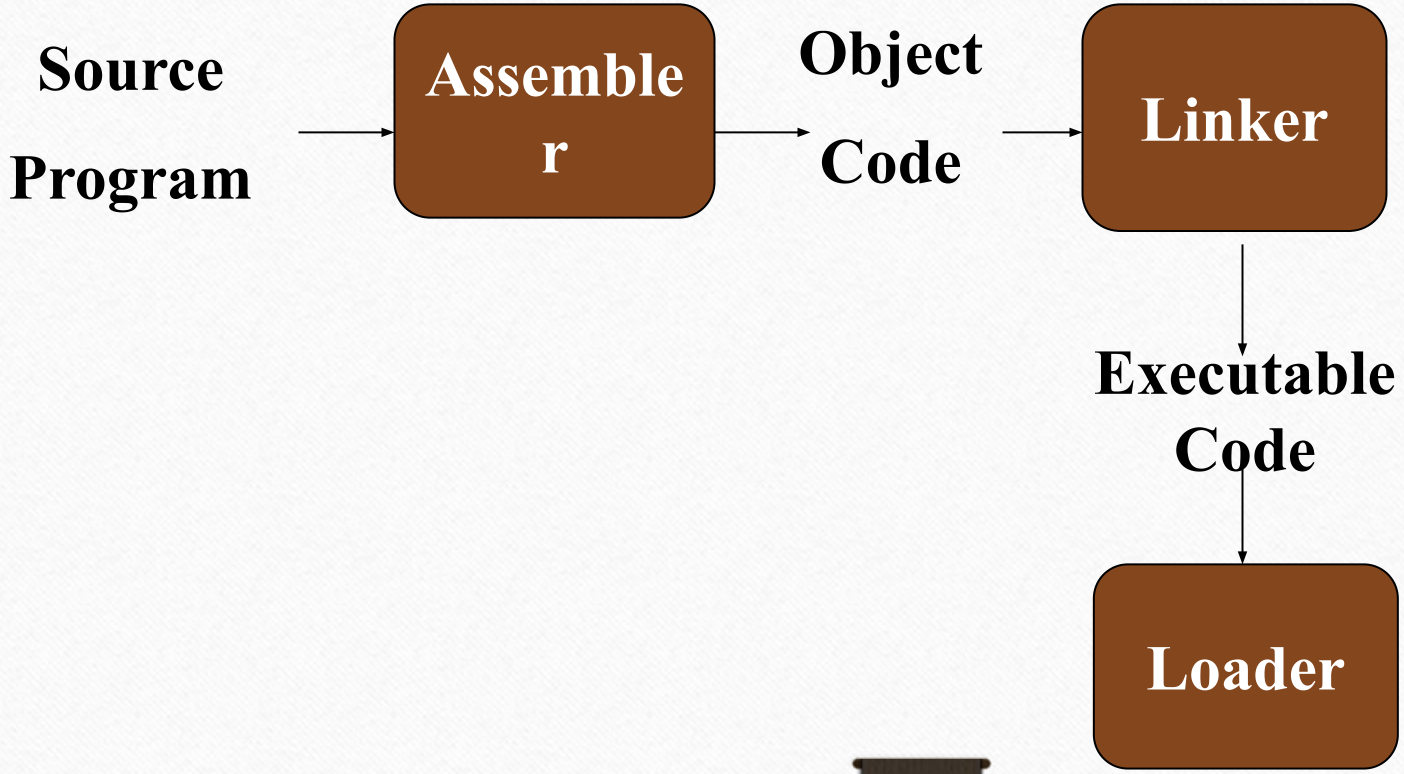
Chapter 02

Assembler

Role of Assembler



Introduction to Assemblers

- Fundamental functions
  - translating mnemonic operation codes to their machine language equivalents
  - assigning machine addresses to symbolic labels used by the programmer

- Machine dependency
  - different machine instruction formats and codes

Example Program (Fig. 2.1)

- Purpose
  - The main routine reads records from input device (code F1)
  - copies them to output device (code 05)
  - The main routine calls subroutine RDREC to read a record into the buffer and subroutine WRREC to write a record from the buffer to the o/p device

