

Noorul Islam College of Engineering
Department of Software Engineering

Part A Questions and Answers
System Software

M.Germanus Alex

Unit I

1. Define system software?

It consists of a variety of programs that support the operation of computer. This software makes it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

2. What are the components of system software?

1. Text editor – To write programs in a high-level language.
2. Compiler – To translate high level language programs into machine language.
3. Loader or linker – To load the machine language program into memory.
4. Debugger – To help detect errors in the program.
5. Assembler - To translate assembly language program into machine language.
6. Macro processor – it is a part of assembler for translation.

3. What are the functions of assemblers?

To translate assembly language program into machine language and assigning machine addresses to symbolic labels used by the programmer.

4. What are the functions of loaders?

To load the machine language program into memory

5. What are the functions of linkers?

Linkers are used to perform the linking operation. Linking is to combines two or more separate object programs and supplies the information needed to allow references between them.

6. What are the functions of compilers?

The function of compiler is to translate high level language programs into machine language. In order to convert into machine language the source program

is split into tokens by lexical analysis. The statement is recognized by syntax analysis or parsing. The last step is the generation of object code.

7. What are the functions of text editors?

An interactive text editor is a program that allows a user to create and revise a target document. The term document includes objects such as computer programs, text, equations, tables diagrams line art etc.

8. What are the functions of debuggers?

To help detect errors in the program.

9. Name the registers available in SIC/XE machine?

Table below indicates the additional register details

Mnemonic	Number	Special use
B	3	Base register; used for addressing.
S	4	General working register – no special use.
T	5	General working register – no special use.
F	6	Floating-point accumulator (48 bits)

10. Write the addressing mode of SIC/XE machine?

Two new relative addressing modes with format 3

Base relative $b=1, p=0$ $TA (\text{target address}) = (B) + \text{disp} (0 \leq \text{disp} \leq 4095)$
Program_counter relative $b = 0, p = 1$ $TA = (PC) + \text{disp} (-2048 \leq \text{disp} \leq 2047)$

For base relative addressing the disp field is interpreted as 12 bit unsigned integer.

For program_counter relative addressing the disp field is interpreted as 12 bit signed integer with negative values are in 2's complement form.

For direct addressing b and p are set to 0. the disp field indicates the target address.

Indexed addressing if x is set to 1.

For immediate addressing $i = 1$ and $n = 0$.

For indirect addressing $i = 0$ and $n = 1$ the value contained in this word is taken as the address of the operand value.

For simple addressing i and n are both 0. the address is taken as the location of the operand.

11. Write a data movement instruction for SIC machine?

```
LDA FIVE
STA ALPHA
LDCH CHARZ
STCH C1
-
-
-
ALPHA RESW 1
FIVE WORD 5
CHARZ BYTE C 'Z'
C1 RESB 1
```

12. Write a data movement instruction for SIC/XE machine?

```
LDA #5
STA ALPHA
LDA #90
STCH C1
-
-
-
ALPHA RESW 1
C1 RESB 1
```

13. Write notes on CISC machines?

Complex instruction set computers (CISC) has a relatively large and complicated instruction set, several different instruction formats and lengths, and many addressing modes.

14. Write the data formats of VAX architecture?

Integers are stored as binary numbers in byte, word, longword, quadword, or octaword ; 2's complement representation is used for negative values. Character are stored using 8-bit ASCII codes. Four different floating point data formats on the VAX ranging from 4 to 16 bytes.

VAX provides a **packed decimal** data format. Each byte represents two decimal digits, with each digit encoded using 4 bits of the byte. The sign is encoded in the last 4 bits of the byte.

Numeric format is used to represent numeric values with one digit per byte. The sign may appear either in the last byte or as a separate byte preceding the first digit. These variations are called **trailing numeric** and **leading separate numeric**.

VAX supports queues and variable-length bit strings.

15. How the instruction sets are formed in VAX architecture?

Variable length instruction formats. Each instruction consist of an operation code (1 or 2 bytes) followed by upto 6 operand specifiers depending on the type of instruction.

16. List the registers available in Pentium Pro microprocessor?

There are eight general-purpose registers, which are named as EAX, EBX, ECX, EDX, ESI, EBX and ESP. Each general purpose registers is 32 bits long. Registers EAX, EBX, ECX and EDX are generally used for data manipulation. EIP is a 32 bit register that contains a pointer to the next instruction to be executed.

Floating point computations are performed using a floating point unit (FPU) . this unit contain eight 80 bit data registers.

17. Write notes on RISC machines?

Reduced instruction set computers. This design can result in faster and less expensive process development, greater reliability, and faster instruction execution time.

A RISC system is characterized by a standard, fixed instruction length, and single-cycle execution of most instructions. Memory access is usually done by load and store instructions only. All instructions except for load and store are register-to register operations. There are typically large numbers of general purpose registers.

18. Name the various registers available in UltraSPARC architecture?

SPARC architecture includes a large register file that usually contains more than 100 general-purpose registers. Any procedure can access only 32 registers. The first eight of these registers can be accessed by all procedures on the system. The other 24 registers are visualized as a window thorough which part of the register file can be seen. In original SPARC architecture, the general purpose registers were 32 bits long. Later it was expanded to 64 bits.

19. Write the addressing mode of UltraSPARC machines?

1. Immediate mode – operand value specified as part of the instruction.
2. Register direct mode – in register
3. PC relative
4. Register indirect with displacement
5. Register indirect indexed.

20. Write notes on delayed branches?

Instruction execution is **pipelined** – while one instruction is being executed, the next one is being fetched from memory and decoded. SPARC branch instructions are **delayed branches** – the instruction immediately following the branch instruction is executed before the branch is taken.

21. Write the memory organization of Power PC architecture?

Memory consists of 8-bit bytes; all addresses used are byte addresses. Two consecutive bytes form a halfword; four bytes form a word; eight bytes form a doubleword; sixteen bytes form a quadword.

Power PC program can be written using a virtual address space of 2^{64} bytes. This address space is divided into fixed-length **segments**, which are 256 megabytes long. Each segment is divided into **pages**, which are 4096 bytes long. The virtual address specified by instructions is automatically translated into a physical address.

22. Write the addressing mode of Power PC architecture?

Immediate, register direct. Instruction that address memory are load and store, and branch instruction. Load and store uses the following addressing modes

Mode	Target address calculation
Register direct	$TA = (\text{register})$
Register indirect with index	$TA = (\text{register} - 1) + (\text{register} - 2)$
Register indirect with immediate index	$TA = (\text{register}) + \text{displacement}$

23. What are methods for performing I/O operations in Power PC Architecture?

Two different methods for performing I/O operations. In one approach, the segments in the virtual address space are mapped onto an external address space. Segments that are mapped in this way are called *direct-store* segments.

I/O is performed using the regular virtual memory management hardware and software.

24. Write the register details of Cray T3E architecture?

32 general purpose registers. 64 bits long. Used to store and manipulate integer data and address.

32 floating-point registers. 64 bits long. 64 bit PC and several other status and control registers.

25. Write the addressing modes of Cray T3E architecture?

The operand value may be specified as part of the instruction itself or it may be in a register

Operands in memory are addressed using one of the following two modes

PC-relative $TA = (PC) + \text{displacement}$

Register indirect with displacement $TA = (\text{register}) + \text{displacement}$

Unit II

1. What are the functions of assembler?

The assembler perform functions such as translating mnemonic operation codes to their machine language equivalents and assigning machine addresses to symbolic labels used by the programmer.

