

LIST OF EXPERIMENTS AND RECORD OF PROGRESSIVE ASSESSMENT

Sr. No.	Name of experiment	Page No.	Date of performance	Date of submission	Assessment Max.10 marks	Sign. of teacher and remark
1	To understand different classes of system softwares	9				
2	To study the internal structure of computer system and its assembly language (Model IBM 360/370)	15				
3	To understand table processing techniques using searching methods	23				
4	To understand table processing techniques using sorting methods	29				
5	To understand assembler design and implementation steps	35				
6	To understand the design and implementation of various phases of compiler.	45				
7	To understand the design of loader and linkage editor	53				
8	To understand macros and basic design of macro processor	59				
				Total marks Average marks out of 10*		

* To be transferred to proforma of CIAAN-2006 (Proforma A-1/A-2)

Note : The curriculum of this subject is referred and the above list of experiments is finalized to achieve the desired objectives.

EXPERIMENT No. 1

1.0 Title:

To understand different classes of Computer System Software

2.0 Prior concepts:

- Computer System organisation
- Hardware / software components
- programming language processors
- Assembly Language programming
- Operating system environment.

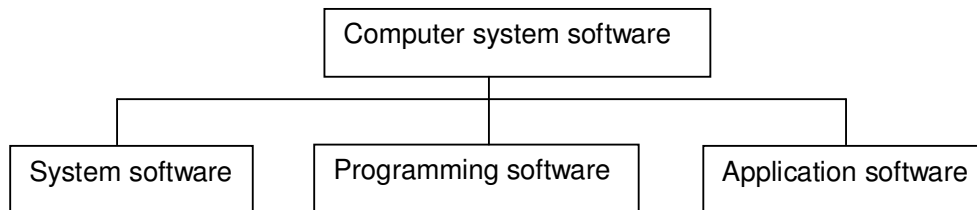
3.0 New concepts:

Proposition 1 : Computer Software

It is a general term used to describe a collection of computer programs, procedures and documentation that perform some tasks on a computer system. It is divided into three classes:

1. System software.
2. Programming software
3. Application software.

Concept structure



• **System software:** It is any computer software which manages and controls computer hardware so that application software can perform a task. System software performs tasks like transferring data from memory to disk ,disk to memory, or rendering text onto a display device.

Examples:

1. Compilers /Interpreters
2. Assemblers
3. Loaders/Linker
4. Text editors
5. Debuggers
6. Device drivers
7. Operating systems. etc

Two major categories of system softwares are

1. Machine Dependent System Software

- These softwares rely on Internal physical architecture of machine (Machine code, Instruction format ,Addressing mode , Registers.)

2. Machine Independent System Software

- These softwares do not rely on internal physical architecture of machine (Machine code, Instruction format Addressing mode, Registers)

• Programming software:

It is usually provides tools to assist a programmer in writing computer programs and software using different programming languages in a more convenient way.

Examples:

Different Kinds of the tools include:

1. End-User Programming:
 - Word, Excel, Paint, Chat, Explorer etc
2. Utilities and Analysis Programming:
 - BASIC ,Visual Basic, C,C++,PASCAL,COBOL,FORTRAN, JavaScript , HTML, etc

• **Application software:**

It is a set of programs comprising various modules that are written for specific computer application.

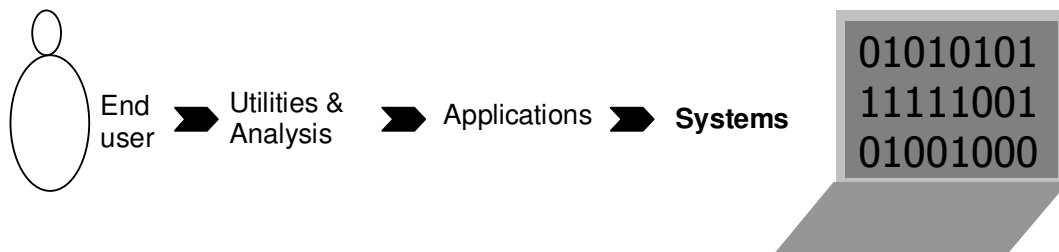
Examples

- Banking, Inventory control ,Payroll , Engineering, Education, GIS etc

Proposition 2 : System software

System software consists of a variety of programs that support the operation of a computer and enables the end-user to perform specific, productive tasks

Concept structure



Main goal of system software is to reduce the communication gap between humans and computers.

System Software performs variety of functions:

- File editing
- Resource Accounting
- I/O management
- Storage management
- Complete various stages of its processing by computer system like – translation relocation, linking, editing, eventual execution

Proposition 3: System Programming and its components

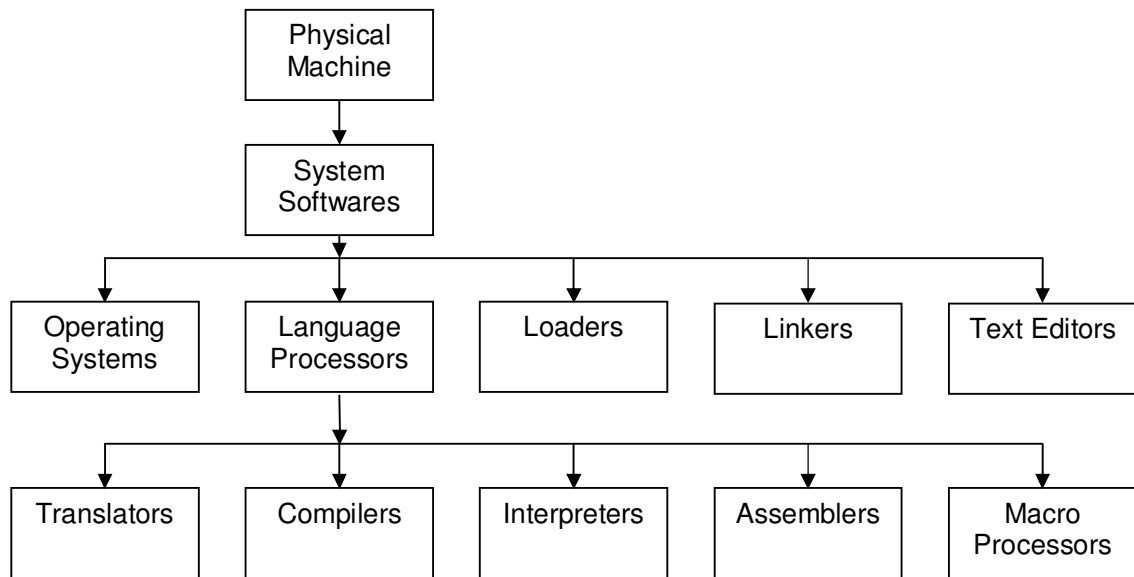
System Program: It is a program which aids in effective execution of general user’s computational requirements on a computer system.

Systems programming: It is the activity of designing and implementing the system programs.

People						
Application Programming						
Compiler		Assembler		Macro Processor		
Loader	Text Editor		Debugging aids		Searching & Sorting	
I/O Programs		File System	Scheduler	Libraries	Memory Management	
					Device Management	

Foundation system programming

Components of computer system programming:



- **Operating system:** It is a system software that provides easy interface for user to communicate with computer system. It also manages all the resources of computer system.
- **Language processor:** It is a system software used to convert / translates one form of language into another form. A language processor may fall into any one of the following categories.
 - i. Translator
 - ii. Assembler
 - iii. Compiler
 - iv. Interpreter

Translator: It is a general term used for converting one form of computer language into another form. The input text of the translator is called as source program and output text is called as object program. The output of translator needs not always a machine code.

