CSN 252 SIC – XE

ASSEMBLER

CSN 252 – SYSTEM SOFTWARE

ABHISHEK RAJ
21114004

CSN: 252 - SYSTEM SOFTWARE

- **Memory**: ***The memory in SIC/XE architecture consists of 1 megabyte, equivalent to 220 bytes, each consisting of 8 bits. This is a significant increase from the standard SIC memory size, which was much smaller. With this increase in memory size, the instruction formats and addressing modes have also been modified to accommodate the larger capacity.***

In SIC/XE architecture, a word comprises three consecutive bytes, making up 24 bits. All addresses in this architecture are byte addresses, and terms are accessed by specifying the location of their lowest-numbered byte. This means that comments are addressed through their starting byte address.

- **Register:**

  The SIC/XE architecture has a set of 9 registers, consisting of 5 standard SIC registers and four additional registers. The four additional registers are:

B: The base register stores a base address for indexed addressing. The index register often uses it to access memory locations relative to the base address.

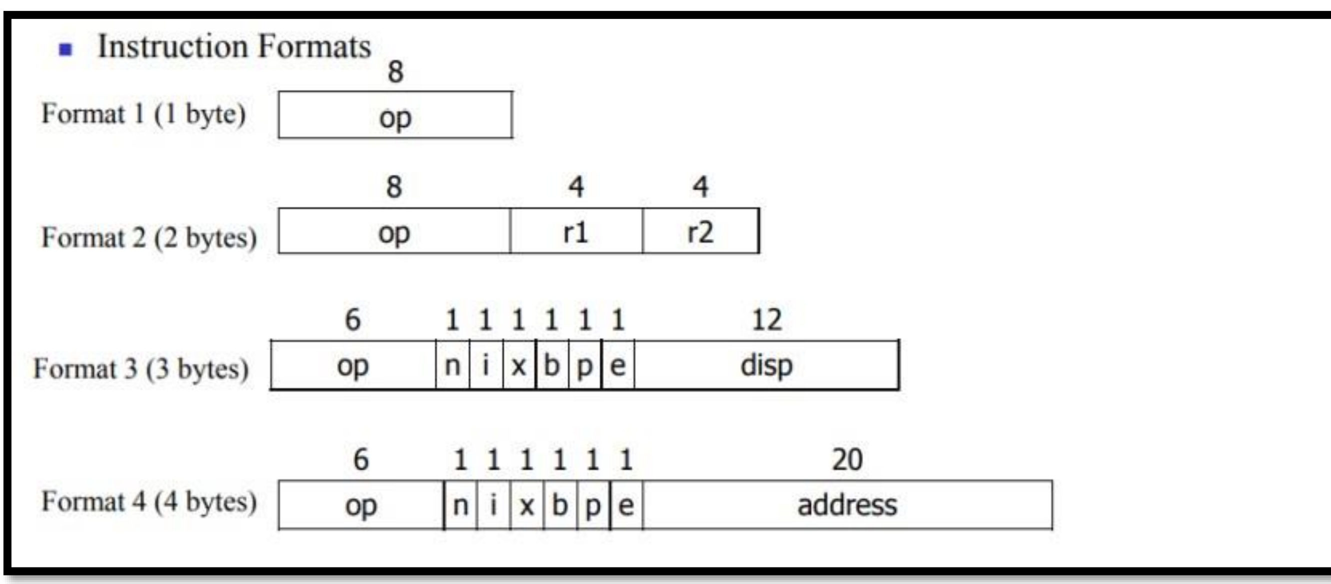
S: The S register is a general-purpose working register that can be used for the temporary storage of data during program execution. It is often used for arithmetic and logical operations.

T: The T register is also a general-purpose working register that temporarily stores data during program execution. It is similar to the S register in its functionality.

F: The F register is specialized for floating-point arithmetic operations. It accumulates floating-point values and uses other floating-point registers to perform arithmetic operations.

- **Data Formats**: Integers are typically represented in binary form using a fixed number of bits. Characters are commonly represented using ASCII codes, which assign a unique numerical value to each character. ASCII codes use 7 bits to represent each character, allowing a maximum of 128 unique characters to be defined. Floating-point numbers are typically represented using a fixed number of bits, with the number of bits used to describe the number determining the precision and range of values that can be defined. In SIC/XE architecture, floating-point numbers are represented using 48 bits, which consists of a sign bit, a 39-bit mantissa, and an 8-bit exponent.

