SOLUTIONS TO SYSTEM PROGRAMMING QUESTIONS BPUT 2010

1) **Answer the following questions:** 2x10

(a) What is the work of *USING* pseudo-op?

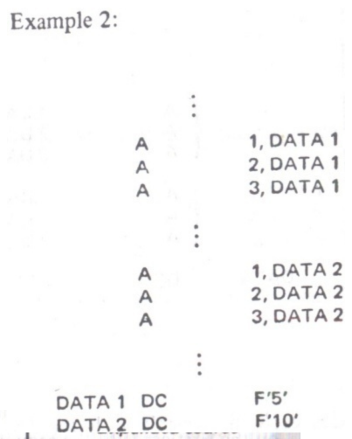
Ans USING pseudo code informs the assembler that R15 (specified register) is used as the base register and during execution will contain the address of the first instruction of the program.

(b) Describe the structure of BT (Base Table) used in pass 2 of assembler?

Ans The Base Table (BT), that indicates which registers are currently specified as base registers by USING pseudo-ops and what are the specified contents of these registers. It also specifies the registers that are unavailable by the DROP pseudo-ops.

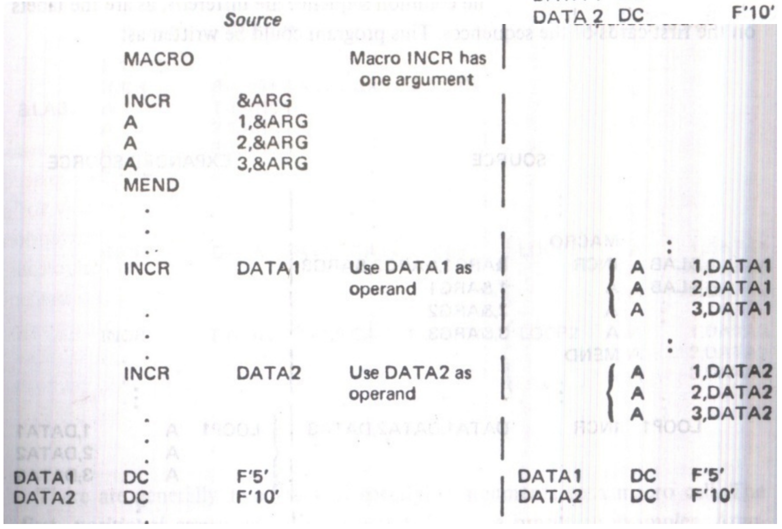
(c) What is dummy argument in macro? Explain with example.

Ans In this case the macro is defined with help of arguments called as *dummy* or *macro instruction arguments.* In the example it is clear that the first block of statements makes use of DATA1 and the second set makes use of the DATA2. So here in the macro definition along with the macro name the arguments are also defined and statement in the body makes use of these arguments in place of data. In the first macro call INCR DATA1, DATA1 is the argument and the second macro call DATA2 is the argument that is passed. Let us now consider a different case where the data in the block of statement are not the same and also the blocks make use of labels. We can define macro for this kind of code by passing sufficient number of arguments to handle the data in the set of statements. We also make of *label argument* that is used to define the label for a set of statement. The label arguments can be defined in two ways i.e. before the name of the macro (*positional argument*)or in the normal argument list (*dummy arguments)*. Whenever these kind of macros are called the corresponding values in the calls are passed the matching arguments in the definition.



Example 2:

```
        :
A               1, DATA 1
A               2, DATA 1
A               3, DATA 1
        :
A               1, DATA 2
A               2, DATA 2
A               3, DATA 2
        :
DATA 1  DC      F'5'
DATA 2  DC      F'10'
```



```
                    Source
MACRO                          Macro INCR has
                               one argument
INCR        &ARG
A           1,&ARG
A           2,&ARG
A           3,&ARG
MEND
  .
  .
INCR        DATA1              Use DATA1 as      :
                               operand         { A      1,DATA1
                                               { A      2,DATA1
  .                                            { A      3,DATA1
  .                                              :
INCR        DATA2              Use DATA2 as    { A      1,DATA2
                               operand         { A      2,DATA2
  .                                            { A      3,DATA2
  .
DATA1   DC      F'5'                           DATA1    DC    F'5'
DATA2   DC      F'10'                          DATA2    DC    F'10'
```

(d) Differentiate between relocation and loading?

Ans Relocation refers to adjust all address dependent locations, such as address constants, to correspond to the allocates space where as loading refers to physically place the machine instructions and data into memory.