Compiler: It is a system software that accepts a source program “In a high-level language” and produces a corresponding object program.

Interpreter: It is a system software that accepts a source program “In a high-level language” and produces a corresponding object program .The major difference between compiler and interpreter are

- i) Translates source program line by line.
- ii) Detects one error at a time.

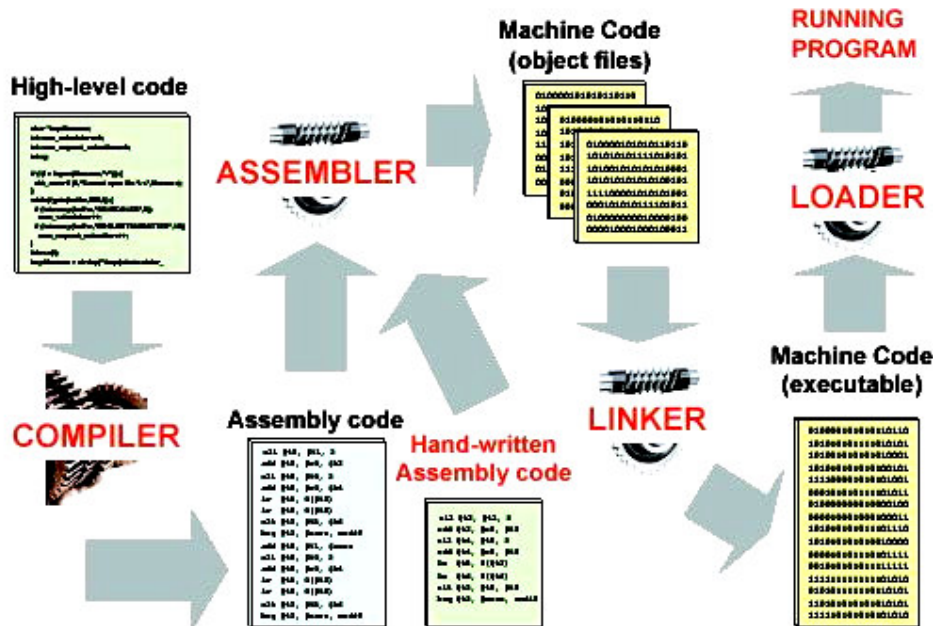
Assembler: It is a system software that accepts a source program in assembly language and produces a corresponding object code

Macro Processor: A Macro call is an abbreviation (or name) for some code. A macro definition is a sequence of code that has name (macro call). Macro Processor is a system software which performs macro expansion (in assembly language) for every occurrence of macro call.

- **Loader:** It is a software which accepts the object programs, prepares these programs for execution and loads them into core memory .
- **Linker:** It is a system software that links various subroutines to the main program with respect to memory addresses.
- **Text Editor:** It is a system software that creates and manages text files.

4.0 Diagram:

The structure explaining steps between the HLL to executable program and role of System Programming in its working sequence.



5.0 Learning Objectives:

Intellectual skills

- To identify different System Software / Programs
- To identify System Components: hardware, software, programs

Motor skills

- Ability to handle different System Software.

6.0 Conclusion:

- To control hardware system while offering services to its user is the function of system software / software system / application software

.....

 • type of programmer must know the internal (hardware) structure of computer system .

7.0 Laboratory practice:

1. Create, delete, rename, copy the files/directories using
 - a. MS-DOS
 - b. MS Windows
 - c. Linux
2. Develop any two programs in C/C++ and VB and debug, compile, execute the same programs.
3. Write assembly language program for addition of two eight bit numbers and execute it using
 - a. MASM software
 - b. 8086 assembler kit

8.0 Questions:

1. What is system software?
2. List out different system software?
3. What do you mean by programming?

4. List all kinds of software in chronological manner.
5. Why should you study the system programming?
6. What is the difference between system software and application software?
7. List the name of operating system, language processors, utility programs in your lab. Write their features.
8. Whether BIOS program and Bootstrap loader are system software?
Justify your answer
9. Select from given options which one of the host language uses to develop Compiler or interpreter.
a. C/C++ b. visual basic c. JAVA d. FORTRAN e. COBOL f. PASCAL
10. Write correct sequence of the following process in program executions.
a. compiling b-coding c-linking d- execution/ run e - loading f- debugging

(Space for Answers)

(Space for Answers)

EXPERIMENT No. 2

(Not to be considered for internal assessment)

1.0 Title:

To study the internal structure of computer system and its assembly language
(Model IBM 360/370)

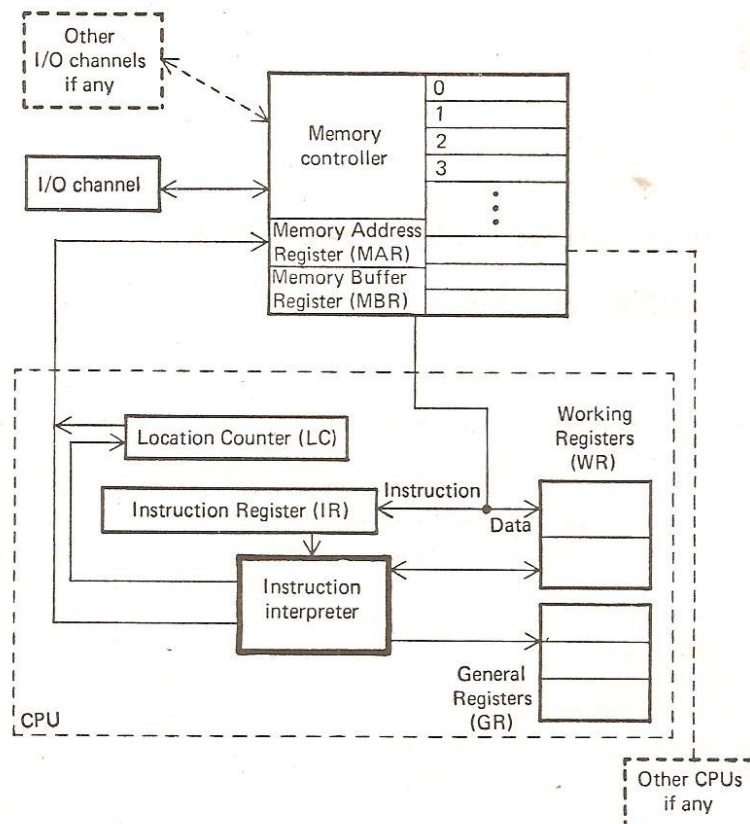
2.0 Prior concepts:

- Internal structure of 8085/8086 processor
- Assembly language programming

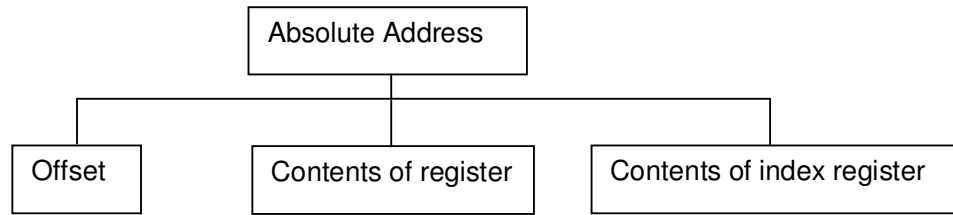
3.0 New concepts:

