# Spring - 2012

(1)(a) Explain the main purpose of system software in computer.

→ System software consists of a variety of programs that support the orientation of a computer. It is a set of programs to perform variety of system functions like file editing, resource management, I/O management, storage management, etc. System softwares support the operation and use of computer. Computer consists of various types of system softwares like operating system, language translators, loaders, linkers and macroprocessors. By the use of OS, computer have interface that communicates between users and system. It makes computer user-friendly and easier to use. It helps in process, memory and resource management, input/output operations, etc. System also provides language translators which takes input in one language and outputs in another language. For e.g.: Interpreter, compiler, assembler, etc.

Since, computer cannot perform it's task or cannot run effectively, system software is necessary in computer.

(b) Write an assembly language program of simple arithmetic operation for CISC Architecture.

→
```
START 2000
    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
```

(c) Differentiate the RISC and CISC machine with example.

→

| RISC machine | CISC machine |
|---|---|
| (1) RISC stands for Reduced Instruction Set Computer. | (1) CISC stands for Complex Instruction Set Computer. |
| (2) It has fixed standard instruction format. | (2) It has variable instruction format. |
| (3) It has less number of addressing modes. | (3) It has many addressing modes. |

| RISC machine | CISC machine |
|---|---|
| (4) It has large numbers of registers. | (4) It has few number of registers. |
| (5) E.g.: UltraSPARC, Power PC, Cray T3E Architecture. | (5) E.g.: VAX, Pentium Pro Architecture, etc. |
| (6) It has heavy pipelining of instructions. | (6) It has low pipelining of instructions. |
| (7) It has few number of instructions. | (7) It has high number of instructions. |
| (8) It is hardwired control unit. | (8) It is microprogrammed control unit. |
| (9) Speed is high. | (9) Speed is slow. |
| (10) It requires single cycle for execution. | (10) It requires more than 1 cycle for execution. |

(2)(a) Write the algorithm of Pass-I assembler.

→ Algorithm for PASS-I assembler:

```
Pass 1:
begin
    read first input line
    if OPCODE = 'START' then
        begin
            save #[OPERAND] as starting address
            initialize LOCCTR to starting address.
            write line to intermediate file.
            read next input line.
    end {if START}.
```

```
else
    initialize LOCCTR to 0.
while OPCODE ≠ 'END' do.
    begin
        if this is not a comment line then
            begin
                if there is a symbol in LABEL field the
                begin
                    search SYMTAB for LABEL
                    if found then
                        set error flag
                    else
                        insert {LABEL, LOCCTR} into
                        SYMTAB.
                    end {if symbol}.
                search OPTAB for OPCODE.
                if found then
                    add 3 {instruction length} to LOCCTR.
                else if OPCODE = 'WORD' then
                    add 3 to LOCCTR.
                else if OPCODE = 'RESW' then
                    add 3 * #[OPERAND] to LOCCTR.
                else if OPCODE = 'RESB' then
                    add # [OPERAND] to LOCCTR.
                else if OPCODE = 'BYTE' then
                    begin
                        find length of constants in bytes
```

add length to LOCCTR.
end {if BYTE}.
else
set error flag (invalid operation code)
end {if not a comment}.
write line to intermediate file.
read next input line.
end {while not END}.
write last line to intermediate file.
save (LOCCTR - starting address) as program length.
end {Pass 1}.

(b) Explain how object codes are generated by assembler with example.

→ Assembler performs following functions to generate object code:

(1) Convert mnemonic operation codes to their machine language equivalents.
(2) Convert symbolic operands to their equivalent machine addresses.
(3) Build machine instructions in proper format.
(4) Convert data constants specified in source program into their internal machine representations.
(5) Write object program and assembly listing.

for example:

| Line | Loc | Source statement | Object code |
|---|---|---|---|
| 10 | 1000 | FIRST STL RETADR | 141033. |

To translate the source code into object code, the assembler converts STL to machine equivalent value 14. The symbolic operand RETADR is translated to 1033.

In the example, the instruction contains a forward reference. i.e. reference to label is defined later in the program. It is impossible to process this line as the address that will be assigned to RETADR is not known. So, most assemblers make two passes over source program where the second pass does actual translation.

The assembler must process statements called assembler directives which are not translated into machine instructions but they provide instructions to assembler. E.g.: RESW and RESB instruct assembler to reserve memory locations.

The assembler writes the generated output code onto the output device, which will be loaded into memory for execution. It writes object code in three formats; header, text and end record.

If RETADR is defined at location 1033, then, the object code ~~is~~ generated ~~as~~ is:

141033.

(3)(a) Explain the use of loader and linker in system ~~syst~~ software.

→ A loader is a system program that performs the loading function. Linker is used to perform linking operations. Loader brings the object program into memory and starts its execution. An absolute loader completes its task in a single pass. It checks ~~whether~~ the header record to verify that ~~is~~ correct program is loaded into memory. The loader reads each text record and moves the object code into the indicated ~~loc~~ address in memory. When End record is encountered, loader jumps to the specified address to begin execution of loaded program.

Linker is used to link ~~relate~~ logically related ~~parts of~~ program. For example, instructions in one control section might need to refer to instructions in another section. As control sections are loaded and relocated independently, assembler is unable to process those references, called ~~as~~ as external references. The assembler generates the information for each external reference so that ~~later~~ loader can perform required linking.

Thus, loaders help in loading the program ~~and~~ into memory and linker links the various programs to provide executable code. Also, loader helps in automatic library search, which helps in handling external references and to use standard subroutines which are required when linking.

(b) How different program linkings is done. Explain with the help of examples.

