4/5/22

SS - Assignment - 01

B1 Explain SIC Architecture.

→ SIC architecture consists of its memory and Registers, Data formats, Instruction formats, Addressing modes, Instruction set, Input and Output.

→ Memory: There are 2^{15} 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, and each location in memory contains 8-bit bytes.

→ Registers: There are 5 registers, each 24 bits in length.

Mnemonic	Number	Use
A	0	Accumulator; used for Arithmetic operation
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)
All machine instructions on the standard version of SIC have the 24-bit format.

→ Addressing modes: There are 2 addressing modes available. Parenthesis are used to indicate the content of a register/memory location.

Mode	Indication	Target address
Direct	x = 0	TA = address
Indirect	x = 1	TA = address + (x)

→ Input and Output: I/O are performed by transferring 1 byte @ a time / from the rightmost 8 bits of Accumulator. The TD (Test Device) instruction tests whether the addressed

device is ready to send / receive a byte of data. Read Data (RD) Write data (WD) are used for reading / writing the data.

→ Instruction Set: SIC provides load and store instructions like LDA, STA, LDX, STX etc; All Arithmetic operations involve registers A and a word in memory with the result being left in the register.

Q2. Explain SIC-XE Architechture

→ 1) Memory: Maximum memory available on a SIC/XE System is 1 Megabyte (2^{20} bytes)

2) 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)

3) Floating-point data type.

There is a 48-bit floating-point data type $F * 2^{(e-1024)}$

1	11	36
s	exponent	fraction

4) Instruction formats: The new set of IF follows: Format 1: Contains only operation code. Format 2 contains first 8 bits for Operation code, next 4 for register 1 & following 4 for register 2. Format 3: Contains first 6 bits for operation code, next 6 bits for flags and last 12 bits for displacement for the address of the operand. Format 4 is same as format 3 with an extra 2 hex digits (8 bits) for address that require more than 12 bits to be represented. format 1 (1 byte), format 2 (2 bytes), format 3 (3 bytes) & format 4 (4 bytes)

1) format 1 (1 byte)

8
op

2) format 2 (2 bytes)

8	4	4
op	r_1	r_2

3) format 3 (3 bytes)

6	1	1	1	1	1	1	12
op	n	i	x	b	p	e	disp

4) format 4 (4 bytes)

6	1	1	1	1	1	1	20
op	n	i	x	b	p	e	address.

5) Addressing modes & Flag Bits :- 5 Addressing mode Combinations

a) Direct : operand address goes as it is. n and i are both set to the same value, either 0 or 1. In general it is 1.

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

c) Relative : either b/p equal to 1 and the other one to 0.

d) Immediate : (i=1, n=0). The operand value is already enclosed on the instruction.

e) Indexed (x=1) : value to be added to the value stored at the register x to obtain real address of the operand.

6) Instruction Set : LDB, STB are load & store instructions, ADDF, SUBF, MULF, DIVF are floating-point arithmetic operations. ~~RMO~~, Register to Register arithmetic operations, Register move ~~operati~~ Instruction (RMO) and SVC i.e Supervisior call Instruction.

7) Input and Output : There are I/O channels that can be used to perform i/o while the CPU is executing other instructions. The instructions SIO, TIO and HIO are used to start, test and halt the operation of I/O channels.

Q3. Explain the pass 1 algorithm and the data structures used.

→ The simple assembler uses 2 major internal data structures: The operation code Table (OPTAB) and the Symbol Table (SYMTAB).

→ OPTAB :- It is used to lookup mnemonic operation codes and translates

them to their machine language equivalents. In pass 1 the OPTAB is used to lookup and validate the opcode in the source program. In pass 2 it is used to translate the opcodes to machine language.

→ SYMTAB: This table includes the name and value for each label in the source program, together with flags to indicate the error conditions. During pass 1: labels are entered into the Symbol table along with their assigned address value as they are encountered. During pass 2: Symbols used as operands are looked up the symbol table to obtain the address value to be inserted in the assembled instructions.

Algorithm 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 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}
```

Q4. Explain the pass-2 algorithm

→ Algorithm for pass2: 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
      else
       begin
         Store 0 as operand addres :
       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}
```

→ Here the first i/p line is read from the intermediate file. If the opcode is START, then this line is directly written to the list file. A header record is written in the object program which gives the starting address and the length of the program. Then the first text record is initialized. An error flag is set indicating it as undefined. If symbol itself is not found then store 0 as operand address and the object code instruction is assembled. Once the whole program is assemble & when the END directive is encountered, the END record is written.

Q5. List and Explain various Addressing modes used in SIC-XE Architechture.

