# SS Module II Important Questions

## 1. Write SIC or SIC/XE assembly language program

## 2. What are the basic functions of an assembler

- Convert mnemonic operation codes to their machine language equivalents. Eg: translate STL to 14
- Convert symbolic operands to their equivalent machine addresses. Eg: translate RETADR to 1033
- Convert the data constants specified in the source program into their internal machine representations.- eg: translate EOF to 454F46
- Write the object program and assembly listing.
- Build machine instruction into proper format

## 3. Write the object program format

- The simple object program contains three types of records:
  - Header record
  - Text record
  - end record
- The header record contains the starting address and length.

Header record:

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

- Text record contains the translated instructions and data of the program
  - Together with an indication of the addresses where these are to be loaded.

Text record:

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

- The end record marks the end of the object program and specifies the address where the execution is to begin.

End record:

| | |
|---|---|
| Col. 1 | E |
| Col 2-7 | Address of first executable instruction in object program (hexadecimal) |

- Figure 2.3 shows the object program corresponding to figure 2.2. The Asymbol

## 4. Explain the datastructures used in an assembler algorithm.

The datastructures used are

- 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 look up and validate the operation code in the source program.
  - In pass 2, it is used to translate the operation codes to machine language
  - OPTAB is usually organized as a hash table, with mnemonic operation code as the key. The hash table organization is particularly appropriate, since it provides fast retrieval with a minimum of searching.
  - Most of the cases the OPTAB is a static table- that is, entries are not normally added to or deleted from it.
- SYMTAB
  - This table includes the name and value for each label in the source program

    - 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
    - SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval. value to be inserted in the assembled instructions.
- LOCCTR
    - LOCCTR is initialized to the beginning address mentioned in the START statement of the program.
    - After each statement is processed, the length of the assembled instruction is added to the LOCCTR to make it point to the next instruction.
    - Whenever a label is encountered in an instruction the LOCCTR value gives the address to be associated with that label

## 5. Write pass 1 of a two pass assembler

- Read the first input line
- if the opcode is start
    - Starting address will be the operand
    - LOCCTR is set as starting address
    - Write this line to the intermediate file
    - Read next input line
- else
    - Set LOCCTR to 0
- Start a loop with condition while OPCODE is not END
    - Write the line to intermediate
    - Now we need to check if there is a symbol in the label field
        - We need search SYMTAB for that particular LABEL
            - IF we found it
                - Set Error flag (Duplicate)
            - Else
                - LABEL and LOCCTR are inserted into SYMTAB
    - Now we need search the OPTAB for OPCODE
        - If we found the OPCODE
            - Incremenet LOCCTR by 3
        - If OPCODE is word
            - Increment LOCCTR by 3
        - If OPCODE is RESW

        - Increment LOCCTR by 3 x Size of words to reserved
    - If OPCODE is RESB
        - Increment LOCCTR by the number of bytes
    - If OPCODE is BYTE
        - Increment LOCCTR by length of the constant in bytes
    - If none of the conditions match
        - Set the error flags
  - Read next input line
- Write last line to intermediate file
- Save LOCCTR - starting address as program length

## 6. Write pass 2 of a two pass assembler

- Read first input line
- if opcode is start
    - Write the line to assembly listing
    - Read next input line
- Write Header records to object program
- Initialize first Text record
- Start a loop with condition while OPCODE not equal to END
    - Search OPTAB for OPCODE
        - If found
            - If there is a symbol in operand field
                - Search SYMTAB for operand
                - If found
                    - Store symbol value as operand address
                - else
                    - Set error flag
            - else
                - Set 0 as operand address
            - Assemble object code instruction
        - if OPCODE = BYTE or WORD
            - Convert constant to object code
        - If object code is not fitting into current text record
            - Write Text Record to Object Program
            - Initialize a new text record (To accomodate the remaining)
        - Add object code to text record

    - Write the line into assembly listing along with object code
    - Read next input line
- Write last text record to object program
- Write End record to object program
- Write Last lisiting line

## 7. Conversion of SIC program

- Translation of PC Relative

Write a sequence of instructions for SIC/XE to set ALPHA equal to 4*BETA-9.