→ Let us consider following examples:

```
LOC          Statement          Object code
0000    PROGA   START   0
                EXTDEF  LISTA, ENDA
        REF     EXTREF  LISTB, ENDB, LISTC, ENDC

0020    REF1    LDA     LISTA
0023    REF2    +LDT    LISTB+4
0027    REF3    LDX     #ENDA-LISTA
                :

0040    LISTA   EQU     *
0054    ENDA    EQU     *
                ;
```

| LOC | | Statements. | |
|---|---|---|---|
| 0000 | PROGB | START | 0 |
| | | EXTDEF | LISTB, ENDB |
| | | EXTREF | LISTA, ENDA, LISTC, ENDC. |
| 0036 | REF1 | +LDA | LISTA |
| 003A | REF2 | LDT | LISTB+4 |
| 003D | REF3 | +LDX | #ENDA-LISTA |
| 0060 | LISTB | EQU | * |
| 0070 | ENDB | EQU | *. |

Here, in both programs, the values of data words are:

| | | |
|---|---|---|
| REF4 | WORD | ENDA-LISTA+LISTC |
| REF5 | WORD | ENDC-LISTC-10 |
| REF6 | WORD | ENDC-LISTC+LISTA-1 |
| REF7 | WORD | ENDA-LISTA-(ENDB-LISTB) |
| REF8 | WORD. | *LISTB-LISTA |

Here, each program contain separate control sections. Each program contain list of items LISTA, B and C.. REF1, REF2 and REF3 are instruction operands. In first reference REF1 for PROGA, no modification for linking is necessary. But in PROGB, the same operand refers to external symbol. The assembler uses extended format instruction with address field set to 00000. The object program for PROGB contains modification ~~to~~ ~~to~~ record instructing

loader to add value of LISTA to that address field when program is linked. for REF2 in PROGA, assembler stores value of constant in address field of instruction and modification record tells loader to add that field value of LISTB to that field. But in PROGB, no linking is required.

Similarly in REF3, in PROGA, assembler has all information necessary for computing the value. But in PROGB, values of labels are unknown. So, expressions must be assembled as external reference.

In this way program linking is done.

(4)(a) Explain the dynamic linking in macro.

→ Dynamic linking is the process of loading a subroutine and linking to the rest of the program when it is first called. It allows several programs to use one copy of a subroutine or library. The commonly used macros are stored in dynamic link library. Then a single copy of the macro is loaded into the computer. All the programs that are in execution are linked to that single copy of macro instead of linking into separate copy of macro.

In object-oriented system dynamic linking is used for references to software objects. It also allows the several programs to share a single object.

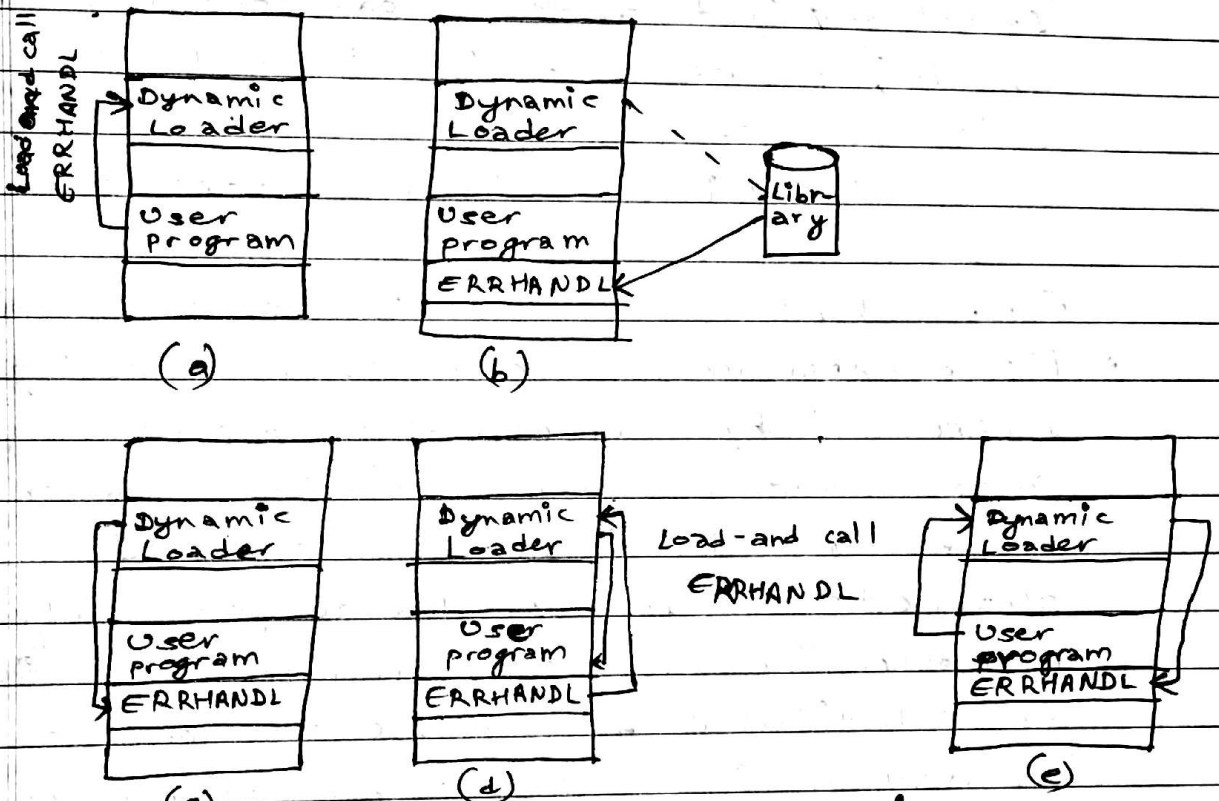



Fig.: Loading and calling using dynamic linking.

In figure, instead of executing JSUB instruction, program makes load and call service request to operating system. The OS examines internal tables to determine whether or not the routine is already loaded. If required, then routine is loaded from user program or system libraries. Then, control is passed from OS to routine being called. When subroutine completes its processing, it returns to its caller and OS returns control

to program that made the request. Also, the allocated memory is released. If subroutine is still in memory, then a second call to it may not required another load operation. Control can be directly passed from dynamic loader to called routine.

In this way, dynamic linking is done.

(b) Explain about macro expansion with an example.