→ 1) Translation involving Register - Memory instructions
2) Program - Counter Relative
3) Base - Relative Addressing mode

4) Immediate Addressing mode
5) Indirect and PC- relative mode.

1) Translation for the Instruction involving Register- Register Addressing mode: This addressing mode can be represented by either using format-3 type or format-4 type instruction format. In format-3, the instruction has the opcode followed by a 12-bit displacement value in the address field. In format-4 the instruction contains the mnemonic code followed by 20bit displacement value in the address field.

2) Program counter Relative: In this usually format-3 instruction format is used. The Instruction contains the opcode followed by a 12-bit displacement value. The range of displacement values are from 0-2048. This displacement value is added to the current contents of the program counter to get the address of the operand required by the instruction.

3) Base-Relative Addressing mode: In this mode the basis register is used to mention the displacement value. Therefore the target address is TA = (base) + displacement value.

4) Immediate Addressing mode: In this mode no memory reference is involved. If immediate mode is used the target address is the operand itself.

5) Indirect and PC- Relative mode:- In this type of Instruction the symbol used in the instruction is the address of the location which contains the address of the operand. The address of this is found using PC- Relative addressing mode. The instruction jumps the control to the address location RETADR which in turn has the address of the operand.

8?. Explain the different machine Independent features.

→ There are the features which don't depend on the arch. of machine. They are Literals, Expressions, program blocks and Control Sections:-

a) Literals: literal is equivalent to define a constant explicitly & assign an address label for it and use the label as the instruction operand. A literal is identified with the prefix =, followed by a specification of literal value.

* Literal pool: All the literal operands are gathered together into one or more literal pools. At the end of the object program, generated immediately following the END statement. At the location where the LTORG directive is encountered.

* Duplicate literals: The same literal used more than once in the program. Only one copy of the specified value needs to be stored. Eg: =X'05' in the example program.

* Implementation of literal: Data structure: a literal table LITTAB
  - literal name
  - operand value and length
  - Address. Littab is often organized as a hash table, using the literal name or value as a key.

b) → Pass:- Expressions: A single term as an instruction operand can be replaced by a expression.

```
STAB    RESB    1100
STAB    RESB    11 * 100
STAB    RESB    (6+3+2) * MAXENTRIES.
```

The assembler has to evaluate the expression to produce a single operand address/value. Expressions can be Relative and Absolute sometimes. To determine the type of expression, we must keep track of the types of all symbols defined in the program.

3) program blocks: As an Eg: 3 blocks are used: default for executable instructions, CDATA: all data areas that are less in length. - CBLKS: all data areas that consists of larger blocks of memory. Each program block may actually contain several separate segments of the source program. The assemble will logically rearrange these segments to gather together the pieces of each block.

4) Control Sections: It is a part of the program that maintain its identity after assembly. It is often used for subroutine or other logical subdivision of a program. It can be assembled, loaded and relocated independently and is more flexible.

B12. Explain various instruction formats used in SIC-XE Architechture.

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

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

format 1 (1 byte)

8
op.

2) format 2 (2 byte): first 8 bits for the operation code, next 4 for register 1 and following 4 for register 2. The numbers for the registers go according to the numbers indicated at the registers section.

format 2 (2 bytes)

8	4	4
op	r1	r2

3) format 3 (3 bytes): first 6 bits contain operation code, next 6 bits contain flags, last 12 bits contain displacement

00
00
10
10

for the address of the operand.

format 3 (3 bytes)