4 Use immediate addressing modes for constants and assume ALPHA and BETA 3
to be floating point numbers.

4 Write an SIC program to swap the values of **ALPHA** and **BETA** 3

b) Write a SIC program to perform linear search in an array of 100 elements. 6

## Object Code generation in SIC

Generate the assembled object program for the below SIC program. The machine code for the instructions used are: LDX – 04, LDA – 00, ADD – 18, TIX – 2C, STA – 0C, JLT – 38 and RSUB – 4C. Show the location counter value for each instruction.

b)

```
SUM       START     4000
FIRST     LDX       ZERO
          LDA       ZERO
LOOP      ADD       TABLE,X
          TIX       COUNT
          JLT       LOOP
          STA       TOTAL
          RSUB
TABLE     RESW      2000
COUNT     RESW      1
ZERO      WORD      0
TOTAL     RESW      1
          END       FIRST
```

- Make a table of Line no, Location counter, label, opcode, operand and object code

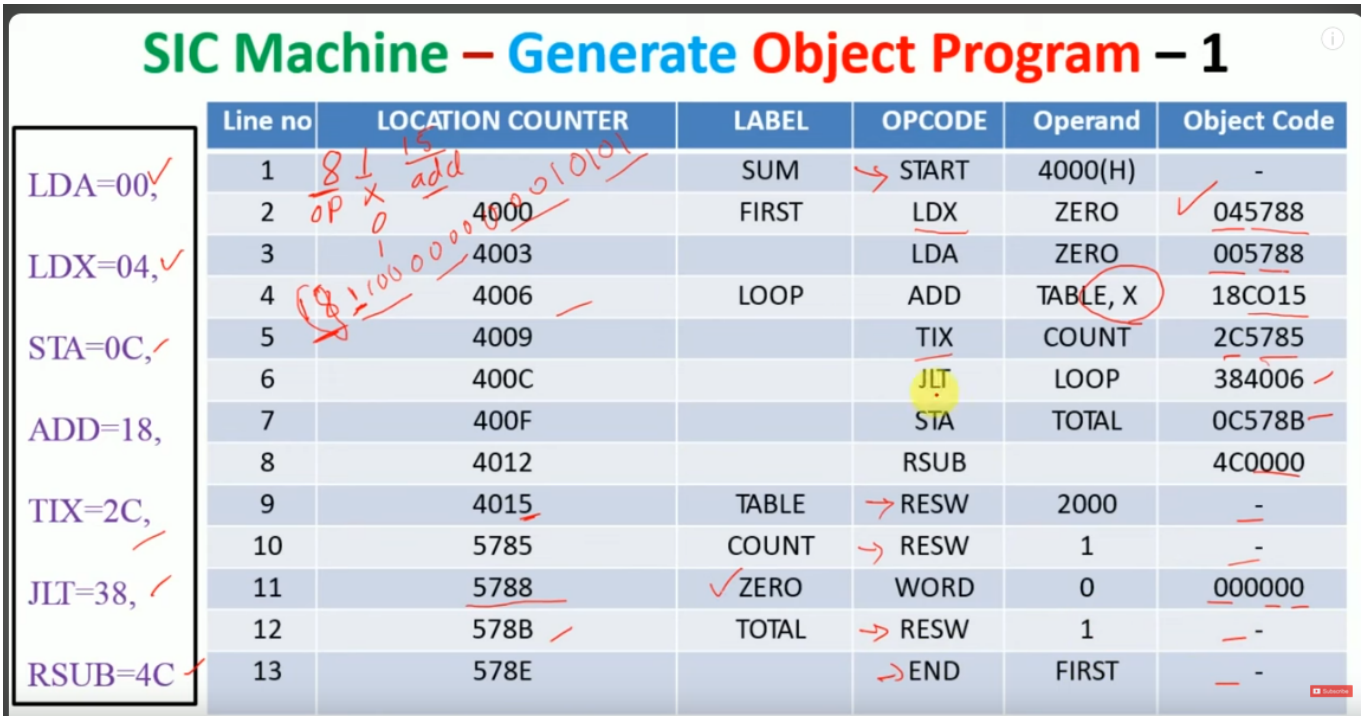
## SIC Machine – Generate Object Program – 1