2. Name the various assembler directives?

START	Specify the name and starting address for the program
END	Indicate the end of the source program and specify the first executable instruction in the program
BYTE	Generate character or hexadecimal constant.
WORD	Reserve the indicated number of bytes for a data area.
RESW	Reserve the indicated number of words for a data area.

3. Write the function of a simple SIC assembler?

- Convert the mnemonic operation codes to their machine language equivalents.
- Convert symbolic operands to their equivalent machine address.
- Build the machine instructions in the proper format.
- Convert the data constants specified in the source program into their internal representation.
- Write the object program and the assembly listing.

4. What is meant by forward reference?

A reference to a label that is defined later in the program is called forward reference.

FIRST	STL	RETADR

RETADR	RESW	1

Here REGADR label is an example of forward reference because; the label is used before it is declared.

5. Write notes on the Header record in the object program?

The Header record contains the program name, starting address, and length. The format is

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

6. Write notes on the Text record in the object program?

Text record contains the translated instructions and data of the program, together with an indication of the address where they are to be loaded. The format is

Col.1	T
Col. 2-7	Starting address for object code in this record
Col 8-9	Length of object code in this record in bytes
Col 10-69	Object code, represented in hexadecimal

7. Write notes on the End record in the object program?

The End record marked the end of the object program and specifies the address in the program where execution is to begin. The format is

Col 1	E
Col. 2-7	Address of first executable instruction in object program

8. What are the functions of Pass 1 of an assembler?

- Assign address to all statements in the program.
- Save the values assigned to all labels for use in Pass2.
- Perform some processing of assembler directives.

9. What are the functions of Pass 2 of an assembler?

- a) Assemble instruction.
- b) Generate data value define by BYTE, WORD, etc.
- c) Perform processing of assembler directives not done during Pass 1.
- d) Write the object program and the assembly listing.

10. Write the data structure for symbol table in assembler?

The symbol table includes the name and value for each label in the source program, together with flags to indicate error conditions. This table may also contain other information about the data area or instruction labeled- for example its type and length.

11. List the details available in the intermediate file generated by Pass1?

The intermediate file contains each source statement together with its assigned address, error indicators, etc. This file is used as the input to Pass 2. This working copy of the source program can also be used to retain the results of certain operations that may be performed during Pass 1, so that these need not be performed during Pass 2. Similarly pointer into OPTAB and SYMTAB may also be retained for each operation code and symbol table used.

12. What is meant by program relocation?

When more than one program at a time, sharing the memory and other resources of the machine. The address for loading is not known in advance.

Since the assembler does not know the actual location where the program will be loaded, it cannot make the necessary changes in the address used in the program. The assembler can identify those part of the object program that need modification. An object program that contains the information necessary to perform his kind of modification is called **relocatable program**.

13. Write notes on the Modification record in the object program?

The command for the loader must also be a part of the object program. We can accomplish it with a modification record. The format is

col 1	M
col 2 -7	Starting address of the address field to be modified, relative to the beginning of the program.
col 8 – 9	Length of the address field to be modified in half-bytes.

14. Define literals in assembly language?

The programmer can be able to write the value of a constant operand as a part of the instruction that uses it. Such type of operand is called a literal. In some assembler languages the literal is identified with the prefix = which is followed by a specification of the literal value.

Ex:-

```
215 1062 WLOOP TD =X'05' E32011
```

All of the literal operands used in a program are gathered together into one or more literal pools.

The basic data structure needed to store literals is literal table LITTAB.

15. What is the structure of a literal table?

The basic data structure needed to store literals is literal table LITTAB. For each literals used this table contains the literal name, the operand value and length, and the address assigned to the operand when it is placed in a literal pool. LITTAB is often organized as a hash table, using the literal name or value as the key.

16. What are the various symbol defining statements?

Most assembler provides an assembler directive that allows the programmer to define symbols and specify their values. The assembler directive generally used is EQU. The general form is

```
symbol EQU value
```

This statement defines the given symbol in the SYMTAB and assigns to it the value specified.

Another common use of EQU is in defining mnemonic names for registers.

```
A EQU 0
X EQU 1
L EQU 2
```

There is another common assembler directive that can be used to indirectly assign values to symbols is ORG.

ORG value

17. What are program blocks?

Program blocks refer to segment of code that are rearranged within a single object program unit, and control sections to refer to segments that are translated into independent object program unit.

The assembler directive USE indicates which portions of the source program belong to the various blocks.

18. What is a control section in assembly language statement?

A control section is a part of the program that maintains its identity after assembly; each such control section can be loaded and relocated independently of each other's.

When control sections form logically related parts of a program, it is necessary to provide some means for linking them together.

19. What are external references?

Control sections are independently loaded and located, the assembler is unable to process these references in the usual way. The assembler has no idea where any other control section will be located at execution time. Such references between control sections are called external references.

20. What are external symbols?

They are symbols that are defined in a control section, and may be used by other sections.

21. Write notes on Define record?

A Define record gives information about external symbols that are defined in this control section. The format is

col 1	D
col 2- 7	Name of external symbol defined in this control section.

col 8 – 13 Relative address of symbol within this control section.
col 14 – 73 Repeat information in col 2-13.

22. Write notes on Refer record?

A Refer record lists symbols that are used external references by the control section. The format is

col 1 R
col 2 – 7 Name of external symbol referred to in this control section.
col 8 – 73 names of other external reference symbols.

23. Define load-and-go assembler?

There are two main types of one-pass assembler.

One type produces object code directly in memory for immediate execution. This type generate object program in memory for immediate execution. No object program is written out, and no loader is needed. This kind of **load-and-go** assembler is useful in a system that is oriented toward program development and testing.

24. What is the function of multi-pass assembler?

Restrictions such as prohibiting forward reference in symbol definition are not normally a serious inconvenience for the programmer. The general solution is the multi-pass assembler that can make as any passes as are needed to process the definitions of symbols. The portions of the program that involve forward references in symbol definitions are saved during pass1.

25. Name the various sections in SPARC assembler?

SPARC assembler language program is divided into units called **sections**. The assembler provides a set of predefined section names. They are

.TEXT	Executable instructions
.DATA	Initialized read/write data
.RODATA	Read only data
.BSS	Unintialized data areas

Unit III

1. Define loader?

A loader is a system program that performs the loading function. Many loaders also support relocation and linking. Some systems have a linker to perform the linking operation and a separate loader to handle relocation and loading.

2. What is meant by loading in system software?

Loading means brings the object program into memory for execution.

3. Define relocation?

Relocation modifies the object program so that it can be loaded at an address different from the location originally specified.

4. Define linking?

Linking means combines two or more separate object programs and supplies the information needed to allow references between them.

5. What is an absolute loader?

All functions of absolute loader are performed in a single pass. The header record is checked to verify that the correct program has been presented for loading. As each Text record is read, the object code it contains is moved to the indicated address in memory. When the End record is encountered, the loader jumps to the specified address to begin execution of the loader program.

6. How the object code for STL instruction is loaded into memory?

Each pair of bytes from the object program record must be packed together into one byte during loading. For example, the machine operation code for an STL instruction would be represented by the pair of characters "1" and "4". When the loader reads these, they will occupy two bytes of memory. However this operation code must be stored as one byte during loading.

7. Write the algorithm for an absolute loader?

Begin

 Read Header record

 Verify program name and length

 Read first Text record

 While record type \neq 'E' do

 Begin

 { if object code is in character form, convert into internal representation }

 move object code to specified location in memory

 read next object program record

```
        end
        jump to address specified in End record
end
```

8. What is a Bootstrap loader?

When a computer is first turned on or restated, a special type of absolute loader, called a bootstrap loader, is executed. This loads the first program to be run by the computer- usually an operating system.