- The end-of-record marked with NULL char(00)
- If (size(rec) > len(buff)) only 1st 4096 bytes are copied
- at the end of the file, writes EOF on the output device, then RSUB to the operating system
- Lines beginning with dot (.) represents Comment
- Line number are just for your reference
- program

```
Line            Source statement

        COPY      START     1000             COPY FILE FROM INPUT TO OUTPUT
        FIRST     STL       RETADR           SAVE RETURN ADDRESS
        CLOOP     JSUB      RDREC            READ INPUT RECORD
                  LDA       LENGTH           TEST FOR EOF (LENGTH = 0)
                  COMP      ZERO
                  JEQ       ENDFIL           EXIT IF EOF FOUND
                  JSUB      WRREC            WRITE OUTPUT RECORD
                  J         CLOOP            LOOP
        ENDFIL    LDA       EOF              INSERT END OF FILE MARKER
                  STA       BUFFER
                  LDA       THREE            SET LENGTH = 3
                  STA       LENGTH
                  JSUB      WRREC            WRITE EOF
                  LDL       RETADR           GET RETURN ADDRESS
                  RSUB                       RETURN TO CALLER
        EOF       BYTE      C'EOF'
        THREE     WORD      3
        ZERO      WORD      0
        RETADR    RESW      1
        LENGTH    RESW      1                LENGTH OF RECORD
        BUFFER    RESB      4096             4096-BYTE BUFFER AREA
        .
        .         SUBROUTINE TO READ RECORD INTO BUFFER
        .
        RDREC     LDX       ZERO             CLEAR LOOP COUNTER
                  LDA       ZERO             CLEAR A TO ZERO
        RLOOP     TD        INPUT            TEST INPUT DEVICE
                  JEQ       RLOOP            LOOP UNTIL READY
                  RD        INPUT            READ CHARACTER INTO REGISTER A
                  COMP      ZERO             TEST FOR END OF RECORD (X'00')
                  JEQ       EXIT             EXIT LOOP IF EOR
                  STCH      BUFFER,X         STORE CHARACTER IN BUFFER
                  TIX       MAXLEN           LOOP UNLESS MAX LENGTH
                  JLT       RLOOP              HAS BEEN REACHED
        EXIT      STX       LENGTH           SAVE RECORD LENGTH
                  RSUB                       RETURN TO CALLER
        INPUT     BYTE      X'F1'            CODE FOR INPUT DEVICE
        MAXLEN    WORD      4096
        .
        .         SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
        WRREC     LDX       ZERO             CLEAR LOOP COUNTER
        WLOOP     TD        OUTPUT           TEST OUTPUT DEVICE
                  JEQ       WLOOP            LOOP UNTIL READY
                  LDCH      BUFFER,X         GET CHARACTER FROM BUFFER
                  WD        OUTPUT           WRITE CHARACTER
                  TIX       LENGTH           LOOP UNTIL ALL CHARACTERS
                  JLT       WLOOP              HAVE BEEN WRITTEN
                  RSUB                       RETURN TO CALLER
        OUTPUT    BYTE      X'05'            CODE FOR OUTPUT DEVICE
                  END       FIRST
```

Figure 2.1 Example of a SIC assembler language program.

Figure 2.1 (Pseudo code)

```
Program copy {
        save return address;
  cloop:   call subroutine RDREC to read one record;
            if length(record)=0 {
          call subroutine WRREC to write EOF;
        }  else {
                                    call subroutine WRREC to write one record;
               goto cloop;
            }
            load return address
            return to caller
  }
```

An Example (Figure 2.1)

EOR:
character
x‘00’

```
Subroutine RDREC {
            clear A, X register to 0;
  rloop:    read character from input device to A register
            if not EOR {
              store character into buffer[X];
              X++;
              if X < maximum length
                        goto rloop;
          }
            store X to length(record);
            return
 }
```

An Example (Figure 2.1, Cont.)

```
Subroutine WDREC {
            clear X register to 0;
  wloop:    get character from buffer[X]
            write character from X to output device
            X++;
            if X < length(record)
                 goto wloop;
            return
  }
```

Assembler Directives

- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START
  - END
  - BYTE
  - WORD
  - RESB
  - RESW

