UNIT-1

Introduction to system programming:

* **Definition of system programming:-**

System programming is the activity of programming computer system software. The primary characteristics of system programming when compared to application programming is that application programming aims to produce software which provides services to the user directly.

Ex:- Word processor

Whereas, system programming aims to produce software and software platforms which provides services to other software.

Ex:- operating system, computational science applications, Game engines & video games, compilers, interpreteur, assembles, Linkers, Loaders --- etc

System software:-

System software consists of variety of programs that supports the operation of a computer. The software makes it possible for the user to focus on application, other problems to be solved without knowing the details of how the machine works internally.

System software and Machine Architecture:-

System programms are intended to support the operations and the use of computers. The machine architecture consists of

* Machine code
* Instruction format
* Addressing mode
* Registers

Machine independency of system software:-

The logic and general designing is basically same in which machine independent of system software consists of two parts such as :- (1) code optimisation
(2) subprogram Linkage

⇒ one of the characteristic in which the most system software differs from Application software is machine dependent

⇒ system programs are intended to support the operations and use of computer itself rather than any particular information

⇒ some of the examples of system software are :- * Text editor
* Assembler
* compiler
* Loader &
* Linker
* macro processor
* Debuger &
* operating system.

* Text editor :-
It is used to create and modify the program
Ex :- Notepad & wordpad

* Assembler & compiler :-
These are the translator which are used to translate the program in to machine language.

* Loader & Linker :-
The Resulting machine program was loaded in to memory and prepared for execution.

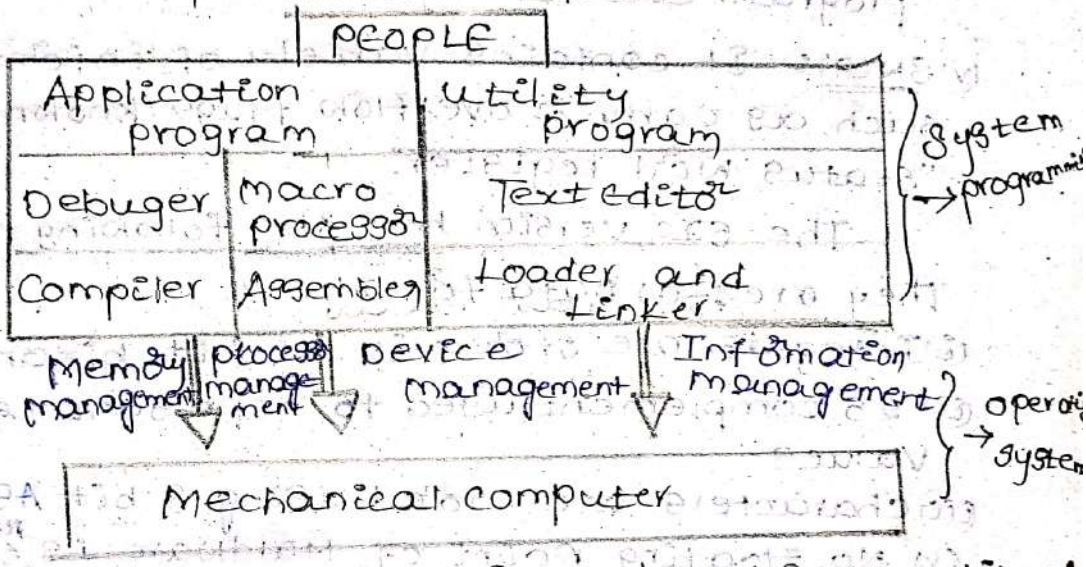
* macro processor :-
It is the program that copies a text from one place to another by making a systematic set of Replacement.

* Debuger:-

It is a system software or a program that is used to detect the errors in the program.

* operating system:-

It is a system software that acts as an interface between user and the Hardware by making a set of replacements.



* SIC Architecture:- [simplified instructional computer]

The SIC machine has basic addressing. It stores most memory address in Hexa decimal integer format. The SIC Architecture stores all data in binary form and uses Two's compliment to represent negative values at machine level. memory storage in SIC consists of 8 bit bytes and all memory address in SIC are byte address. Any three consecutive bytes form 24 bit word addressed by the location.

SIC machine has several registers each of 24 bit long and supports both numeric and character representation.

The following are the SIC registers. They are:- (1) A(0):- It is used for performing basic Arithmetic operations and storing the results known as "Accumulator Registers".

(ii) X(1):- It stores and calculates [illegible] known an "Index Register"

(iii) L(2):- It is used to jump in to specific address and storing return address known as "Linkage Register".