9. Write a subroutine using SIC/XE to read one character from input device and convert it from ASCII code to hexadecimal and is returned in a register A.

```
GETC      TD    INPUT
          JEQ   GETC
          RD    INPUT
          COMP #4
          JEQ   80
          COMP #48
          JLT   GETC
          SUB   #48
          COMP #10
          JLT   RETURN
          SUB   #7
RETURN    RSUB
INPUT     BYTE X'F1'
          END   LOOP
```

10. What are relocating loader?

Loaders that allow for program relocation are called **relocating loader or relative loaders**. There are two methods for relocating.

a) A modification record is used to describe each part of the object code that must be changed when the program is relocated.

b) A machine that uses direct addressing and has a fixed instruction format, it is more efficient to specify relocation. Along with the Text record a **relocation bit** is associated with each word of the object code.

11. Write notes on relocation bit?

A machine that uses direct addressing and has a fixed instruction format, it is more efficient to specify relocation. Along with the Text record a **relocation bit** is associated with each word of the object code.

If the relocating bit corresponding to a word of object code is set to 1, the programs starting address is to be added to this word when the program is relocated. A bit value of 0 indicates no relocation is needed. If a Text record contains fewer than 12 words of object code, the bits corresponding to unused words are set to 0.

12. What are the data structures used for a linking loader?

The various data structures used are

- a) ESTAB - External symbol table
- b) PROGADDR - Beginning of address in memory where the linked program is to be loaded.
- c) CASADDR - Contain the starting address assigned to the control section currently being scanned by the loader.

13. What are the contents of ESTAB?

It is used to store the name and address of each external symbol in the set of control section being loaded. The table also often indicates in which control section the symbol is defined. A hashed organization is typically used for this table.

14. What is the content of CSADDR during pass 1?

CSADDR - It contains the starting address assigned to the control section currently being scanned by the loader. This value is added to all relative addresses within the control section to convert them to actual addresses.

15. What are the functions of pass 1 of loader?

During the first pass, the loader is concerned only with Header and Define record types in the control sections. The control section name from the Header record is entered into ESTAB, with value given by CSADDR. All external symbols appearing in the Define record for the control section are also entered into the ESTAB. When the End record is read, the control section length CSLTH is added to CSADDR.

16. What the function of pass 2 of loader?

Pass 2 performs the actual loading, relocation and linking of the program. CSADDR contains the actual starting address of the control section currently being loaded. As each Text record is read, the object code is moved to the specified address. When Modification record is encountered, the symbol whose value is to be used to modification is looked up in ESTAB. This value is added or

subtracted from the indicated location in memory. The last step is to transfer the control the loaded program to begin execution.

17. Define load map?

At the end of Pass1, ESTAB contains all external symbols defined in the set of control sections together with the address assigned to each. Many loaders include as an option the ability to print a load map that shows these symbols and their addresses.

18. What is meant by automatic library call?

Many linking loaders automatically incorporate the routines from a subprogram library into the program being loaded. This feature allows the programmer to use subroutines from one or more libraries as if they were part of the programming language. On some systems, this feature is referred to as *automatic library call or library search*.

19. What are the commands used for the selection of alternative sources of input in loaders, delete external symbols and to change the external symbol?

One typical loader option allows the selection of alternate sources of input.
Ex:-

INCLUDE program-name(library-name)

Might direct the loader to read the designated object program from a library and treat it as if it were part of the primary loader input

Other commands allow the user to delete external symbols or entire control sections. Ex

DELETE Csect-name

Might instruct the loader to delete the named control section from the set of programs being loaded.

The command

CHANGE name1,name2 might cause the external symbol name1 to be changed to name 2.

20. What is a linkage editor?

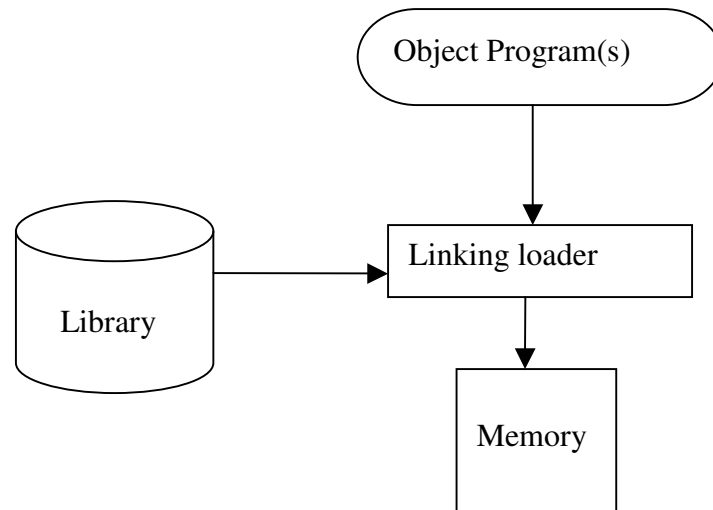
A *linkage editor* produces a linked version of the program, which is written to a file or library for later execution. The linkage editor performs

relocation of all control sections relative to the start of the linked program. This means that the loading can be accomplished in one pass with no external symbol table required. The linked program produced by a linkage editor is generally in a form that is suitable for processing by a relocating loader.

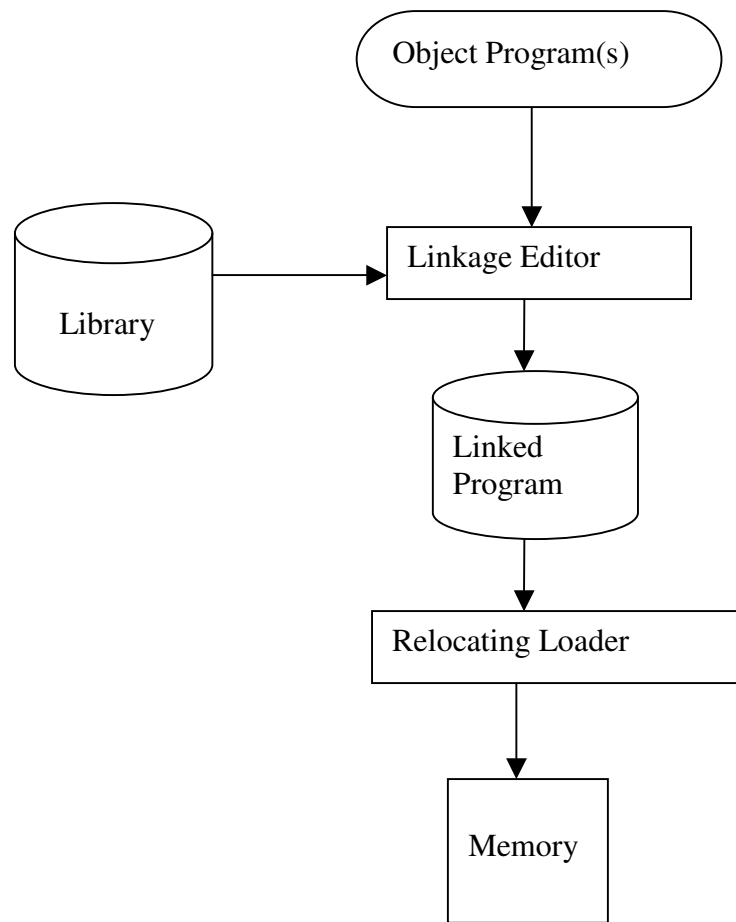
21. What is a linking loader?

A *linking loader* performs all linking and relocation operations, including automatic library search, and loads the linked program directly into the memory for execution. A linking loader searches libraries and resolves external references every time the program is executed.