- **Instruction Formats**: In SIC/XE architecture, four formats are available. The Bit(e) is used to distinguish between Formats 3 and 4. e = 0 means Format 3, and e = 1 means Format 4.



Addressing modes

- Base relative (n=1, i=1, b=1, p=0)
- Program-counter relative (n=1, i=1, b=0, p=1)
- Direct (n=1, i=1, b=0, p=0)
- Immediate (n=0, i=1, x=0)
- Indirect (n=1, i=0, x=0)
- Indexing (both n & i = 0 or 1, x=1)
- Extended (e=1 for format 4, e=0 for format 3)

- **This Assembler also includes Machine – Independent Assembler Features.**
    1. **Literals**
    2. **Symbol Defining Statements**
    3. **Expressions**
    4. **Program Blocks**

- **The Assembler is implemented our assembler using C++ programming language.**

- **The Assembler uses the C++ library fstream to read input from a file and write output on another.**

- **The Assembler includes a shell script to compile the .cpp files.**

Assembler

- **Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.**

- **It generates instructions by evaluating the mnemonics in the operation field and finding the value of symbols and literals to produce machine code.**

- **Pass 1:**
    1. **Define symbols and literals and remember them in the symbol and literal tables, respectively.**
    2. **Keep track of the location counter.**
    3. **Process pseudo–operations.**

- **Pass 2:**
    1. **Generate object code by converting symbolic op-code into respective numeric op-code.**
    2. **Generate data for literals and look for values of symbols**.

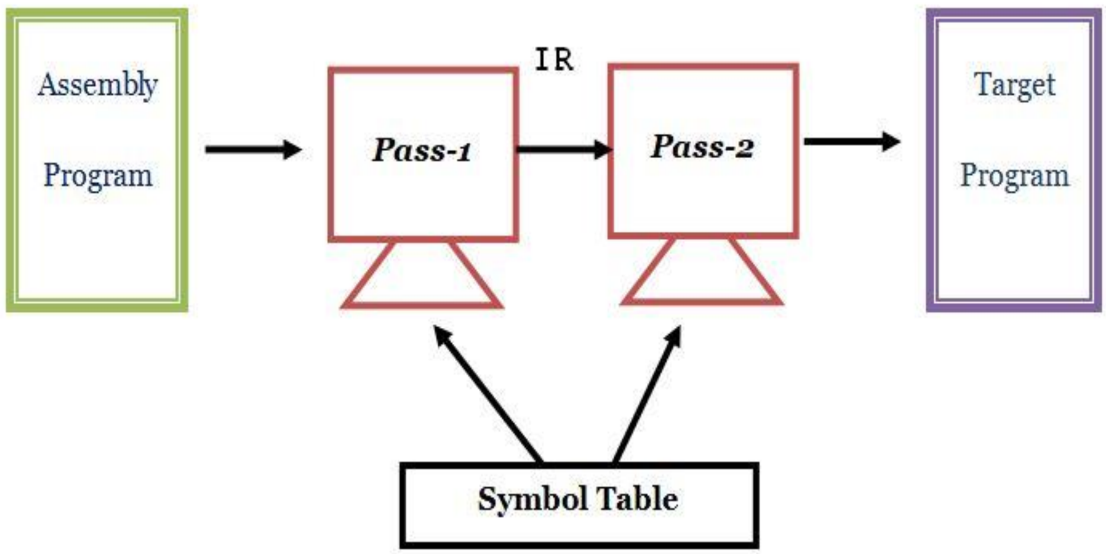


Fig: - Whole Assembler work as.

Assembler: Design

- **Tables.CPP:** **It contains all the data structures required for our assembler to run. It includes the structs for labels, opcode, literal, and blocks. Maps are defined for various tables with their indices as strings with the names of labels or opcodes as required.**

- **Utility.CPP:** **It contains valuable functions that other files will require.**

1. **Convert int to string hexadecimal (): converts the input number into a string which is the hexadecimal representation of that number.**

2. **Convert string hex to int (): Converts the hexadecimal string to an integer and returns the integer value of that hexadecimal number.**
3. **Convert_string_to_hexadecimal_string(): Takes in the string as input, converts the string into its hexadecimal equivalent, and then returns the equivalent as a string.**
4. **expandString(): To transform the input string to the given size. It takes a string to be expanded as a parameter and the length of the output string and character to be inserted to expand the string.**
5. **If_all_num(): Checks if the given input string only has all the element's digits!**
6. **REAL_OPCODE(): for opcode of format 4, it returns the opcode leaving the first flag bit.**
7. **FLAG_FORMAT(): If there is a flag bit in the string, return true otherwise, return a null string.**
8. **Check_comment_line(): Checks the comment by looking at the first character of the input string and then accordingly returns true if a comment or else false**
9. **read_first_non_white_space(): Iterates until it gets the first non-space character in the input string.**
10. **check_white_space(): if blanks are present, return true else, false ;**
11. **Class AbhiEvaluateString contains the functions peek(), get() and number();**

