組合語言與系統程式

ASSEMBLY LANGUAGE AND SYSTEM PROGRAMMING

Speaker: Shu-Yu Kuo

Department of Computer Science

National Chung Hsing University

SYSTEM SOFTWARE: AN INTRODUCTION TO SYSTEMS PROGRAMMING

LELAND .L. BECK

Chapter 2 Assemblers

OUTLINE

- **2.1 Basic Assembler Functions**
- 2.2 Machine-Dependent Assembler Features
- 2.3 Machine-Independent Assembler Features
- 2.4 Assembler Design Options
- 2.5 Implementation Examples

INTRODUCTION TO ASSEMBLERS

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

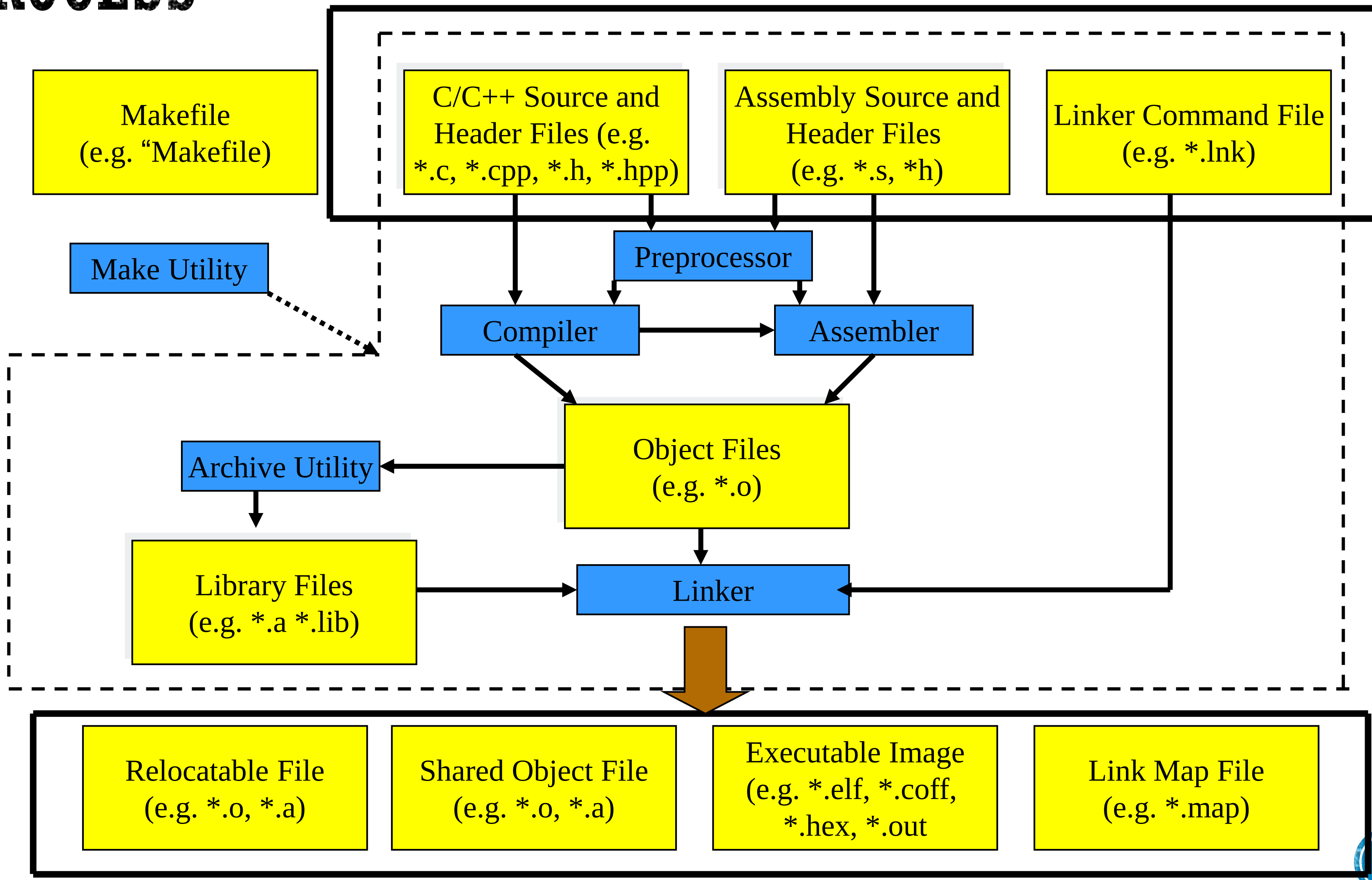
EXAMPLE OF A SIC ASSEMBLER LANGUAGE PROGRAM (FIG 2.1,2.2)