22. Write the block diagram of processing of an object program using a linking loader?



23. Write the block diagram of processing of an object program using a linkage editor?



24. What is meant by dynamic linking?

This scheme postpones the linking function until execution time: a subroutine is loaded and linked to the rest of the program when it is first called. This type of function is usually called *dynamic linking*, *dynamic loading* or *load on call*.

25. Define binding?

When dynamic linking is used, the association of an actual address with the symbolic name of the called routine is not made until the call statement is executed. Another way of describing this is to say that the binding of the name to an actual address is delayed from load time until execution time.

Unit IV

1. What is the operation of macroprocessor?

A macroinstruction is simply a notational convenience for the programmer. A macro represents a commonly used group of statements in the source programming language. The macro processor relocates each macroinstruction with the corresponding group of source language statements. This called expanding macros.

The function of a macro processor essentially involves the substitution of one group of characters or lines for another.

2. Write notes on macro definition and expansion?

Two assembler directives are used in macro definitions. The first MACRO statement identifies the beginning of a macro definition. The second MEND assembler directive marks the end of the macro definition.

```
RDBUF      MACRO      &INDEV,&BUFADR,&RECLTH
           ---
           ---
           TD      =X'&INDEV'
           --
           --
           MEND
```

In the above source code, the symbol in the label field (RDBUF) is the name of the macro, and the entries in the operand field identify the parameters of the macroinstruction.

3. What are the data structures involved in macro processor?

DEFTAB – macro definition table, the macro definitions are stored in this table.

NAMTAB – macro names are entered into this table. It is served as the index to DEFTAB.

ARGTAB – argument table, which is used during the expansion of macro invocation.

4. Write the procedure PROCESSLINE in the one-pass macro processor algorithm?

Procedure PROCESSLINE

```

Begin
  Search NAMTAB for OPCODE
  If found then
    EXPAND
  Else if OPCODE = 'MACRO' then
    DEFINE
  Else write source line to expanded file
end

```

5. What are the features of machine independent macro processors?

The various features are

1. Concatenating macroinstruction parameters with other character strings.
2. Generating unique labels within macro expansions, which avoid the need for extensive use of relative addressing.
3. Macro expansions. This ability is to alter the expansion of a macro by using control statements.
4. The definition and use of keyword parameters in macroinstructions.

6. How concatenation of macro parameters is performed?

Most macro processors allow parameters to be concatenated with other character strings. The following statement will do the concatenation

```
DA X&ID1
```

In which the parameter &ID is concatenated after the character string X and before the character string 1.

One problem about this is that the beginning of the macro parameter is identified by the starting symbol & , however the end of the parameter is not marked.

Most macro processors deal with this problem by providing a special *concatenation operator*. In SIC macro language, this operator is the character → . Thus the previous statement would be written as

```
LDA X&ID→1
```

7. Write notes on generation of unique labels in macroprocessors?

The duplicate definition would prevent correct assembly of the resulting expanded program. Many macro processors avoid these problems by allowing the creation of special types of labels within macro instructions.

Labels used within the macro body begin with the special character \$. Each symbol beginning with \$ has been modified by replacing \$ with \$AA. More generally the character \$ will be replaced by \$xx, where xx is a two character alphanumeric counter of the number of macro instruction expanded. For example AA,AB,AC, etc.

8. Write the conditional statement in macros?

The use of macro time variable makes it clear that the same logical condition is involved in to IF statements. Examining the values of the variable may also be faster than repeating the original test.

The implementation of conditional macro expansion is very simple. The macro processor must maintain a symbol table that contains the values of all macro time variables used. The table is used to look up the current value of a macro time variable whenever it is required.

The testing of Boolean expression in IF statements occur at the time macros is expanded.

9. Write notes on WHILE statement in macros?.

The WHILE statement specifies that the following lines , until the next ENDW statement ate to be generated repeatedly as long as the looping are done while true.

Ex:

```
WHILE      (&CTR ≤ &EORCT)
---
---
ENDW
```

When a WHILE statement is encountered during macro expansion, the specified Boolean expression is evaluated. If the value of this expression is FALSE the macro processor skips ahead until it finds the next ENDW statement, and then resumes normal macro expansion. If the value of this expression is TRUE, the macro processor continues to processes lines in the usual way until the next ENDW statement. Then proceed to the WHILE for revaluation.

This method does not allow for nested WHILE structures.

10. Write notes on conditional macro expansion?

Most macro processors, however, can also modify the sequence of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation.

The term *conditional assembly* is commonly used to describe this feature.

Ex:

```
RDBUFF    MACRO    &INDEV,&BUFADR,
           &RECLTH,&EOR,&MAXLTH
           F        (&EOR ≠ ' ')
&EORCK    SET      1
           ---
           ---
           ENDIF
           --
           --
           IF      (&MAXLTH EQ ' ')
           ---
           ---
           ELSE
           ---
           ---
           ENDIF
           ---
           ---
           MEND
```

11. Define macro time variable?

```
&EORCK    SET      1
```

The symbol &EORCK is a macro time variable, which can be used to store working values during the macro expansion. Any symbol that begins with the character & and that is not a macro instruction parameter is assumed to be a *macro-time variable*. All such variables are initialized to a value of 0.

The use of macro time variable makes it clear that the same logical condition is involved in IF statements. Examining the values of the variable may also be faster than repeating the original test.

12. Write notes on positional parameters in macroprocessors?

The macro definitions defined in ordinary way are *positional parameters*. That is, the parameters and arguments were associated with each other according to their positions in the macro prototype and the macro invocation statement.

```
GENER    ,,DIRECT,,,,,3.
```

13. Write notes on Keyword parameters in macroprocessors?

If a macro has a large number of parameters, and only a few of these are given values in a typical invocation. A different form of specification is useful. This is called *keyword parameters*. In keyword parameters each argument value is written with a keyword that names the corresponding parameters.

```
RDBUF      MACRO      &INDEV = F1, &BUFADR=, &RECLTH=,
                  &EOR = 04, &MAXLTH = 4096
```

14. How to invoke the macro which has keyword parameters?

In the macro prototype, each parameter name is followed by an equal sign, which identifies a keyword parameter. After the equal sign, a default value is specified for some of the parameters. The parameter is assumed to have this default value if its name does not appear in the macro invocation statement.

```
GENER TYPE = DIRECT, CHANNEL = 3.
```

15. How to define one macro instruction in another macro?

It is also possible to define one macro instruction in another macro.

```
MACROS      MACRO      { --- }
RDBUF      MACRO      &INDEV,&BUFADR,&RECLTH
            ---
            ---
            MEND
            ---
            MEND
```

16. How to invoke a macro by another by another macro?

The invocation of macro by another is as follows

```
RDBUF MACRO &BUFADR, &RECLTH, &INDEV
            ---
            ---
            RDCHAR &INDEG
            ---
            ---
            MEND
```

17. What happened when we apply one pass macro processor algorithm for invoking a macro by another macro?

When we apply the one pass macro processor algorithm for invocation of a macro by another in the above macro code, the procedure EXPAND would be called when the macro was recognized. The arguments from the macro invocation would be entered into ARGTAB. The Boolean variable EXPANDING would be set to TRUE, and expansion of the macro invocation would begin.

18. Write notes on general purpose macro processors?

These are not depend on any particular programming language, but can be used with a variety of different languages. Advantage of general purpose of macro processor is that the programmer does not need to learn about a different macro facility for each compiler of assembler language, so much of the time and expenses involved in training are eliminated.

19. What are the problems in general purpose macro processors?

A more general problem involves the tokens of the programming language. The restriction on the length of identifiers and the rules for the formation of constants. Another problem is the syntax used for macro definitions and macro invocation statements.