6	1	1	1	1	1	1	12
op	n	i	x	b	p	e	disp

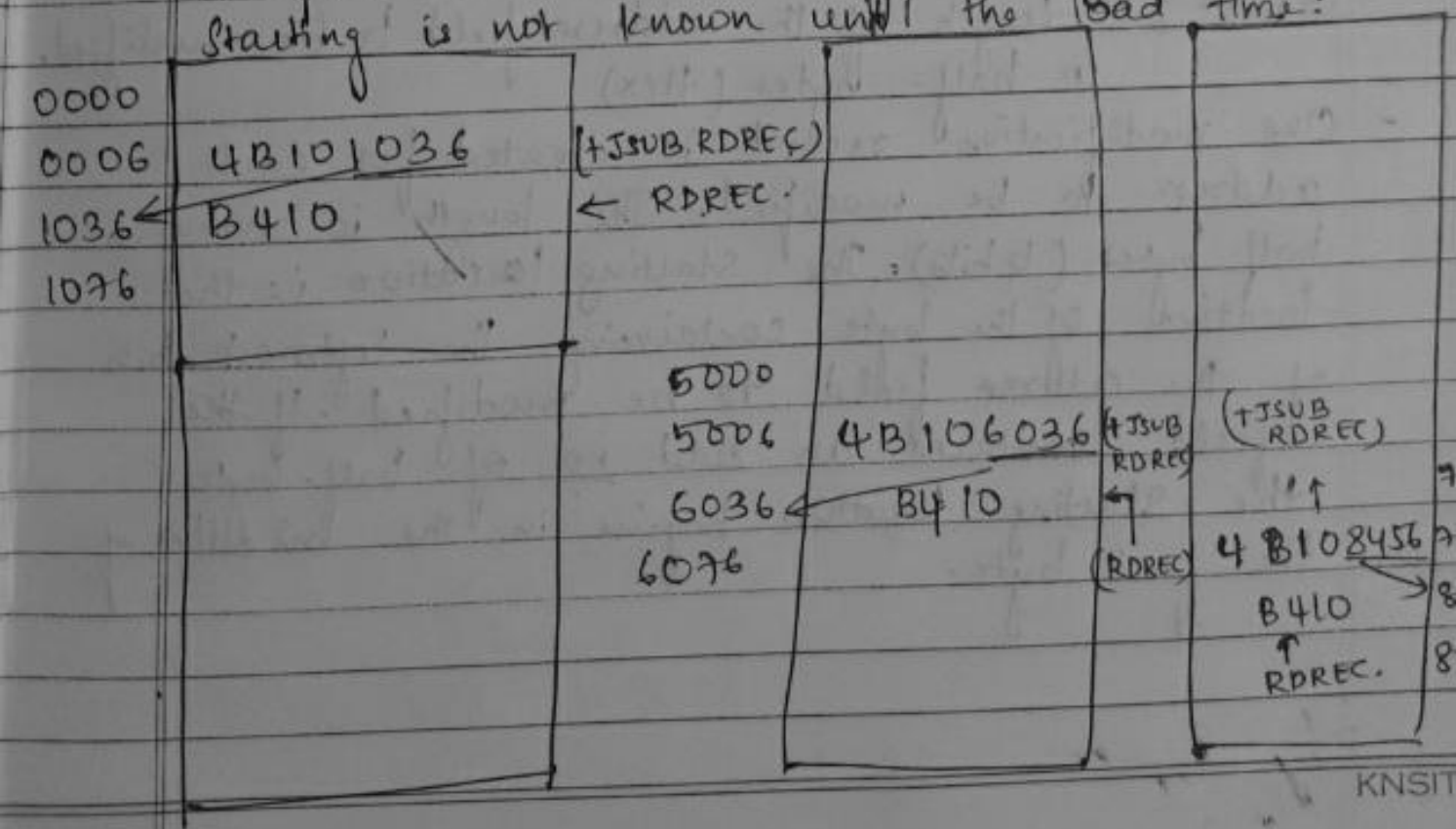
4) format 4 (4 bytes) : same as format 3 with an extra 2 hex digits (8 bit) for addresses that require more than 12 bits to be represented.

format 4 (4 bytes)

6	1	1	1	1	1	1	20
op	n	i	x	b	p	e	address

Q8. Explain program Relocation with a diagram and its modification record with an Example.

→ Sometimes it is required to load and run several programs at the same time. The system must be able to load these programs wherever there is a place in the memory. Therefore the exact starting is not known until the load time.



The above diagram shows the concept of relocation. Initially the program is loaded at location 0000. The instruction JSUB is loaded at location 0006. The address field of this instruction contains 01036, which is the address of the instruction labeled RDREC. The 2nd fig. shows that if the program is to loaded at new location 5000. The address of the instruction JSUB gets modified to new location 6036. The 3rd fig shows that if the program is relocated at location 7420, the JSUB instruction would need to be changed to 4B108456 that correspond to the new address of RDREC.

→ Modification Record:

Col. 1 → M

Col. 2→4 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)

- One modification record is created 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 no of half-bytes, the starting location begins in the middle of the first byte.

- H^COPY ^0000 0000 1077 5 half-bytes
- T^000000^1D^1772029^1692029^14B1 01036 032026^
  290000 332007^ 4B1 0105D ^3F2FEC^032010
- T^00001D^13^0F201 6^01000 3^0F200D^14B
  ^0105D^3E2003^45 4F46
- T^001036^1D^B41 0^B400^B440^7571 000^E320
  19^332FFA^DB201 3^A004^1332008^57C003^
  B850.
- T^001053^1D^3B2FEA^134000^4F000^F1^B
  410^774000^E320 11^332FFA^53C003^
  DF2008^B850.
- T^001D700^73B2FEF^4F000005
- M^000007^05
- M^00001 4^01
- M^000027^05
- E^000000.

6TH and 11TH questions are remaining
ma'am told she will be sending solution on monday