- **pass1.cpp:**

pass1.cpp is the first pass of the assembler, where it reads the input source file and generates an intermediate file along with an error file. It checks each source file line to see if it is a comment line. If it is a comment, the assembler writes it to the intermediate file and updates the line number. If not a comment, the

assembler checks if it is a START opcode and updates the start address and LOCCTR accordingly.

After checking the symbol and opcode, the assembler updates the data to be written to the intermediate file. It then proceeds with a loop that continues until the end of the source file is reached. For each line, the assembler checks if it is a comment line. If it is, it is printed to the intermediate file, and the line number is updated. If it is not a comment line, the assembler verifies the opcode and updates the LOCCTR accordingly.
For opcodes like USE, the assembler inserts a new BLOCK entry in the BLOCK map. For LTORG, a function in pass1.cpp is called to print the literal pool present till that time. For ORG, the assembler points out the LOCCTR to the operand value given, and for EQU, it checks whether the operand is an expression, and if it is valid, it enters the symbols in the SYMTAB. If the opcode does not match with any of the above-given opcodes, the assembler prints an error message in the error file.
Once the loop ends, the assembler stores the program length and prints the SYMTAB, LTTAB, and other tables if present.

Accordingly, we then update our data to be written in the intermediate file. After the loop ends, we store the program length and then go on to print the SYMTAB, LTTAB, and other tables if present.

The to_handle_the_LTORG() function is called by the assembler when it encounters the LTORG opcode in the input file during Pass 1. This function takes an argument, which refers to the literal pool present till the current line is processed in Pass 1.
Using an iterator, the function first prints all the literals in the LITTAB (literal table) until that point in the input file. Then it updates the line number to the following line in the intermediate file. Next, the function checks if any literals do not have an address assigned yet. For such literals, the present value of LOCCTR is stored in the LITTAB, and LOCCTR is incremented based on the literal size. This function is used to handle the LTORG directive, which indicates the beginning of a new literal pool

in the program. The LTORG directive causes all the literals present till that point in the input file to be assigned an address in memory. It also indicates the beginning of a new literal pool that may be present in the program.

Manipulate_EXPRESSION(): A while loop retrieves the symbols from the expression in this function. If the symbol is not found in the SYMTAB (Symbol Table), the assembler writes an error message in the error file. There is also a variable called paircount that is used to keep track of whether the expression is absolute or relative. If paircount gives an unexpected value, the assembler will print an error message. This function is used to check whether an expression is valid. It's likely used in the EQU opcode to validate the expression used in the operand.

- **pass2.cpp:**

  In pass2.cpp, the assembler takes the intermediate file generated by pass1 as input and develops the final object program. If the assembler is unable to open the file, it will print an error message in the error file. The first line of the intermediate file is read, and until the lines are comments, they are taken as input and printed to the intermediate file, and the line number is updated. If the opcode is "START," the start address is initialized as the LOCCTR, and the line is written to the listing file. Then, the assembler checks whether the number of sections in the intermediate file is greater than one. If it is, the program length is updated as the length of the first control section; otherwise, it is kept unchanged. For instructions with immediate addressing, the assembler writes the modification record. For writing the end record, there is a function in pass2.cpp that takes in the final address of the program as an argument and writes the end record to the object program.

function_for_reading_till_TAB(): takes in the string as input and reads the string until tab(‘\t’) occurs.

function_for_reading_the_intermediate_file(): **Takes in location counter, opcode, operand, and label. If the line is a comment returns true and takes in the following input line.**