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	

2.1 BASIC ASSEMBLER FUNCTIONS

- ***Assembler:*** a program
  - Accepts an *assembly language program* as input
  - Produces its *machine language equivalent* along with *information for the loader*

OVERVIEW OF LINKERS AND THE LINKING PROCESS



ASSEMBLER DIRECTIVES

- **Pseudo-instructions**
  - Not translated into machine instructions
  - Provide instructions to the assembler itself
- **Basic assembler directives**
  - **START:** specify *name* and *starting address of the program*
  - **END:** specify *end of program* and (option) *the first executable instruction in the program*
    - If not specified, use the address of the first executable instruction
  - **BYTE:** *generate character or hexadecimal constants*
  - **WORD**： generate one-word integer constant
  - **RESB:** : instruct the assembler to *reserve memory location* without generating data values
  - **RESW**

EXAMPLE OF A SIC ASSEMBLER LANGUAGE PROGRAM (FIG 2.1,2.2)

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	

EXAMPLE OF A SIC ASSEMBLER LANGUAGE PROGRAM (FIG 2.1,2.2) (CONT.)

```
                .
                .       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
```

EXAMPLE OF A SIC ASSEMBLER LANGUAGE PROGRAM (FIG 2.1,2.2) (CONT.)

```
                .
                .       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
```

Figure 2.2 Program from Fig. 2.1 with object code.

EXAMPLE OF A SIC ASSEMBLER LANGUAGE PROGRAM (FIG 2.2)

- ***Loc*** column shows the *machine address (memory address)* for each part of the program

  - Assume program starts at address 1000

  - A ***location counter*** is used to keep track the address changing

FUNCTIONS OF A BASIC ASSEMBLER

- Convert *mnemonic operation codes* to their *machine language equivalents*
  - E.g. STL -> 14 (line 10)
- Convert *symbolic operands* to their equivalent *machine addresses*
  - E.g. RETADR -> 1033 (line 10)
- Build the machine instructions in the proper format
- Convert the *data constants* to *internal machine representations*
  - E.g. EOF -> 454F46 (line 80)
- Write the *object program (object file)*

FUNCTIONS OF A BASIC ASSEMBLER (CONT.)

- All of above functions can be accomplished by *sequential processing* of the source program
  - Except step 2 in processing symbolic operands
- Example
  - **10 STL RETADR**
    - RETADR is not yet defined when we encounter STL instruction
    - Called **forward reference**

ADDRESS TRANSLATION PROBLEM

- **Forward reference**
  - A reference to a label that is defined later in the program
    - We will be unable to process this statement sequentially

- As a result, most assemblers make **2 passes** over the source program
  - ***1st pass***: scan *label definitions* and *assign addresses*
  - ***2nd pass***: actual translation (object code)

OBJECT PROGRAM

- Finally, assembler must write the generated object code to some output device
  - Called *object program*
  - Will be later loaded into memory for execution

OBJECT PROGRAM FOR FIG 2.2 (FIG 2.3)

Program name, Starting address (hex),Length of object program in bytes (hex)

```
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
```

Starting address (hex),Length of object code in this record (hex),Object code (hex)

Address of first executable instruction (hex)

F... 2.3 Object program corresp...

OBJECT PROGRAM (CONT.)

- Contains 3 types of records:
  - **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)
  - **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 (hex) (2 columns per byte)
  - **End record**

    Col.1 E
    Col.2~7 Address of first executable instruction (hex) (END program_name)

2.1.2 ASSEMBLER ALGORITHM AND DATA STRUCTURES

- Algorithm
  - Two-pass assembler

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

INTERNAL DATA STRUCTURES

OPTAB (operation code table)

- Content
  - Mnemonic machine code and its machine language equivalent
  - May also include instruction format, length etc.
- Usage
  - Pass 1: used to look up and validate operation codes in the source program
  - Pass 2: used to translate the operation codes to machine language
- Characteristics
  - Static table, predefined when the assembler is written
- Implementation
  - Array or hash table with mnemonic operation code as the key (preferred)
- Ref. Appendix A

OPTAB

Mnemonic	OPCODE	SIC/XE-Specific	Format
ADD	18		3/4
ADDF	58	v	3/4
ADDR	90	v	2
AND	40		3/4
……	……	……	……

