

System Software

| | |
|---|-----------------------------|
| Semester : VI | Course Code : 15CS63 |
| Course Title : System Software AND Compiler Design | |
| Faculty : Niranjan Murthy C | |
| Dept : Computer Science & engineering | |

| | |
|--|---|
| Prerequisites: | Basic concepts of microprocessors (10CS45) |
| Description | This course gives an introduction to the design and implementation of various types of system software. A central theme of the course is the relationship between machine architecture and system software. The design of an assembler or an operating system is greatly influenced by the architecture of the machine on which it runs. These influences are emphasized and demonstrated through the discussion of actual pieces of system software for a variety of real machines. |
| <p>Outcomes</p> <p>The students should be able to:</p> <ol style="list-style-type: none"> 1. Student able to Define System Software such as Assembler and Macroprocessor. 2. Student able to Define System Software such as Loaders and Linkers 3. Student able to lexical analysis and syntax analysis Familiarize with source file, object and executable file structures and libraries 4. Describe the front and back end phases of compiler and their importance to students | |

MODULE- 1

- **Introduction to System Software,**
- **Machine Architecture of SIC and SIC/XE.**
- **Assemblers: Basic assembler functions, machine dependent assembler features,**
- **machine independent assembler features, assembler design options.**
- **Macroprocessors: Basic macro processor functions, ->10 Hours**

MACHINE ARCHITECTURE**System Software:**

- System software consists of a variety of programs that support the operation of a computer.
- Application software focuses on an application or problem to be solved.
- System softwares are the machine dependent softwares that allows the user to focus on the application or problem to be solved, without bothering about the details of how the machine works internally.

Examples: Operating system, compiler, assembler, macroprocessor, loader or linker, debugger, text editor, database management systems, etc.

Difference between System Software and application software

| System Software | Application Software |
|---|---|
| System software is machine dependent | Application software is not dependent on the underlying hardware. |
| System software focus is on the computing system. | Application software provides solution to a problem |
| Examples: Operating system, compiler, assembler | Examples: Antivirus, Microsoft office |

SIC – Simplified Instructional Computer

Simplified Instructional Computer (SIC) is a hypothetical computer that includes the hardware features most often found on real machines. There are two versions of SIC, they are, standard model (SIC), and, extension version (SIC/XE) (extra equipment or extra expensive).

SIC Machine Architecture:

We discuss here the SIC machine architecture with respect to its Memory and Registers, Data Formats, Instruction Formats, Addressing Modes, Instruction Set, Input and Output.

Memory:

There are 215 bytes in the computer memory, that is 32,768 bytes. It uses Little Endian format to store the numbers, 3 consecutive bytes form a word , each location in memory contains 8-bit bytes.

Registers:

There are five registers, each 24 bits in length. Their mnemonic, number and use are given in the following table.

| Mnemonic | Number | Use |
|----------|--------|---|
| A | 0 | Accumulator; used for arithmetic operations |
| X | 1 | Index register; used for addressing |
| L | 2 | Linkage register; JSUB |
| PC | 8 | Program counter |
| SW | 9 | Status word, including CC |

Data Formats:

Integers are stored as 24-bit binary numbers. 2's complement representation is used for negative values, characters are stored using their 8-bit ASCII codes. No floating-point hardware on the standard version of SIC.

Instruction Formats:

| | | |
|-----------|---|--------------|
| Opcode(8) | x | Address (15) |
|-----------|---|--------------|

X is used to indicate indexed-addressing mode.

All machine instructions on the standard version of SIC have the 24-bit format as shown above.

Addressing Modes:

Only two modes are supported: Direct and Indexed

| Mode | Indication | Target address calculation |
|---------|------------|----------------------------|
| Direct | x= 0 | TA = address |
| Indexed | x= 1 | TA = address + (x) |

() are used to indicate the content of a register.