→ Macro expansion is the precess of replacing macro call by corresponding sequence of instructions. In macro expansion, whenever a macro invocation is done, the invoked statements will be expanded to form a body of macro. The arguments from macro invocation are substituted for parameters in macro prototypes. The arguments and parameters are associated according to their positions. i.e. first argument in invocation corresponds to first parameter in macro prototype.

For e.g.: START 0

```
INCR  MACRO &arg1, &arg2, &arg3
      A     1, &arg1
      A     2, &arg2
      A     3, &arg3
      MEND
```

```
LOOP1  INCR  data1, data2, data3.
        :
LOOP2  INCR  data2, data3, data1
        :
data1  WORD  3
data2  WORD  2
data3  WORD  1
```

Here, the program is expanded at LOOP 1 and LOOP 2. When the statement "LOOP1 INCR data1, data2, data3" is invoked, the control is transferred to the statement "INCR MACRO &arg1, &arg2, &arg3". Then, the arguments data1, data2, data3 replace the parameters arg1, arg2, arg3 respectively. Then there will be a new block of statements as:

```
LOOP1  A  1, data1
       A  2, data2
       A  3, data3
```

Similarly, for LOOP2, we will have,

```
LOOP2  A  1, data2
       A  2, data3
       A  3, data1.
```

(c) Explain briefly about machine independent loader features.

→ Loading and linking are the OS service function. The machine independent loader features are:

(i) Automatic Library Search:

Many linking loaders can automatically incorporate routines from a subprogram library into the program being loaded. The linking loader must keep track of external symbols that are referred to but not defined in primary input to loader. The use of automatic library search helps in handling external references. It allows to use standard subroutines without explicitly including them in program to be loaded. The routines are automatically retrieved from a library as they are needed during linking.

(ii) Loader Options:

Loaders allow user to specify options that modify the standard processing. Loaders have special commands language to specify options. Mainly there are 3 types of loader options. They are:

(1) Typical loader option 1:

It allows selection of alternative sources of input. E.g.: INCLUDE program-name(library name)

(2) Loader option 2:

It allows user to delete external symbols or entire control sections.

Eg: DELETE csect-name.

(3) Loader option 3:

It involves the automatic inclusion of library routines to satisfy external references:

E.g.: LIBRARY MYLIB.

Similarly, NOCALL STDDEV, PLOT, CORREL

It avoids overhead of loading and linking the unneeded routines and saves memory space.

(5)(a) Define two different development processes that Booch suggested.

→ Booch suggested two development processes micro and macro. Booch's macro process represents the overall activities of development team on a long-range scale. It includes following activities:

(1) Establish requirements for software.
(2) Develop an overall model of the system's behavior.
(3) Create an architecture for the implementation.
(4) Develop implementation through successive refinements.
(5) Manage continued evolution of a delivered system.

Similarly, Booch's micro process represents daily activities of system developers:

(1) Identify classes and objects of system.
(2) Establish behaviour and other attributes of classes and objects.
(3) Analyze relationships among classes and objects.
(4) Specify the implementation of classes and objects.

(6)(b) Explain about ANSI Macro Language.

→ In ANSI C language, definitions and invocations of macros are handled by a preprocessor. The preprocessor is generally not integrated with the rest of the compiler.

For e.g.: #define NULL 0
#define EOF (-1)

These two ANSIC macro definitions means that the every occurrence of NULL will be replaced by 0 and EOF by (-1).

Similary, if the macro #define EQ == is defined then,

while (I EQ 0) means that statement is converted into

while (I == 0)

by the macro processor.

ANSI C macros can be defined with

parameters such as

#define ABSDIF(X, Y) ((X)>(Y)?(X)-(Y):(Y)-(X))

Here, macro name is ABSDIF, parameters are X, Y. If (X)>(Y) condition is true, value of the expression is (X)-(Y) otherwise value is (Y)-(X).

In ANSI C, macro invocation can be done as

ABSDIF (3+1, 10-8)

Then, it is expanded as,

3+1>10-8? 3+1-10-8 : 10-8-3+1

which would produce output -14 instead of 2.

ANSI C provides stringizing operator #. When name of macro parameter is preceded by #, argument is substituted.

for e.g.: #define DISPLAY (EXPR) printf (#EXPR "= %d \n", EXPR)

Then, invocation:

DISPLAY (I*J+1)

would be expanded as

printf ("I*J+1" "= %d \n", EXPR).

Moreover, ANSI C provides conditional compilation statements to be sure that a macro is defined at least once.

(7) Write short notes on:

(a) Multi-pass assembler.

→ Multipass assembler completes its task in more than one pass. In multipass assembler, only the parts of the program having forward references must be processed in multiple passes, but not the whole program need multiple passes. To implement multipass assembler, we use a symbol table to store the symbols that are not defined. We store name and numbers on the table. When a symbol is defined, we use it's value to recvaluate values of other symbols.

Thus multipass assembler completes its task in two pass. In first pass, it scans code, validates it and creates symbol table. In second pass, it solves forward references and converts assembly code to machine code.

(b) Algorithm and data structure for a linking loader.

→ The algorithm for linking loader is more complicated than absolute loader algorithm. A linking loader makes two passes over it's input, which is similar to that of assembler. Pass 1 assigns address to all external symbols.

Pass 2 performs actual loading, relocation and linking.

The main data structure needed for linking loader is an external symbol table ESTAB. This table is analogous to SYMTAB in assembler algorithm. It is used to store name and address of each external symbol in the set of control sections being loaded. A hashed organization is used for this table. The other two variables are PROGADDR (program load address) and CSADDR (control section address). PROGADDR is beginning address in memory where linked program is to be loaded. Its value is supplied by OS to loader. CSADDR contains starting address assigned to control section currently being scanned by loader.

(c) Program blocks in assembler:

→ Program block allows the generated machine instructions and data to appear in object program in a different order from the corresponding source statements. Program blocks refer to the segments of code that are rearranged within a single object program unit. The assembler directive USE indicates which portion of source program belongs to various blocks. At beginning, program statements are assumed to be of default block. If no USE statement is used then, entire program belongs to single block.

Program block can contain several segments of source program. The assembler rearranges the segments to gather the pieces of each blocks together. These blocks will be assigned addresses in object program in the order in which they were first begun in source program.