20. What is a line by line macro processor?

Using this approach, the macro processor reads the source program statements and performs all of its function as usual. However, the output lines are passed to the language translator as they are generated, instead of being written to an expanded source file.

21. What are the advantages of line-by-line macro processor?

It avoids making an extra pass over the source program, so it can be more efficient than using a macro preprocessor. Some of the data structures required by the macro processor and the language translator can be combined. A line-by-line macro processor can also makes it easier to give diagnostic messages that are related to the source statement containing error.

22. What is an integrated macro processor?

The main form of communication between the two functions is the passing of source statements from one to another. It is possible to have even closer

cooperation between the macroprocessor and the assembler or compiler. Such a scheme can be thought of as a language translator with an integrated macro processor.

An integrated macro processor can potentially make use of any information about the source program that is extracted by the language translator.

23. Write notes on MASM Macro processor?

The macro processor of MASM is integrated with pass1 of the assembler. It supports all of the main macro processor functions, including the definition and invocation of macro instructions within macros.

The comment line which begins with ; is a macro comment. It serves only as documentation for the macro definition, it is ignored when the macro is expanded. The comment line begins with ; is an ordinary assembler language comment.

24. Write notes on ANSIC Macro language?

In the ANSIC language, definition and invocations of macros are handled by a preprocessor. This preprocessor is generally not integrated with the rest of the compiler.

Two examples of ANSIC definitions are

```
# define NULL 0
# define EOF (-1)
```

After these definitions appear in the program, every occurrence of NULL will be replaced by 0, and every occurrence of EOF will be replaced by (-1).

ANSI macros can also be defined with parameters.

```
# define ABSDIFF(X,Y) ((X) > (Y) ? (X) - (Y) : (Y) - (X) )
```

25. Write notes on ELENA Macro processor?

ELENA was developed as a research tool, not as a commercial software product.

Macro definitions in ELENA are composed of a header and a body, as with most macro processors. It consists of a sequence of keyword and parameter markers.

ELENA also provides macro-time variables and macro-time instructions that can be used to control the macro expansion.

The ELENA macro processor uses a macro definition table that is similar to the one we discussed for SIC. The macro is identified by the sequence of keywords that appear in its header.

Unit V

1. What are the tasks of an editing process?

The document-editing process is an interactive user-computer dialogue designed to accomplish four tasks:

1. Select the part of the target document to be viewed and manipulated.
2. Determine how to format this view on-line and how to display it.
3. Specify and execute operations that modify the target document.
4. Update the view appropriately.

2. Write notes on selecting a document?

Selection involves first **traveling** through the document to locate the area of interest. This is accomplished by **next screenful, bottom, and find pattern**. **Filtering** extracts the point of interest such as the next screenful of text or next statement.

3. Write notes on formatting a document?

Formatting then determines how the result of the filtering will be seen as visible representation on a display screen or other device.

4. What are the input devices for editing text?

Input devices are used to enter elements of the text being edited, to enter commands, and to designate editable elements. These devices are categorized as

1. Text devices.
2. Button devices.
3. Locator devices.

5. What are text devices?

Text or string devices are typically typewriter-like keyboards on which a user presses and releases keys, sending a unique code for each key.

6. What are button devices?

Button or choice devices generate an interrupt or set a system flag, usually causing invocation of an associated application-program action. They are set of special function keys on an alphanumeric keyboard. Buttons can be simulated in software by displaying text strings or symbols on the screen.

7. What are locator devices?

Locator devices are two-dimensional analog-to-digital converters that position cursor symbol on the screen by observing the user's movement of the device. The most common such devices are **mouse** and the **data tablet**. The data tablet is a flat, rectangular, electromagnetically sensitive panel. Either a ballpoint-pen-like stylus is moved over the surface. The tablet returns to a system program the coordinates of the position of the data tablet at which the stylus is located.

8. Write notes on voice input devices?

Voice-input devices, which translate spoken words to their textual equivalents, may prove to be the text input of the future. The more modern **professional workstation** support multiple proportionally spaced character fonts to produce realistic facsimiles of hard-copy documents.

9. Write notes on command language processor?

It accepts input from the user's input devices and analyzes the tokens and syntactic structure of the commands. So it functions much like a lexical and syntactic phase of a compiler.

The command language processor may produce an intermediate representation of the desired editing operations. This intermediate representation is then decoded by an interpreter that invokes the appropriate semantic routines.

10. What are the various component involved in editor?

1. Editing component
2. Traveling component
3. Viewing component
4. Display component

11. Write notes on editing component?

In editing a document, the start of the area to be edited is determined by the **current editing pointer** maintained by the **editing component** which is the collection of modules dealing with editing tasks. The current editing pointer can be set or reset by the commands such as **next paragraph and next screen**.

12. Write notes on traveling component?

The **traveling component** of the editor actually performs the setting of the current editing and viewing.

13. Write notes on viewing component?

In viewing a document, the start of the area to be viewed is determined by the **current viewing pointer**. This pointer is maintained by the viewing component of the editor, which is a collection of modules responsible for determining the next view. When the display needs to be updated, the viewing component invokes the **viewing filter**. This component filters the document to generate a new **viewing buffer** based on the current viewing pointer as well as on the viewing filter parameters.

14. Write notes on display component?

In line editors, the viewing buffer may contain the current line. In screen editors, this buffer may contain a rectangular cutout of the quarter-plane of text. This viewing buffer is then passed to the **display component** of the editor, which produces a display by mapping the buffer to a rectangular subset of the screen, usually called a **window**.

15. What are the various types of computing environment of editors?

Editors function in three basic types of computing environments: time-sharing, stand-alone, and distributed.

16. What is time sharing environment?

The **time-sharing** editor must function rapidly within the context of the load on the computers processor, central memory, and I/O devices.

17. What is stand-alone environment?

The editor on a **stand-alone** system must have access to the functions that the time-sharing editor obtains from its host operating system.

18. What is distributed environment?

The editor operating in a **distributed** resource-sharing local network must run independently on each user's machine. Many workstations and intelligent terminals have their own microprocessors and local buffer memories in which editing manipulation can be done.

19. Define gaits?

Given a good graphical representation of the progress, it may even be useful to run the program at various speeds, called **gaits**.

20. What is tracing in debugging process?

Tracing can be used to track the flow of execution logic and data modifications. The control flow can be traced at procedure, branch, individual instruction and so on. It can also be based on conditional expression also.

21. What is traceback in debugging process?

Traceback can show the path by which the current statement was reached. It can also show which statements have modified a given variable or parameter. This kind of information is displayed symbolically.

22. What are the capabilities of a debugging system?

The debugging system has good **program-display** capabilities. It is to display the program being debugged, complete with statement numbers. It is also useful to recompile the program during the debugging session.

Most user environments and many applications systems involve the use of several different programming languages. A debugging system must be sensitive to the specific language being debugged.

23. What are the two options for display of source code during the debugging session?

In display of source code during the debugging session. There are two main options.

1. The language translator may provide the source code or source listing tagged in some standard way so that the debugger has a uniform method of navigation about it.

2. The translator may supply an interface module that does the navigation and display in response to request from the debugger.

24. Write notes on the relationship of debugger with other parts of the system?

One very important requirement for an interactive debugger is that it always be available. When an error is discovered, immediate debugging must be possible. The debugger must communicate and cooperate with other operating system components such as interactive subsystems. The debugger must also exist in a way that is consistent with the security and integrity components of the system. The debugger must coordinate its activities with those of existing and future language compilers and interpreters.