Assembler: Data Structures

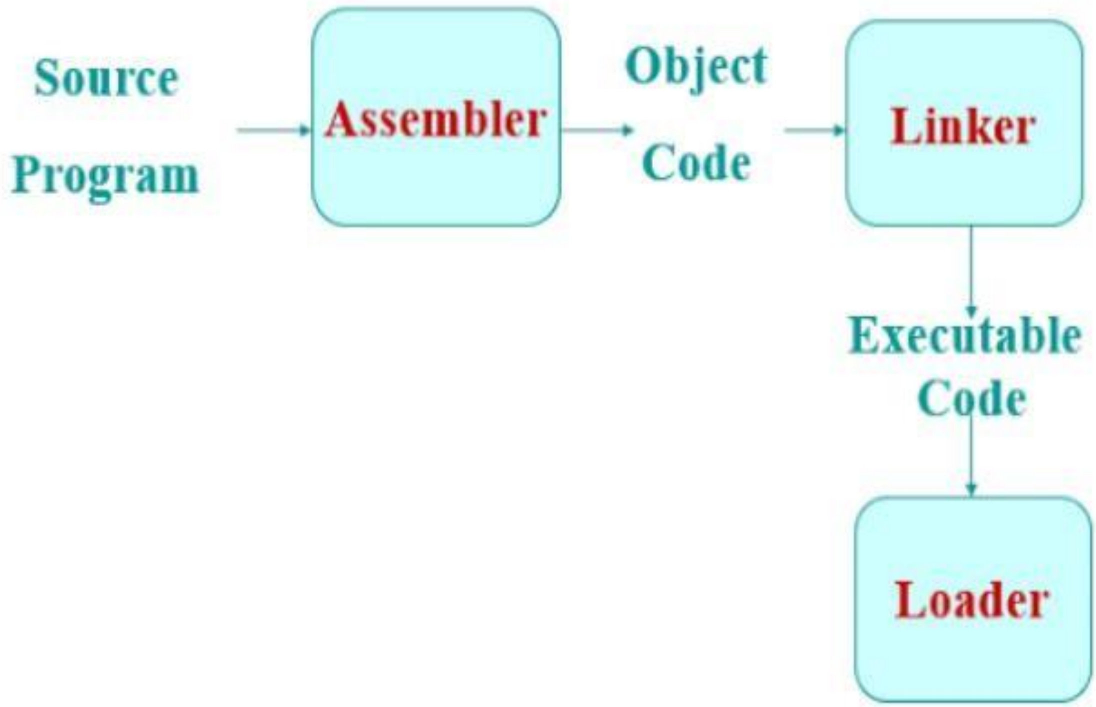
BLOCKS: **Contains information about blocks (name, start address, block number, location counter, character representing whether the block exists).**

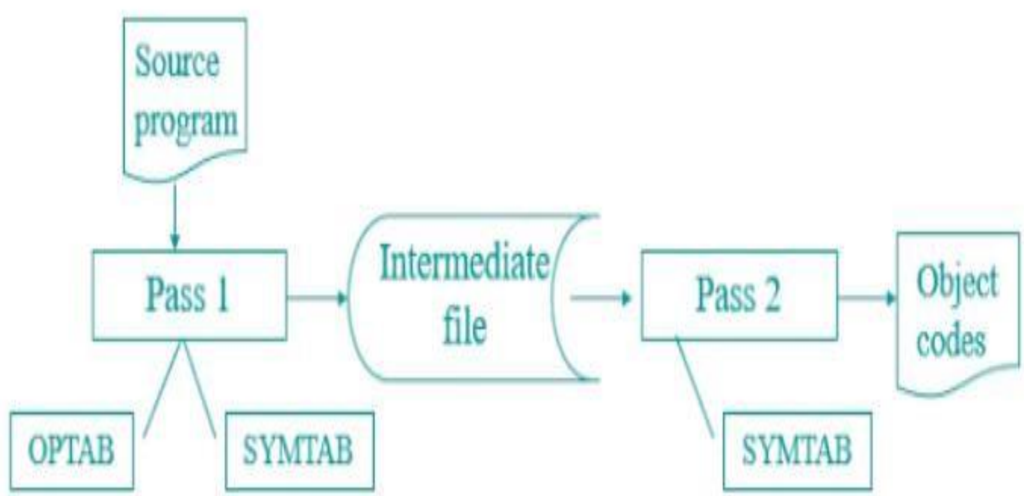
REGTAB: **Contains information on the registers like its numeric equivalent, a character representing whether such register exists.**

LITTAB: **Contains information about literals like their value, address, block number, and a character representing whether the literal exists in the literal table or not.**

SYMTAB: **Contains information on labels like name, address, block number, a character representing whether the label exists in the symbol table, and an integer representing whether the label is relative or not.**

OPTAB: **Contains information about the opcode, like name, format, and a character representing whether the opcode is valid or not.**