Proposition 1 General machine structure of IBM 360/370

- The IBM 360/370 is based on the “stored program computer” concept.
- Major functional blocks:
 - Instruction interpreter (II) interprets (understand) the intent of instruction fetched from memory
 - Location counter(LC) - Stores the address of the instruction to be executed next.
 - Working register(WR) - Scratch pad registers used for temporary storage
 - General register - Storage register used by programmer
 - Memory address register (MAR) - Contains the address of memory location that is to be read from or write into
 - Memory buffer register (MBR) - Contains copy of designated memory location specified by MAR
 - Memory controller - Transfers data between MBR and memory according to the address given by MAR



Memory Every memory location consists of 8 bit information. Memory address space 2^{24} byte. The addressing on 360 memory consists three components

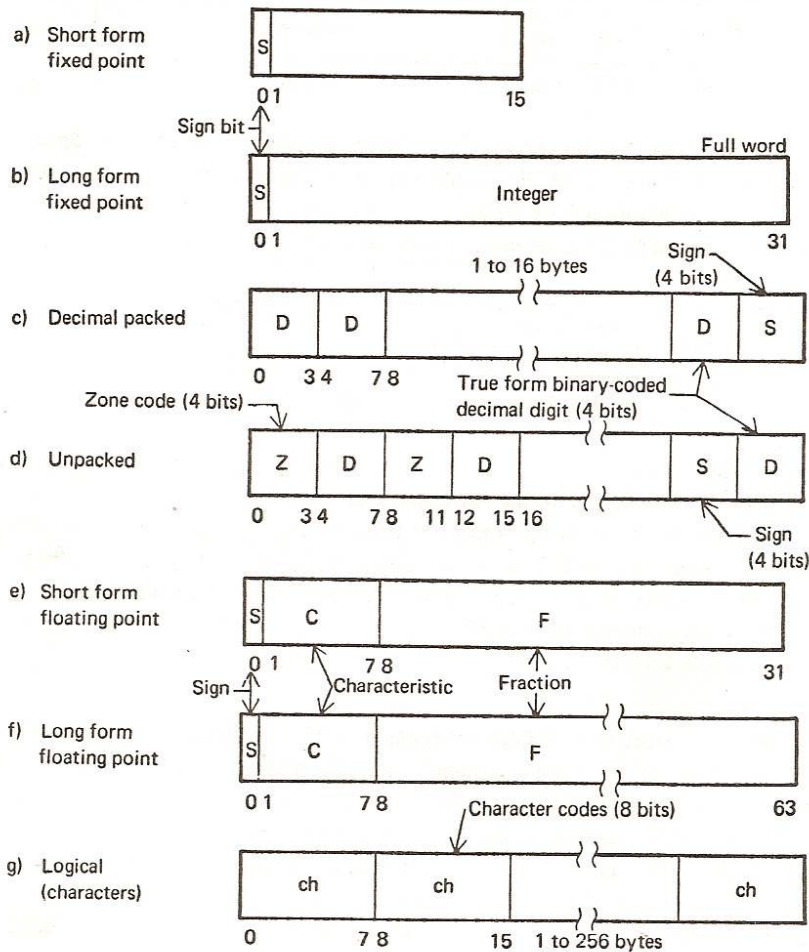


Registers 16 general purpose registers (32 bit each) The GPRS may be used as base register or index.

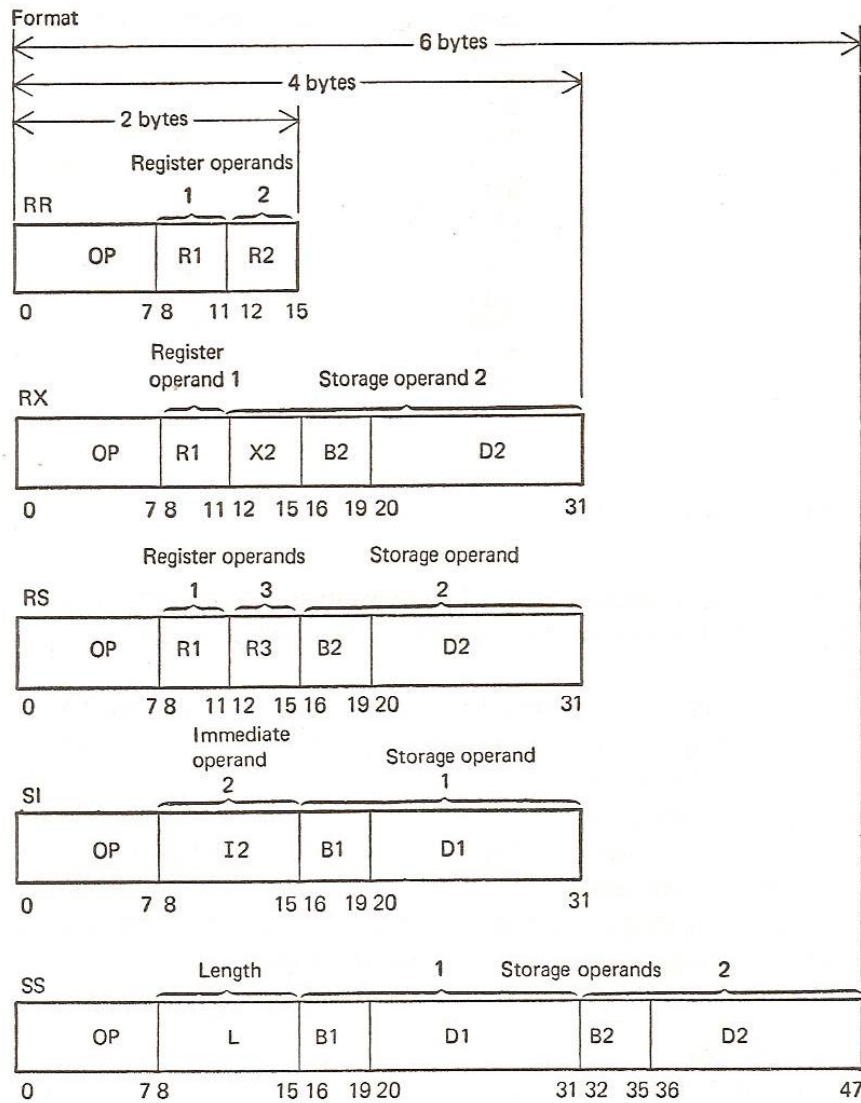
4 floating point registers (64 bit each)

1 program status word (64 bit)

Data The 360 operates on following data formats



Instructions The 360 instruction formats are as follows



Mnemonics used:

- OP ~ operation code
- Ri ~ contents of general register used as operand
- Xi ~ contents of general register used as index
- Bi ~ contents of general register used as base
- Di ~ displacement
- li ~ immediate data
- L ~ operand length

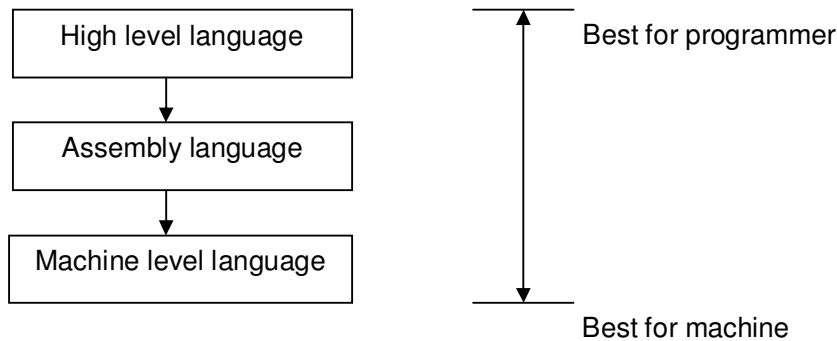
The 360 has arithmetic, logical control or transfer and special instructions.