25. Write notes on user-interface criteria of a debugger?

The user interaction should make use of full-screen displays and windowing systems as much as possible. Menus should have titles that identify the task they help program. A debugging system should also support interactive users when a full-screen terminal device is not present. The command language should have a clear, logical, simple syntax. Command formats should be as flexible as possible. A good interactive system should have an on-line HELP facility.

16 marks

Unit I

1. Explain the SIC machine architecture?

Memory

Memory consist of 8-bit bytes; any consecutive bytes form a word (24bits). Addresses on SIC are byte addresses; words are addressed by the location of their lowest numbered byte. Total 32,768 (2^{15}) bytes in the computer memory.

Registers

There are **five** registers. Each register is 24 bits in length. Table below indicates the register details

Mnemonic	Number	Special use
A	0	Accumulator; arithmetic operation
X	1	Index register; used for addressing
L	2	Linkage register; the JSUB jump to subroutine instruction stores the return address in this register.

PC	8	Program counter; contains address of next instruction fetched for execution.
SW	9	Status word; contains variety of instruction, including condition code (CC).

Data formats

Integers – 24 bit binary numbers; for negative values use 2's complement.
 Characters – 8-bit ASCII codes.

Instruction formats

24 bit format.

8	1	15
Opcode	x	address

X is used to indicate indexed addressing mode.

Addressing modes

Two addressing modes

Direct	x = 0	TA (target address) = address
Indirect	x = 1	TA = address + (x)

Instruction set

Instruction that load and store registers – LDA, LDX, STA, STX, etc.

Integer arithmetic operations – ADD, SUB, MUL, DIV.

All arithmetic operations involved register A and a word in memory. The result is left in register.

Compare instruction – COMP – compare value in register A with word in memory, this sets a condition code to indicate the result.

Conditional jump instruction - JLT, JEQ, JGT can test the setting of CC and jump accordingly.

Subroutine linkage instruction – JSUB –placing the return address in register in L; RSUB – returns by jumping to the address contained in register L.

Input and output

I/O are performed by transferring 1 byte at a time. Three I/O instruction.

TD – Test whether the addressed device is ready to send or receive a byte of data.
If CC is < the device is ready to send , = = the device is not ready.

RD – Read data.

WD – Write data.

2. Write a assembly code to implement the arithmetic instruction on SIC and SIC/XE ?

a) Arithmetic operation for SIC machine

```
LDA ALPHA
ADD INCR
SUB ONE
STA BETA
LDA GAMMA
ADD INCR
SUB ONE
STA DELTA
---
---
---
ONE      WORD      1
ALPHA    RESW      1
BETA     RESW      1
GAMMA    RESW      1
DELTA    RESW      1
INCR     RESW      1
```

a) Arithmetic operation for SIC/XE machine

```
LDS      INCR
LDA      ALPHA
ADDR     S,A
SUB      #1
STA      BETA
LDA      GAMMA
ADDR     S,A
SUB      #1
STA      DELTA
---
---
---
ALPHA    RESW      1
BETA     RESW      1
GAMMA    RESW      1
DELTA    RESW      1
```

3. Explain the architecture of VAX family of computers?

VAX family of computers was introduced by Digital Equipment Corporation (DEC).

Memory

The VAX memory consists of 8-bit bytes. All addresses used are byte addresses. Two consecutive bytes form a word; four bytes form a longword; eight bytes form a quadword; sixteen bytes form an octaword.

All VAX programs operate in a virtual address space of 2^{32} bytes. It allow the programs to access extremely large memory. One half of the VAX virtual address space is called **system space**, which contains the operating system, and is shared by all programs. The other half of the address space is called **process space**, for program. A part of process space is stacks.

Registers

There are 16 general purpose registers denoted by V0 through V15. All general registers are 32 bits in length. Register R15 is a program counter(PC). R14 is a stack pointer (SP). R13 is the frame pointer (FP) , VAX procedure call convention build a data structure called stack frame, and places its addresses in FP.R12 is the argument pointer (AP). The procedure call convention uses AP to pass a list of argument associated with the call.

R6 – R11 available for general use by the program. R0 – R5 used by some machine instructions.

Processor status longword (PSL) , which contains state variables and flags associated with a process.

There are also number of control registers that are used to support various OS functions.

Data formats

Integers are stored as binary numbers in byte, word, longword, quadword, or octaword ; 2's complement representation is used for negative values. Character are stored using 8-bit ASCII codes. Four different floating point data formats on the VAX ranging from 4 to 16 bytes.

VAX provides a **packed decimal** data format. Each byte represents two decimal digits, with each digit encoded using 4 bits of the byte. The sign is encoded in the last 4 bits of the byte.

Numeric format is used to represent numeric values with one digit per byte. The sign may appear either in the last byte or as a separate byte preceding the first digit. These variations are called **trailing numeric** and **leading separate numeric**.

VAX supports queues and variable-length bit strings.

Instruction formats

Variable length instruction formats. Each instruction consist of an operation code (1 or 2 bytes) followed by upto 6 operand specifiers depending on the type of instruction.

Addressing modes

Register mode – operand itself may be in a register.

Register deferred mode – operands address may be specified by a register.

Autoincrement and autodecrement mode – register content increment/decrement

Program counter relative modes – displacement related with PC.

Deferred modes – indirect addressing.

Instruction set

Instruction mnemonics are formed by combining the following elements

1. a prefix that specifies the type of operation,
2. a suffix that specifies the data type of the operands,
3. a modifier that gives the number of operands involved.

Example of instructions

ADDW2 – add operation with two operands each a word in length.

MULL3 – multiply operation with three longword operands

CVTWL – conversion from word to long word

Input and output

I/O on VAX accomplished by device controllers. The portion of address space into which the device controller registers are mapped is called I/O **space**. The association of an address in I/O space with a physical register in a device controller is handled by the memory management routine.

4. Explain the architecture of UltraSPARC?

The UltraSPARC processor, announced by Sun Microsystems. The name SPARC stands for scalable processor architecture. This is suitable for wide range of implementations from microcomputers to supercomputers.

Memory

Memory consists of 8-bit bytes; all addresses used are byte addresses. Two consecutive bytes form a halfword; four bytes form a word; eight bytes form a doubleword.

UltraSPARC programs can be written using a virtual space of 2^{64} bytes. This address space is divided into **pages**. The virtual address space is translated in to physical address by UltraSPARC Memory Management Unit (MMU).

Registers

SPARC architecture includes a large register file that usually contains more than 100 general-purpose registers. Any procedure can access only 32 registers. The first eight of these registers can be accessed by all procedures on the system. The other 24 registers are visualized as a window thorough which part of the register file can be seen.

If a set of currently running procedures needs more windows than are physically available, a “window overflow” interrupt occurs. The operating system save the contents of some registers in the file and provide the additional windows.

In original SPARC architecture , the general purpose registers were 32 bits long. Later it was expanded to 64 bits.

Floating point computations are performed using a special floating-point unit (FPU). This unit contains a file of 64 double-precision floating point registers, and other control an status registers.

Besides these there are program counter PC, condition code registers and a number of other control registers.

Data formats

Integers are stored as 8-, 16-, 32-, or 64-bit binary numbers. Both signed and unsigned numbers are supported; 2's complement is used for negative values. In SPARC the most significant part of a numeric value is stored at the lowest-numbered address. This is called big-endian byte ordering.

Three different floating-point data formats. The single-precision format is 32 bit long. The double precision is 64 bits long. The quad-precision stores 63 significant bits. Characters are stored as one per byte in ASCII codes.

Instruction formats

Three basic instruction formats. All formats are 32 bits long. The first 2 bits of the instruction word identify which format is being used.

Format 1 – call instruction.

Format 2 – branch instruction.