Instruction Set

- Load and store registers (LDA, LDX, STA, STX)
- Integer arithmetic (ADD, SUB, MUL, DIV), all involve register A and a word in memory.
- Comparison (COMP), involve register A and a word in memory.
- Conditional jump (JLE, JEQ, JGT, etc.)
- Subroutine linkage (JSUB, RSUB)

Input and Output

- One byte at a time to or from the rightmost 8 bits of register A.
- Each device has a unique 8-bit ID code.
- Test device (TD): test if a device is ready to send or receive a byte of data.
- Read data (RD): read a byte from the device to register A
- Write data (WD): write a byte from register A to the device.

SIC/XE Machine Architecture:

Memory

- Maximum memory available on a SIC/XE system is 1 Megabyte (2²⁰ bytes).

Registers

- Additional B, S, T, and F registers are provided by SIC/XE, in addition to the registers of SIC.

| Mnemonic | Number | Special use |
|----------|--------|--------------------------------------|
| B | 3 | Base register |
| S | 4 | General working register |
| T | 5 | General working register |
| F | 6 | Floating-point accumulator (48 bits) |

Floating-point data type:

- There is a 48-bit floating-point data type, F*2(e-1024)

Instruction Formats :

The new set of instruction formats for SIC/XE machine architecture are as follows.

Format 1 (1 byte): contains only operation code (straight from table).

Format 2 (2 bytes): first eight bits for operation code, next four for register 1 and following four for register 2. The numbers for the registers go according to the numbers indicated at the registers section (ie, register T is replaced by hex 5, F is replaced by hex 6).

Format 3 (3 bytes): First 6 bits contain operation code, next 6 bits contain flags, last 12 bits contain displacement for the address of the operand. Operation code uses only 6 bits, thus the second hex digit will be affected by the values of the first two flags (n and i). The flags, in order, are: n, i, x, b, p, and e. Its functionality is explained in the next section. The last flag e indicates the instruction format (0 for 3 and 1 for 4).

Format 4 (4 bytes): same as format 3 with an extra 2 hex digits (8 bits) for addresses that require more than 12 bits to be represented.

Addressing Modes:

Five possible addressing modes plus the combinations are as follows.

1. Direct (x, b, and p all set to 0): operand address goes as it is. n and i are both set to the same value, either 0 or 1. While in general that value is 1, if set to 0 for format 3 we can assume that the rest of the flags (x, b, p, and e) are used as a part of the address of the operand, to make the format compatible to the SIC format.

2. Relative (either b or p equal to 1 and the other one to 0): the address of the operand should be added to the current value stored at the B register (if b = 1) or to the value stored at the PC register (if p = 1)

3. Immediate(i = 1, n = 0): The operand value is already enclosed on the instruction (ie. lies on the last 12/20 bits of the instruction)

4. Indirect(i = 0, n = 1): The operand value points to an address that holds the address for the operand value.

5. Indexed (x = 1): value to be added to the value stored at the register x to obtain real address of the operand. This can be combined with any of the previous modes except immediate.

The various flag bits used in the above formats have the following meanings

e -> e = 0 means format 3, e = 1 means format 4

Bits x,b,p : Used to calculate the target address using relative, direct, and indexed addressing Modes.

Bits i and n: Says, how to use the target address b and p - both set to 0, disp field from format 3 instruction is taken to be the target address.

For a format 4 bits b and p are normally set to 0, 20 bit address is the target address

x - x is set to 1, X register value is added for target address calculation

i=1, n=0 Immediate addressing, TA: TA is used as the operand value, no memory reference

i=0, n=1 Indirect addressing, ((TA)): The word at the TA is fetched. Value of TA is taken as the address of the operand value

i=0, n=0 or i=1, n=1 Simple addressing, (TA):TA is taken as the address of the operand value

Two new relative addressing modes are available for use with instructions assembled using format 3.

Instruction Set:

SIC/XE provides all of the instructions that are available on the standard version. In addition we have, Instructions to load and store the new registers LDB, STB, etc, Floating-point arithmetic operations, ADDF, SUBF, MULF, DIVF, Register move instruction : RMO, Register-to-register arithmetic operations, ADDR, SUBR, MULR, DIVR and, Supervisor call instruction : SVC.