	Hexadecimal op code	Mnemonic	Meaning (format)	
Load-Store-Register ↑		Load group		
	58	L	Load (RX)	
	48	LH	Load half word (RX)	
	98	LM	Load multiple (RS)	
	18	LR	Load (RR)	
	12	LTR	Load and test (RR)	
↓		Store group		
	50	ST	Store (RX)	
	40	STH	Store half word (RX)	
	90	STM	Store multiple (RS)	
	Fixed point arithmetic ↑		Add group	
		5A	A	Add (RX)
4A		AH	Add half word (RX)	
1A		AR	Add (RR)	
		Compare group		
59		C	Compare (RX)	
49	CH	Compare half word (RX)		
19	CR	Compare (RR)		
↓		Divide group		
	5D	D	Divide (RX)	
	1D	DR	Divide (RR)	
		Multiply group		
	5C	M	Multiply (RX)	
	4C	MH	Multiply half word (RX)	
1C	MR	Multiply (RR)		
↓		Subtract group		
	5B	S	Subtract (RX)	
	4B	SH	Subtract half word (RX)	
1B	SR	Subtract (RR)		

	Hexadecimal op code	Mnemonic	Meaning (format)	
Fixed point arithmetic	55	CL	Compare logical (RX)	
	D5	CLC	Compare logical (SS)	
	95	CLI	Compare logical (SI)	
	15	CLR	Compare logical (RR)	
		Move-group		
	D2	MVC	Move (SS)	
	92	MVI	Move (SI)	
		And-group		
	54	N	Boolean AND (RX)	
	D4	NC	Boolean AND (SS)	
	94	NI	Boolean AND (SI)	
	14	NR	Boolean AND (RR)	
		Or-group		
	56	O	Boolean OR (RX)	
	D6	OC	Boolean OR (SS)	
	96	OI	Boolean OR (SI)	
	16	OR	Boolean OR (RR)	
	Exclusive-group			
57	X	Exclusive-or (RX)		
D7	XC	Exclusive-or (SS)		
97	XI	Exclusive-or (SI)		
17	XR	Exclusive-or (RR)		
	Shift			
8D	SLDL	Shift left (double logical) (RS)		
89	SLL	Shift left (single logical) (RS)		
8C	SRDL	Shift right (double logical) (RS)		
88	SRL	Shift right (single logical) (RS)		
	Linkage group			
45	BAL	Branch and link (RX)		
05	BALR	Branch and link (RR)		
	Branch group			
47	BC	Branch on condition (RX)		
07	BCR	Branch on condition (RR)		
46	BCT	Branch on count (RX)		
06	BCTR	Branch on count (RR)		
Transfer				

	Hexadecimal op code	Mnemonic Compare-group	Meaning (format)
Miscellaneous ↑ ↓	9E	HIO	Halt I/O (RX)
	41	LA	Load address (RX)
	9C	SIO	Start I/O (RX)
	0A	SVC	Supervisor call (SI)
	9D	TIO	Test I/O (RX)
	43	IC	Insert character (RX)
	91	TM	Test under mask (SI)
	42	STC	Store character (RX)

Proposition 2 : Assembly language of IBM 360/370



An example of assembly language program

To write a program that will add 49 to the content of 10 adjacent full words in memory under the following set of assumptions.

- a. The 10 numbers to be added are in contiguous locations beginning at absolute location 952.
- b. The program in core memory is steaming at absolute location 48.
- c. '49' is at absolute location 948.
- d. Register 1 contains 48.

	Programs	Comments
	START	Identifies names of program
	BALR 15,0	Start register 15 to address of the next instruction
	USING	Pesudo-op indicating to assembler register 15 is base register and its content is address of next instruction
	BEGIN+2,15	
	SR 4,4	Clear register 4 (set index=0)
LOOP	L 3,TEN	Load the number 10 into register 3
	L 2,DATA(4)	Load data (index) into register 2
	A 2,FORTY9	Add 49
	ST 2,DATA(4)	Store updated value of data (index)
	A 4,FOUR	Add 4 to register 4 (set index = index+4)
	BCT 3,LOOP	Decrement register 3 by 1, if result non zero, branch back to loop
TEN	BR 14	branch back to caller
FOUR	DC F'10'	Constant 10
FOURTY9	DC F'4'	Constant 4
DATA	DC F'49'	Constant 49
	DC F'1,3,3,3,3,4,5,8,9,0	Words to be processed
	END	

4.0 Learning Objectives:**Intellectual skills**

- To know the working of registers and functional blocks in IBM 360/370
- To know the memory, data formats, instructions formats and addressing of IBM 360 / 370

Motor skills

- Ability to develop assembly level language programs (IBM 360/370)

5.0 Laboratory practice:

1. Write an IBM 360/370 ALP to add two 8 bit numbers
2. Write an IBM 360/370 ALP to multiply two 8 bit numbers

6.0 Conclusion:

- The IBM 360/370 is based on concept.
- In RX instruction format, to get a absolute address, the offset is added to contents of register andregister.
- Relative addresses/absolute addresses are the actual addresses of core memory

7.0 Questions:

Write answers to Q.....Q.....Q.....Q.....and Q..... (Teacher shall allot the questions)

1. Define
 - High level language
 - Assembly language
 - Machine language
2. What do you mean by
 - Machine -OP
 - Pseudo-OP
 - Literal
 - Symbol
3. Why IBM 360/370 uses 12 bit offset instead of 24 bit offset?
4. What is the use of BASE register and index register?
5. State the difference between USING and BALR pseudo-OP
6. How 8088's addressing differs from IBM 360/370?

(Space for answers)

(Space for answers)

EXPERIMENT No. 3

1.0 Title:

To understand the Table Processing Techniques using Searching Techniques.

2.0 Prior concepts:

- Data Structure concept (Array, Pointer, Stack, searching and sorting)
- Basic Assembly language 8085/86 or IBM 360/370 .

3.0 New Concepts:

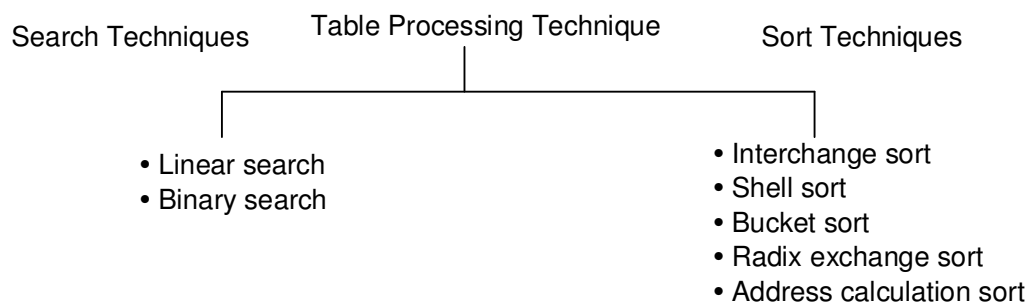
Proposition1: Table Processing Technique

It is often required to maintain large tables of information in such a way that items may be moved in and out quickly and easily. For example, in assembler symbol table is composed of multiple-word entries in a fixed format. In the table there is, symbol name (key), its value, and re-locatability. It is necessary to maintain large tables of information in such a manner of symbol corresponding location and its value.