INTERNAL DATA STRUCTURES

- **SYMTAB (symbol table)**
  - Content
    - Label name and its value (address)
    - May also include flag (type, length) etc.
  - Usage
    - Pass 1: labels are entered into SYMTAB with their address (from LOCCTR) as they are encountered in the source program
    - Pass 2: symbols used as operands are looked up in SYMTAB to obtain the address to be inserted in the assembled instruction
  - Characteristic
    - Dynamic table (insert, delete, search)
  - Implementation
    - Hash table for efficiency of *insertion* and *retrieval*

SYMTAB

Label Name	Address
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	102A
……	……
RETADR	1033

INTERNAL DATA STRUCTURES

- **Location Counter**
  - A variable used to *assign addresses*
  - Initialized to the beginning address specified in the START statement

ALGORITHM FOR 2 PASS ASSEMBLER (FIG 2.4)

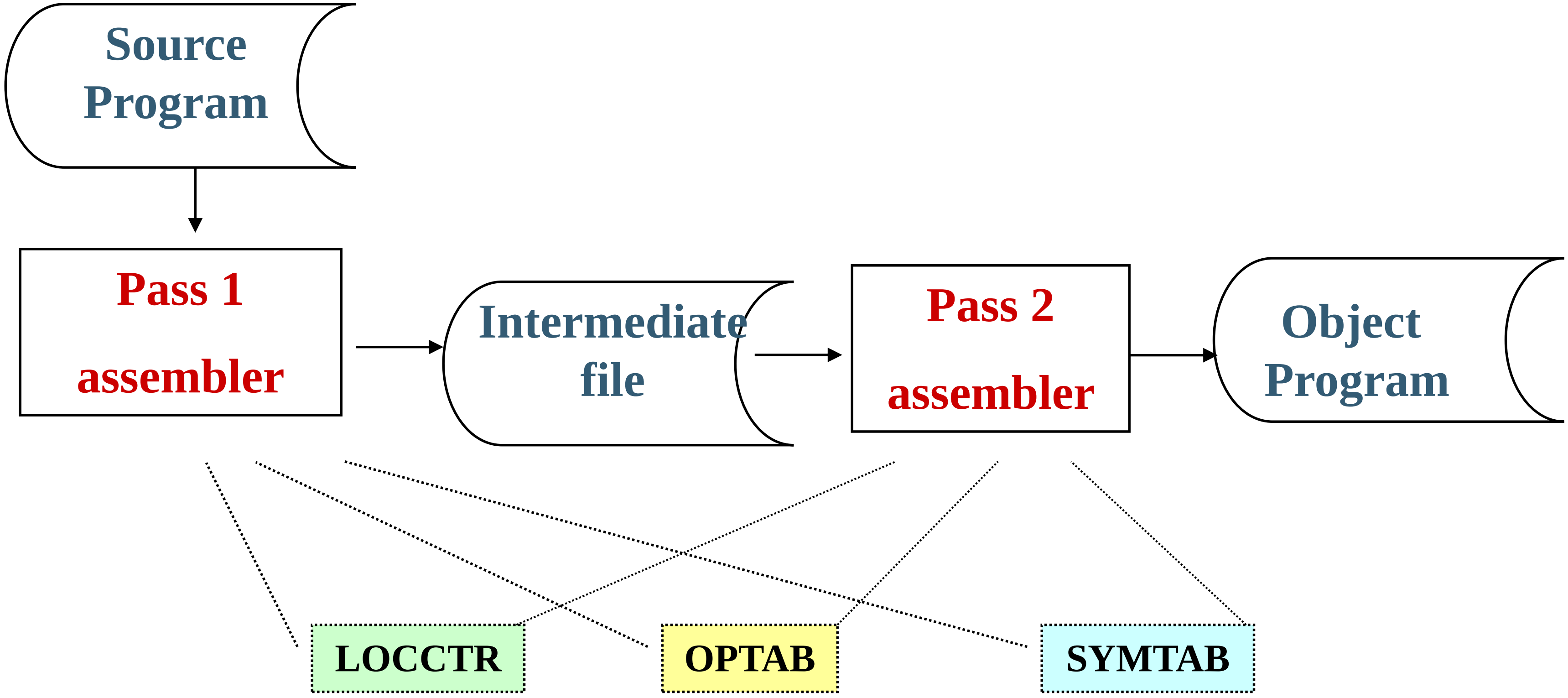
- Figure 2.4 (a): algorithm for pass 1 of assembler
- Figure 2.4 (b): algorithm for pass 2 of assembler

ALGORITHM FOR 2 PASS ASSEMBLER (FIG 2.4)

- Both pass1 and pass 2 need to read the source program.
  - However, pass 2 needs more information
    - *Location counter value*, error flags
- **Sol: An intermediate file**
  - Contains each source statement with its assigned address, error indicators, etc.
  - Used as the **input to Pass 2**

