CST 305- SYSTEM SOFTWARE

MODULE 2

Module 2

- (Assembly language programming and Assemblers)
- SIC/XE Programming.
- Basic Functions of Assembler.
- Assembler Output Format – Header, Text and End Records.
- Assembler Data Structures.
- Two Pass Assembler Algorithm.
- Hand Assembly of SIC/XE Programs.

1.3.3 SIC Programming Examples

- **Sample data movement operations**
  - No memory-to-memory move instructions (Fig. 1.2)

```
        LDA     five            LDA     #5
        ...                     ...
five    word    5
```

```
        LDA     FIVE        LOAD CONSTANT 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDCH    CHARZ       LOAD CHARACTER 'Z' INTO REGISTER A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW    1           ONE-WORD VARIABLE
FIVE    WORD    5           ONE-WORD CONSTANT
CHARZ   BYTE    C'Z'        ONE-BYTE CONSTANT
C1      RESB    1           ONE-BYTE VARIABLE
```

1.3.3 SIC Programming Examples

```
        LDA     #5          LOAD VALUE 5 INTO REGISTER A
        STA      ALPHA      STORE IN ALPHA
        LDA     #90         LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH     C1         STORE IN CHARACTER VARIABLE C1
         .
         .
         .
ALPHA   RESW     1          ONE-WORD VARIABLE
C1      RESB     1          ONE-BYTE VARIABLE
```

(b)

Figure 1.2 Sample data movement operations for (a) SIC and (b) SIC/XE.

1.3.3 SIC Programming Examples

- Sample arithmetic operations
  - (ALPHA+INCR-1) assign to BETA (Fig. 1.3)
  - (GAMMA+INCR-1) assign to DELTA

```
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     DELTA       STORE IN DELTA
        .
        .
        .
ONE     WORD    1           ONE-WORD CONSTANT
.                           ONE-WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

1.3.3 SIC Programming Examples

```
        LDS     INCR           LOAD VALUE OF INCR INTO REGISTER S
        LDA     ALPHA          LOAD ALPHA INTO REGISTER A
        ADDR    S,A            ADD THE VALUE OF INCR
        SUB     #1             SUBTRACT 1
        STA     BETA           STORE IN BETA
        LDA     GAMMA          LOAD GAMMA INTO REGISTER A
        ADDR    S,A            ADD THE VALUE OF INCR
        SUB     #1             SUBTRACT 1
        STA     DELTA          STORE IN DELTA
        .
        .
        .
.                              ONE WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

1.3.3 SIC Programming Examples

- **String copy**

```
        LDX     ZERO           INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH    STR1,X         LOAD CHARACTER FROM STR1 INTO REG A
        STCH    STR2,X         STORE CHARACTER INTO STR2
        TIX     ELEVEN         ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT     MOVECH         LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE    C'TEST STRING'    11-BYTE STRING CONSTANT
STR2    RESB    11                11-BYTE VARIABLE
.                                 ONE-WORD CONSTANTS
ZERO    WORD    0
ELEVEN  WORD    11
```

1.3.3 SIC Programming Examples

```
        LDT     #11              INITIALIZE REGISTER T TO 11
        LDX     #0               INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH     STR1,X          LOAD CHARACTER FROM STR1 INTO REG A
        STCH     STR2,X          STORE CHARACTER INTO STR2
        TIXR     T               ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT      MOVECH          LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE     C'TEST STRING'   11-BYTE STRING CONSTANT
STR2    RESB     11               11-BYTE VARIABLE
```

1.3.3 SIC Programming Examples

```
        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
```

1.3.3 SIC Programming Examples

```
        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.

1.3.3 SIC Programming Examples

```
INLOOP   TD     INDEV      TEST INPUT DEVICE
         JEQ    INLOOP     LOOP UNTIL DEVICE IS READY
         RD     INDEV      READ ONE BYTE INTO REGISTER A
         STCH   DATA       STORE BYTE THAT WAS READ
         .
         .
         .
OUTLP    TD     OUTDEV     TEST OUTPUT DEVICE
         JEQ    OUTLP      LOOP UNTIL DEVICE IS READY
         LDCH   DATA       LOAD DATA BYTE INTO REGISTER A
         WD     OUTDEV     WRITE ONE BYTE TO OUTPUT DEVICE
         .
         .
         .
INDEV    BYTE   X'F1'      INPUT DEVICE NUMBER
OUTDEV   BYTE   X'05'      OUTPUT DEVICE NUMBER
DATA     RESB   1          ONE-BYTE VARIABLE
```

Figure 1.6 Sample input and output operations for SIC.

1.3.3 SIC Programming Examples