E.g.:

```
COPY    START   0
FIRST   RSTL    RETADR
CLOOP   JSUB    RDREC
        :
        JSUB    WRREC
        :
        USE
RDREC   CLEAR   X
        CLEAR   A
        :
        USE
WRREC   CLEAR   X
        :
```

Here, there are different program blocks RDREC and WRREC to read and write record into buffer.

Name: Bishworaj Damichhane
ROLL: 8
2013 Subject: S.P

Himal Date Page 1

① What are system s/w ? Explain its role in a system.

⇒ 1st part:

System software is a set of program to perform a variety of system functions file editing, resource management, I/O management and storage management. The characteristic in which system software differ from application software machine dependency.

Rules of s/w system are:

- An operating system is the most important system program that act as an interface both the user and the system. It makes computer easier to use.
- Language Translator is the program that takes an input program in one language and produces an output in another language.
- A Compiler is a language program that translates program written in any high-level language into its equivalent machine language program and interpreter executes it immediately.

Himal
Date
Page

(b) Describe the roles of data structure OPTAB, SYMTAB & LOCCTR in machine dependent pass 1 Algorithm.

⇒ The machine dependent pass 1 Algorithm defines symbols, assign address to all statements in program, save the addresses assigned to all labels for use in pass 2. and also perform some processing of assembler directives.

- An operation code (OPTAB) contain the memonic code and translate them into their machine Language equivalents. It also contains the information about instruction format & length. In pass 1, OPTAB is used to LOOKUP and validate operation codes in the source code program.

- Symbol Table (SYMTAB) include their name and value for each label in the source condition. During pass 1 of the assembler, labels are entered into SYMTAB as they are encountered in the source program along their assigned address

AB.

- Location counter (LOCCTR) is initialized to the beginning address specified in the START statement. After each source statement is processed, the length of the assembled instruction or data area is added to LOCCTR. Whenever a label is reached in the source program, the current value of LOCCTR gives the address to be associated with that label.

2(a) Describe the architecture of SIC & SIC/XE machines.

- Simplified Instruction Code (SIC) is a typical microcomputer which comes in two version i.e The standard model & XE version. SIC machine architecture consist of memory, Registers, Data formats, Instruction formats, Addressing mode & I/O.

- Memory consist of bytes (8 bits), words addressed by the location of the lowest numbered byte. There are totally, 32768 byte in memory.

Himal
Date
Page

- Registers: There are five registers namely
  * Accumulator (A)
  * Index (X)
  * Linkage (L)
  * Programming counter (PC)
  * Status word (SW)

- Data format: Integers are stored as 24-bit binary numbers. They don't support floating point data items.

- Instruction formats are of 24-bits word

| Opcode (8) | x(1) | Address (15) |
|---|---|---|

Where X represent flag bit.

- Addressing modes:

| Mode | Indication | Target Address |
|---|---|---|
| Direct | x = 0 | TA = Address |
| Indexed | x = 1 | TA = Address |

- Instruction set: Data movement (LDA, LDX, STA, STX etc), Arithmetic Operating ins, Ex- ADD, SUB etc, Branching ins. Eg - JLT, JEQ, etc. Subroutine Linkage Instruction Ex: JSUB, RSUB etc.

I/O : Input output is performed by transferring one byte at a time to or from the right most 8 bits of register A.

SIC / XE Architecture

→ Memory - 1 word = 24 bits
- Total (SIC/XE) = $2^{20}$ (1048576) bytes.

→ Register - 10 x 24 bit register
- Accumulator (A), Index (X), Linkage (L), Base (B), General register (S, T) Floating point accumulator (F) - 48 bits, Programming Counter (PC), status word (SW) are register

→ Data format: Integers are stored on 24 bit, 2's complement format, Characters - 8-bit ASCII and floating point 48 bit signed - exponent fraction format.

→ Instruction format:
format 1 (1 byte): | OP(8) |
format 2 (2 byte): | OP(8) | r1(4) | r2(4) |
format 3 (3 byte): | OP(6) | n | i |

→ Instruction: SIC provides 26 instruction, SIC/XE provides an additional 33 instruction (59 total)

→ I/O: $2^8$ (256) I/O device, has unique 8-bit address
- SIC/XE has 6 I/O instruction i.e Read Data (RD), WD, TD, SIO-Start I/O, Halt I/O (HIO), TIO-Test I/O

Q(b) How does program relocation takes place in machine dependent assembler? Also explain how forward reference concept is used.

⇒ The assembler doesn't know the actual location where the program will be loaded, it cannot make the necessary changes in the address used by the program. However, the assembler can identify for the loader those parts of the object program that need modification. An object program contains the information necessary to perform this kind of modification. During the modification process program relocation takes place in machine dependent assembler.

Date
Page

9b 2nd part

que
3 (a) What is program linking? Explain with the help of an example.

⇒ Program Linking is the program that combine two or more separate object programs and supplies information needed to allow cross-references.

Example:

| Loc | Source statement | Object code |
|---|---|---|
| 000 | PROGB START 0 | |
| | EXTDEF LISTB, ENDB | |
| | EXTDEF LISTA, ENDA | |
| | LISTC, ENDC. | |

| | | | | |
|---|---|---|---|---|
| 0036 | REF1 | +LDA | LISTA | 031000 |
| 003A | REF2 | LDA | LISTB+4 | 772027 |
| 003D | REF3 | +LDX | #ENDA-LISTA | 051000 |

Consider the program.
— Each of which consist of single control Section. A Lists of items, LISTA - ENDA, LISTB - ENDB, LISTC - ENDC. Each program contains exactly the same set of refe -rences to these external symbols. Instruction operands are REF1, REF2, & REF3. No involved in the linking are omited.

3(b) Illustrate briefly how linkage editor works in machine independant linkers.

⇒ A linkage editor produces a linked version of the program i.e load module or executable image which is written to turn the a file or library for later execution. When the user is ready to run the linked program, a simple relocating loader can be used to load the program into memory. The only object code modification necessary is the addition of an actual load address to relative

values within the program. The LE performs relocation of all control sections relative to the start of the linked program.