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		.			

```
                 .
                 .        SUBROUTINE TO READ RECORD INTO BUFFER
                 .
2039    RDREC     LDX      ZERO                    041030
203C              LDA      ZERO                    001030
203F    RLOOP     TD       INPUT                   E0205D
2042              JEQ      RLOOP                   30203F
2045              RD       INPUT                   D8205D
2048              COMP     ZERO                    281030
204B              JEQ      EXIT                    302057
204E              STCH     BUFFER,X                549039
2051              TIX      MAXLEN                  2C205E
2054              JLT      RLOOP                   38203F
2057    EXIT      STX      LENGTH                  101036
205A              RSUB                             4C0000
205D    INPUT     BYTE     X'F1'                   F1
205E    MAXLEN    WORD     4096                    001000
```

```
                .
                .       SUBROUTINE TO WRITE RECORD FROM BUFFER
                .
        2061    WRREC   LDX     ZERO            041030
        2064    WLOOP   TD      OUTPUT          E02079
        2067            JEQ     WLOOP           302064
        206A            LDCH    BUFFER,X        509039
        206D            WD      OUTPUT          DC2079
        2070            TIX     LENGTH          2C1036
        2073            JLT     WLOOP           382064
        2076            RSUB                    4C0000
        2079    OUTPUT  BYTE    X'05'           05
                        END     FIRST
```

Assembler's functions

- Convert mnemonic operation codes to their machine language equivalents

  Ex Translate STL to 14 (line no 10)

- Convert symbolic operands to their equivalent machine addresses

  Ex: RETADR TO 1033 (line no 10)

- Build the machine instructions in the proper format
- Convert the data constants to internal machine representations

  Translate EOF to 454F46 (line no 80)

- Write the object program and the assembly listing

Difficulties: Forward Reference

- Forward reference: reference to a label that is defined later in the program.

Loc	Label	Operator	Operand
1000	FIRST	STL	RETADR
1003	CLOOP	JSUB	RDREC
…	…	…	…
1012		J	CLOOP
…	…	…	…
1033	RETADR	RESW	1

- Because of forward reference most assemblers make two passes over the source program
- In addition the assembler must process assembler directives
- The assembler must write the generated object code onto some o/p device. This object code later loaded to memory for execution
- The simple object program format consist of 3 types of records

Object Program Format

- Header Record

  Col. 1 H

  Col. 2~7 Program name

  Col. 8~13 Starting address (hex)

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

Object Program Format

- Text Record

  Col.1 T

  Col.2~7 Starting address in this record (hex)

  Col. 8~9 Length of object code in this record in bytes (hex)

  Col. 10~69 Object code (69-10+1)/6=10 instructions

- End Record

  Col.1 E

  Col.2~7 Address of first executable instruction (hex)

  (END program_name)

Fig. 2.3

H COPY 001000 00107A

T 001000 1E 141033 482039 001036 281030 301015 482061 ...