LDA=00,
LDX=04,
STA=0C,
ADD=18,
TIX=2C,
JLT=38,
RSUB=4C

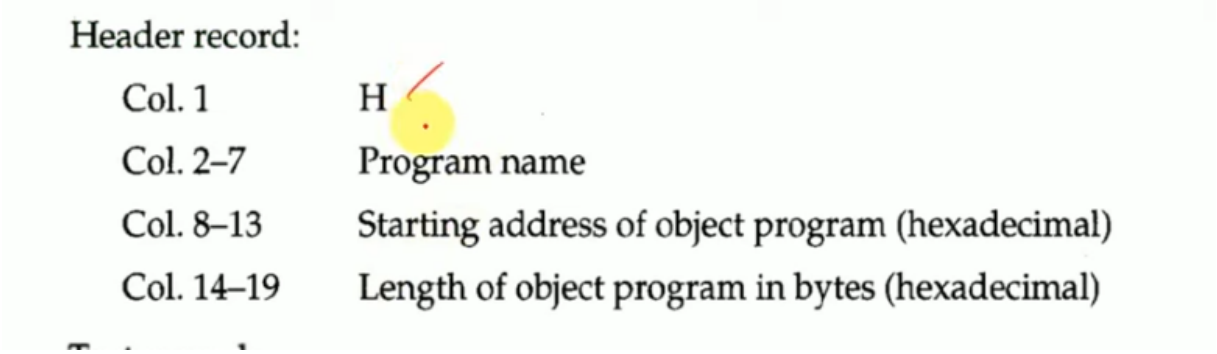
| Line no | LOCATION COUNTER | LABEL | OPCODE | Operand | Object Code |
|---|---|---|---|---|---|
| 1 | | SUM | START | 4000(H) | |
| 2 | | FIRST | LDX | ZERO | |
| 3 | | | LDA | ZERO | |
| 4 | | LOOP | ADD | TABLE, X | |
| 5 | | | TIX | COUNT | |
| 6 | | | JLT | LOOP | |
| 7 | | | STA | TOTAL | |
| 8 | | | RSUB | | |
| 9 | | TABLE | RESW | 2000 | |
| 10 | | COUNT | RESW | 1 | |
| 11 | | ZERO | WORD | 0 | |
| 12 | | TOTAL | RESW | 1 | |
| 13 | | | END | FIRST | |

- Take start, Note the start address
  - Put the start address below inside location counter
- Increment The lOCCTR by 3 until RSUB
- At line no 9, we have RESW
  - It has 2000 bytes,
    - We need to multiple 2000 x 3, and convert the result to hex
  - Add that value to LOCCTR
- Follow the algorithm to see how to increment LOCCTR

### Creating object code



- No object code for START, END as they are directives
  - No object code for RESW
- First 2 bytes = Numerical value of OPCODE
- Next 4 Byes = LOCCTR of OPERAND
- When theres ,X, it means its an indexed addressing mode
  - Here TABLE, X is indexed addressing
  - In the case of TABLE, actually it should be 18 4015
  - Since its indexed addressing mode
    - We convert them to binary, First bit in 3 bits and the remaining in 4 bits
    - 4015 = 100 0000 0001 0101
    - Since its indexed addressing mode, we add one to 100
    - We get 1100 0000 0001 0101 = C015
- Example
  - Line no 2
  - OPCODE = LDX
    - LDX = 04
  - Operand = ZERO
    - ZERO is defined in Line 11
    - ZERO = 5788
- Creating Header Record



| Header record: | |
|---|---|
| Col. 1 | H |
| Col. 2–7 | Program name |
| Col. 8–13 | Starting address of object program (hexadecimal) |
| Col. 14–19 | Length of object program in bytes (hexadecimal) |

  - We get

H^SUM___^004000^00178E

- Creating Text Record

Text record:

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

- End record:
  - T^004000^15^045788^005788^18C015^2C5785^384006^0C578B^4C0000
- for Getting length of object code
  - bits in object code x number of object codes = 7x6 = 21
  - Convert 21 decimal to hex = 15
- Remaining, add the object codes one by one

- Creating next text record
  - T^005788^3^000000
- it starts at 5788
- Length is 3

- End record
  - E^004000

Write a sequence of instructions for SIC/XE to divide BETA by GAMMA, setting ALPHA to the integer portion of the quotient and DELTA to the remainder. Use register to-register instructions to make the calculation as efficient as possible. 5