(iv) PC(8):- It contains the address of next instruction to be executed known as "program counter register".

(v) SW(9):- It contains variety of information such as carry & overflow flags known as "Status word register".

The SIc version has the following features. They are:- a) Data format:-

(i) Integers are stored in 24 bit binary number

(ii) 2's complement used to represent negative values

(iii) Characters are stored using 8 bit ASCI code

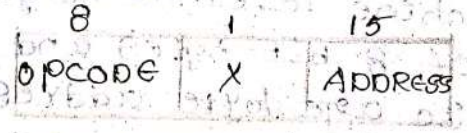
(iv) No floating point of Hardware is NOT supported in SIc version

(v) There are a total number of 32,768 bytes [32kb] in computer memory

(2) Instruction format:-

(i) It is represented in 24 bit format

(ii) The flat bit 'x' is used to indicate Address mode.



(3) Addressing modes:-

(i) There are two Addressing modes Available in SIc machine such as Direct mode and Indirect mode.

(ii) It is indicated by x-bit in the instruct

(iii) 'x' Represents the contents of regis te

Mode	Indication	Target Address calculation
Direct A.M	x=0	TA=ADDRESS
Indirect Addressing mode	x=1	TA=ADDRESS+1

(4) Instruction Set :-
(i) It supports format-3 i.e., 24 bit format
(ii) It loads and stores registers [LDA, LDX, STA, STX]
(iii) It performs integer Arithmetic operations [ADD, SUB, MUL, DIV]
(iv) It compares instructions [COMP]
(v) It has conditional jump instructions [JLT, JEQ, JGT]
(vi) It supports JSUB [Jumps to sub routine by placing the return address in the register 'L']
(vii) It supports RSUB [Returns by Jumping to the address contain in the registers].

(5) Input & output :-
(i) Input & output can be performed by transfering one byte at a time, two are from the right most of 8 bits of register 'A'
(ii) Each device is assigned a unique 8 bit code
(iii) Test device :- [TD]
It Tests whether the device is ready to send & receive and performs the operations such as Read data [RD] and Write Data [WD]

SIC/XE machine Architecture :-
[Extra equipments]
There is a more complicated machine built on Top of SIC called "Simplified Instruction computer" with extra equipment [XE]. The XE expansion of SIC adds a 48 bit floating point data type and additional memory addressing mode with extra memory of 1 mb. All SIC code is upward compatable with SIC/XE. In Addition to the standard SIC registers, there are also 4 additional General purpose Registers such as :-

(1) B(3) :-
It is used for Addressing known as "Base Register"

(2) S(4) :- It is a general purpose Register.
(3) ST(5) :- It is " " "
(4) F(6) :- It is a floating point accumulator Register.

These 9 Registers allows the SIc & SIc/xe machine to perform simple Tasks in a customized assembly language.

* SIc/xe features :-

The following are the various SIc/xe features such as,

(1) Data format :-

(i) Integers are stored in 24 bit binary number.
(ii) Two's complement is used for representing negative values.
(iii) Floating point values are represented in 48 bit format
(iv) The exponents is in b/w 0 and 2047
(v) It is represented as $f * 2^{(e-1024)}$
(vi) 0 :- (Zero)
It sets all the bits to zero

(2) Instruction format :-

(i) It contains Relative Addressing and supports format-3 and format-4.
(ii) It extends the address to 20 bits
(iii) There are four different formats supported in SIc/xe version. They are,

* Format-1 :-
It consists of 8 bits of allocated memory to store instructions.

1 byte
| OP(8) |

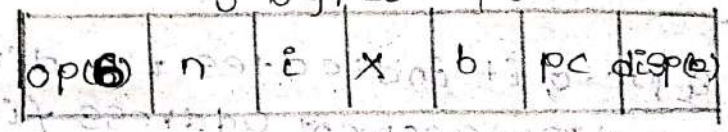
* Format-2 :-
It consists of 16 bits of allocated memory to store 8 bits of instructions and two four bits to store the operands

2 bytes
| OP(8) | R1(4) | R2(4) |

*format-3:-

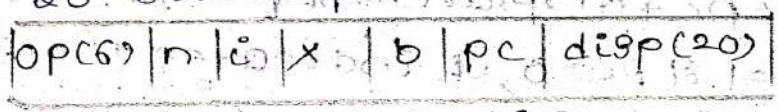
It consists of 6 bits to store an instructions 6 bits for flag values and 12 bits of displacement

3 bytes = 24 bits



* format-4:-

It is valid on SIC/XE machine consisting of four bytes [32 bits] of the same elements as format-3 but instead of 12 bit displacement, it stores 20 bit displacement [Address].