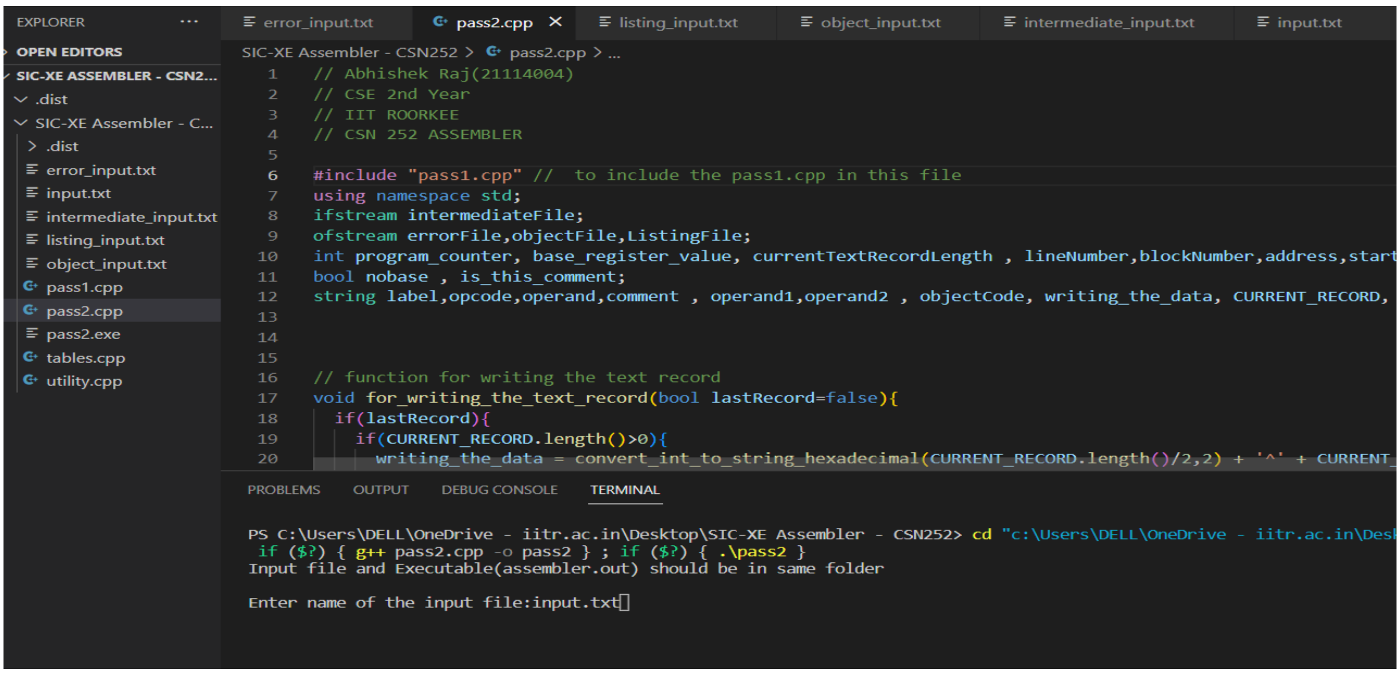

Source
Program
Assembler
Object
Code
Linker
Executable
Code
Loader


Source
program
Pass 1
Intermediate
file
Pass 2
Object
codes
OPTAB
SYMTAB
SYMTAB

Steps to COMPILE and EXECUTE the ASSEMBLER

Step 1: **Open the terminal and the folder where you have stored all the assembler files.**