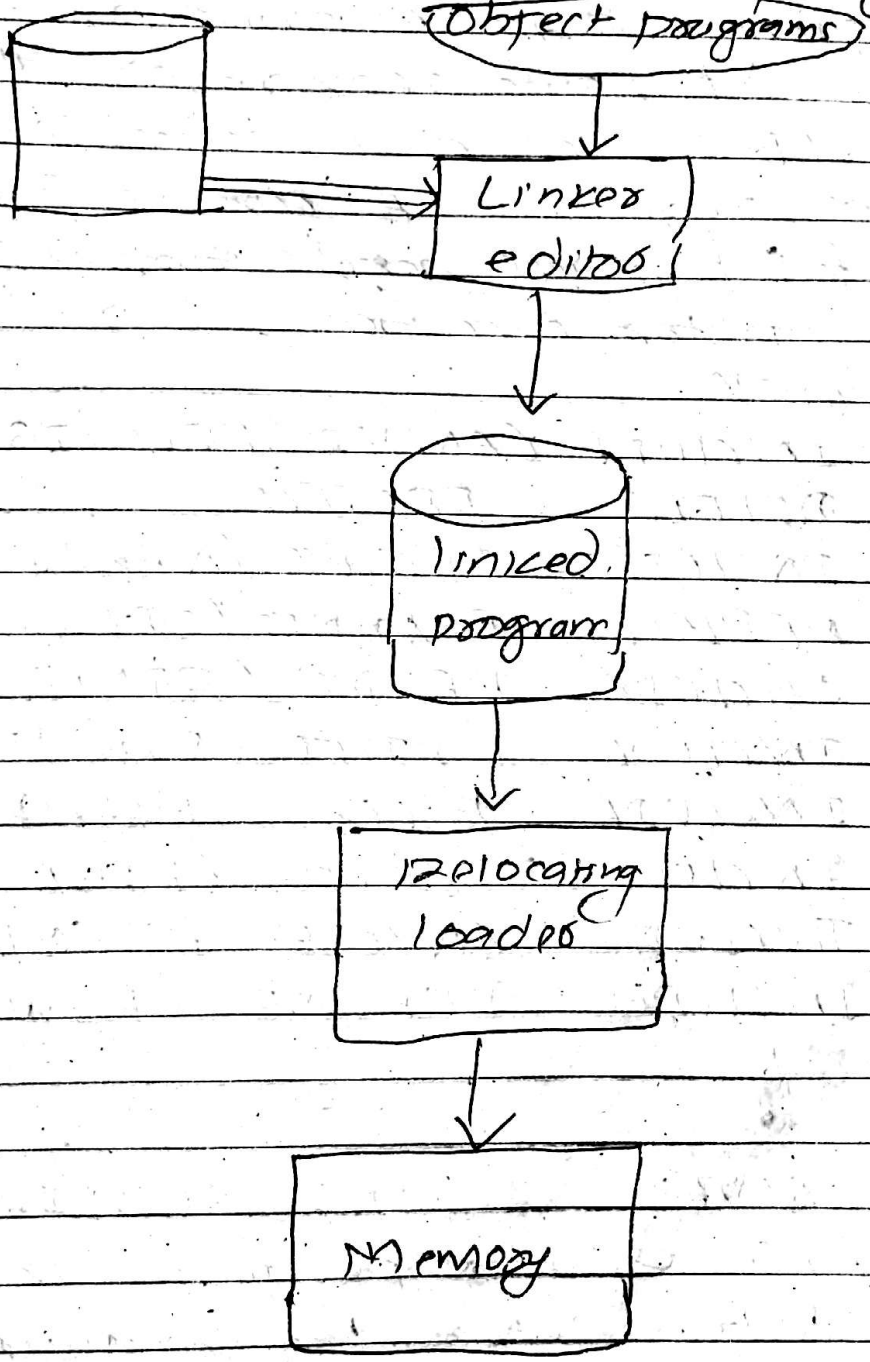



Fig :- Processing of an object code using linkage editor.

Himal
Date
Page

All the items that needs to be modified at load time have values that are relative to the start of the linked program. If a program is to be ex-ecuted many times without being reassembled, the use of LE substantially reduces the over head required. LE can perform many useful functions besides simply preparing an OP for execution:
Example.

```
INCLUDE CANNER(DROGLIB)
DELETE  DBOJECT
INCLODE DROJECT(NEWLIB)
REPLACE DBANNER(DROGLIB
INCLUDE READR(FINLIB)
INCLUDE WRITER(FINLIB)
INCLUDE BLOCK(FINLIB)
INCLUDE DEBLOCK(FINLIB)
INCLUDE CNCODE(FINLIB)
INCLUDE DENCODE(FINLIB)
  .
  .
SAVE    FINIOCSOBLIB
```

Linking editors perform the linking operation before the program is loaded from execution time.

5 (a) Explain how conditional macro operation is implemented?

⇒ macroprocessor maintains a symbol table that contain the current value of all macro variables. Implementation is easy as (IF-ELSE-ENDIF structure). A symbol table:

(i) The table contains the value of all macro-time variable used.

(ii) Entries in the table are made or modified when SET statements are processed.

(iii) This table is used to look up the current value of macro time variable whenever it is required.

When an IF statement is encountered during the expansion of macro, the specified Boolean expression is evaluated

(1) TRUE

(a) The macro processor continues to process line from DEFTAB until it encounters the next ELSE or ENDIF statement.

(b) If ELSE is encountered, then skips to END IF.

(2) FALSE:

- The macro processor skips ahead in DEFTAB until it finds the next ELSE or ENDIF statement.

Himal
Date
Page

5(b) What is the process to generate unique labels in macro process?

⇒ It is in general not possible for the body of a macro instruction to contain labels of usual kind.