Input and Output:

There are I/O channels that can be used to perform input and output while the CPU is executing other instructions. Allows overlap of computing and I/O, resulting in more efficient system operation. The instructions SIO, TIO, and HIO are used to start, test and halt the operation of I/O channels.

Example programs SIC:

Example 1: Simple data and character movement operation

| | | |
|-------|------|-------|
| | LDA | FIVE |
| | STA | ALPHA |
| | LDCH | CHARZ |
| | STCH | C1 |
| ALPHA | RESW | 1 |
| FIVE | WORD | 5 |
| CHARZ | BYTE | C'Z' |
| C1 | RESB | 1 |

Example 2: Arithmetic operations

| | | |
|--|-----|-------|
| | LDA | ALPHA |
| | ADD | INCR |
| | SUB | ONE |
| | STA | BETA |

.....

.....

.....

| | | |
|-------|------|---|
| ONE | WORD | 1 |
| ALPHA | RESW | 1 |
| BEETA | RESW | 1 |
| INCR | RESW | 1 |

Example 3: Looping and Indexing operation

| | | | | |
|--------|------|---------|--|---------|
| | LDX | ZERO | | ; X = 0 |
| MOVECH | LDCH | STR1, X | | |
| | STCH | STR2, X | | |
| | TIX | ELEVEN | | |
| | JLT | MOVECH | | |

.....

.....

.....

| | | |
|--------|------|-----------------|
| STR1 | BYTE | C 'HELLO WORLD' |
| STR2 | RESB | 11 |
| ZERO | WORD | 0 |
| ELEVEN | WORD | 11 |

Example 4: Input and Output operation

| | | | |
|--------|------|--------|------------------------------|
| INLOOP | TD | INDEV | ; TEST INPUT DEVICE |
| | JEQ | INLOOP | ; LOOP UNTIL DEVICE IS READY |
| | RD | INDEV | ; READ ONE BYTE INTO A |
| | STCH | DATA | ; STORE A TO DATA |
| . | | | |
| . | | | |
| OUTLP | TD | OUTDEV | ; TEST OUTPUT DEVICE |
| | JEQ | OUTLP | ; LOOP UNTIL DEVICE IS READY |
| | LDCH | DATA | ; LOAD DATA INTO A |
| | WD | OUTDEV | ; WRITE A TO OUTPUT DEVICE |
| . | | | |
| . | | | |
| INDEV | BYTE | X 'F5' | ; INPUT DEVICE NUMBER |
| OUTDEV | BYTE | X '08' | ; OUTPUT DEVICE NUMBER |
| DATA | RESB | 1 | ; ONE-BYTE VARIABLE |

Example 5: To transfer two hundred bytes of data from input device to memory

| | | |
|--------|------|-----------|
| | LDX | ZERO |
| CLOOP | TD | INDEV |
| | JEQ | CLOOP |
| | RD | INDEV |
| | STCH | RECORD, X |
| | TIX | B200 |
| | JLT | CLOOP |
| . | | |
| . | | |
| INDEV | BYTE | X 'F5' |
| RECORD | RESB | 200 |
| ZERO | WORD | 0 |
| B200 | WORD | 200 |

Example Programs (SIC/XE)**Example 1: Simple data and character movement operation**

```

                LDA      #5
                STA      ALPHA
                LDA      #90
.
.
.
ALPHA          RESW      1
C1             RESB      1

```

Example 2: Arithmetic operations

```

                LDS      INCR
                LDA      ALPHA
                ADD      S,A
                SUB      #1
                STA      BETA
.....
.....
ALPHA          RESW      1
BETA          RESW      1
INCR          RESW      1

```

Example 3: Looping and Indexing operation

```

                LDT      #11
                LDX      #0           ;X = 0
MOVECH        LDCH     STR1, X       ; LOAD A FROM STR1
                STCH     STR2, X       ; STORE A TO STR2
                TIXR     T
                JLT      MOVECH
.
.
STR1          BYTE     C 'HELLO WORLD'
STR2          RESB     11