4 bytes = 32 bits

Where, disp = displacement Address

n = indirect Addressing mode (or) flag bits

i = immediate Addressing mode (or) flag bits

x = Index Register or flag bits

b = base Register mode or " "

pc = program counter or " "

e = floating point

(3) Addressing mode:-

(1) It uses n, i flag bits to represent the addressing mode.

(2) It also uses x, b, pc addressing mode

Immediate Addressing:-

if n=0 and i=1, the target address is used as operand value

if n=1 and i=0, in this Addressing the target address is fetched and used as an operand to fetch the operand.

(3) Simple Addressing:-

if n=0 and i=0 (or) n=1, and i=1, then the target address represents the location of operand.

(4) Flag 'x':-
If x=1, the index address is used to store the contents of x-register for target address calculation.

(5) flag b, pc:-
If b=0 & pc=0, it indicates the direct addressing, the displacement address field contains the target address
If b=0 & pc=1, it indicates the program counter register
TA = pc + displacement [-2048 ≤ disp ≤ 2047]
If b=1 & pc=0, it indicates the base relative Addressing
TA = B + disp (0 ≤ disp ≤ 4095)

(A) Instruction set:-
(1) SIC provides 26 instructions whereas SIC/XE provides an Additional 33 [26+33=59]
(2) SIC/XE 6 categories of instructions and it uses format 1, 2, 3 & 4
(3) It supports Load & store instructions [LDA, LDX, STA, STX, LDCH, STCH]
(4) It supports integer Arithmetic operations [ADD, SUB, MUL, DIV]
(5) It supports floating point Arithmetic operations [ADDF, SUBF, MULF, DIVF]
(6) It supports register to Register Arithmetic operations. [ADDR, SUBR, MULR, DIVR]
(7) A Special supervisor called instruction is provided with Arithmetic operations such as RSUB, COMPR, SHIFTR, SHIFTL ... etc
(8) SIC/XE has Additional instructions available in the following category. It has four additional instructions such as JUMP, JSUB, RSUB, COMPR.

Compare:-[comp]

It compare the contents of Register 'A' with a word in the memory and sets the conditional code.

Conditional Jumps:-[JLT, JEQ, JGT]

It Jumps according to the setting of the compiler.

Sub Routine Linkage:-[JSUB, RSUB]

It Jumps in to and return from Subroutine using the register.

5-I/O operations:-

SIC/XE has the capability of programmed I/O. Three Additional instructions are provided in SIC/XE version which are as follows.

⇒ It occupies one byte (8 bits) at a time

Test data device:-[TD].

It determines that the Address of I/O device is ready to send and receive a byte of data. Thus condition code gets the results from this test.

<:::- It indicates that the device is ready to Send & Receive.

=:::- It indicates that the device is not ready to send or Receive

Read data [RD]:-

It Reads the data from I/O devices in to Register 'A'.

Write data [WD]:-

It writes the data to the I/O device from Register, 'A'.

START I/O, TEST I/O, HALT I/O :-

These are used to start, test and halt the operations of Inputs output.

SIc and SIc/xe programming Examples:-

An example program to show the assembly language syntax for SIc version.

```
LDA  FIVE   Load constant five into Register A
STA  ALPHA  Store Alpha in Register A
LDCH CHARZ  Load character Z into Register A
STCH C1     Store character in variable C1
 .
 .
 .
ALPHA RESW1  one word variable
FIVE  WORD5  one word constant
CHARZ BYTE C'Z'  one byte constant
C1    RESB1  one byte constant
```

An example program to show assembly language for SIc/xe version

```
LDA #5     : Load constant five into Register A
STA ALPHA  : Store Alpha in Register A
LDCH #90   : Load character Z into Register A.
STCH C1    : Store character in variable C1.
 .
 .
ALPHA RESW1  one word variable
C1    RESB1  one byte constant
```

#:- immediate addressing
@:- Indirect addressing
CC:- conditional code
C :- character string

Write a program in assembly language to perform Arithmetic operations in SIc version.

```
LDA     ALPHA   Load Alpha into Register A
ADD     INCR    Add the value of INCR
SUB     ONE     Subtract one
STA     BETA    Store Beta into Register A
LDA     GAMA    Load Gama into Register A
ADD     INCR    Add the value of INCR
SUB     ONE     Subtract one
STA     DELTA   Stores the value delta into
                Register A
 :
 :
ONE     WORD    one word constant
ALPHA   RESW1   one word variable
BETA    RESW1   one word variable
GAMA    RESW1   one word variable
INCR    RESW1   one word variable
DELTA   RESW1   one word variable
```