□ W

Himal
Date
Page

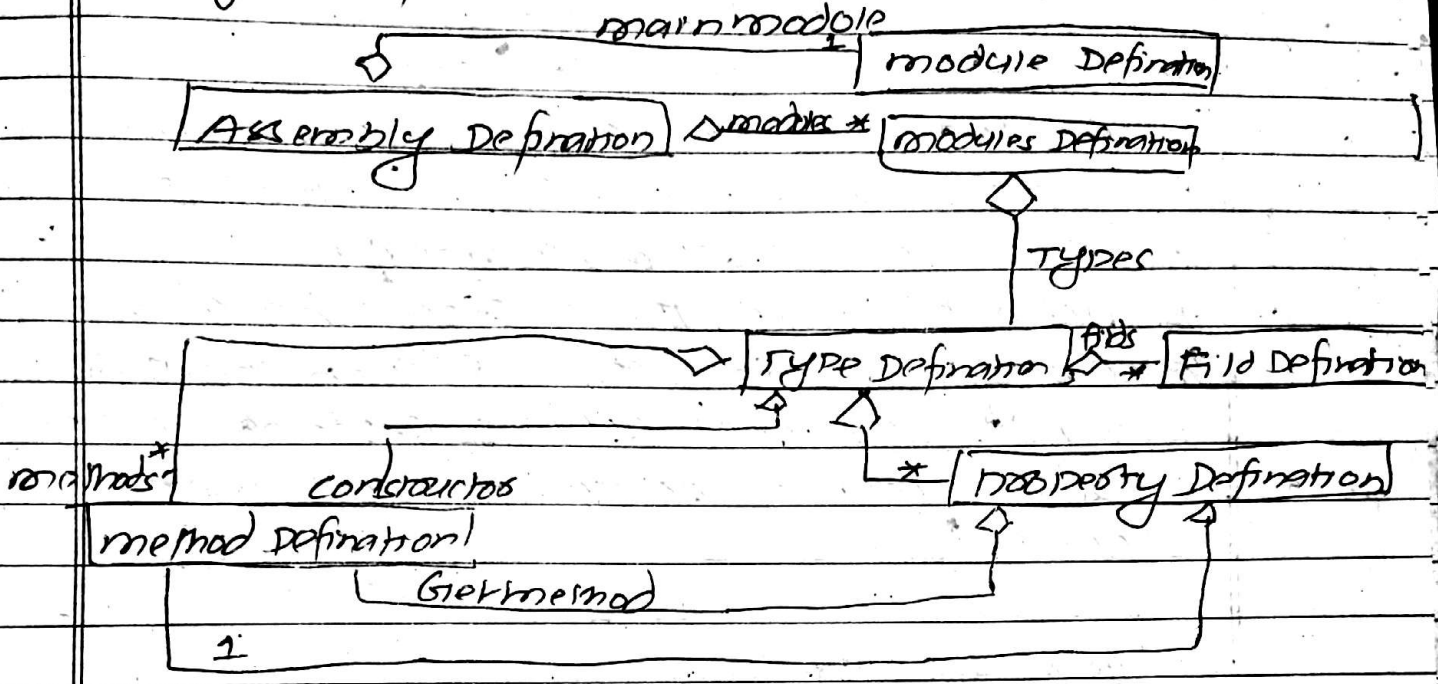
5 © What are the advantage of combining macro processor with language translator.

- Macro processing within language translators are: line-by-line macro processor and Integrated macro processor. It can be used with variety of different language

Advantage

- programmer doesn't need to learn about a different macro facility for each compiler or assembler language.
- costs involved in producing a different macro processor for each language is not needed.
- Avoids making a extra pass over the source program.
- More efficient - some of the data structure can be combined.
- Make it easier to give diagnostic messages related to the source statement containing the error.

6(a) Explain the content and method of object diagram for assemblies.




module Definition contains TypeDefinitions. Each of them contains collection of:

- method Definition, Field Definition & property Definition. We can get the constructor of a type by using the Constructor property. A constructor is a method-Definition. A property Definition own two method Definition which corresponds to the Get-& set. A methodDefinition contains a method body and Instruction property.

Himal
Date
Page

7@ Control section & program linking:

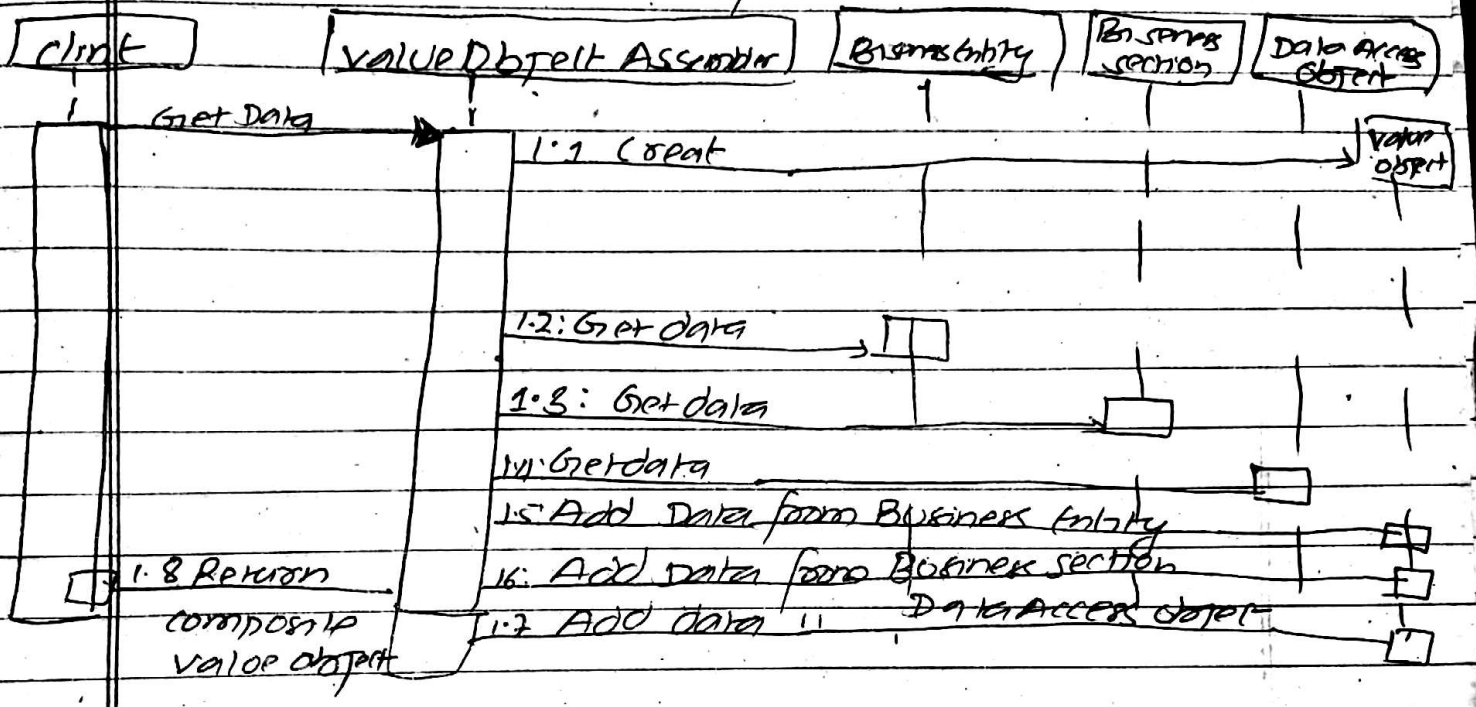
- Control section - can be loaded and relocated independently of the others.
- are most often used for subroutines or other logical subdivisions of a program.
- The programmer can assemble, load & each of these control section separately.
- Because of these, there should be some means for linking control section together.
- assembler directive: CSECT; sectname: CSECT
- Separate counter location for each control section