```

Assemblers - 1**A Simple Two-Pass Assembler****Main Functions**

- Translate mnemonic operation codes to their machine language equivalents
- Assign machine addresses to symbolic labels used by the programmers
- Depend heavily on the source language it translates and the machine language it produces.
- E.g., the instruction format and addressing modes

Basic Functions of an Assembler

```

5 COPY START 1000 COPY FILE FROM IN
10 FIRST STL RETADR SAVE RETURN ADDRESS
15 CLOOP JSUB RDREC READ INPUT RECORD
20 LDA LENGTH TEST FOR EOF (LENG
25 COMP ZERO
30 JEQ ENDFIL EXIT IF EOF FOUND
35 JSUB WRREC WRITE OUTPUT RECO
40 J CLOOP LOOP
45 ENDFIL LDA EOF INSERT END OF FIL
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

110 .
115 . SUBROUTINE TO READ RECORD INTO BUFFER
120 .
125 RDREC LDX ZERO CLEAR LOOP COUNTY
130 LDA ZERO CLEAR A TO ZERO
135 RLOOP TD INPUT TEST INPUT DEVIC
140 JEQ RLOOP LOOP UNTIL READY
145 RD INPUT READ CHARACTER ID
150 COMP ZERO TEST FOR END OF I
155 JEQ EXIT EXIT LOOP IF EOR
160 STCH BUFFER, X STORE CHARACTER
165 TIX MAXLEN LOOP UNLESS MAX
170 JLT RLOOP HAS BEEN REACH
175 EXIT STX LENGTH SAVE RECORD LENG
180 RSUB RETURN TO CALLER
185 INPUT BYTE X'F1' CODE FOR INPUT D
190 MAXLEN WORD 4096
195 .
    
```



```

195 .
200 . SUBROUTINE TO WRITE RECORD FROM BUFFER
205 .
210 WRREC LDX ZERO CLEAR LOOP COUNT
215 WLOOP TD OUTPUT TEST OUTPUT DEVICE
220 JEQ WLOOP LOOP UNTIL REACHED
225 LDCH BUFFER, X GET CHARACTER FROM
230 WD OUTPUT WRITE CHARACTER TO
235 TIX LENGTH LOOP UNTIL ALL
240 JLT WLOOP HAVE BEEN WRITTEN
245 RSUB RETURN TO CALLER
250 OUTPUT BYTE X'05' CODE FOR OUTPUT
255 END FIRST

```

- It is a copy function that reads some records from a specified input device and then copies them to a specified output device

- Reads a record from the input device (code F1)
- Copies the record to the output device (code 05)
- Repeats the above steps until encountering EOF.
- Then writes EOF to the output device
- Then call RSUB to return to the caller
-

RDREC and WRREC

- Data transfer
 - A record is a stream of bytes with a null character (0016) at the end.
 - If a record is longer than 4096 bytes, only the first 4096 bytes are copied.
 - EOF is indicated by a zero-length record. (I.e., a byte stream with only a null character.
 - Because the speed of the input and output devices may be different, a buffer is used to temporarily store the record
- Subroutine call and return
 - On line 10, “STL RETADDR” is called to save the return address that is already stored in register L.
 - Otherwise, after calling RD or WR, this COPY cannot return back to its caller.

Assembler Directives

- Assembler directives are pseudo instructions
 - They will not be translated into machine instructions.
 - They only provide instruction/direction/information to the assembler.
- Basic assembler directives :
 - START : Specify name and starting address for the program
 - END : Indicate the end of the source program, and (optionally) the first executable instruction in the program. Assembler Directives (cont'd)
 - BYTE : Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.