Write a program in assembly language to perform Arthimetic operations in SIC/XE version

```
LDA     ALPHA   Load Alpha into register A
ADD     INCR    Add the value of INCR
SUB     #1      Subtract one
STA     BETA    Store Beta into Register A
LDA     GAMA    Load Gama into Register A
ADD     INCR    Add the value of INCR
SUB     #1      Subtract one
STA     DELTA   Stores the value delta in to
                Register A
 :
 :
ALPHA   RESW1   one word variable
BETA    RESW1   "   "    "
GAMA    RESW1   "   "    "
INCR    RESW1   "   "    "
DELTA   RESW1   "   "    "
```

*Write a sample program for Indexing and looping Operations for SIC/XE version.

```
LDA   Zero    Initialize value zero to Register A.
STA   Index   Store Index value in, Register A
LDX   Index   Load Index value in to Register X.
ADDLP LDA  AlphA,x  Load word from Alpha in to Register A
ADD   BETAX  Add word from Register A
STA   GAMAX  Store the Result in Gama
LDA   Index  Load Index value in to Register A
ADD   Three  Add value 3 in to Index
STA   Index  Store value Index in to Register A
comp  K300   compare new Index value to 300
ILT   AddLP  Loop if Index is less than 300.
      .
      .
      .
Index  RESW 1   one word variable for Index value
ALPHA  RESW 100  Array variable 100 for each word
Zero   WORD 0    one word constant
K300   WORD 300  one word constant
Three  WORD 3    one word constant
Indexing and Looping operations using SIC/XE
LDS    #3     Initialize value 3 to Register S
LDT    #300   Initialize value 300 to Register T.
LDX    #0     Initialize value 0 to X.
ADDLP LDA  ALPHA,X  Load word from Alpha in to Register A
ADD    BETA,X  Add word from Beta in to Register A
STA    GAMA,X  Store the result in Gama
ADDR   3,X     Add 300 to Index
COMPR  X,T     compare the new Index value to 300
ILR    ADDLP   Loop if index value is 300.
       .
       .
       .
ALPHA  RESW 100  Array variables 100 for each word
BETA   RESW 100  Array variables 100 for each word.
GAMA   RESW100   Array variables 100 for each word.
```

* CISC and RISC Architecture:-

* CISC Architecture:-

CISC Architecture stands for "complex Instruction Set computer". CISC is a processor design where single instruction can execute several low level operations or capable of Handling Multistep operations or Addressing Modes with in a single Instruction. This term was contrast to the RISC and therefore became an Ambiguous.

In 70's hardware oriented Approach has give rise to CISC Architecture in which more complexities have been added in the hardware to allow the software to be simpler. Most of the instructions are implemented by using Hardware.

For example,

System/300 through 'z' architecture, PDP-11 and VAX Architecture, Data General Nova [DGN] and many other are the well known processors and micro controllers that have been labelled to use CISC and also includes Motorola 6800, 6809, 68000 and its families.

* RISC Architecture:-

RISC Architecture stands for "Reduced instruction set computer". RISC is a C.P.U designed strategy based on the simplified instruction set provides high performance when combined with a micro processor Architecture. It is capable of executing instructions using fewer micro processor cycles per Instruction.

A computer based on this strategy (i.e). a reduce instruction set computer are called RISC processors. The opposing Architecture of RISC is called as CISC

Apple's software approach has led to the Introduction of RISC Architecture which utilises a small but highly optimised set of instruction. It has been found that 80% of work can be performed very fastly by using this computer The main principle of RISC is that it takes long & complex instructions from CISC and reduces to shorter & simpler instructions which can execute fast. Although RISC machines are less complex & less expensive, their plans extra demand on programmers to implement complex instructions by using simple instructions.

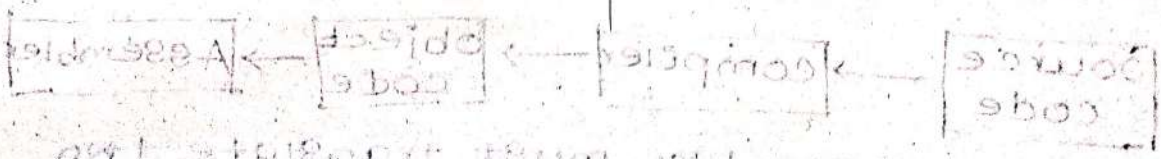
Ex:-

RISC families includes DEC corporation, ALPHA, AMD, AM2900 /ARC, ARM, ABR, INTEL 1860, INTEL 1960 & MOTOROLA 88000.