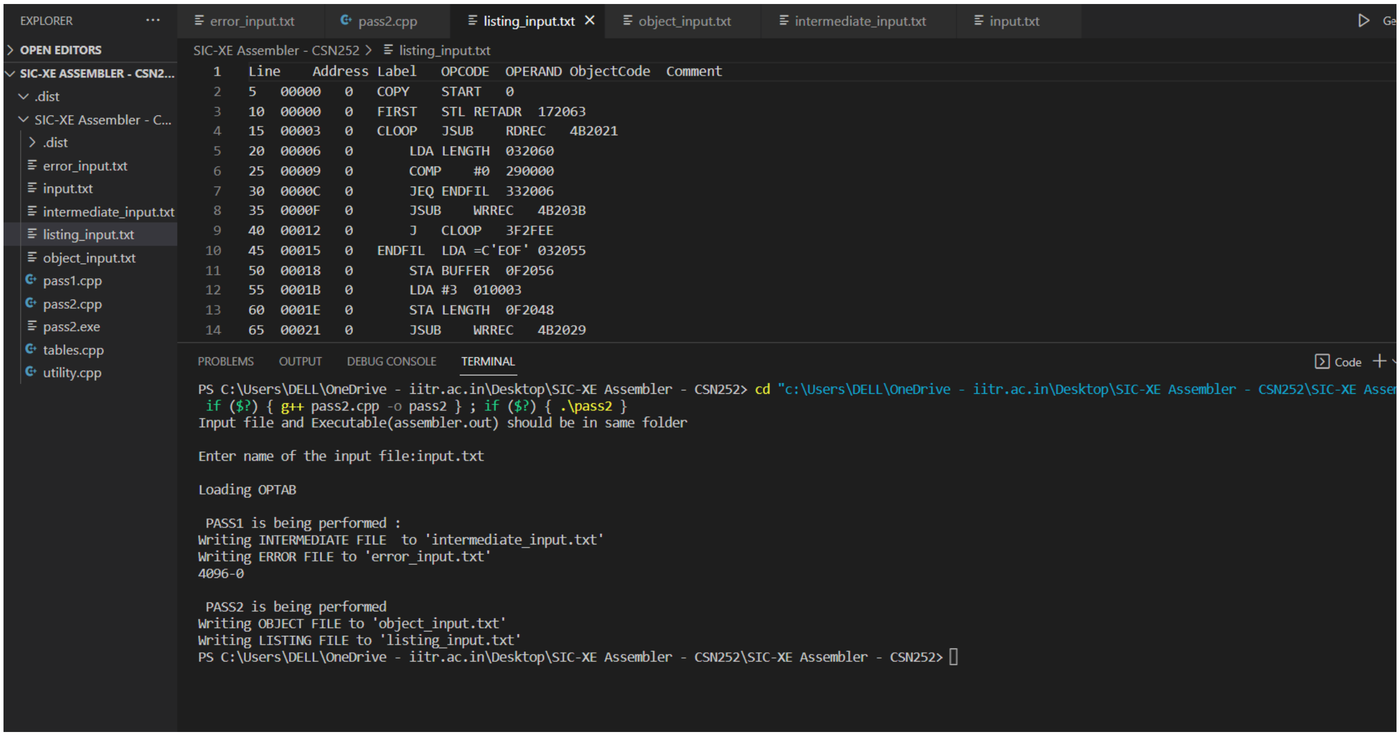
Step2: **Run the command:**



Step 3: After Step 2,Copy the file and paste it into the **test_inputs** folder.

Pasted the file in test_inputs.

Step4: Now open the folder test_inputs in the terminal and run a command :



Step 5: Copy the program you want to run on the assembler and paste in into the file **input.txt** in folder **test_inputs**.

input.txt

File Edit View

```
COPY        START       0
FIRST       STL         RETADR
CLOOP       JSUB       RDREC
            LDA         LENGTH
            COMP        #0
            JEQ         ENDFIL
                JSUB        WRREC
            J           CLOOP
ENDFIL      LDA         =C'EOF'
            STA         BUFFER
            LDA         #3
            STA         LENGTH
            JSUB       WRREC
            J           @RETADR
                USE CDATA
RETADR      RESW        1
LENGTH      RESW        1
                USE CBLKS
BUFFER      RESB        4096
BUFFEND     EQU         *
MAXLEN      EQU         BUFFEND-BUFFER
.
.        SUBROUTINE TO READ RECORD INFO BUFFER
.
              USE
RDREC       CLEAR       X
            CLEAR       A
            CLEAR       S
            +LDT        #MAXLEN
RLOOP       TD          INPUT
            JEQ         RLOOP
            RD          INPUT
            COMPR       A,S
            JEQ         EXIT
            STCH        BUFFER,X
            TIXR        T
            JLT         RLOOP
EXIT        STX         LENGTH
            RSUB
              USE CDATA
INPUT       BYTE        X'F1'
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
              USE
WRREC       CLEAR       X
            LDT         LENGTH
WLOOP       TD          =X'05'
            JEQ         WLOOP
            LDCH        BUFFER,X
            WD          =X'05'
            TIXR        T
            JLT         WLOOP
            RSUB
              USE CDATA
              LTORG
            END         FIRST
```

Step 6: Type **input.txt** in the terminal in front of : “Enter the input file name” and press ENTER.