- WORD : Generate one-word integer constant
- RESB : Reserve the indicated number of bytes for a data area
- RESW : Reserve the indicated number of words for a data area

An Assembler's Job

- Convert mnemonic operation codes to their machine language codes
- Convert symbolic (e.g., jump labels, variable names) operands to their machine addresses
- Use proper addressing modes and formats to build efficient machine instructions
- Translate data constants into internal machine representations
- Output the object program and provide other information (e.g., for linker and loader)

Object Program Format

- Header
 - Col. 1 H
 - Col. 2~7 Program name
 - Col. 8~13 Starting address of object program (hex)
 - Col. 14-19 Length of object program in bytes (hex)
- Text
 - Col.1 T
 - Col.2~7 Starting address for object code in this record (hex)
 - Col. 8~9 Length of object code in this record in bytes (hex)
 - Col. 10~69 Object code, represented in hexa (2 col. per byte)
- End
 - Col.1 E
 - Col.2~7 Address of first executable instruction in object program (hex)

The Object Code for COPY

```
H COPY 001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003
    00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081044 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

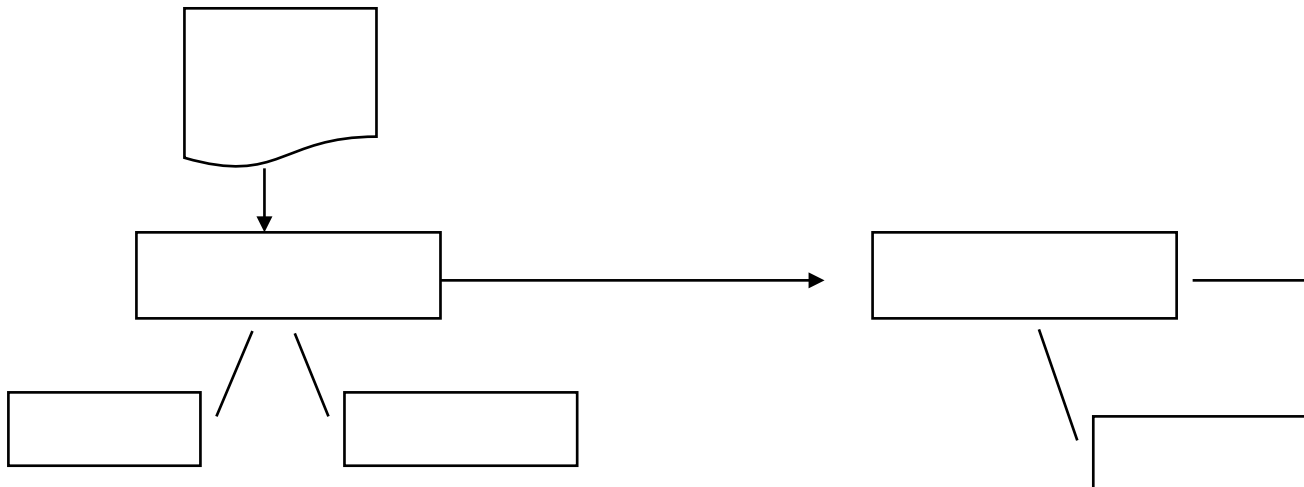
E 001000

NOTE: There is no object code corresponding to addresses 1033-2038. This storage is simply reserved by the loader for use by the program during execution.

Two Pass Assembler

- Pass 1
 - Assign addresses to all statements in the program
 - Save the values (addresses) assigned to labels (including label and variable names) for use in Pass 2 (deal with forward references)
 - Perform some processing of assembler directives (e.g., BYTE, RESW, these can affect address assignment)
- Pass 2
 - Assemble instructions (generate opcode and look up addresses)
 - 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

A Simple Two Pass Assembler Implementation



Algorithms and Data Structures

Three Main Data Structures

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

OPTAB (operation code table)

- Content
 - The mapping between mnemonic and machine code. Also include the instruction format, available addressing modes, and length information.
- Characteristic
 - Static table. The content will never change.
- Implementation

- Array or hash table. Because the content will never change, we can optimize its search speed.
- In pass 1, OPTAB is used to look up and validate mnemonics in the source program.
- In pass 2, OPTAB is used to translate mnemonics to machine instructions.

Location Counter (LOCCTR)

- This variable can help in the assignment of addresses.
- It is initialized to the beginning address specified in the START statement.
- After each source statement is processed, the length of the assembled instruction and data area
- to be generated is added to LOCCTR.
- Thus, when we reach a label in the source program, the current value of LOCCTR gives the address to be associated with that label.

Symbol Table (SYMTAB)

- Content
 - Include the label name and value (address) for each label in the source program.
 - Include type and length information (e.g., int64)
 - With flag to indicate errors (e.g., a symbol defined in two places)
- Characteristic
 - Dynamic table (I.e., symbols may be inserted, deleted, or searched in the table)
- Implementation
 - Hash table can be used to speed up search – Because variable names may be very similar (e.g., LOOP1, LOOP2), the selected hash function must perform well with such non-random keys.

The Pseudo Code for Pass 1

Begin