In 21st century the use of ARM Architecture processors are all used in smartphones, tables, computers, i-Phones i-pad, android, android devices, which provides base for RISC.

* It emphasis on hardware components

* It emphasis on hardware components that includes multi clock complex instruction	* It emphasis on software components that includes single clock reduced instructions
* memory to memory load and store are incorporated through instructions	* Register to register load & store that are independent of instructions
* small code size and have high clock cycle per second	* Large code size & lower clock cycle per second
* Transistors are used for storing complex instructions	* It uses more transistors & memory register
* It has been introduced by INTEL corporation	* It has been introduced by Apple corporation
* In CISC, complex instruction are executed which takes much time.	* In RISC complex instructions are reduced & executed which takes less time
* It is the oldest version of the Architecture	* It is the modern & updated version of the Architecture.



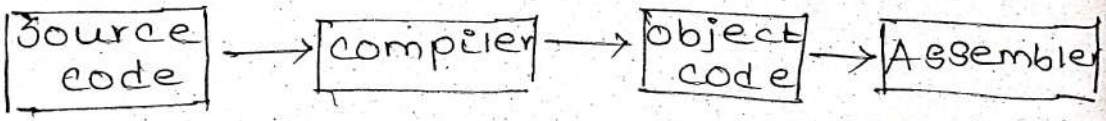
Chapter-2
Assemblet

*Assembler:-

An assemblet is a type of computer program (or) system software that converts Assembly language in to machine code. It takes the basic commands & operation from assembly code and converts them into binary code that can be recognised by a specific type of processor.

Assembles are similar to compiler they produces executable code & assembler provides assemble code. However assembles are more simplistic since they only convert low level code [Assembly code] to machine code

Most programs are written in high level programming languages & are compiled directly to machine code using a compiler. However, in some cases, the assembly code may be used to customize functions & ensure that they perform in a specific way.



The Assembler must translate two different kinds of symbols like Assembler defined symbols & programmer defined symbols. The Assembler defined symbols are NEMNOMICS for the machine instructions & pseudocode instructions.

Characteristics of Assembler:-

*It allows complex jobs (or) takes to run in a simpler way.

*It is memory efficient as it requries less memory

* It is faster in speed as its execution time is less.
* It is mainly hardware oriented
* It requires less instruction to get the results
* It is not require to keep track of memory locations.
* It is a low level embeded system.

Features of Assembler:-

* It can use MNEMONIc then numeric operation code & it also provides the information of any error in the code.
* This language helps in specifying the symbolic operand that means it doesn't need the specifying the need address of that operand. It can be represented in the form of symbol.
* The data can be declared by using decimal notation function
* It converts the data constants in to internally machine representation.
* It converts NMEMON operation code to that machine language equivalent.
* It converts symbolic operands to their equivalent machine address.
  Ex.- variable names
* It builds the machine instructions in a proper format

Basic Assembler functions:-

(1) It translates MNEMONIC Language, to its equivalent object code [machine code]
(2) It assigns machine address took symbolic cables.

There are different types of basic assembler functions.

They are: 1. Assembler directives [pseudo code]
(2) Data transfer [RD/WD]
(3) Subroutine function [JSUB, RSUB]
(4) Translation function

Assembler directives:-

It is also called pseudocode instructions that cannot be translated in to machinary code. these instructions allows the Assemblers to perform the basic operations.
It provides information to the assembler.
Ex:- START, END, BYTE, WORD, RESW, RESB.
START:- It specifies the start time M.A of object program
END: It marks & indicates the end of the program
BYTE:- It generates the character in hexa decimal constant [1 byte = 8 bits]
WORD:- It generates one word constant
RESW:- It reserved word the indicated no. of words for data area
RESB:- It reserved bytes the indicated no. of bytes for data area.

Data transfer [RD/WD]:-

* It allows to transfer the Input & output to and from the memory
* A Buffer is used to store the record.
* The end of each recorded is marked with null character.
* The Buffer length is 4096 bytes = 4KB
* The end of the file is indicated with 0 length record
* RD & WD is an instruction that is used to read the input & ~~output~~ write the output data.