(e) Name the cards generated in a direct –linking loader.

Ans 1. External Symbol Dictionary cards (ESD)
2. Instructions and data cards, called "text" of program (TXT)
3. Relocation and Linkage Directory cards (RLD)
4. End card (END)

(f) Define a formal system mathematically.

Ans A formal system is an uninterrupted calculus or logistic system. It consists of an alphabet, a set of words called axioms, and a finite set of relations called rules of inference. Examples of formal systems are: set theory, Boolean algebra, propositional and predicate calculus, Post systems, Euclid's plane geometry, Backus Normal Form, and Peano arithmetic.

(g) Name the analysis phases of a compiler.

Ans The analysis phases of compiler are as follows:

i) Lexical analysis
ii) Syntax analysis
iii) Semantic analysis
iv) Intermediate code generation
v) Code optimization
vi) Machine code generation.

(h) For the C statement: c = a + b; find the codes generated in code generation phases of a compiler.

Ans The code generated for the C statement c = a +b is as follows:

i) L 1,A	i) //Load A in the register 1
ii) A 1,B	ii) //Add B to the contents of register 1
iii) ST 1,C	iii) //Store the contents of register 1 in C

(i) Define BNF.

Ans BNF is a notation for writing grammars that is commonly used to specify the syntax of programming languages. In BNF non-terminals are written as names enclosed in corner-bracket'<>'. The sign -> is written ::= (read "is replaced by"). Alternatively ways of rewriting a given non-terminal are separated by a vertical bar, |, (read "or"). Example

i) <Σ>::=<A><B> *A followed by B"*

ii) //Read as *"the variable A is replaced*

ii) <A>::=a<A>|a *by a followed by A or a"*

iii) //Read as *"the variable Bis replaced*

iii) <B>::=<B>b|b *by B followed by b or b"*

i) //Read as *"the sign Σ is replaced by*

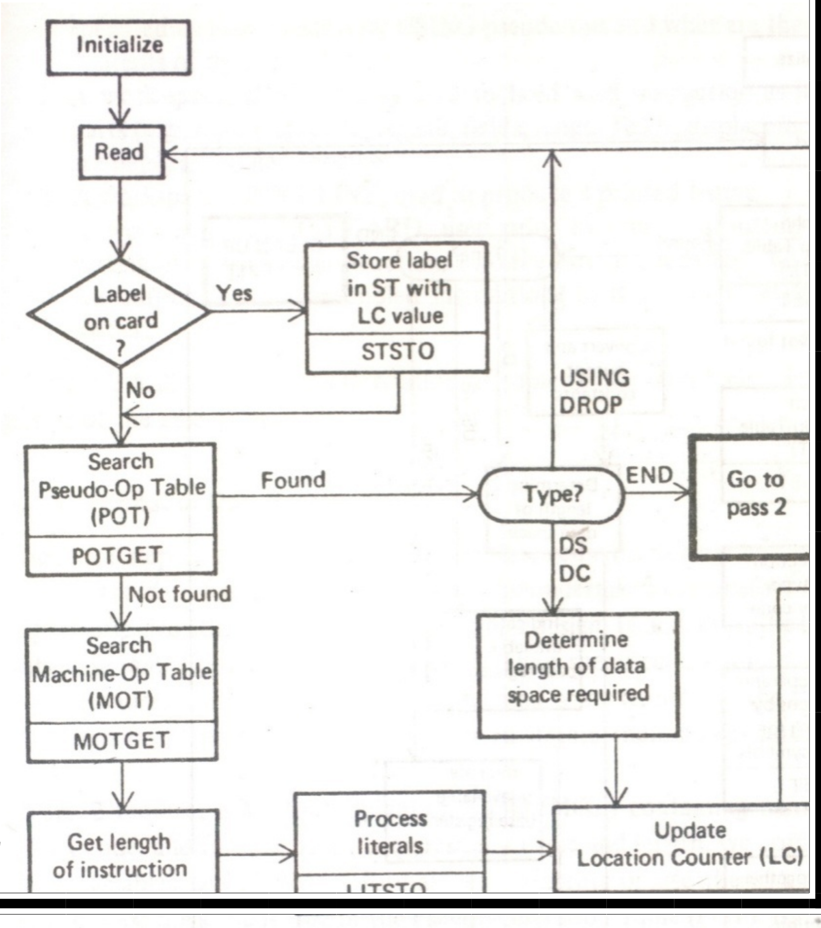
(j) What is boothstrap loader? Why it is used?