INTERMEDIATE FILE

- LABEL, OPCODE, OPERAND



ALGORITHM FOR PASS 1 OF ASSEMBLER (FIG 2.4A)

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}
```

Label

LOC

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

ALGORITHM FOR PASS 2 OF ASSEMBLER (FIG 2.4B)

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.

ALGORITHM FOR 2 PASS ASSEMBLER

- Pass 1 (define symbol)
  1. Assign addresses to all statement in the program
  2. Save the value (addresses) assigned to all labels for user in Pass 2.
  3. Perform some processing of assembler directives
- Pass2 (assemble instruction and generate object program)
  1. Assemble instructions (translating operation codes and looking up address)
  2. Perform processing of assembler directives not done during Pass1.
  3. Generate data values defined by BYTE, WORD, etc.
  4. Write the object program

OUTLINE

- 2.1 Basic Assembler Functions
- **2.2 Machine-Dependent Assembler Features**
  - Instruction Formats and Addressing Modes
  - Program Relocation
- 2.3 Machine-Independent Assembler Features
- 2.4 Assembler Design Options
- 2.5 Implementation Examples

2.2 MACHINE DEPENDENT ASSEMBLER FEATURES

- Machine Dependent Assembler Features
  - Example: SIC/XE supports
    - More instruction formats and addressing modes
    - Program relocation

SIC/XE ASSEMBLER

- Previous, we know how to implement the 2-pass SIC assembler.

- What's new for SIC/XE?
  - More addressing modes and instruction formats.
  - Program Relocation.

SIC/XE ASSEMBLER (CONT.)

- SIC/XE
  - Immediate addressing: op #c
  - Indirect addressing: op @m
  - PC-relative or Base-relative addressing: op m
    - The assembler directive **BASE** is used with base-relative addressing
    - If displacements are too large to fit into a 3-byte instruction, then 4-byte extended format is used
  - Extended format: +op m
  - Indexed addressing: op m, x
  - Register-to-register instructions
  - Large memory
    - Support multiprogramming and need *program reallocation* capability

EXAMPLE OF A SIC/XE PROGRAM (FIG 2.5)

- Improve the execution speed of Fig. 2.2 (SIC version)
  - Register-to-register instructions
  - Immediate addressing: op #c
    - Operand is already present as part of the instruction
  - Indirect addressing: op @m
    - Often avoid the need of another instruction

EXAMPLE OF A SIC/XE PROGRAM (FIG 2.5,2.6)

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	

EXAMPLE OF A SIC/XE PROGRAM (FIG 2.5,2.6) (CONT.)

```
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
```

EXAMPLE OF A SIC/XE PROGRAM (FIG 2.5,2.6) (CONT.)