Subroutine function:-

* It is used to read the record [RD, REC) and write the record (WR, REC)
* Sub routine function is mainly used to access the calling function & called function
* JSUB is used to jump the subroutine functions & RSUB Return to the subroutine function
* These subroutine function save the register with linkage 'L'

Translation function:-

* It is mainly used to translate the object code in to assembly code & vice versa.
* It translates STL to 14 (storage location)
* It translates return address [RETADDR] to 33
* It builds the machine instructions in proper format
* It translates EOF to 454 [End of the file] to 454/46

The following example shows an assembler language program for SIC version

Line	SOURCE	STATEMENT
5	COPY START	1000 copies from Input to output
10	FIRST TL	RETADDR Stores the return address
15	CLOOP JSUB	RDREC Read the input record
20	LDA	Length Length of Record
25	comp	Zero
30	JEQ	ENDFIL Exit if EOF is found.
35	JSUB	WRREC Write output record
40	J	CLOOP Loop
45	ENDFIL LDA	EOF End of file
50	STA	Buffer
55	LDA	THREE set length = 3
60	STA	Length

```
65      JSUB  WRREC [Write the record]
70      LDL   RETADDR get the return address
75      RSUB  -       return to the caller
80      EOF   BYTE    C'EOF'
85      THREE WORD    3       one word constant
90      ZERO  WORD    0       one word constant
95      RETADDR RESW  1
100     LENGTH  RESW  1
105     BUFFER  RESB  4096    4096 Byte Buffer Area
```

Write a program in Assembly language to use Sub routine function to read the record into buffer

```
LINE    SOURCE    STATEMENT
110     -
115     -
120     -
125 RDREC   LDX   Zero clear loop counter
130         LDA   Zero clear 'A' to zero
135 RLOOP   TD    Input Test Input device
140         JEQ   RLOOP Loop until ready
145         Input read input character
                  into Register A
150         Zero  Test for end of the
                  record
155         EXIT  Exit Loop
160         Buffer store a character
                  in buffer
165         MAXLEN loop unless max, length
                  has been reached
170     EXIT  JLT RLOOP
175     EXIT  STX Length Save the record length
180     RSUB  -   -   return to the caller
185     INPUT BYTE
```

A FORWARD REFERENCE:-

* It is a referrence to enable that is define later in the program.
* most of the Assemblers makes two passes over the source program such as pass1 & pass2
* pass 1 is used to scan the source lables definition & assign address
* pass 2 performance most of the actual translations
* The functions of two pass is as follows
* pass-1:

It defines symbols & literals and remember them in symbol table & literal table Respectively

* It assigns address to all the statements
* It process pseudo operations
* It checks the correctness of instruction
* It saves the value assigned to all lables into the symbol table for pass 2
* It keeps track of location contour

* pass-2:

* It generates the object code by converting Symbolic opcodes into respective numeric opcodes
* It generates data values defined by byte, Word
* It performs processing of assembler directives which is not done during pass 1

A simple SIC assembler:-

* The object program will be loaded into memory for executed
* In simple assembler these are three types of records such as,

(i) Header: It contains the program names, starting address, length

(ii) Text: It contain starting address. Object code length.

(iii) END :- It contains the address of first executable instruction.

Assembler tables & logic :-

our simpler assembler uses two internal tables such as (1) OP TAB
(2) SIM TAB

OPTAB :-

OPTAB is used to look in (or) store all by nemonic operation code translated them into and its equivalence

ex :- LAD → STL → 14

SIMTAB :-

SIMTAB is used to store the values assign to tables

ex :- copy → 1000, first → 100

1. OPTAB :-

* It is nothing but operation table
* It contains instruction format and length
* pass 1, opTAB is used to look up and validate operation codes
* pass 2, opTAB is used to translate the operation code in to machine language code
* In SIC/XE assembler searches optab in pass 1 to find the instruction length for implementing location counter [LOCCTR]
* It is organised as a Hash Table.

2. SIMTAB :-

* It is nothing but symbolic table
* It contains length of a instruction
* It includes flags to indicate error conditions
* pass 1, Lables are entered in to SIMTAB along with the assigned address
* pass 2, symbols are used as are operands that look up in the SIMTAB to obtain the address
* It is organised as Hash Table

LOCCTR:-

* It is the variable for assignment address
* LOCCTR is initialised to address specified Location
* When we reach a table, the current value of LOCCTR gives the address to be associated with the table.

Machine dependent and Machine Independent assembler features,

Machine dependent assembler features:-

* There are some assembler features which are machine dependent such features are called machine dependent assembler features
* Machine dependent assembler features enables the assembler by executing the assembly language instructions in an efficient manner
* It enhances the efficiency the performance of computer by executing the instruction very fastly that takes less time.
* There are different types of machine dependent assembler features such as:

(1) Instruction format & Addressing mode.
(2) program relocation

(1) Instruction format & Addressing mode:-

a) It specifies the format of an instruction that is represented in bits (or) bytes.
b) In SIC machine, the instruction is 24 bit format using such as fields, opcode, Index Register [X] & Addresses
(3) In SIC/XE machine, the instruction format using 32 bit format for storing the instruction and it uses several fields like opcode, n, x, i, b, pc and displacement as address.