Ans

2) Describe the overall design of a pass1 of an assembler with the description of the data structure used in it. 10

Ans *Pass 1:* databases:

1. Input source program.
2. A Location Counter (LC), used to keep track of each instruction's location.
3. A table, the Machine-Operation Table (MOT), that indicates the symbolic mnemonic for each instruction and its length (two, four, or six bytes).
4. A table, the Pseudo-Operation Table (POT), that indicates the symbolic mnemonic and action to be taken for each pseudo-op in pass 1.



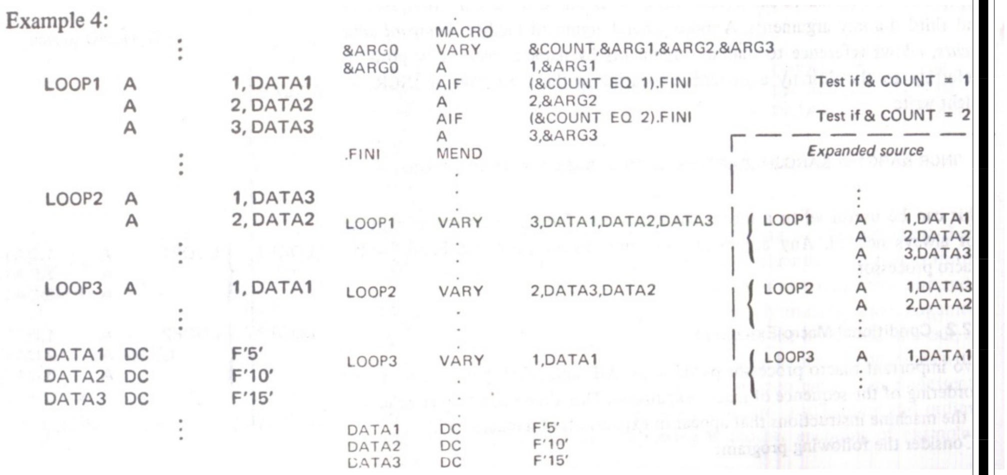
Initialize
Read
Label on card ?
Yes
Store label in ST with LC value
STSTO
No
USING DROP
Search Pseudo-Op Table (POT)
POTGET
Found
Type?
END
Go to pass 2
Not found
DS DC
Search Machine-Op Table (MOT)
MOTGET
Determine length of data space required
Get length of instruction
Process literals
LITSTO
Update Location Counter (LC)

FIGURE 3.3 Pass 1 overview: define symbols

5. A table, the Symbol Table (ST), that is used to store each literal and its corresponding value.
6. A table, the Literal Table (LT), that is used to store each literal encountered and its corresponding assigned location.
7. A copy of the input to be used later by pass 2. This may be stored in a secondary storage device, such as magnetic tape, disk, or drum, or the original source deck may be read by the assembler a second time for pass 2.

3) **(a) Describe the conditional macro expansion facilities with suitable examples.** 5

<u>Ans</u> The AIF and AGO pseudo codes provides conditional operations within a macro. This will be clear from example 4 where we have set of statements having variable number of statements. In this case we define a macro that makes use of AIF (conditional statement). Even in some cases AGO (unconditional statement) is used. In this macro we make use of one additional argument COUNT that keeps a count of statement used in the block. The AIF with help of COUNT finds the number of statements that is to be expanded. Here .FINI is used as a *macro label*. Here the label is used to mark the end of the macro. With the help of conditional macros we can produce program equivalent to that of high level languages.


Example 4:

```
        :
LOOP1   A       1, DATA1
        A       2, DATA2
        A       3, DATA3
        :
LOOP2   A       1, DATA3
        A       2, DATA2
        :
LOOP3   A       1, DATA1
        :
DATA1   DC      F'5'
DATA2   DC      F'10'
DATA3   DC      F'15'
        :
```

```
        MACRO
&ARG0   VARY    &COUNT,&ARG1,&ARG2,&ARG3
&ARG0   A       1,&ARG1
        AIF     (&COUNT EQ 1).FINI          Test if & COUNT = 1
        A       2,&ARG2
        AIF     (&COUNT EQ 2).FINI          Test if & COUNT = 2
        A       3,&ARG3
.FINI   MEND
        :
LOOP1   VARY    3,DATA1,DATA2,DATA3
        :
LOOP2   VARY    2,DATA3,DATA2
        :
LOOP3   VARY    1,DATA1
        :
DATA1   DC      F'5'
DATA2   DC      F'10'
DATA3   DC      F'15'
```

Expanded source

```
        :
LOOP1   A       1,DATA1
        A       2,DATA2
        A       3,DATA3
        :
LOOP2   A       1,DATA3
        A       2,DATA2
        :
LOOP3   A       1,DATA1
        :
```

(b) **Write an assembly language program for executing the C instruction;** 5   d = a + b – c;   assume that address of a = 100, b =200, c = 300, d = 400.