Format 3 – remaining instructions (register loads and stores, and three-operand arithmetic operations).

Addressing modes

6. Immediate mode – operand value specified as part of the instruction.
7. Register direct mode – in register
8. PC relative
9. Register indirect with displacement
10. Register indirect indexed.

Instruction set

SPARC has fewer than 100 machine instructions. Load and store are the instructions that access memory. Other instructions are register to register operations.

Instruction execution is **pipelined** – while one instruction is being executed, the next one is being fetched from memory and decoded. SPARC branch instructions are **delayed branches** – the instruction immediately following the branch instruction is executed before the branch is taken.

SPARC includes special-purpose instructions to provide support for operating systems and optimizing compilers.

Input and output

Communication with I/O devices is accomplished through memory. A range of memory area is logically replaced by device registers. When a load or store instruction refers to this device register area of memory, the corresponding device is activated.

5. Briefly discuss on Cray T3E architecture?

T3E is a massively parallel processing (MPP) system, for use on technical application in scientific computing. A T3E system contains a large number of processing elements (PE), arranged in a three-dimensional network.

Each PE consists of a DEC Alpha EV5 RISC microprocessor, local memory and performance-accelerating control logic. A T3E system may contain from 16 to 2048 processing elements.

Memory

Each T3E processing element has its own local memory with a capacity of from 64 megabytes to 2 gigabytes. The local memory is physically distributed, logically shared memory system.

The memory is 8 bit bytes. Two consecutive bytes form a **word**, four bytes form a **longword**, and eight bytes form a **quadword**.

Registers

32 general purpose registers. 64 bits long. Used to store and manipulate integer data and address.

32 floating-point registers. 64 bits long. 64 bit PC and several other status and control registers.

Data formats

Integers are stored as longwords or quadwords, 2's complement is used for negative values. Characters are stored one per byte, using their 8-bit ASCII code.

Instruction formats

Five basis instruction formats. 32 bits long. The first 6 bits of the instruction word specify the opcode.

Addressing modes

The operand value may be specified as part of the instruction itself or it may be in a register

Operands in memory are addressed using one of the following two modes

PC-relative $TA = (PC) + displacement$

Register indirect with displacement $TA = (register) + displacement$

Instruction set

Approximately 130 machine instructions. It is designed to implement the architecture as fast as possible. There are no byte or word load and store instructions. This means that memory access interface does not need to include shift-and-mask operations.

Input and output

The T3E system performs I/O through multiple ports into one or more I/O channels, which can be configured in a number of ways. A system may be configured with upto one I/O channel for every eight PEs. All channels are accessible and controllable form all PEs.

Unit II

1. Explain about a simple SIC assembler?

The translation of source program to object code by a simple SIC assembler requires us to accomplish the following functions

- a) Convert the mnemonic operation codes to their machine language equivalents.
- b) Convert symbolic operands to their equivalent machine address.
- c) Build the machine instructions in the proper format.
- d) Convert the data constants specified in the source program into their internal representation.
- e) Write the object program and the assembly listing.

A reference to a label that is defined later in the program is called forward reference.

```
FIRST      STL      RETADR
           ---
           ---
RETADR     RESW     1
```

Here REGADR label is an example of forward reference because; the label is used before it is declared.

To translate the instructions of the source program, the assembler must process statements called assembler directives. These statements are not translated into machine instructions. Finally, the assembler must write the generated object code onto some output device. This object program will later be loaded into memory for execution. The simple object program format we use contains three types of records Header, Text and End.

The Header record contains the program name, starting address, and length. The format is

```
Col.1      H
Col. 2-7   Program name
Col 8-13   Starting address of object program
```

Col 14-19 Length of object program in bytes

Text record contains the translated instructions and data of the program, together with an indication of the address where they are to be loaded. The format is

Col.1	T
Col. 2-7	Starting address for object code in this record
Col 8-9	Length of object code in this record in bytes
Col 10-69	Object code, represented in hexadecimal

The End record marked the end of the object program and specifies the address in the program where execution is to begin. The format is

Col 1	E
Col. 2-7	Address of first executable instruction in object program

The general descriptions of the functions of the two pass assembler are

Pass 1

- a) Assign address to all statements in the program.
- b) Save the values assigned to all labels for use in Pass2.
- c) Perform some processing of assembler directives.

Pass 2

- a) Assemble instruction.
- b) Generate data value define by BYTE, WORD, etc.
- c) Perform processing of assembler directives not done during Pass 1.
- d) Write the object program and the assembly listing.

2. Write the algorithm for Pass 1 of assembler?
3. Write the algorithm for Pass 2 of assembler?

4. i) Explain about program relocation?

When more than one program at a time, sharing the memory and other resources of the machine. The address for loading is not known in advance.

Since the assembler does not know the actual location where the program will be loaded, it cannot make the necessary changes in the address used in the program. The assembler can identify those part of the object program that need

modification. An object program that contains the information necessary to perform his kind of modification is called **relocatable program**.

The command for the loader must also be a part of the object program. We can accomplish it with a modification record. The format is

col 1	M
col 2 -7	Starting address of the address field to be modified, relative to the beginning of the program.
col 8 – 9	Length of the address field to be modified in half-bytes.

Text records are exactly the same as those that would be produced by an absolute assembler. However the load addresses in the text records are interpreted as relative, rather than absolute locations.

ii) Write notes on symbol defining statements?

Most assembler provides an assembler directive that allows the programmer to define symbols and specify their values. The assembler directive generally used is EQU. The general form is

symbol EQU value

This statement defines the given symbol in the SYMTAB and assigns to it the value specified.

Another common use of EQU is in defining mnemonic names for registers.

A	EQU	0
X	EQU	1
L	EQU	2

...

These statements cause the symbols A,X,L, to be entered into SYMTAB with their corresponding values 0,1,2..

Now the programmer can write RMO 0,1 instead of RMO A,X.

The usage of register is changed from one program to next is done by

BASE EQU R1

```
COUNT    EQU  R2
INDEX    EQU  R3
```

There is another common assembler directive that can be used to indirectly assign values to symbols is ORG.

```
ORG value
```

Where value is a constant or an expression. When this statement is encountered during assembly of a program, the assembler resets its location counter (LOCCTR) to the specified value.

5. Explain control section and program linking in assembler?

A control section is a part of the program that maintains its identity after assembly; each such control section can be loaded and relocated independently of each other's.

When control sections form logically related parts of a program, it is necessary to provide some means for linking them together.

Because control sections are independently loaded and located, the assembler has no idea where any other control section will be located at execution time. Such references between control sections are called *external references*.

The assembler also includes information in the object program that will cause the loader to insert the proper values where they are required. We need two new record types.

The two new record types are Define and Refer. A Define record gives information about external symbols that are defined in this control section. A Refer record lists symbols that are used external references by the control section. The format of these records is

Define record

```
col 1      D
col 2- 7   Name of external symbol defined in this control section.
col 8 - 13 Relative address of symbol within this control section.
col 14 - 73 Repeat information in col 2-13.
```

Refer record

```
col 1      R
```

col 2 – 7 Name of external symbol referred to in this control section.
col 8 – 73 names of other external reference symbols.

Modification record

col 1 M
col 2-7 Starting address of eth field to be modified, relative to the
beginning of the control section.
col 8 – 9 Length of the field to be modified.
col 10 Modified flag
col 11 – 16 External symbol whose flag value is added or subtracted
from the indicated field.

Unit III

1. i) Explain about the design of an absolute loader?
ii) Explain about a simple bootstrap loader?
- 2.
3. Explain the algorithm for Pass 1 of a linking loader?