T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000

T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 …

T 002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 …

T 002073 07 382064 4C0000 05

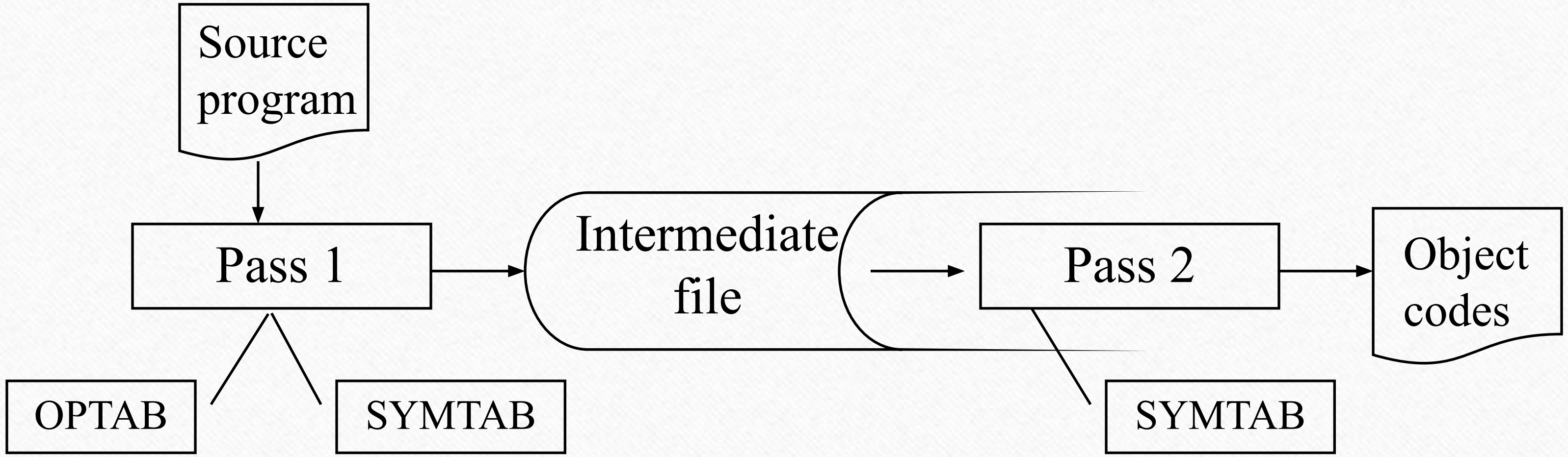
E 001000

Two Pass Assembler

- Pass 1
  - Assign addresses to all statements in the program
  - Save the values assigned to all labels for use in Pass 2
  - Perform some processing of assembler directives
- Pass 2
  - Assemble instructions
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done in Pass 1
  - Write the object program and the assembly listing

Two Pass Assembler

- Read from input line
  - LABEL, OPCODE, OPERAND



Data Structures

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter(LOCCTR)

OPTAB (operation code table)

- Content
  - menmonic, machine code (instruction format, length) etc.
    - Used for
      - PASS1: Used to lookup mnemonic opcode and validate opcode in source program
      - PASS2: translate them to their machine language equivalent
- Characteristic
  - static table
- Implementation
  - array or hash table, easy for search

SYMTAB (symbol table)

- Content
  - label name, value, flag, (type, length) etc.
  - Used for
  - PASS 1: Labels are entered into Symbol table as they are encountered in source program along with their assigned address (from LOCCTR)
  - PASS 2: Symbols used as operands are looked up in Sym tab to obtain the addresses to be inserted in the assembler instructions
- Characteristic
  - dynamic table (insert, delete, search)
- Implementation
  - hash table, non-random keys, hashing function

COPY	1000
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	1024
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039

Location Counter(LOCCTR)

- A variable used to help in assignment of addresses
- LOCCTR is initialized to the beginning address specified in the START statement
- After each source statement is processed the length of the assembled instruction or data area o be generated is added to LOCCTR
- Whenever we reach a label in the source program, the current value of LOCCTR gives the address to be associated with the label

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}
```

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}
```

Example program

```
SUM     START   1000
FIRST   LDX     ZERO
        LDA     ZERO
LOOP    ADD     TABLE,X
        TIX     COUNT
        JLT     LOOP
        STA     TOTAL
        RSUB
TABLE   RESW    2000
COUNT   RESW    1
ZERO    WORD    0
TOTAL   RESW    1
        END     FIRST
```

Example program

```
1000    SUM     START   1000
1000    FIRST LDX       ZERO
1003            LDA     ZERO
1006    LOOP ADD        TABLE,X
1009            TIX     COUNT
100C            JLT     LOOP
100F            STA     TOTAL
1012            RSUB
1015    TABLE   RESW 2000
1018    COUNT   RESW 1
101B    ZERO WORD       0
101E    TOTAL   RESW 1
1021            END     FIRST
```

OPTAB	
Mnemonic	Opcode
LDX	04
LDA	00
ADD	18
STA	0C
RSUB	4C
TIX	2C
JLT	38

SYMTAB	
Name	LOCCTR
FIRST	1000
LOOP	1006
TABLE	1015
COUNT	1018
ZERO	101B
TOTAL	101E

- Machine Dependent Assembler Features
  - instruction formats and addressing modes
  - program relocation

Machine-dependent Assembler Features

Instruction formats and addressing modes

Program relocation

Instruction Format and Addressing Mode

- SIC/XE
  - PC-relative or Base-relative addressing: op m
  - Indirect addressing:(avoids another instruction)op @m
  - Immediate addressing(no memory fetch) op #c
  - Extended format: +op m

  (programs responsibility to specify this mode when required)
  - Index addressing: op m,x
  - register-to-register instructions in place of reg-to-memory
  - larger memory -> multi-programming (program allocation)
  - BASE is used in conjunction with base relative address