Table processing technique classified as given below

1. Search techniques
2. Sort techniques

Concept structure



Proposition 2 : Searching Technique

The problems of searching is as follows

1. More than one entry with the same keyword (for defined and undefined symbols)
 2. No entry found (for defined and undefined symbols)
- required individual treatment depending on the function of the table searching has two different class

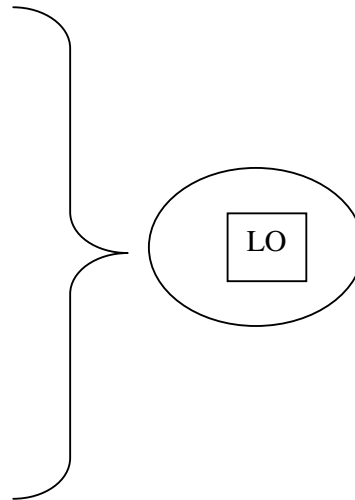
- Linear searching
- Binary searching

Linear searching: It is a simple searching technique. This technique uses table in which the items have not been ordered .One way to look for a given keyword is to compare every entry in the table with a given keyword. This procedure is fine for short tables and has the simplicity in processes, but for long tables it can be very slow.

Illustration of linear search:

Symbol Table

Number	Symbol
1	TI
2	EX
3	FN
4	RN
5	OR
6	IW
7	LE
8	LO
9	NC
10	OP
11	IF
12	RD
13	FU
14	TE
15	AL



The symbol LO is compared with each symbol in the table sequentially. If the symbol LO is found, the search is successful otherwise not successful.

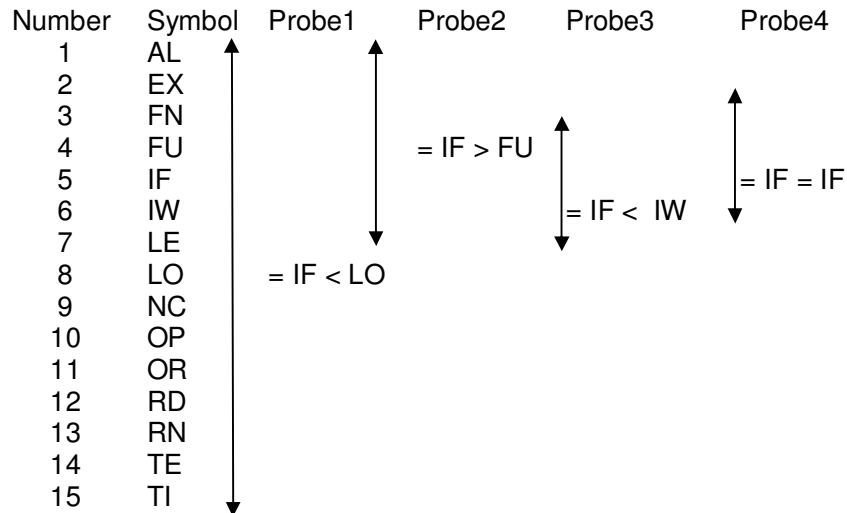
Binary Searching : A more systematic way of searching an ordered table . This technique. uses following steps for searching a keywords from the table.

- 1-Finds the middle entry ($N/2$ or $(N+1)/2$)
- 2- Start at the middle of the table and compare the middle entry with the keyword to be searched.
- 3- The keyword may be equal to , greater than or smaller than the item checked.
- 4-The next action taken for each of these outcomes is as follows
 - If equal, the symbol is found.
 - If greater, use the top half of the given table as a new table search.
 - If smaller, use the bottom half of the table

Illustration of Binary search :

Consider a table of 15 items , suppose we are searching for item IF.

1. First compare IF with middle item LO and find that IF must be in the top half of the table
2. Second comparison with middle term in this half of the table.
3. Shows IF to be in the second quarter.
4. Third comparison with IW shows IF to be in the third eighth of the table (i.e between item 4 and 6)
5. Final comparison made with the item in position 5.
6. A comparison failure on the fourth probe would have revealed that the item did not exists in the table.



4.0 Learning Objectives :

Intellectual skills

- To understand the process of searching an element (symbol) from table
- To compare the searching techniques.

Motor skills

- Ability to develop the source code for linear search and binary search
- Ability to compile correct logical , syntax errors , and execute program.

5.0 Stepwise procedure:

Binary search algorithm :

```

BinarySearch(A[0..N-1], value, low, high)
{
    if (high < low)
        return -1 // not found
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid // found
}

```

Optional algorithm for binary search

```

1. location = -1;
2. while ((more than one item in list) and (haven't yet found
   target))
2A. look at the middle item
2B. if (middle item is target)
    have found target
    else
2C. if (target < middle item)
    list = first half of list
2D. else (target > middle item)
    list = last half of list
    end while
3. if (have found target)
   location = position of target in original list

```

4. return location as the result

Linear search algorithm :

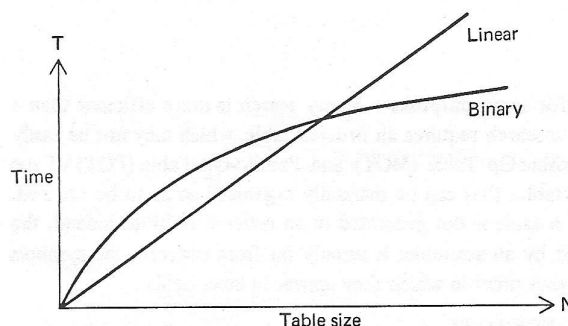
```
for i:=N:1:-1 do           //Search from N down to 1. (The step is -1)
  if A[i] = x then QuitLoop i;
next i;
Return(i);                //Or otherwise employ the value.
```

Optional algorithm for linear search

```
int linearSearch(int a[], int first, int last, int key)
{
// function:
// Searches a[first]..a[last] for key.
// returns: index of the matching element if it finds key,
//         otherwise -1.
// parameters:
// a in array of (possibly unsorted) values.
// first, last in lower and upper subscript bounds
// key in value to search for.
// returns:
// index of key, or -1 if key is not in the array.
for (int i=first; i<=last; i++) {
  if (key == a[i]) {
    return i;
  }
}
return -1; // failed to find key
}
```

6.0 Diagram:

Comparison of linear and Binary search technique



7.0 Laboratory practice:

- 1 - Develop source code for linear search technique.
- 2 - Develop source code for Binary search technique.

8.0 Conclusion:

- Linear search average length of Time is.....($T(\text{lin})=A*N$ / $T(\text{bin})=B*\log_2(N)$)
- Binary search average length time is.....($T(\text{lin})=A*N$ / $T(\text{bin})=B*\log_2(N)$)

9.0 Questions:

Write answers to Q.....Q.....Q.....Q.....and Q..... (Teacher shall allot the questions)

1. Define the following terms in one or two sentences.
 - a. Searching
 - b. Sorting
 - c. Hashing or random searching
2. Show the result of each pass for the Binary search .
(81,52,57,22,95,04,83,96,42,32,48,82)
3. Write assembly sample code for binary search method.
4. Draw the flowchart for linear search technique.
5. Draw the flowchart for Binary search technique

(Space for answer)

(Space for answer)

EXPERIMENT No. 4

1.0 Title:

To understand the Table Processing Techniques using sorting techniques.

