Solution and scheme of Evaluation

Q1.Explain SIC/XE machine instruction formats and all addressing modes by clearly indicating the setting of different flag bits. 10m

Instruction Formats

SIC/XE supports these 4 different instruction formats .

Format 1 (1 byte) **1m**

Opcode 8bits

Format 1 (1 byte): contains only operation code.

Format 2 (2 bytes) **1m**

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).

Opcode 8bits	R1 4bitS	R2 4bits

Format 3 (3 bytes) **2m**

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.

n-indirection , i-immediate , x-indexing , b-Base relative , p-PC relative , e-Extended

Opcode 6bits	n	i	x	b	p	e	displacement 12 bits

Format 4 (4 bytes) **1m**

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.

Opcode 6bits	n	i	x	b	p	e	Address 20 bits

Formats 3 & 4 introduce addressing mode flag bits

Addressing Modes 5m

SIC/XE supports the following addressing modes

1. n=0 & i=1
Immediate addressing - TA is used as an operand value (no memory reference)

2. n=1 & i=0
Indirect addressing - word at TA (in memory) is fetched & used as an address to fetch the operand from

3. n=0 & i=0
Simple addressing TA is the location of the operand

4.n=1 & i=1
Simple addressing same as n=0 & i=0

5.Flag x: Indexed addressing

x=1 Indexed addressing add contents of X register to TA calculation

Flag b & p (Format 3 only):

· **b=0 & p=0**

Direct addressing displacement/address field containsTA (Format 4 always uses direct addressing)

· **b=0 & p=1**

PC relative addressing - TA=(PC)+disp (-2048<=disp<=2047)*

· **b=1 & p=0**
Base relative addressing - TA=(B)+disp (0<=disp<=4095)**

Flag e:e=0 use Format 3
e=1 use Format 4

Q2a. What is System software? Differentiate it from application software. (4 Marks)

Definition 1m

System software consists of a variety of programs that support the operation of a computer. Assemblers ,loaders .linkers are examples of system software.

Differences between system software and application software 3m

System software	**Application Software**
These are intended to support the operation &use of computers	Concerned primarily on the solution of a problem using computer as a tool.
Focus is on the architecture of computing system.	Focus is on application not on the computing system.
Machine dependent	Dependent on system software
Ex:Assemblers,Compilers	EX:MS paint,MS word

Q2 b. Write a sequence of instructions for SIC/XE to clear a 20-byte string to all blanks 6 Marks

SIC/XE PROGRAM: 4m

```
        LDT    #20      Initialize register T to 20
        LDX    #0       Initialize index register to 0
LOOP    LDCH   #0       Load 0 into register A
        STCH   STR1,X   Store 0 into str1
        TIXR   T        Add 1 to index,compare result to 20
        JLT    LOOP     Loop if index less than 20
.
.
STR1    RESW   20
```

Q3a. Write a short note on parser-lexer communication 5 MArks

- When you use a lex scanner and a yacc parser together a communication is needed
  The parser is the higher level routine.
- It calls the lexer **yylex()** whenever it needs a token from the input.
- The lexer then scans through the input recognizing tokens. As soon as it finds a token of interest to the parser, it returns to the parser,

With the token in **yylval**

The lexer and the parser have to agree what the token codes are. yacc defines the token code

- Yacc write a **y.tab.h** C header file containing all of the token definitions.
  Include *y.tab.h* in the lexer

Q3b. Write a Lex program to count the number of vowels and consonants in a string: (5 M)

```
 %{
     int vowels = 0;
    int  consonents  = 0;
%}
%%
    [  \t\n ]+      ;
   [aeiouAEIOU]     vowels++;
   [bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]    consonents++;
   .               ;
%%
main()
{
yylex();
Printf("The No of vowels are %d",vowels);
Printf("The no of consonents are %d",consonents);
}
```

Q4a. Calculate the target address generated for the following machine instructions. (6 Marks)

i. 032600h ii.03C300h iii.0310C303h. Consider (B)=006000, (PC)=003000 (X)=000090

I)032600H 2M

000000	1	1	0	0	1	0	0110 0000 0000

Format3

b=0 p=1 x=0 disp=600 PC relative addressing mode in format3 PC=3000h ,

TA=(PC) + disp=3000 +600=3600

ii.03C300h 2M

000000	1	1	1	1	0	0	0011 0000 0000

Format3

b=1 p=0 x=1 disp=300 base relative addressing mode with indexing in format3

B=6000h , TA=(B) +(x)+ disp=6000 +90+0300=6390

iii.0310C303h. 2m

000000	1	1	0	0	0	1	0000 1100 0011 0000 0011

Format 4, TA=20 bit address =0C303

Q4b. What is regular expression? Explain the lex specification with an example. (4 Marks)

Structure of a Lex_specification

%{definitions%}
%%
{rules} 2m
%%
{user subroutine}

Lex program has three parts ,where the definitions and the user subroutines are optional.