Program Linking

- consist of control section
- a list of items, LISTA ENDA, LISTB - ENDB, LIST C - ENDC
- each program contains exactly the same set, of references to these external symbols
- Instruction operands, (REF1, REF2, REF3) and value of data words are from REF4 to REF8

Himal
Date
Page

(b) Interaction diagram from Assembler:



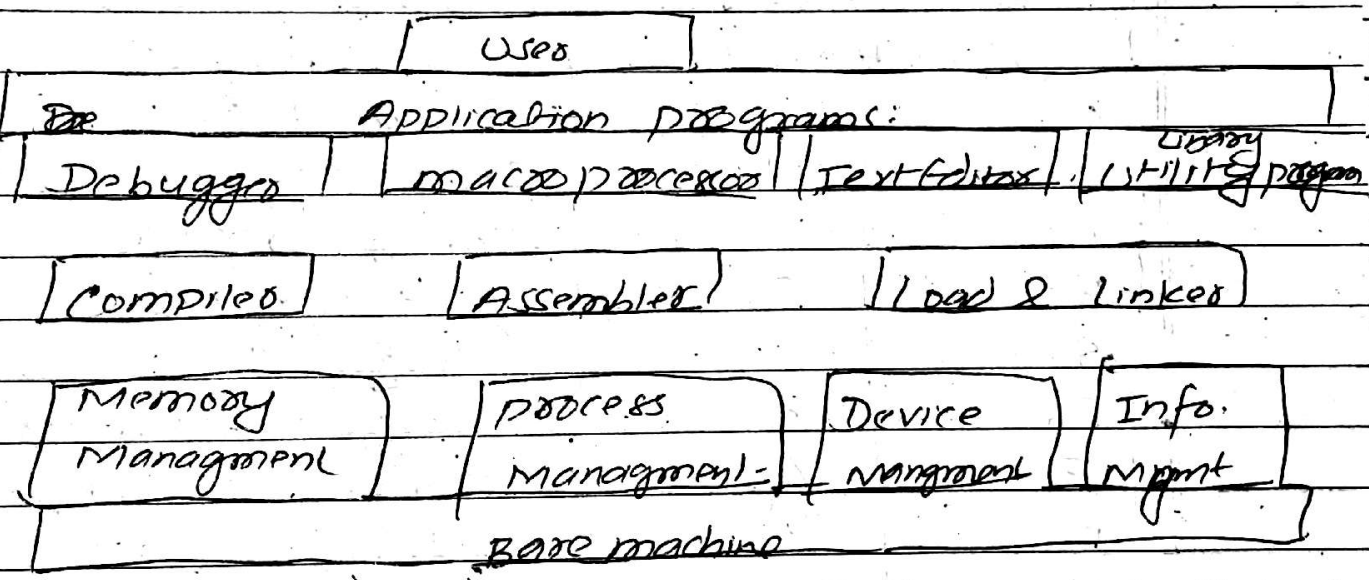
2014

Himal Date Page 1

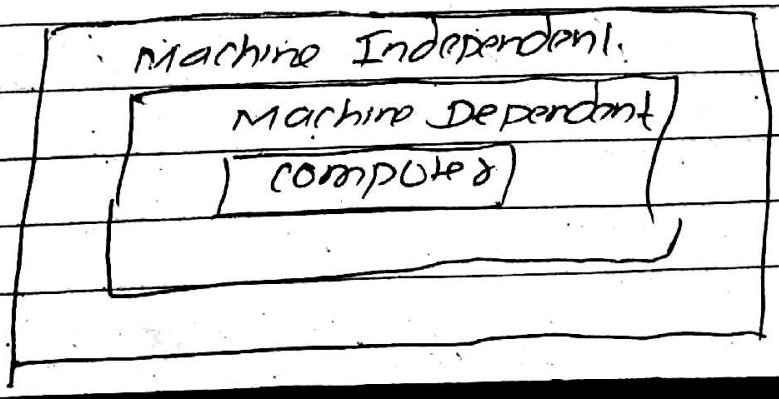
Q① Define system software. Berifly explain the machine dependence & independence part of system s/w.

Ans → System software consist of variety of program that support the operation of computer. It is a set of program to perform a variety of system function as file editing, resource managment, I/O managment and storage managment

System software concept




- Machine dependent - contains Instruction set, Instruction format, Addressing mode. Assembly language.
- Machine Independent General design logic / Strategy. Two pass assembler



Himal
Date
Page

Explain about SIC/XE architecture.