2.0 Prior concepts:

- Data Structure concept (Array, Pointer, Stack, searching and sorting)
- Basic Assembly language 8085/86 or IBM 360/370

3.0 New Concepts :

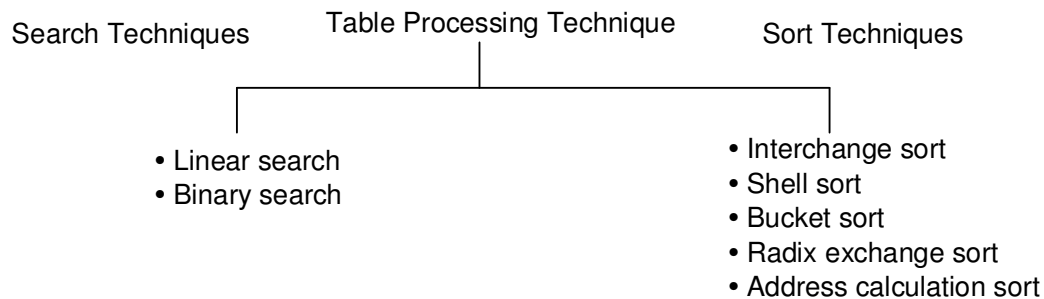
Proposition1 Sorting techniques

It is a technique required to rearrange unordered large tables like Machine-Op table (MOT) and pseudo-Op table (POT) and symbol table (ST) in sequential order.

Table processing technique classified as given below

1. Search techniques
2. Sort techniques

Concept structure



1. Interchange sort : It is also known as bubble sort, sinking sort, sifting sort .

This simple sort takes adjacent pairs of items in the table and put them in order as required.

Illustration of Interchange sort :

In this method the elements is rearranged by exchanging the two adjacent element If they are not in order

Unsorted List	1 st pass	2 nd pass	3 rd pass	4 th pass	5 th pass	6 th pass	7 th & final pass
19	13	05	05	01	01	01	01
13	05	13	01	05	05	05	02
05	19	01	13	13	13	<u>02</u>	05
27	01	19	19	16	02	09	09
01	26	26	16	02	09	11	11
26	27	16	02	09	<u>11</u>	13	13
31	16	02	09	<u>11</u>	16	16	16
16	02	09	<u>11</u>	19	19	19	19
02	09	11	21	21	21	21	21
09	11	<u>21</u>	26	26	26	26	26
11	<u>21</u>	27	27	27	27	27	27
21	31	31	31	31	31	31	31

2. Shell sort : This sort is similar to the interchange sort in that it moves data items by exchange. A fast comparative sort algorithm. It begins by comparing items a distance 'd' apart. The items that are way out of place will be moved rapidly than a simple interchange sort.


Illustration of Shell sort :

Unsorted List	pass 1 (d ₁ =6)	pass 2 (d ₂ =3)	pass 3 (d ₃ =2)	pass 4 (d ₄ =1)
19	19		*02	*01
13	13	*09	01	*02
05	*02	*01	*09	*05
27	*09	02	*05	*09
01	01	*19	11	11
26	*21	**11	**13	13
31	31	*05	**16	16
16	16	*27	*19	19
02	*05	**13	***21	21
09	*27	*21	*26	26
11	11	*31	*27	27
21	*26	*16	*31	31
		26		


- * = Exchange
- ** = Dual Exchange
- *** = Triple Exchange.

3. Bucket sort : It is one of the simple distributed sort is called the radix or bucket sort. The sort involve examining the least significant digit of the keyword first and the item is then assigned to a bucket uniquely dependent on the value of the digits.

Original Table	First distribution	Merge	Second distribution	Final merge
19		01	0-01,02,05,09	01
13	0	31	1-11,13,16,19	02
05	1- 01,31,11,21	11	2-21,26,27	05
27	2- 02	21	3-31	09
01	3- 13	02	4-	11
26	4	13	5-	13
31	5- 05	05	6-	16
16	6- 26,16	26	7-	19
02	7-27	16	8-	21
09	8	27	9-	26
11	9- 19,09	19		27
21	09	09		31



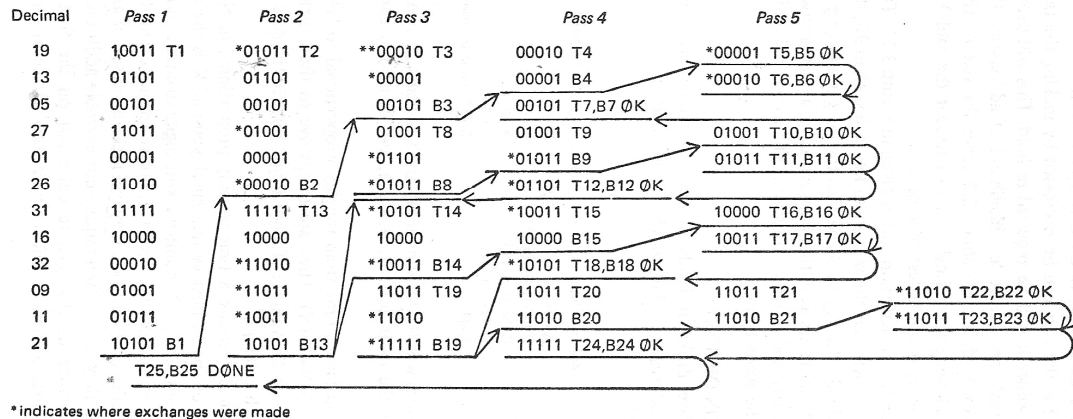
Separate based on last digit



Separate based on last digit

4. Radix exchange sort : A considerably better distributed sort , its used when keys are expressed (expressible) in binary . This sorting is accomplished by considering groups with the same (M) first bits and ordering that group with respect to the (M+1)st bits.

Scanning down from the top of the group for a one bit and up from the bottom for zero bit . Thus that two process are exchange and sorting continues.



5. Address calculation sort : It is one of the fastest type of sort if enough storage space is available. Sorting is done by transforming the key into an address in the table that " represents " the key .