```
                .
                .       SUBROUTINE TO WRITE RECORD FROM 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
```

Figure 2.6 Program from Fig. 2.5 with object code.

OUTLINE

- 2.1 Basic Assembler Functions
- **2.2 Machine-Dependent Assembler Features**
  - **Instruction Formats and Addressing Modes**
  - Program Relocation
- 2.3 Machine-Independent Assembler Features
- 2.4 Assembler Design Options
- 2.5 Implementation Examples

2.2.1 INSTRUCTION FORMATS AND ADDRESSING MODES

- START now specifies a beginning program address of 0
  - Indicate a *relocatable program*
- Register translation
  - For example: *COMPR A, S => A004*
  - Must keep the register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
    - Keep in **SYMTAB**

ADDRESS TRANSLATION

- Most register-to-memory instructions are assembled using *PC relative* or *base relative* addressing
  - Assembler must calculate a *displacement* as part of the object instruction
  - If displacement can be fit into 12-bit field, format 3 is used.
  - Format 3: 12-bit address field
    - **Base-relative: 0~4095**
    - **PC-relative: -2048~2047**
  - *Assembler attempts to **translate using PC-relative first, then base-relative***
    - If displacement in PC-relative is out of range, then try base-relative

ADDRESS TRANSLATION (CONT.)

- If displacement can not be fit into 12-bit field in the object instruction, format 4 must be used.
  - Format 4: 20-bit address field
  - No displacement need to be calculated.
    - 20-bit is large enough to contain the full memory address
  - Programmer must specify extended format: +op m
  - For example: *+JSUB RDREC => 4B101036*
    - *LOC(RDREC) = 1036, get it from SYMTAB*

PC-RELATIVE ADDRESSING MODES

- *10 0000 FIRST STL RETADR 17202D*
  - Displacement= RETADR – (PC) = 30-**3** = 2D
  - opcode (6 bits) =14_{16}=00010100_2
  - nixbpe=110010
    - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
    - p = 1: indicate *PC-relative* addressing

OPCODE	n	i	x	b	p	e	Displacement
0001 01	1	1	0	0	1	0	$(02D)_{16}$

Object Code = 17202D

PC-RELATIVE ADDRESSING MODES (CONT.)

- 40 0017 J CLOOP 3F2FEC
  - Displacement= CLOOP - (PC) = 6 - **1A** = -14 = FEC (2's complement for negative number)
  - opcode=$3C_{16}$ = 00111100_2
  - nixbpe=110010

OPCODE	n	i	x	b	p	e	Displacement
0011 11	1	1	0	0	1	0	$(FEC)_{16}$

Object Code = 3F2FEC

BASE-RELATIVE ADDRESSING MODES

- Base register is under the control of the programmer
  - Programmer use **assembler directive** ***BASE*** to specify which value to be assigned to base register (B)
  - **Assembler directive** ***NOBASE***: inform the assembler that the contents of base register no longer be used for addressing
  - ***BASE*** and ***NOBASE*** produce no executable code

BASE-RELATIVE ADDRESSING MODES

- 175 1056 STX LENGTH 134000
  - Try PC-relative first
    - Displacement= LENGTH - (PC) = 0033 - 1059 = -1026 (hex)
  - Try base-relative next
    - displacement= LENGTH – (B) = 0033 – 0033 =0
    - Opcode=10_{16} = $(00010000)_2$
    - nixbpe=110100
      - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
      - b = 1: *base-relative* addressing

OPCODE	n	i	x	b	p	e	Displacement
000100	1	1	0	1	0	0	$(000)_{16}$

BASE-RELATIVE ADDRESSING MODES (CONT.)

- *160 104E STCH BUFFER, X 57C003*
  - The displacement of PC-relative is out of range
  - Displacement= BUFFER – (B) = 0036 – 0033(=LOC(LENGTH)) = 3
  - opcode=54
  - nixbpe=111100
    - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
    - x = 1: *indexed* addressing
    - b = 1: *base-relative* addressing

OPCODE	n	i	x	b	p	e	Displacement
0101 01	1	1	1	1	0	0	$(003)_{16}$

Object Code = 57C003

IMMEDIATE ADDRESS TRANSLATION

- Convert the *immediate* operand to its internal representation and insert it into the instruction
- 55 0020 LDA #3 010003
  - opcode=00
  - nixbpe=010000
    - i = 1: *immediate addressing*

OPCODE	n	i	x	b	p	e	Displacement
0000 00	0	1	0	0	0	0	$(003)_{16}$

Object Code = 010003

IMMEDIATE ADDRESS TRANSLATION (CONT.)

- 133 103C +LDT #4096 75101000
  - opcode=74
  - nixbpe=010001
    - i = 1: *immediate addressing*
    - e = 1: *extended instruction format* since 4096 is too large to fit into the 12-bit displacement field

OPCODE	n	i	x	b	p	e	Address
0111 01	0	1	0	0	0	1	$(01000)_{16}$

Object Code = 75101000

IMMEDIATE ADDRESS TRANSLATION (CONT.)

- 12 0003 LDB   #LENGTH   69202D
  - The immediate operand is the symbol LENGTH
    - The address of LENGTH is loaded into register B
  - Displacement=LENGTH – (PC) = 0033 – 0006 = 02D
  - opcode=68_{16} = 01101000_2
  - nixbpe=010010
    - Combined *PC relative* (p=1) with *immediate addressing* (i=1)

OPCODE	n	i	x	b	p	e	Displacement
0110 10	0	1	0	0	1	0	$(02D)_{16}$

IMMEDIATE ADDRESS TRANSLATION (CONT.)

- 55 0020 LDA #3 010003
  - opcode = 00_{16} = 00000000_2
  - nixbpe=010000
    - i = 1: immediate addressing

OPCODE	n	i	x	b	P	e	Displacement
0000 00	0	1	0	0	0	0	$(003)_{16}$

INDIRECT ADDRESS TRANSLATION

- Indirect addressing
  - The contents stored at the location represent the *address* of the operand, not the operand itself
  - Target addressing is computed as usual (PC-relative or BASE-relative)
  - *n* bit is set to 1

INDIRECT ADDRESS TRANSLATION (CONT.)

- 70 002A J @RETADR 3E2003
  - Displacement= RETADR- (PC) = 0030 – 002D =3
  - opcode= 3C
  - nixbpe=100010
    - n = 1: *indirect addressing*
    - p = 1: *PC-relative addressing*

OPCODE	n	i	x	b	p	e	Displacement
0011 11	1	0	0	0	1	0	$(003)_{16}$