An SIC/XE Example

Loc		Source statement		Object code
0000	COPY	START	0	
0000	FIRST	STL	RETADR	17202D
0003		LDB	#LENGTH	69202D
		BASE	LENGTH	
0006	CLOOP	+JSUB	RDREC	4B101036
000A		LDA	LENGTH	032026
000D		COMP	#0	290000
0010		JEQ	ENDFIL	332007
0013		+JSUB	WRREC	4B10105D
0017		J	CLOOP	3F2FEC
001A	ENDFIL	LDA	EOF	032010
001D		STA	BUFFER	0F2016
0020		LDA	#3	010003
0023		STA	LENGTH	0F200D
0026		+JSUB	WRREC	4B10105D
002A		J	@RETADR	3E2003
002D	EOF	BYTE	C'EOF'	454F46
0030	RETADR	RESW	1	
0033	LENGTH	RESW	1	
0036	BUFFER	RESB	4096	

```
          .
          .         SUBROUTINE TO READ RECORD INTO BUFFER

1036      RDREC      CLEAR      X          B410
1038                 CLEAR      A          B400
103A                 CLEAR      S          B440
103C                +LDY        #4096      75101000
1040      RLOOP      TD         INPUT      E32019
1043                 JEQ        RLOOP      332FFA
1046                 TD         INPUT      DB2013
1049                 COMPR      A,S        A004
104B                 JEQ        EXIT       332008
104E                 STCH       BUFFER,X   57C003
1051                 TIXR       T          B850
1053                 JLT        RLOOP      3B2FEA
1056      EXIT       STX        LENGTH     134000
1059                 RSUB                  4F0000
105C      INPUT      BYTE       X'F1'      F1
```

```
            .
            .          SUBROUTINE TO READ RECORD INTO BUFFER

105D    WRREC      CLEAR        X           B410
105F               LDT          LENGTH      774000
1062    WLOOP      TD           OUTPUT      E32011
1065               JEQ          WLOOP       332FFA
1068               LDCH         BUFFER,X    53C003
106B               WD           OUTPUT      DF2008
106E               TIXR         T           B850
1070               JLT          WLOOP       3B2FEF
1073               RSUB                     4F0000
1076    OUTPUT     BYTE         X'05'       05
                   END          FIRST
```

PC-Relative Addressing Modes

- PC-relative
  - 10 0000 FIRST STL RETADR 17202D

op(6)	n	I	x	b	p	e	disp(12)

$(14)_{16}$ 1 1 0 0 1 0 $(02D)_{16}$

  - displacement= RETADR - PC = 30-3 = 2D
  - 40 0017 J CLOOP 3F2FEC

op(6)	n	I	x	b	p	e	disp(12)

$(3C)_{16}$ 1 1 0 0 1 0 $(FEC)_{16}$

  - displacement= CLOOP-PC= 6 - 1A= -14= FEC

Base-Relative Addressing Modes

- Base-relative
  - base register is under the control of the programmer
  - 12 LDB #LENGTH
  - 13 BASE LENGTH
  - 160 104E STCH BUFFER, X 57C003

op(6)	n	I	x	b	p	e	disp(12)

$(54)_{16}$ 1 1 1 1 0 0 $(003)_{16}$

(54) 1 1 1 0 1 0 0036-1051= $-101B_{16}$

  - displacement= BUFFER - B = 0036 - 0033 = 3

- NOBASE is used to inform the assembler that the contents of the base register no longer be relied upon for addressing

Immediate Address Translation

- Immediate addressing
  - 55 0020 LDA #3 010003

op(6)	n	I	x	b	p	e	disp(12)
$(00)_{16}$	0	1	0	0	0	0	$(003)_{16}$

  - 133 103C +LDT #4096 75101000

op(6)	n	I	x	b	p	e	disp(20)
$(74)_{16}$	0	1	0	0	0	1	$(01000)_{16}$

Immediate Address Translation (Cont.)

- Immediate addressing

  - 12   0003   LDB   #LENGTH 69202D

op(6)	n	I	x	b	p	e	disp(12)

$(68)_{16}$ 0 1 0 0 1 0 $(02D)_{16}$

$(68)_{16}$ 0 1 0 0 0 0 $(033)_{16}$ 690033

- the immediate operand is the symbol LENGTH
- the address of this symbol LENGTH is loaded into register B
- LENGTH=0033=PC+ displacement = 0006+02D
- if immediate mode is specified, the target address becomes the operand

Indirect Address Translation

- Indirect addressing
  - target addressing is computed as usual (PC-relative or BASE-relative)
  - only the n bit is set to 1
  - 70 002A J @RETADR 3E2003

op(6)	n	I	x	b	p	e	disp(12)
$(3C)_{16}$	1	0	0	0	1	0	$(003)_{16}$

- TA=RETADR=0030
- TA=(PC)+disp=002D+0003

Program Relocation

- More than one program share the memory
- In advance programs running concurrently??
- Then we would load without overlap or waste of space
- But not practical to plan program execution (jobs summited nor time)
- So load whenever there is room in memory
- Then actual staring address is not known until loads time.