Begin

Get PROGADDR from operating system

Set CSADDR to PROGADDR{ for first control section}

while not end of input **do**

begin

 read next input record{Header record for control section}

 set CSLTH to control section length

 search ESTAB for control section name

if found **then**

 set error flag{duplicate external symbol}

else

 enter control section name into ESTAB with value CSADDR

while record type \neq 'E' **do**

begin

 read next input record

if record type = 'D' **then**

for each symbol in the record **do**

begin

 search ESTAB for symbol name

if found **then**

 set error flag (duplicate external symbol)

else

 enter symbol into ESTAB with value
 (CSADDR + indicated address)

end {for}

```

    end {while ≠ 'E'}
    add CSLTH to CSADDR {{starting address for next control section}}
  end {while not EOF}
end {Pass 1}

```

During the first pass, the loader is concerned only with Header and Define record types in the control sections. The control section name from the Header record is entered into ESTAB, with value given by CSADDR. All external symbols appearing in the Define record for the control section are also entered into the ESTAB. When the End record is read, the control section length CSLTH is added to CSADDR.

At the end of pass 1, ESTAB contains all external symbols defined in the set of control sections together with the address assigned to each.

4. Explain the algorithm for Pass 2 of a linking loader?

```

begin
  set CSADDR to PROGADDR
  set EXECADDR to PROGADDR
  while not end of input do
    begin
      read next input record {Header record}
      set CSLTH to control section length
    while record type ≠ 'E' do
      begin
        read next input record
      if record type = 'T' then
        begin
          {if object code is in character form, convert into
            internal representation}
          move object code from record to location
            (CSADDR + specified address)
        end {if 'T'}
        else if record type = 'M' then
          begin
            search ESTAB for modifying symbol name
          if found then
            add or subtract symbol value at location
              (CSADDR + specified address)
          else
            set error flag (undefined external symbol)
          end {if 'M'}
        end {while ≠ 'E'}
      if an address is specified {in End record} then
        set EXECADDR to (CSADDR + specified address)
      end
    end
  end

```

```
        add CSLTH TO CSADDR
    end {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

Pass 2 performs the actual loading, relocation and linking of the program. CSADDR contains the actual starting address of the control section currently being loaded. As each Text record is read, the object code is moved to the specified address. When Modification record is encountered, the symbol whose value is to be used to modification is looked up in ESTAB. This value is added or subtracted from the indicated location in memory.

The last step is to transfer the control the loaded program to begin execution. The End record for each control section may contain the address of the first instruction in that control section. If no control section contains a transfer record, the loader used the beginning of the linked program.

The pass 2 algorithm is modified by assigning a reference number to each external symbol referred to in a control section. This reference number is used in the Modification records. This mechanism avoids multiple searches of ESTAB for the same symbol during loading of a control section.

5. Explain the various loader design options?

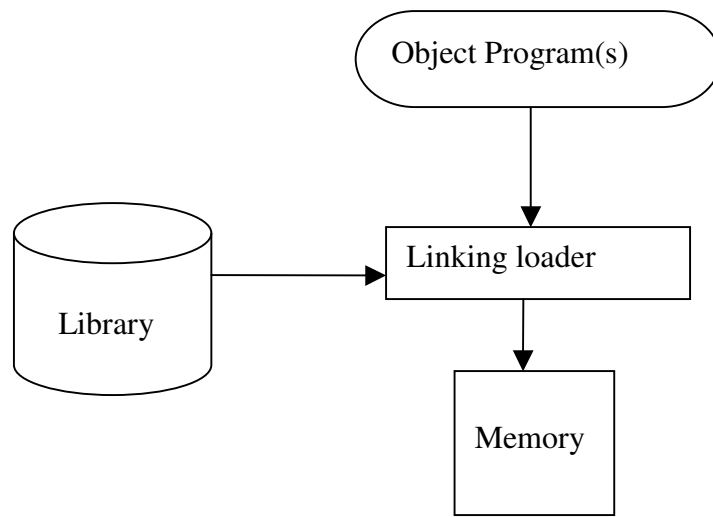
Linkage editors

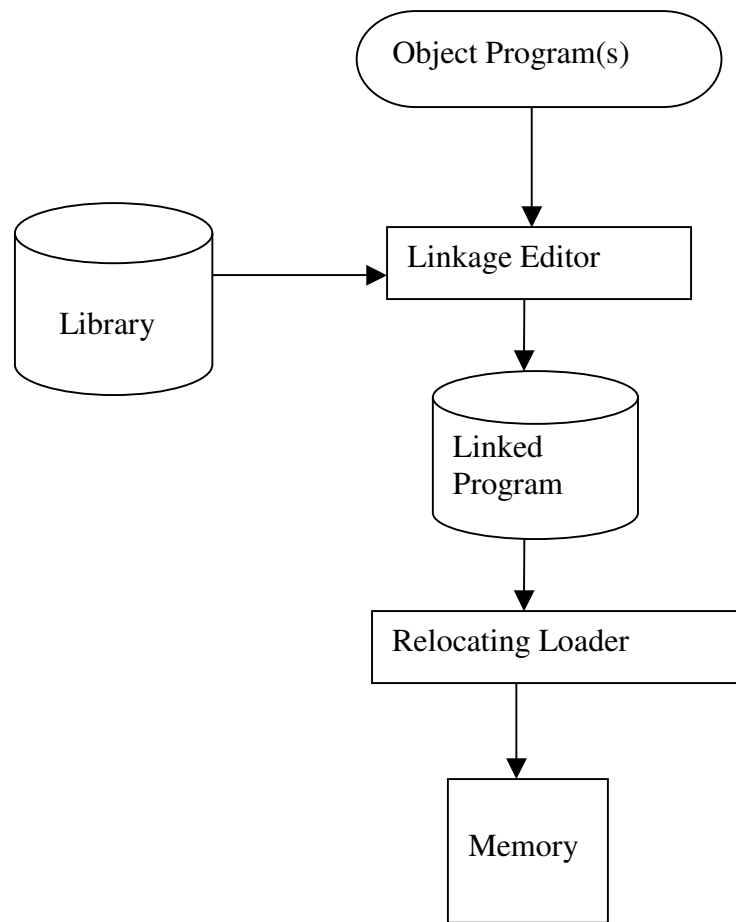
The source program is first assembled or compiled, producing an object program.

A *linking loader* performs all linking and relocation operations, including automatic library search, and loads the linked program directly into the memory for execution. A linking loader searches libraries and resolves external references every time the program is executed.

A *linkage editor* produces a linked version of the program, which is written to a file or library for later execution. The linkage editor performs relocation of all control sections relative to the start of the linked program. This means that the loading can be accomplished in one pass with no external symbol table required. The linked program produced by a linkage editor is generally in a form that is suitable for processing by a relocating loader.

Processing of an object program using linking loader and linkage editor is given below.





If the actual address at which the program will be loaded is known in advance, the linkage editor can perform all the needed relocation. The content and processing of such an image are the same as for an absolute object program.

Linkage editors can be used to build packages or subroutines or other control sections that are generally used together. If a program using formatted I/O were linked in the usual way, all of the cross-references between these libraries subroutines would have to be processed individually. The linkage editor could be used combine the appropriate subroutines into a package.

Linkage editors often allow the user to specify that external references are not to be resolved by automatic library search.

Dynamic linking

This scheme postpones the linking function until execution time: a subroutine is loaded and linked to the rest of the program when it is first called. This type of function is usually called *dynamic linking*, *dynamic loading* or *load on call*.

Dynamic linking is often used to allow several execution programs to share one copy of a subroutine or library.

Refer Book

Bootstrap loaders

Refer Book