1)Definition Section 1m

Enclosed between **%{** And **%}**

Definition Section contains the following

- Header file inclusions
- Variable declarations .

Definition Section of word counting program

%{

charcount = **0, wordcount** = **0, linecount** = **0;**

% **}**

Word [^' **\t\n]+**

Eol \n

The code block here declares three variables used within

the program to track the number of characters, words, and lines encountered.

The last two lines are definitions. Lex provides a simple substitution mechanism

to make it easier to define long or complex patterns. We have added two definitions here. The first provides our description of a word: any non-empty combination of characters except space, tab, and newline. The second describes our end-of-line character, newline. We use these definitions in the second section of the file, the rules section

2)Rules Section

Rules are enclosed between **%%** and **%%**

Rules Section defines
R1 {*action1*}
R2 {*action2*}

Where

- each **Ri** is a regular expression
- ***action i,*** is a program fragment defining what action the lexical analyzer should take when pattern Ri matches lexeme.

In Lex actions are written in C;in general,however,they can be in any implementation language.

The rules section contains the patterns and actions that specify the lexer. Here is our sample word count's rules section:

```
%%
{word} { wordcount++; charcount += yyleng;}
{eol} { chartount++; linecount++; }
 .    {charcount++;}
%%
```

2m

variable **yyleng** which contains the length of the string our lexer recognized. lexer recognizes a newline, it will increment both the character count and the line count. Similarly, if it recognizes any other character it increments the character count. For this lexer, the only "other characters" it could recognize would be space or tab; anything else would match the first regular expression and be counted as a word.

3) **User subroutine** **2m**

- The third section holds whatever auxiliary procedures are needed by the actions.

Lex copies it to the C file after the end of the lex generated code

main ( )

{

YYlexO ;

printf("%d %d %d\nm,lineCount, wordcount, charcount);

}

It first calls the lexer's entry point yyIex() and then calls **printf()** to print the results of this run

When yylex() reaches the end of its input file, it calls yywrap(), which returns a value of 0 or 1. If the value is 1, the program is done and there is no more input. If the value is 0, on the other hand, the lexer assumes that w r a p ( ) has opened another file for it to read, and continues to read from yyin. The default yywrap() always returns 1. By providing our own version of yywrap(), we can have our program read all of the files named on the command line, one at a time.

Q5.Write and explain the algorithm of PASS-2 of an Assembler (10 Marks)

The Algorithm for Pass 2:

- Initialization of records – 2 Marks
- Checking Symbol table and Operation table for opcode and label – 5 Marks
- Checking optab for assembler directives – 3 Marks

```
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}
```

6a. What is program relocation? Explain the problem associated with it and solutions?

The need for program relocation **1 Mark**

· It is desirable to load and run several programs at the same time.

· The system must be able to load programs into memory wherever there is room.

· The exact starting address of the program is not known until load time.

Example:

```
55      101B        LDA     THREE       00102D
```

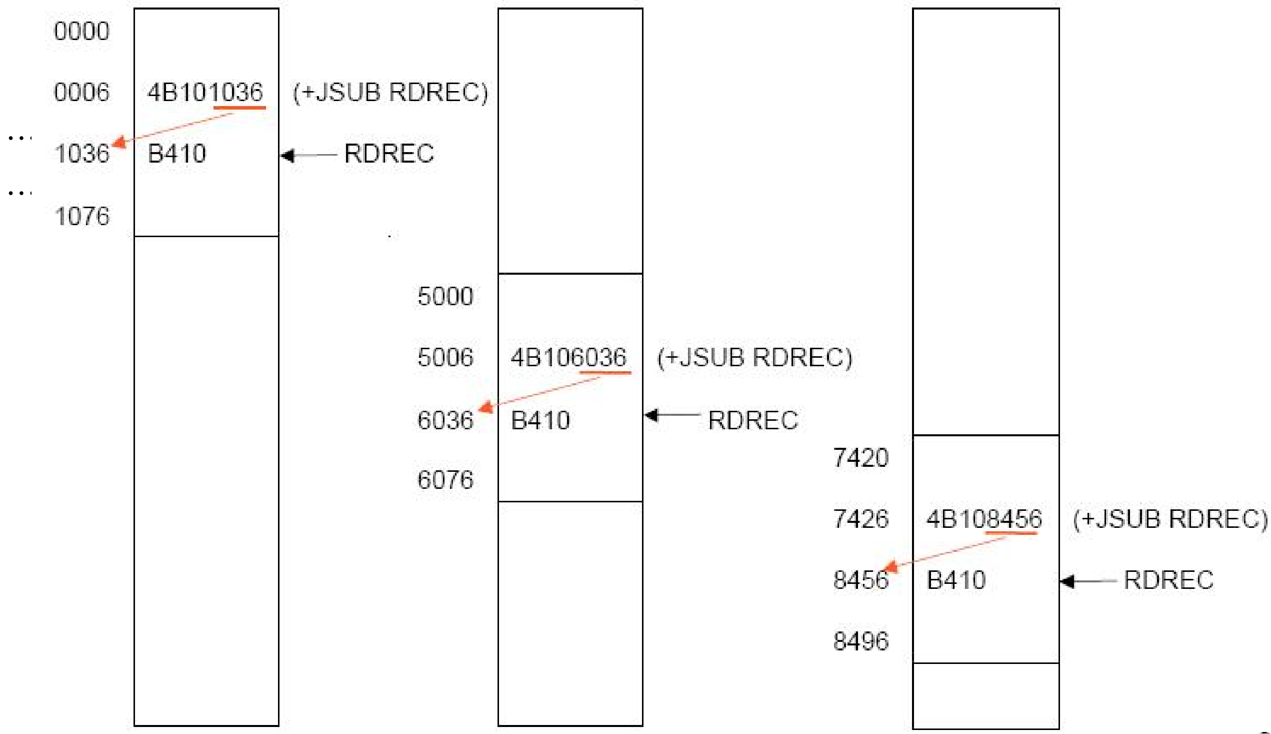
Calculate based on the starting address 1000