Step7 : Now all the data has been written in the designated files in the folder **test_inputs .**

object_input.txt

SIC-XE Assembler - CSN252 > object_input.txt

```
H^COPY  ^000000^001071
T^000000^1E^1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
T^00001E^09^0F20484B20293E203F
T^000027^1D^B410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T^000044^09^3B2FEA13201F4F0000
T^00006C^01^F1
T^00004D^19^B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T^00006D^04^454F4605
E^000000
```

listing_input.asm

SIC-XE Assembler - CSN252 > ≡ listing_input.txt

```
Line    Address Label   OPCODE  OPERAND ObjectCode  Comment
5   00000   0   COPY    START   0
10  00000   0   FIRST   STL RETADR  172063
15  00003   0   CLOOP   JSUB    RDREC   4B2021
20  00006   0       LDA LENGTH  032060
25  00009   0       COMP    #0  290000
30  0000C   0       JEQ ENDFIL  332006
35  0000F   0       JSUB    WRREC   4B203B
40  00012   0       J   CLOOP   3F2FEE
45  00015   0   ENDFIL  LDA =C'EOF' 032055
50  00018   0       STA BUFFER  0F2056
55  0001B   0       LDA #3  010003
60  0001E   0       STA LENGTH  0F2048
65  00021   0       JSUB    WRREC   4B2029
70  00024   0       J   @RETADR 3E203F
75  00000   1       USE CDATA
80  00000   1   RETADR  RESW    1
85  00003   1   LENGTH  RESW    1
90  00000   2       USE CBLKS
95  00000   2   BUFFER  RESB    4096
100 01000   2   BUFFEND EQU *
105 01000       MAXLEN  EQU BUFFEND-BUFFER
110 .
115 .       SUBROUTINE TO READ RECORD INFO BUFFER
120 .
125 00027   0       USE
130 00027   0   RDREC   CLEAR   X   B410
135 00029   0       CLEAR   A   B400
140 0002B   0       CLEAR   S   B440
145 0002D   0       +LDT    #MAXLEN 75101000
150 00031   0   RLOOP   TD  INPUT   E32038
155 00034   0       JEQ RLOOP   332FFA
160 00037   0       RD  INPUT   DB2032
165 0003A   0       COMPR   A,S A004
170 0003C   0       JEQ EXIT    332008
175 0003F   0       STCH    BUFFER,X    57A02F
180 00042   0       TIXR    T   B850
185 00044   0       JLT RLOOP   3B2FEA
190 00047   0   EXIT    STX LENGTH  13201F
195 0004A   0       RSUB        4F0000
200 00006   1       USE CDATA
205 00006   1   INPUT   BYTE    X'F1'   F1
210 .
215 .       SUBROUTINE TO WRITE RECORD FROM BUFFER
220 .
225 0004D   0       USE
230 0004D   0   WRREC   CLEAR   X   B410
235 0004F   0       LDT LENGTH  772017
240 00052   0   WLOOP   TD  =X'05'  E3201B
245 00055   0       JEQ WLOOP   332FFA
250 00058   0       LDCH    BUFFER,X    53A016
255 0005B   0       WD  =X'05'  DF2012
260 0005E   0       TIXR    T   B850
265 00060   0       JLT WLOOP   3B2FEF
270 00063   0       RSUB        4F0000
275 00007   1       USE CDATA
280 00007   1       LTORG
285 00007   1   *   =C'EOF'     454F46
290 0000A   1   *   =X'05'      05
295 00066           END FIRST
```

Testing on sample INPUTS

Program 1 : Sample program from book L L Beck section (2.2)

```
COPY        START       0
FIRST       STL         RETADR
            LDB         #LENGTH
            BASE        LENGTH
CLOOP       +JSUB       RDREC
            LDA         LENGTH
            COMP        #0
            JEQ         ENDFIL
          +JSUB       WRREC
            J           CLOOP
ENDFIL            LDA          EOF
            STA         BUFFER
            LDA         #3
            STA         LENGTH
            +JSUB       WRREC
            J           @RETADR
EOF       BYTE      C'EOF'
RETADR      RESW        1
LENGTH      RESW        1
BUFFER      RESB        4096
.
.         SUBROUTINE TO READ RECORD INFO BUFFER
.
RDREC       CLEAR       X
            CLEAR       A
            CLEAR       S
            +LDT        #4096
RLOOP       TD          INPUT
            JEQ         RLOOP
            RD          INPUT
            COMPR       A,S
            JEQ         EXIT
            STCH        BUFFER,X
            TIXR        T
            JLT         RLOOP
EXIT        STX         LENGTH
            RSUB
INPUT       BYTE        X'F1'
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.|
WRREC       CLEAR       X
            LDT         LENGTH
WLOOP       TD          OUTPUT
            JEQ         WLOOP
            LDCH        BUFFER,X
            WD          OUTPUT
            TIXR        T
            JLT         WLOOP
            RSUB
OUTPUT      BYTE            X'05'
            END         FIRST
```