```

read first input line

if OPCODE = 'START' then begin
    save #[Operand] as starting addr
    initialize LOCCTR to starting address
    write line to intermediate file
    read next 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
                (if symbol)
                search OPTAB for OPCODE
                if found then
                    add 3 (instr 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
                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}

The Pseudo Code for Pass 2

Begin

 read 1st input line

 if OPCODE = 'START' then

 begin

 write listing line

 read next input line

 end

 write Header record to object program

 initialize 1st Text record

 while OPCODE != 'END' do

 begin

 if this is not 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 field then

 if found then

 begin

 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
        else if OPCODE = 'BYTE' or 'WORD' then
            convert constant to object code
        if object code doesn't fit into current Text record then
        begin
            Write text record to object code
            initialize new Text record
        end
        add object code to Text record
    end {if not comment}
    write listing line
    read next input line
end
write listing line
read next input line
write last listing line
End {Pass 2}
```

Machine dependent Assembler Features

Assembler Features

- Machine Dependent Assembler Features
 - Instruction formats and addressing modes (SIC/XE)
 - Program relocation
- Machine Independent Assembler Features
 - Literals
 - Symbol-defining statements
 - Expressions

- Program blocks
- Control sections and program linking

A SIC/XE Program

```

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
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
250      OUTPUT    BYTE      X'05'     CODE FOR OUTPUT DEVICE
255      .          END        FIRST

```

SIC/XE Instruction Formats and Addressing Modes

- PC-relative or Base-relative (BASE directive needs to be used) addressing: **op m**
- Indirect addressing: **op @m**
- Immediate addressing: **op #c**
- Extended format (4 bytes): **+op m**
- Index addressing: **op m,X**
- Register-to-register instructions

Relative Addressing Modes

- PC-relative or base-relative addressing mode is preferred over direct addressing mode.
 - Can save one byte from using format 3 rather than format 4.
 - Reduce program storage space
 - Reduce program instruction fetch time
 - Relocation will be easier.

The Differences Between the SIC and SIC/XE Programs

- Register-to-register instructions are used whenever possible to improve execution speed.
 - Fetch a value stored in a register is much faster than fetch it from the memory.
- Immediate addressing mode is used whenever possible.
 - Operand is already included in the fetched instruction. There is no need to fetch the operand from the memory.
- Indirect addressing mode is used whenever possible.

- Just one instruction rather than two is enough.

The Object Code

| 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 | | . | |
| 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

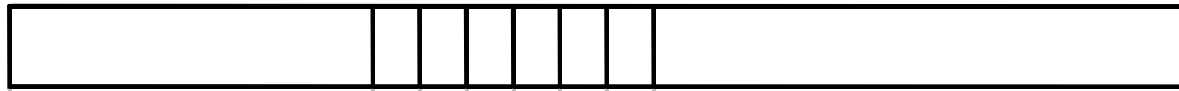
```

Generate Relocatable Programs

- Let the assembled program starts at address 0 so that later it can be easily moved to any place in the physical memory.
 - Actually, as we have learned from virtual memory, now every process (executed program) has a separate address space starting from 0.
- Assembling register-to-register instructions presents no problems. (e.g., line 125 and 150)
 - Register mnemonic names need to be converted to their corresponding register numbers.
 - This can be easily done by looking up a name table.