```
          JSUB    READ          CALL READ SUBROUTINE
          .
          .
          .
.                               SUBROUTINE TO READ 100-BYTE RECORD
READ      LDX     ZERO          INITIALIZE INDEX REGISTER TO 0
RLOOP     TD      INDEV         TEST INPUT DEVICE
          JEQ     RLOOP         LOOP IF DEVICE IS BUSY
          RD      INDEV         READ ONE BYTE INTO REGISTER A
          STCH    RECORD,X      STORE DATA BYTE INTO RECORD
          TIX     K100          ADD 1 TO INDEX AND COMPARE TO 100
          JLT     RLOOP         LOOP IF INDEX IS LESS THAN 100
          RSUB                  EXIT FROM SUBROUTINE
          .
          .
          .
INDEV     BYTE    X'F1'         INPUT DEVICE NUMBER
RECORD    RESB    100           100-BYTE BUFFER FOR INPUT RECORD
.                               ONE-WORD CONSTANTS
ZERO      WORD    0
K100      WORD    100
```

37

1.3.3 SIC Programming Examples

```
         JSUB    READ          CALL READ SUBROUTINE
         .
         .
         .
.                              SUBROUTINE TO READ 100-BYTE RECORD
READ     LDX     #0            INITIALIZE INDEX REGISTER TO 0
         LDT     #100          INITIALIZE REGISTER T TO 100
RLOOP    TD      INDEV         TEST INPUT DEVICE
         JEQ     RLOOP         LOOP IF DEVICE IS BUSY
         RD      INDEV         READ ONE BYTE INTO REGISTER A
         STCH    RECORD,X      STORE DATA BYTE INTO RECORD
         TIXR    T             ADD 1 TO INDEX AND COMPARE TO 100
         JLT     RLOOP         LOOP IF INDEX IS LESS THAN 100
         RSUB                  EXIT FROM SUBROUTINE
         .
         .
         .
INDEV    BYTE    X'F1'         INPUT DEVICE NUMBER
RECORD   RESB    100           100-BYTE BUFFER FOR INPUT RECORD
```

BASIC ASSEMBLER FUNCTIONS

Fundamental functions of an assembler:

- Translating mnemonic operation codes to their machine language equivalents.
- Assigning machine addresses to symbolic labels used by the programmer.

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.

- Indexed addressing is indicated by adding the modifier " X" following the operand.
- Lines beginning with "." contain comments only.
- The following assembler directives are used:
- **START:** Specify name and starting address for the program.
- **END :** Indicate the end of the source program and specify the first executable instruction in the program.

- **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.

- The program contains a main routine that reads records from an input device(code F1) and copies them to an output device(code 05).

The main routine calls subroutines:

- RDREC – To read a record into a buffer.
- WRREC – To write the record from the buffer to the output device.
- The end of each record is marked with a null character (hexadecimal 00).

A Simple SIC Assembler

The translation of source program to object code requires the following functions:

1. Convert mnemonic operation codes to their machine language equivalents. Eg: Translate STL to 14 (line 10).
2. Convert symbolic operands to their equivalent machine addresses. Eg: Translate RETADR to 1033 (line 10).
3. Build the machine instructions in the proper format.
4. Convert the data constants specified in the source program into their internal machine representations. Eg: Translate EOF to 454F46(line 80).
5. Write the object program and the assembly listing.

Consider the statement

10 1000 FIRST STL RETADR 141033

- This instruction contains a forward reference (i.e.) a reference to a label (RETADR) that is defined later in the program.
- It is unable to process this line because the address that will be assigned to RETADR is not known.
- Hence most assemblers make two passes over the source program where the second pass does the actual translation.
- The assembler must also process statements called assembler directives or pseudo instructions which are not translated into machine instructions.

- Instead they provide instructions to the assembler itself. Examples: RESB and RESW instruct the assembler to reserve memory locations without generating data values.

- The assembler must write the generated object code onto some output device.

- This object program will later be loaded into memory for execution.

Object program format contains three types of records:

- **Header record:** Contains the program name, starting address and length.
- **Text record:** Contains the machine code and data of the program.
- **End record:** Marks the end of the object program and specifies the address in the program where execution is to begin.

Record format is as follows:

Header record:

Col. 1	H
Col.2-7	Program name
Col.8-13	Starting address of object program
Col.14-19	Length of object program in bytes

Text record:

Col.1	T
Col.2-7	Starting address for object code in this record
Col.8-9	Length of object code in this record in bytes
Col 10-69	Object code, represented in hexadecimal (2 columns per byte of object code)

End record:

Col.1	E
Col.2-7	Address of first executable instruction in object program.