4.0 Learning objectives :

Intellectual skills

- To understand the process of sorting by applying different methods
- To compare the sorting method by passes , exchanges, and comparisons
- Understand the efficiency of sorting method

Motor skills

- Ability to Convert the different sorting method process into assembly code (representative 360/370 instructions / SYMTLB)

5.0 Stepwise procedure / algorithms:

1 Interchange sort algorithm:

procedure bubbleSort(A : list of sortable items) **defined as:**

```

n := length( A )
do
  swapped := false
  n := n - 1
  for each i in 0 to n do:
    if A[ i ] > A[ i + 1 ] then
      swap( A[ i ], A[ i + 1 ] )
      swapped := true
    end if
  end for
while swapped
end procedure

```

2. Shell sort algorithm :

```

void shellSort(int numbers[], int array_size)
{

```

```

    int i, j, increment, temp;
    increment = 3;
    while (increment > 0)
    {
        for (i=0; i < array_size; i++)
        {
            j = i;
            temp = numbers[i];
            while ((j >= increment) && (numbers[j-increment] > temp))
            {
                numbers[j] = numbers[j - increment];
                j = j - increment;
            }
            numbers[j] = temp;
        }
        if (increment/2 != 0)
            increment = increment/2;
        else if (increment == 1)
            increment = 0;
        else
            increment = 1;
    }
}

```

3. Bucket sort algorithm :

function bucket-sort(array, n) **is**

```

buckets ← new array of n empty lists
for i = 0 to (length(array)-1) do
    insert array[i] into buckets[msbits(array[i], k)]
for i = 0 to n - 1 do
    next-sort(buckets[i])
return the concatenation of buckets[0], ..., buckets[n-1]

```

6.0 Laboratory practice :

Teachers shall allot one of the sorting technique to each group for implementation .
(in c or c++)

1. write a source code for an Interchange sort algorithm.
2. write a source code for a bucket sort algorithm .
- 3 write a source code for a shell sort algorithm

7.0 Conclusion:

Different kinds of sorting methods with its Average time (Approx) and Extra storage (wasted space)

Sr. No.	Type	Average time (Approximate)	Extra storage (Wasted space)
1	Interchange		
2	Shell		
3	Radix		
4	Radix Exchange		
5	Address calculation		

8.0 Questions:

Write answers to Q.....Q.....Q.....Q.....and Q..... (Teacher shall allot the questions)

1. Show the result of each pass for the following lists using
 - a. Inter change sort
 - b. Shell sort
 - c. Radix sort
(SWATI, ALKA, VARSHA, RUPALI, MRUNAL, ROHINI, ZARINA, YAMINI, BABALI, DIVYA)
2. Teacher shall make the group of 5 students and allot them to implement one of the sorting techniques to each group. (draw the flow chart)
 - a. Interchange
 - b. Shell
 - c. Radix exchange
 - d. Address calculation
 - e. Radix
3. Define the following terms in two or three sentences.
 1. searching
 2. sorting
 3. hashing

(Space for answers)

(Space for answers)

EXPERIMENT No. 5

1.0 Title:

To understand assembler design and implementation steps

2.0 Prior concepts:

- Data Structure.
- Assembly language programming concepts.

3.0 New concepts:

Proposition 1 : Assembler definition , features.

The assembler is the system program that translate source code written in assembly language to object code(Machine Language) and other information for loader.

Concept structure



General design procedure :

1. Specify the problem
2. Specify data structures
3. Define format of data structures
4. Specify algorithm
5. Look for modularity (i.e capability of one program to be subdivided into independent programming units)
6. Repeat 1 through 5 on modules

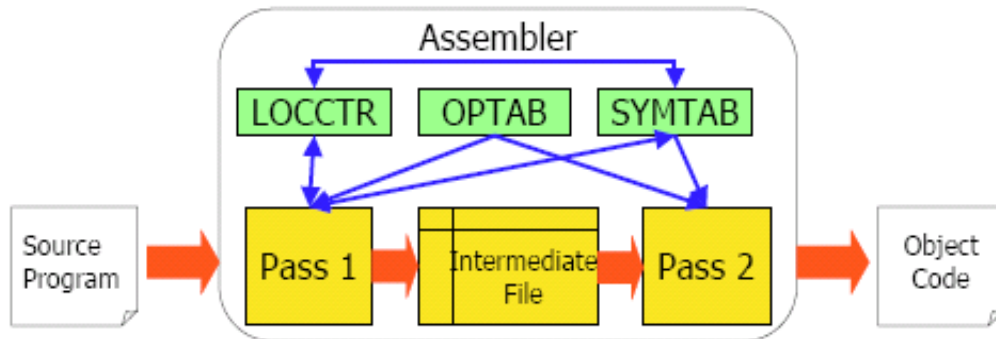
Assembler Design Features

There are two different assembler design features. Each feature has its own method for design and implement assembler. Following features define assembler design and implementation methods.

- Machine Dependent Assembler Features
 - In this method specific instruction format and addressing modes has PC relative and PC Based .
 - Specified Instruction formats and addressing modes
 - Needs Program relocation
- Machine independent Assembler Features
 - Literals
 - Symbol defining statements
 - Expressions
 - Program blocks
 - Control Sections and program linking.
 - Convert mnemonic operation codes to their machine language equivalents.
 - Convert symbolic operands to their equivalent machine addresses.
 - Decide the proper instruction format Convert the data constants to internal machine representations.
 - Generate relative address.

Proposition 2: statement of problems in assembler design.

Concept structure



The intermediate file include each source statement, assigned address and error indicator
 An assembler must do the following :

1. generate instructions:
 - i. Evaluate the mnemonic in the operation field to produce its machine code.
 - ii. Evaluate the subfield means find the value of each symbol, process literals , and assign addresses.
2. process pseudo ops :

Example:

Source program		First pass		Second pass	
		Relative address	Mnemonic Instruction	Relative address	Mnemonic Instruction
JOHN	START 0				
	USING *,15 ~				
	L 1, FIVE	0	L 1,-(0,15)	0	L 1,16(0,15)
	A 1, FOUR	4	A 1,-(0,15)	4	A 1,12(0,15)
	ST 1, TEMP	8	ST 1,-(0,15)	8	ST 1,20(0,15)
FOUR	DC F'4'	12	4	12	4
FIVE	DC F'5'	16	5	16	5
TEMP	DS 1F	20	-	20	-
	END				

Inter mediate steps in assembling a program

During assembling the above program the symbols (FOUR,FIVE,TEMP) appear before they are defined, so it is convenient to make two passes for assembling.

Pass 1: purpose – define symbols and literals.

1. determine length of machine instructions (MOTGET1)
2. keep track of location counter(LC).
3. remember values of symbols until pass 2 (STSTO)
4. process some pseudo ops e.g. EQU, DS (POTGET1)
5. remember literals (LITSTO)

Pass 2 : purpose- generate object program

1. Look up value of symbols(STGET)
2. generating instructions (MOTGET2)
3. generate data (for DS,DC, and literals)
4. process pseudo ops(POTGET2)

Proposition 3 : Data structures for assembler design.**Data structures for pass1:**

- Input source program
- Location counter(LC) to keep track of location of each instruction.
- Machine Operation Table(MOT) that consists symbolic mnemonic for each instruction along with its length.
- Pseudo Operation Table(POT) that indicates symbolic mnemonic and action to be taken for each pseudo op in pass1
- Symbol table(ST) that is used to store each label and its corresponding value.
- Literal table (LT) that is used to store each literal and its corresponding assigned location.
- A copy of input to be used later by pass 2

Data structures for pass2:

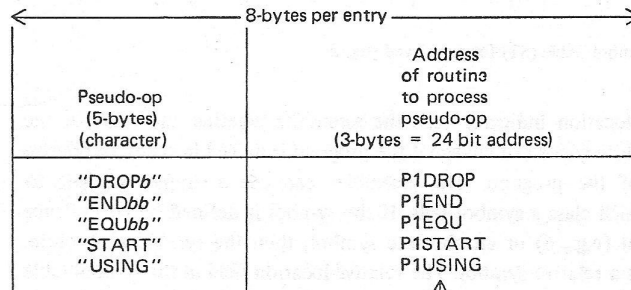
- A copy of source program input pass1
- Location counter(LC) to keep track of location of each instruction.
- Machine Operation Table(MOT) that indicates for each instruction i) symbolic mnemonic ii) length iii) binary machine opcode iv) format of instruction.
- Pseudo Operation Table(POT) that indicates symbolic mnemonic and action to be taken for each pseudo op in pass2
- Symbol table(ST) prepared by pass1,containing each label and its corresponding value.
- Base Table(BT) that indicates which registers are currently specified as base registers and their contents.
- Workspaces for
 - a)holding various parts of instruction during assembling,
 - b)producing a printed list
- Storage for holding the copy of assembled instructions in the format needed by the loader.