(4) It uses program counter relative addressing, base relative addressing that is defined as op-M.
(5) It uses Indirect Addressing that is defined as op@M.
(6) It uses Immediate Addressing that is defined as op#M.
(7) It uses extended format that is defined as +opM
(8) It uses index addressing that is defined as opM,X
(9) It uses Register to Register instruction that is defined as COMPR
(10) It uses larger memory for multiprogramming Allocation.

Register Translation:-

(1) It Translates the Register in to Relevant object code.

Ex:-

A	X	L	B	S	T	F	PC	SW
↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	8	9

(2) These Registers object code values are preloaded in SIMTAB

Address Translation:-

(1) most memory instruction uses pc relative (or) base relative addressing
(2) It supports format-3 with 12 bit displacement address speed
(3) It uses pc Relative that ranges from -2048 to 2047
(4) It uses base relative that ranges from 0 to 4095
(5) It uses format-4 with 20 bit displacement field.

Assembler design options:-

⇒ In computer every Assembler has various design options that are used to perform various operation
⇒ The design option helps the user (or) machine to scan all the statements, Labels addresses and converting them into its corresponding mnemonic [pseudo] codes, and stores them in the operation table [opTAB].
⇒ It also converts the symbolic statements in assembly code into its corresponding equivalent symbolic codes and stores in symbolic table (or) symbol table [SYMTAB]
⇒ The design option of assembler allows to perform further translations easily and effectively.
⇒ There are various design options in assembler that are used by machines. They are:-
(1) one pass Assembler
(2) Two pass Assembler
(3) multi pass Assembler

(1) one pass Assembler:-
(i) A single pass Assembler scans the program only once and create equivalent binary program.
(2) A one pass assembler passes over the source file exactly once, in the same pass collecting the labels resolving future references and doing the actual assembly
(3) The difficult part is to resolve future label referrences i.e, it prohibits forward referrences.
(4) It vomits the operand address if the symbol has not yet been defined.

(2) TWO pass Assembler:-
(1) The first type produces the object code directly in the memory for execution using load and go assembler.
(2) NO object program is written out and no Loader is needed.

(3) It can save the time to
code only once.
(4) The second type produces the usual kind of
object program for later referrence.

Two pass Assembler:-
(1) Most Assembler uses Two pass Assembler to
perform its operation.
(2) The processing of source code into two passes
is called Two pass Assembler.
(3) In other word, the two pass Assembler uses
two passes such as pass 1 and pass-2 and
scans the source program twice.
(4) The internal table [OPTAB & SYMTAB] &
subroutines that are used only during
pass-1.
(5) The SYMTAB, LITTAB and OPTAB are used by
both the passes.

pass-1:-
(1) It scans and assigns the addresses to all
statements in the program
(2) Addresses of symbolic labels are stored
(3) Some Assembler directives will be processed

pass-2:-
(1) It translates opcode & symbolic operands
in to equivalent machine code.
(2) It generates data value defined by Byte,
Word etc..
(3) It writes the object program and assembly
Listing.
(4) The main advantage of Two pass Assembler is
that it allows forward referrence in source
code because when the assembler is
generating the code it has already found all
referrences.

The drawback is it takes lot of time

Multipass Assembler:-

(1) Multipass Assembler means more than one pass is used by the Assembler.

(2) It is used to eliminate forward references in symbol definition.

(3) It creates an number of passes that is necessary to process the defenition of symbols.

(4) Multipass Assembler does the work in two pass by resolving forward referrence.

Pass-1:-

(1) It scans the code, validate the tokens and create opcode & symbol table

Pass-2:-

(1) It solves forward referrences and converts the pseudo code to machine code

*Implementation examples:-

There are two implementation example of assembler such as:-(1) MASM Assembler [Microsoft Macro Assembler]

(2) SPARC Assembler [Scalable processor Architecture]

(1) MASM Assembler:-

Some of the features of MASM Assembler is as follows:-

*A MASM Assembler language program is written as a collection of segments.

*Each segment is defined as a particular class, corresponding to its contents. commonly used classes are CODE, DATA, CONST and STACK.

*During the program execution segments are addressed through 'x86' segment register.