PC or Base-Relative Modes

- Format 3: 12-bit displacement field (in total 3 bytes)
 - Base-relative: 0~4095
 - PC-relative: -2048~2047
- Format 4: 20-bit address field (in total 4 bytes)
- The displacement needs to be calculated so that when the displacement is added to PC (which points to the following instruction after the current instruction is fetched) or the base register (B), the resulting value is the target address.
- If the displacement cannot fit into 12 bits, format 4 then needs to be used. (E.g., line 15 and 125)
 - Bit e needs to be set to indicate format 4.
 - A programmer must specify the use of format 4 by putting a + before the instruction. Otherwise, it will be treated as an error.

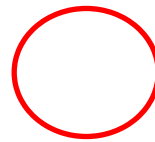


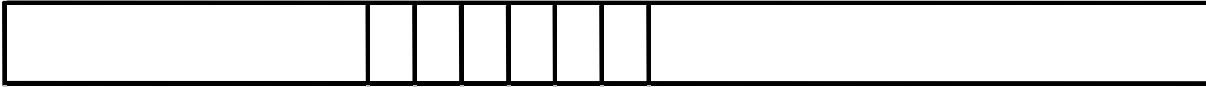
Base-Relative v.s. PC-Relative

- The difference between PC and base relative addressing modes is that the assembler knows the value of PC when it tries to use PC-relative mode to assemble an

instruction. However, when trying to use base-relative mode to assemble an instruction, the assembler does not know the value of the base register.

- Therefore, the programmer must tell the assembler the value of register B.
- This is done through the use of the BASE directive. (line 13)
- Also, the programmer must load the appropriate value into register B by himself.
- Another BASE directive can appear later, this will tell the assembler to change its notion of the current value of B.
- NOBASE can also be used to tell the assembler that no more base-relative addressing mode should be used.