Proposition 4: Format of database in assembler design**MOT: Format of machine operation table**

←———— 6 bytes per entry —————→

Mnemonic Opcode (4 bytes) (characters)	Binary Opcode (1 byte) (hexadecimal)	Instruction length (2 bits) (binary)	Instruction format (3 bits) (binary)
“Abbbb”	5A	10	001
“AHbbb”	4A	10	001
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

POT: Format of pseudo operation table



These are presumably labels of routines in pass 1; the table will actually contain the physical addresses.

ST: Format of symbol table

14-bytes per entry			
Symbol (8-bytes) (characters)	Value (4-bytes) (hexadecimal)	Length (1-byte) (hexadecimal)	Relocation (1-byte) (character)
"JOHNbbbb"	0000	01	"R"
"FOURbbbb"	000C	04	"R"
"FIVEbbbb"	0010	04	"R"
"TEMPbbbb"	0014	04	"R"

BT: Format of machine operation table

4-bytes per entry	
Availability indicator (1-byte) (character)	Designated relative-address Contents of base register (3-bytes = 24-bit address) (hexadecimal)
1 "N"	-
2 "N"	-
:	:
14 "N"	-
15 "Y"	00 00 00

↑
15 entries
↓

Code=

Availability

- Y ~ register specified in USING pseudo-op
- N ~ register never specified in USING pseudo-op or subsequently made unavailable by the DROP pseudo-op

A sample assembly program with illustration for symbol, literal, and base tables:

Sample assembly source program

Sample Assembly Source Program

```

Statement no.
9      6 SETUP      EQU      *
10     USING      SETUP, 15
11     L          DATABASE, = A(DATA1)
12     USING      DATAAREA, DATABASE
13     SR         INDEX, INDEX
14     LOOP      L          AC, DATA1 (INDEX)
15     AR         TOTAL, AC
16     A          AC, = F'5'
17     ST         AC, SAVE (INDEX)
18     A          INDEX, = F'4'
19     C          INDEX, = F'8000'
20     BNE       LOOP
21     LR         1, TOTAL
22     BR         14
23     LTORG
24     SAVE      DS         2000F
25     DATAAREA EQU      *
26     DATA1    DC         F'25, 26, 97, 101 . . . .
                               [2000 numbers]
27     END
    
```

Variable Tables

Symbol Table

Symbol	Value	Length	Relocation
PRGAM2	0	1	R
AC	2	1	A
INDEX	3	1	A
TOTAL	4	1	A
DATABASE	13	1	A
SETUP	6	1	R
LOOP	12	4	R
SAVE	64	4	R
DATAAREA	8064	1	R
DATA1	8064	4	R

Literal Table

A(DATA1)	48	4	R
F'5'	52	4	R
F'4'	56	4	R
F'8000'	60	4	R

Base table (showing only base registers in use)

1) After statement 2:

Base	Content
15	0

2) After statement 10:

Base	Content
15	6

3) After statement 12:

Base	Content
13	8064
15	6

Proposition 5 : use of one pass and two pass assemblers

One-pass assemblers are used when

1. It is necessary or desirable to avoid a second pass over the source program
 2. the external storage for the intermediate file between two passes is slow or is inconvenient to use
- Main problem: forward references to both data and instructions
 - One simple way to eliminate this problem: require that all areas be defined before they are referenced.
 1. It is possible, although inconvenient, to do so for data items.
 2. Forward jump to instruction items cannot be easily eliminated.

How to Handle Forward References

Two pass assembler can resolve the forward references by using following method.

- Load-and-go assembler
- Omits the operand address if the symbol has not yet been defined
- Enters this undefined symbol into ST and indicates that it is undefined
- Adds the address of this operand address to a list of forward references associated with the ST entry
- Scans the reference list and inserts the address when the definition for the symbol is encountered.
- Reports the error if there are still ST entries indicated undefined symbols at the end of the program
- Search ST for the symbol named in the END statement and jumps to this location to begin execution if there is no error

4.0 Learning objectives :**Intellectual skill**

1. To understand design procedure of assembler
2. To understand data structures like LC,ST,BT,LT,MOT,POT etc.
3. To understand working sequence of assembler with another system programs loader, linker and compiler.

Motor skills

1. Ability to design symbol table and literal table.
2. Ability to handle the entries in symbol table and literal table.

5.0 Stepwise procedure/ Algorithm:**Algorithm : one pass**

- Begin
- read first input line
- if OPCODE = 'START' then begin
- save #[Operand] as starting addr
- initialize LC to starting address
- write line to intermediate file
- read next line
- end(if START)
- else
- initialize LC to 0
- While OPCODE != 'END' do
- begin
- if this is not a comment line then
- beg
- if there is a symbol in the LABEL field then
- begin
- search ST 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 LC
- else if OPCODE = 'WORD' then
- add 3 to LC
- else if OPCODE = 'RESW' then
- add 3 * #[OPERAND] to LC
- else if OPCODE = 'RESB' then
- add #[OPERAND] to LC
- else if OPCODE = 'BYTE' then
- begin
- find length of constant in bytes
- add length to LC
- 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 (LC – starting address) as program length
- End {pass 1}

Algorithm : Two Pass

-
- 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 ST 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}
- write listing line
- read next input line
- end
- write listing line
- read next input line
- write last listing line
- End {Pass 2}

6.0 Conclusion:

1. The design of assembler requires to follow the steps
 1. Specify the problem
 2.
 3. Define format of data structures
 4.
 5.
 6. Repeat 1 through 5 on modules
2. While assembler design two passes are needed due to

7.0 Laboratory practice:

- 1) For the following assembly code, design
 - a) symbol table
 - b) literal table
 - c) base table

	SIMPLE	START
	BALR	15, 0
	USING	*, 15
LOOP	L	R1, TWO
	A	R2, TWO
	ST	R1, FOUR
	CL1	FOUR +3,4
	BNE	LOOP
	BR	14
R1	EQU	1
TWO	DC	F'2'
FOUR	DS	F
	END	

2) For a given a data structure MOT. Write a "C" program to enter mnemonic opcode in to MOT and search the particular mnemonic opcode form MOT.

← 6 bytes per entry →			
Mnemonic Opcode (4 bytes) (characters)	Binary Opcode (1 byte) (hexadecimal)	Instruction length (2 bits) (binary)	Instruction format (3 bits) (binary)
"Abbbb"	5A	10	001
"AHbbb"	4A	10	001
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

7.0 Questions:

Write answers to Q.....Q.....Q.....Q.....and Q..... (Teacher shall allot the questions)

1. What do you mean by assembler.
2. List the assembler design steps.
3. State the use of following data structures.
 - a) Symbol Table
 - b) Literal Table.
 - c) Base table.
 - d) Location Counter .
 - e) MOT
 - f) POT
4. Draw the flowchart for Pass1 of assembler design.
5. Draw the flowchart for Pass2 of assembler design.

(Space for answers)

(Space for answers)