```
HCOPY  ^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
```

Figure 2.3 Object program corresponding to Fig. 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	

```
                .       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
```

Figure 2.2 Program from Fig. 2.1 with object code.

Functions of the two passes of assembler

Pass 1 (Define symbols)

1. Assign addresses to all statements in the program.
2. Save the addresses assigned to all labels for use in Pass 2.
3. Perform some processing of assembler directives.

Pass 2 (Assemble instructions and generate object programs)

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 in Pass 1.
4. Write the object program and the assembly listing.

Assembler Algorithm and Data Structures

- Assembler uses two major internal data structures:

Operation Code Table (OPTAB) :

- Used to lookup mnemonic operation codes and translate them into their machine language equivalents.

Symbol Table (SYMTAB) :

- Used to store values(Addresses) assigned to labels

Location Counter (LOCCTR) :

- Variable used to 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 or data area is added to LOCCTR.
- Whenever a label is reached in the source program, the current value of LOCCTR gives the address to be associated with that label.

Operation Code Table (OPTAB)

- Contains the mnemonic operation and its machine language equivalent.
- Also contains information about **instruction format and length.**
- In Pass 1, OPTAB is used to **lookup and validate operation codes** in the source program.
- In Pass 2, it is used to **translate the operation codes to machine language program.**
- During Pass 2, the information in OPTAB tells which instruction format to use in assembling the instruction and any peculiarities of the object code instruction.

- OPTAB is usually organized as a hash table with mnemonic operation code as the key.
- The information in the OPTAB is predefined when the assembler itself is written, rather than being loaded into the table at execution time.
- Hash table organization is appropriate , since it provides fast retrieval with a minimum search.
- OPTAB is a static table , entries are not normally added to or deleted from it.

Symbol Table (SYMTAB)

- Includes the **name and value(address) for each label** in the source program and flags to indicate error conditions.
- **Error** means symbol defined in 2 different places.
- This table also contains other information about the **data area or instruction labeled(its type or length).**

- **During Pass 1** of the assembler, labels are entered into SYMTAB as they are encountered in the source program along with their assigned addresses.
- **During Pass 2**, symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.
- It is usually organized as a **hash table** for efficiency of insertion and retrieval .
- Since entries are rarely deleted from this table , efficiency of deletion is not an important consideration.

- Pass 1 usually writes an intermediate file that contains each source statement together with its assigned address, error indicators.

- This file is used as the input to Pass 2.

- This copy of the source program can also be used to retain the results of certain operations that may be performed during Pass 1 such as scanning the operand field for symbols and addressing flags, so these need not be performed again during Pass 2

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.

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	.			

```
.           SUBROUTINE TO READ RECORD INTO BUFFER
.
RDREC       CLEAR    X                 CLEAR LOOP COUNTER
            CLEAR    A                 CLEAR A TO ZERO
            CLEAR    S                 CLEAR S TO ZERO
           +LDT      #4096
RLOOP       TD       INPUT             TEST INPUT DEVICE
            JEQ      RLOOP             LOOP UNTIL READY
            RD       INPUT             READ CHARACTER INTO REGISTER A
            COMPR    A,S               TEST FOR END OF RECORD (X'00')
            JEQ      EXIT              EXIT LOOP IF EOR
            STCH     BUFFER,X          STORE CHARACTER IN BUFFER
            TIXR     T                 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
.
```

```
.         SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC     CLEAR     X                  CLEAR LOOP COUNTER
          LDT       LENGTH
WLOOP     TD        OUTPUT             TEST OUTPUT DEVICE
          JEQ       WLOOP              LOOP UNTIL READY
          LDCH      BUFFER,X           GET CHARACTER FROM BUFFER
          WD        OUTPUT             WRITE CHARACTER
          TIXR      T                  LOOP UNTIL ALL CHARACTERS
          JLT       WLOOP                HAVE BEEN WRITTEN
          RSUB                         RETURN TO CALLER
OUTPUT    BYTE      X'05'              CODE FOR OUTPUT DEVICE
          END       FIRST
```

Figure 2.5 Example of a SIC/XE program.

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.

```
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
```

Figure 2.8 Object program corresponding to Fig. 2.6.

University questions….

1. Explain the syntax of the records in the Object Program File.
2. Let NUMBERS be an array of 100 words. Write a sequence of instructions for SIC to set all I 00 elements of the array to I.
3. Describe the format of the object program generated by the two-pass SIC assembler algorithm .
4. Write a sequence of instructions for SIC/XE to divide BETA by GAMMA and to store the integer quotient in ALPHA and remainder in DELTA.

5. Let A,B & C are arrays of 10 words each. Write a SIC/XE program to add the corresponding elements of A & B and store the result in C
6. Write a subroutine for SIC/XE that will read a record into a buffer. The record may be any length from 1 to 100 bytes. The end of the record is marked with a “null” character (ASCII code 00). The subroutine should place the length of the record read into a variable named LENGTH. Use immediate addressing and register-to register instructions to make the process as efficient as possible.

7. Write a sequence of instructions for SIC to set ALPHA = BETA*9+GAMMA.
8. Explain the different data structures used in the implementation of Assemblers.
9. Explain the two passes of the assembler algorithm with a proper example.
10. List out the basic functions of Assemblers with proper examples
11. What is meant by forward reference? How is it resolved by two pass assemblers?

12. Write down the format of the Modification record. Describe each field with the help of an example.
13. With the aid of an algorithm explain the Second pass of a Two Pass Assembler.
14. Describe the data structures used in the two pass SIC assembler algorithm.
15. Give the algorithm for pass 1 of a two pass SIC assembler.
16. Describe the format of the object program generated by the two-pass SIC assembler algorithm .