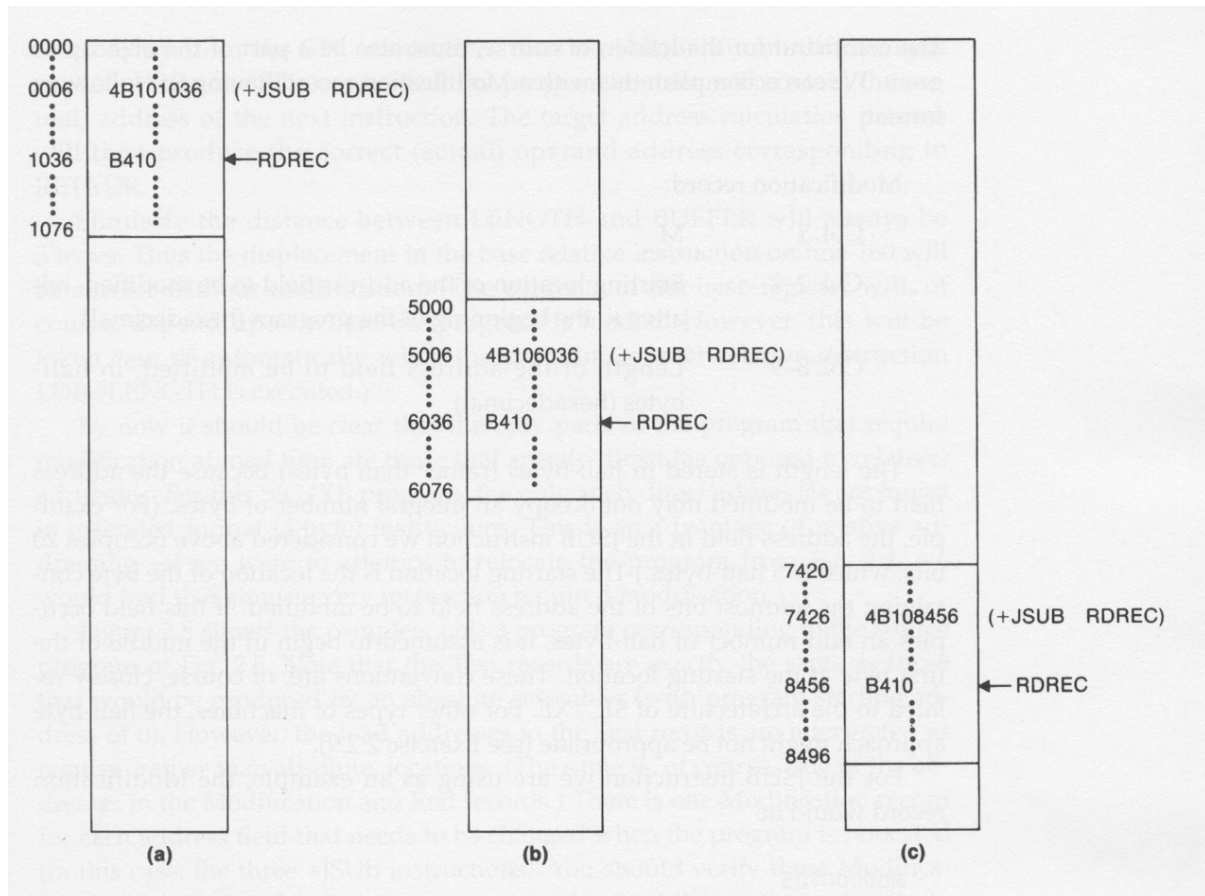


Relocatable Is Desired

- The program in Fig. 2.1 specifies that it must be loaded at address 1000 for correct execution. This restriction is too inflexible for the loader.
- If the program is loaded at a different address, say 2000, its memory references will access wrong data! For example:
 - 55 101B LDA THREE 00102D
- Thus, we want to make programs relocatable so that they can be loaded and execute correctly at any place in the memory.

Address Modification Is Required

If we can use a hardware relocation register (MMU), software relocation can be avoided here. However, when linking multiple object Programs together, software relocation is still needed.



What Instructions Needs to be Modified?

- Only those instructions that use absolute (direct) addresses to reference symbols.
- The following need not be modified:
 - Immediate addressing (no memory references)
 - PC or Base-relative addressing (Relocatable is one advantage of relative addressing, among others.)
 - Register-to-register instructions (no memory references)

The Modification Record

- When the assembler generate an address for a symbol, the address to be inserted into the instruction is relative to the start of the program.
- The assembler also produces a modification record, in which the address and length of the need-to-be-modified address field are stored.
- The loader, when seeing the record, will then add the beginning address of the loaded program to the address field stored in the record.

Modification record:

Col. 1 M

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

Col. 8-9 Length of the address field to be modified in bytes (hexadecimal)

The Relocatable Object Code

```

H^C^O^P^Y  ^0^0^0^0^0^0^1^0^7^7
T^0^0^0^0^0^1^D^1^7^2^0^2^D^6^9^2^0^2^D^4^B^1^0^1^0^3^6^0^3^2^0^2^6^2^9^0^0^0^0^3^3^2^0^0^7^4^B^1^0^1^0^5^D^3^F^2^F^E^C^0^3^2^0^1^0
T^0^0^0^0^1^D^1^3^0^F^2^0^1^6^0^1^0^0^0^3^0^F^2^0^0^D^4^B^1^0^1^0^5^D^3^E^2^0^0^3^4^5^4^F^4^6
T^0^0^1^0^3^6^1^D^B^4^1^0^B^4^0^Q^B^4^4^0^7^5^1^0^1^0^0^0^E^3^2^0^1^9^3^3^2^F^F^A^D^B^2^0^1^3^A^0^0^4^3^3^2^0^0^8^5^7^C^0^0^3^B^8^5^0
T^0^0^1^0^5^3^1^D^3^B^2^F^E^A^1^3^4^0^0^0^4^F^0^0^0^0^F^1^B^4^1^0^7^7^4^0^0^0^E^3^2^0^1^1^3^3^2^F^F^A^5^3^C^0^0^3^D^F^2^0^0^8^B^8^5^0
T^0^0^1^0^7^0^0^7^3^B^2^F^E^F^4^F^0^0^0^0^0^5
M^0^0^0^0^0^7^0^5
M^0^0^0^0^1^4^0^5
M^0^0^0^0^2^7^0^5
E^0^0^0^0^0^0

```