Program Relocation

- Example
  - *Absolute program*, starting address 1000

    e.g. 55 101B LDATHREE 00102D
  - Relocate the program to 2000

    e.g. 55 101B LDATHREE 00202D
  - Each Absolute address should be modified
- Example
  - Except for absolute address, the rest of the instructions need not be modified
    - not a memory address (immediate addressing)
    - PC-relative, Base-relative
  - The only parts of the program that require modification at load time are those that specify direct addresses

Program Relocation

- Eg SIC absolute program
- Line 55 00102D
- Load at 2000
- Then 102D will not contain the value needed
- Assembler doesn’t know the actual address, so it cannot make the changes
- But identification and modification can be told by the assembler to loader
- An object program containing modification information is relocatable program

Example SIC/XE

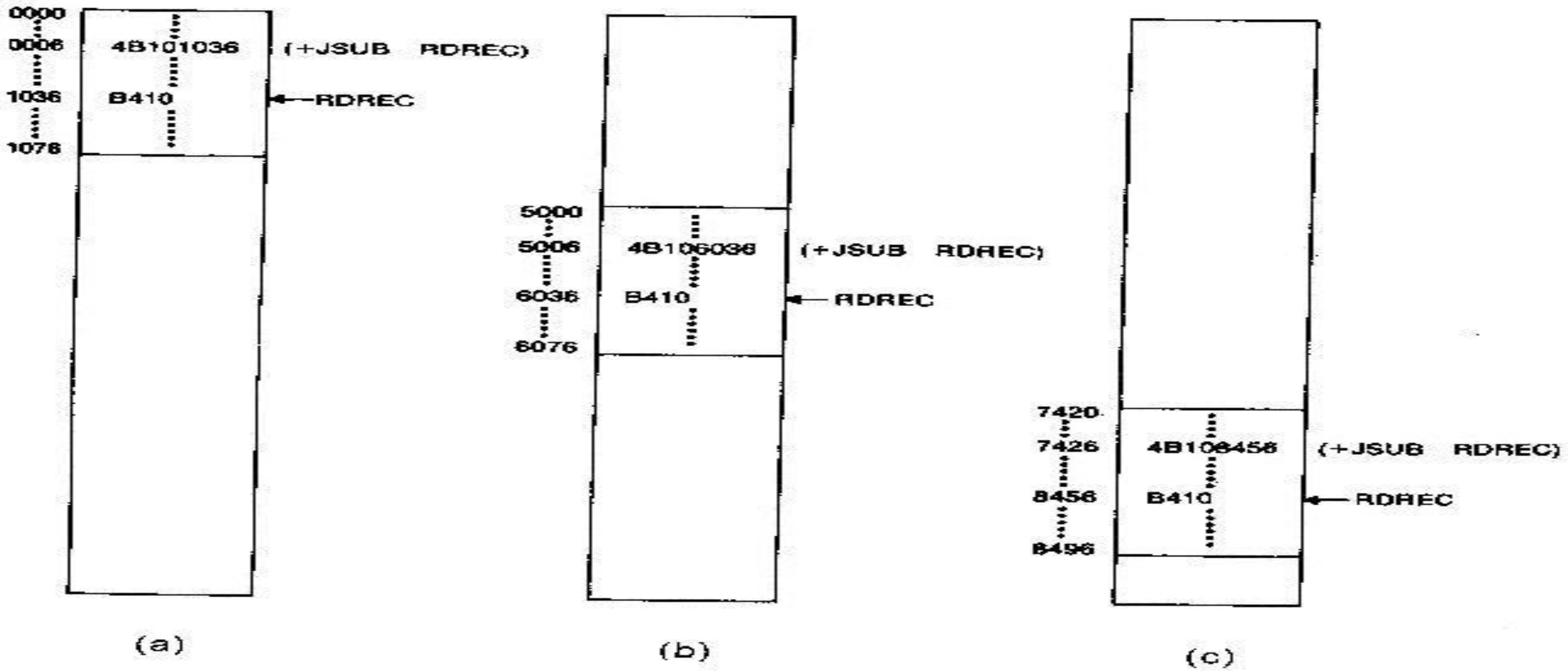


圖2.7 程式重定位範例

Relocatable Program

- Modification record
  - Col 1 M
  - Col 2-7 Starting location of the address field to be modified, relative to the beginning of the program
  - Col 8-9 length of the address field to be modified, in half- bytes

Object Code

```
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

End of Chapter 02

Assembler