Object Code for Program 1 :

object_input.asm

```
H^COPY   ^000000^001077
T^000000^1D^17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
T^00001D^13^0F20160100030F200D4B10105D3E2003453046
T^001036^1D^B410B400B44075101000E32019332FFADB2013A00433200857C003B850
T^001053^1D^3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T^001070^07^3B2FEF4F000005
M^000007^05
M^000014^05
M^000027^05
E^000000
```

Program 2 : L L Beck program from fig 2.9 including Literals and Expressions

```
COPY        START       0
FIRST       STL         RETADR
            LDB         #LENGTH
            BASE        LENGTH
CLOOP       +JSUB       RDREC
            LDA         LENGTH
            COMP        #0
            JEQ         ENDFIL
             +JSUB       WRREC
            J           CLOOP
ENDFIL      LDA         =C'EOF'
            STA         BUFFER
            LDA         #3
            STA         LENGTH
            +JSUB       WRREC
            J           @RETADR
            LTORG
RETADR      RESW        1
LENGTH      RESW        1
BUFFER      RESB        4096
BUFFEND     EQU         *
MAXLEN      EQU         BUFFEND-BUFFER
.
.         SUBROUTINE TO READ RECORD INFO BUFFER
.
RDREC       CLEAR       X
            CLEAR       A
            CLEAR       S
            +LDT        #MAXLEN
RLOOP       TD          INPUT
            JEQ         RLOOP
            RD          INPUT
            COMPR       A,S
            JEQ         EXIT
            STCH        BUFFER,X
            TIXR        T
            JLT         RLOOP
EXIT        STX         LENGTH
            RSUB
INPUT       BYTE        X'F1'
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC       CLEAR       X
            LDT         LENGTH
WLOOP       TD          =X'05'
            JEQ         WLOOP
            LDCH        BUFFER,X
            WD          =X'05'
            TIXR        T
            JLT         WLOOP
            RSUB
            END         FIRST
```

Object code for Proagram 2 :

object_input.asm

```
H^COPY  ^000000^001077
T^000000^1D^17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
T^00001D^13^0F20160100030F200D4B10105D3E2003454F46
T^001036^1D^B410B400B44075101000E32019332FFADB2013A00433200857C003B850
T^001053^1D^3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T^001070^07^3B2FEF4F000005
M^000007^05
M^000014^05
M^000027^05
E^000000
```

Program3 : L L Beck program from fig 2.11 involving program blocks

```
COPY        START     0
FIRST       STL       RETADR
CLOOP       JSUB      RDREC
            LDA       LENGTH
            COMP      #0
            JEQ       ENDFIL
           JSUB      WRREC
            J         CLOOP
ENDFIL      LDA       =C'EOF'
            STA       BUFFER
            LDA       #3
            STA       LENGTH
            JSUB      WRREC
            J         @RETADR
           USE CDATA
RETADR      RESW      1
LENGTH      RESW      1
           USE CBLKS
BUFFER      RESB      4096
BUFFEND     EQU       *
MAXLEN      EQU       BUFFEND-BUFFER
.
.          SUBROUTINE TO READ RECORD INFO BUFFER
.
           USE
RDREC       CLEAR     X
            CLEAR     A
            CLEAR     S
            +LDT      #MAXLEN
RLOOP       TD        INPUT
            JEQ       RLOOP
            RD        INPUT
            COMPR     A,S
            JEQ       EXIT
            STCH      BUFFER,X
            TIXR      T
            JLT       RLOOP
EXIT        STX       LENGTH
            RSUB
           USE CDATA
INPUT       BYTE      X'F1'
.
.         SUBROUTINE TO WRITE RECORD FROM BUFFER
.
           USE
WRREC       CLEAR     X
            LDT       LENGTH
WLOOP       TD        =X'05'
            JEQ       WLOOP
            LDCH      BUFFER,X
            WD        =X'05'
            TIXR      T
            JLT       WLOOP
            RSUB
           USE CDATA
           LTORG
            END       FIRST
```

Objectcode for Program3 :

object_input.asm

```
H^COPY  ^000000^001071
T^000000^1E^1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
T^00001E^09^0F20484B20293E203F
T^000027^1D^B410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T^000044^09^3B2FEA13201F4F0000
T^00006C^01^F1
T^00004D^19^B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T^00006D^04^454F4605
E^000000
```

THANKU