Reload the program starting at 3000

```
55      101B        LDA     THREE       00302D
```

The absolute address should be modified

Explanation **3 Marks**



· The only parts of the program that require modification at load time are those that specify direct addresses.

· The rest of the instructions need not be modified.

Not a memory address (immediate addressing)

PC-relative, Base-relative

· From the object program, it is not possible to distinguish the address and constant.

The assembler must keep some information to tell the loader about those portions of the program that needs modification.Modification record is used for this purpose

The object program that contains the modification record is called **Relocatable program.**

The way to solve the relocation problem 2 Marks

· For an address label, its address is assigned relative to the start of the

program(START 0)

· Produce a Modification record to store the starting location and the length of the address field to be modified.

Modification record

· One modification record for each address to be modified

· The length is stored in half-bytes (4 bits)

· The starting location is the location of the byte containing the leftmost bits of the address field to be modified.

· If the field contains an odd number of half-bytes, the starting location begins in the middle of the first byte.

Modification record

Col. 1	M
Col. 2-7	Starting location of the address field to be modified, relative to the beginning of the program (Hex)
Col. 8-9	Length of the address field to be modified, in half-bytes (Hex)

M^000007^05

M^000014^05

6.b. Give the format of the following 4 MArks
i). Header record ii)Text record iii) End record

Header record: 1 Mark

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: 2 Marks

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: 1 Mark

Col.1 E

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

7. Generate the complete object code for the ALP.Assume suitable machine equivalents from the mnemonic opcodes. (10 Marks)

OPCODES:LDX-04 LDA-00 LDB-68 ADD-18 TIX-2C STA-0C JLT-38 RSUB-4C

- **Calculation of Addresses – 2 Marks**
- **Generating Object Code – 8 Marks**

SUM	START	0000		
0000	FIRST	LDX	#0	050000h
0003		LDA	#0	010000h
0006		+LDB	#TABLE2	69100320h
		BASE	TABLE2	
000A	LOOP	ADD	TABLE,X	1BA013
000D		ADD	TABLE2,X	1BA310
0010		TIX	COUNT	2F200A
0013		JLT	LOOP	3B2FF4
0016		+STA	TOTAL	0F100920h
001A		RSUB		4F0000
001D	COUNT	RESW	1	
0020	TABLE	RESW	0100H	
0320	TABLE2	RESW	0200H	

0920	TOTAL	RESW	1
0923		END	FIRST

1. LDX #0 ,LDX-04

000001	0	1	0	0	0	0	0000 0000 0000

Object Code : 050000h

2. LDA #0 ,LDA - 00

000000	0	1	0	0	0	0	0000 0000 0000

Object Code : 010000h

3. +LDB #TABLE2 ,LDB-68 –Format 4 instruction

0110 10	0	1	0	0	0	1	0000 0000 0011 0010 0000

Object Code : 69100320h

4. ADD TABLE , X , ADD-18 – Format 3 Instruction

0001 10	1	1	1	0	1	0	0000 0001 0011

Displacement = TA – PC

= 0020 – 000D = 013

Object Code : 1BA013

5. ADD TABLE2 , X , ADD-18 – Format 3 instruction

0001 10	1	1	1	0	1	0	0011 0001 0000

Displacement = TA-PC = 0320-0010 = 0310

Object Code : 1BA310

6. TIX COUNT ,TIX – 2C ,Format 3 Instruction

0010 11	1	1	0	0	1	0	0000 0000 1010

Displacement = TA-PC = 001D – 0013 = 00A

Object Code : 2F200A

7. JLT LOOP , JLT - 38

0011 10	1	1	0	0	1	0	1111 1111 0100

Displacement = TA-PC = 000A – 0016 = FF4

Object Code : 3B2FF4

8. +STA TOTAL ,STA -0C

0000 11	1	1	0	0	0	1	0000 0000 1001 0010 0000

Object Code : 0F100920

9. RSUB , RSUB – 4C
   Object Code : 4F0000