*In most cases code segments are addressed using register 'CS' and stack segments are addressed using 'SS'. These segments are automatically set by the segment loader when a program is loaded for execution.

*Data segments are normally addressed using DS (or) ES (or) FS (or) GS

* By default, the assemblers assumes that all referrences to data segments use register 'DS'.
* It tells the assembler that register 'DS' indicates the segment Data seg 2.

Ex:-
DATASEG2

* Registers ES, FS and GS must be loaded by the program before they can be used to address data segments.

Ex:- MOV ES, DATASEG2
MOV AX, ES

* MASM can also produce an instruction listing that shows a number of clock cycle required to execute each machine instruction.

(D) SPARC Assembler:-

* Some of the features of Sun operating System SPARC Assembler is as follows,
* A SPARC Assembler language program is divided in to units called 'sections'. The assembler provides a set of predefined section name which are as follows,

TEXT:- Executable instruction
DATA:- Initialised Read/write data
RBDATA:- Read only data
BSS :- uninitialised data area.

* It is also possible to define other sections specifying section attributes such as executable and writable
* The Assembler maintain a seperate location counter [LOCCTR] for each named section.
* Each time the assembler switches to a different sections, it also switches to the Location counter.
* The Referrences between different section are resolved by the linker, not by the assembler.

Machine Independent assembler features:-

* There are some features which behaves independent of machine such features are called machine independent Assembler features.
* machine independent assembler features donot involve in representing in the instruction formats addressing mode etc
* It mainly focusing on representation of literals on its operands.
* It also focus on representing symbol defining statement that represent the statement in the form of symbols
* It also uses program blocks & control sections as well as program linking which are specifically independent of machine characterstic.
* There are different types of machine independent assembler feature. They are (1) Literals
  (2) Symbol defining statement
  (3) expressions
  (4) program blocks.
  (5) control sections & program linking

Literals:-

⇒ It defines a constant value explicity and assign an address label for it.
⇒ It uses the table as the instruction operan
⇒ A Literal is identefy with the prefix symbol '=' following by a specification of the literal value.

Ex:- LDA = c 'EOF'
LDA = X '05'

⇒ It is convinient to write the value of a constant operand as a part of instruction

⇒ All the literal operands are combined together into one (or) more literal pools.
⇒ A literal can be identified with literal name, it's operand value & length & address.

*Symbol defining statement:-
1- It allow the programmer to define symbols and specify their values.
Syntax:- symbol EQU value
ex:- Replace LNT #4096 with MAX Length EQU 4096
2) It improves the program readability and makes it easier to find and change constant values
(3) It defines mnemonics names for registers.

*Expressions:-
(1) The assembler allows the use of expression as operand.
(2) A expression is nothing but combination of operators and operands.
(3) The Assembles evaluates the expression and produces assign operand address (or) value
(4) An expression consists of operators (+, −, *, /) individual term such as constants user defined symbols etc
(5) An expression can be absolute that can be represent in relative terms in pairs with opposite signs for each pair.

*program blocks:-
(1) The program block is nothing but ~~block~~ block of statements contained in the program.
(2) It uses an assembler directive that contains block name with its keyword 'use'.

Syntax: USE[Blockname]

(3) At the beginning statements are assumed to be part of unnamed or default block

(4) It seperates the program blocks in a particular order

(5) Each program block may Actually contain several seperate segments of the source program.

(6) It enhances the program Readability by dividing the program into seperate blocks

(7) If 'no use' keyword statement are included the entire program belongs to single blocks.

* control section and program linking :-

(1) A control section is a part of program that maintains its identity

(2) A Large program can be divided into control sections

(3) Each control section can be loaded and Relocated Independently

(4) Different control sections are used for subroutines (or) other logical subdivisions of a program

(5) The programmer can Assembles load and manipulate each of the control sections separately

(6) Symbol that are defined in control section cannot be used directly by Another control section.

(7) It uses different symbols like START, CSECT, EXTDEF and EXTREF.

(8) START is used to start the first control section

(9) CSECT is used to start new control section.

(10) EXTDEF [External definition] is used to identify the symbols that are defined in control section

(11) EXTREF [External References] which is used to identify the symbols used in other control sections.

*Program Linking:-

(1) Linking is the process in which Reference to externally defined objects [code and data] are processed so as to make them operational.

(2) Traditionaly linking is used to perform a task after basic Translation of program file.

(3) There are two types of linking such as Static linking and dynamic linking.

(4) In static Linking, all the moduls that are required to complete a programs are physically placed together which remains static.

(5) In dynamic linking the actual task of linking is performed prior to running the program and individual modules are dynamicaly linked during execution.