⇒ - Memory
1 word = 24 bits (3 8-bit byte)
Total (SIC/XE) = $2^{20}$ (1,048,576) byte

- Registers
- 10 x 20 bit registers

| MNEMONIC | Registers | purpose |
|---|---|---|
| A | 0 | Accumulator |
| X | 1 | Index register |
| L | 2 | Linkage register |
| B | 3 | Base register |
| S | 4 | General register |
| T | 5 | General Floating Point Accumulator |
| F | 6 | Floating point Accu |
| PC | 8 | PC |
| SW | 9 | Status word. |

Data format.
- I

Already Done (2013)

Himal
Date
Page

9. Write the function of loader. Write an algorithm for an absolute loader

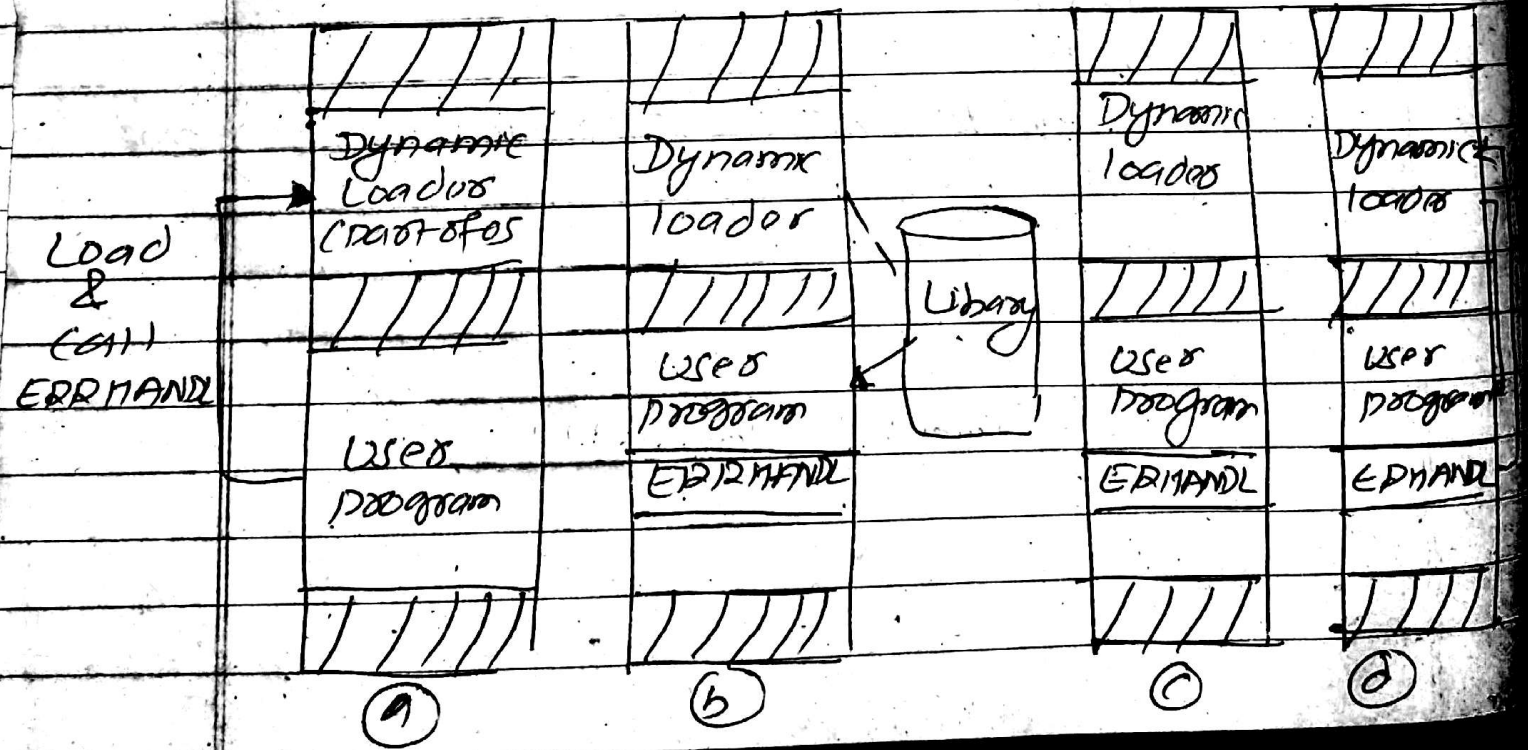
Loader is use to bring an object program into memory and starting its execution.

Algorithm:

```
begin
    read Header record
    verify program length and name
    read first text record
    while record type ≠ '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
```

(b) Explain the term load on call with suitable example?

⇒ Dynamic linking or load on call is often used to allow several executing programs to share one copy of a subroutine or library. Load on call postpones the linking function until the execution time. The programs makes load & call service request to OS. OS examines its internal tables to determine whether or not the routine is already loaded. Control is then passed from OS to the routine being called. When the called subroutine completes its processing OS then return control to the program that issued the request. If a subroutine is still in the memory a second call to it may not require another load operation.



* ERRMAND is the symbolic name of routine to be loaded.
* Dynamic Linking (dynamic loading, load on call).

Q. what do you understand by automatic library search.

⇒ Many linking loaders can automatically incorporate routines from a subprogram library into the program being loaded. A standard system library is used for automatic library search. The subroutine called by the program being loaded are automatically fetched from the library, linked with the main program and loaded. Automatically library call, at the end of pass 1, the symbols in ESTAB that remain undefined represent unresolved external references. The loader search the library.

Himal
Date
Page

Ques④ Explain briefly load and go assemblers.

⇒ load and go assembler generates their object code in memory for immediate execution. No object program is written out, no loader is needed. It is useful in a system with frequent program development and testing. The efficiency of the assembly process is an important consideration. programs are re-assembled nearly every time they are run. Efficiency of the assembler process is a important process.

6ⓐ & 6ⓑ

~~दोनों हैं~~ ।