<u>Ans</u> The assembly code for d = a +b –c is as follows:

```
TEST    START                   Name of the program
BEGIN   BALR    15,0            Set Reg15 to addr. of next inst.
        USING   BEGIN+2,15      Pseudo instr. to assembler
        L       2,ANUM          Load a in Reg2
        A       2,BNUM          Add b to content of the Reg2
        S       2,CNUM          Subtract c to content of Reg2
        ST      2,DNUM          Store the content of Reg2 to d
ANUM    DC      a               Constant a
BNUM    DC      b               Constant b
CNUM    DC      c               Constant c
DNUM    DC      d               Constant d
        END                     End of the program.
```

4) **Describe the design of a relocation loader. What is the advantage of a relocation loader over absolute loader? Discuss.** 10

<u>Ans</u> The design of a relocation loader is as follows:

The advantages of relocation loader over absolute loader are as follows:

i) To avoid possible reassembling of all subroutines when a single subroutine is changed, and to perform the tasks of allocation and linking for the programmer, the general class of *relocating loaders* was introduced.

ii)In addition, there is information (relocation information) as to locations in this program that need to be changed if it is to be loaded in an arbitrary place in core, i.e.. the locations which are dependent on the core allocation.

5) Describe the various loading schemes with suitable example from each.
10

Ans The various loading schemes are as follows:

i) Compile and go	iii) Absolute loader	v) Relocating loader
ii) General loading	iv) Subroutine linkage	vi) Direct linking loader

6) Describe the phases of a compiler with neat sketches with their functionality and suitable example from each.
10

Ans The analysis phases of compiler are as follows:

i) Lexical analysis	iii) Semantic analysis	v) Code optimization
ii) Syntax analysis	iv) Intermediate code generation	vi) Machine code generation.

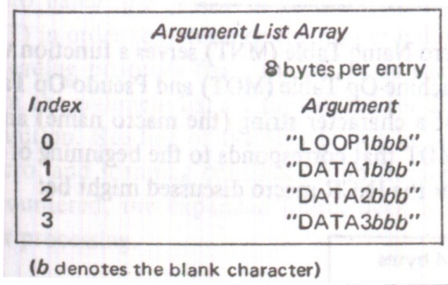
7) (a) What are the various code optimization techniques, example with suitable examples. 5

Ans The code optimization techniques are as follows:

Machine dependent   Machine independent

(b) Describe the databases used in pass 1 of macro processing.
5

Ans *ARGUMENT LIST ARRAY:* used during both pass1 and pass2 but some the functions are just the reverse. During pass 1 the ALA stores the arguments in the macro definition with positional indicator when the definition is stored in MDT i.e. the ith argument is stored in the ALA as #i. This arrangement is in according to the order of argument in which they appear in the MDT. Later on in the pass2, when

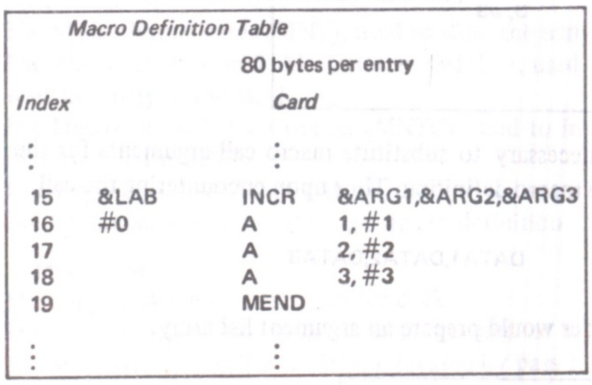


Argument List Array

Index	Argument (8 bytes per entry)
0	"LOOP1*bbb*"
1	"DATA1*bbb*"
2	"DATA2*bbb*"
3	"DATA3*bbb*"

(*b* denotes the blank character)

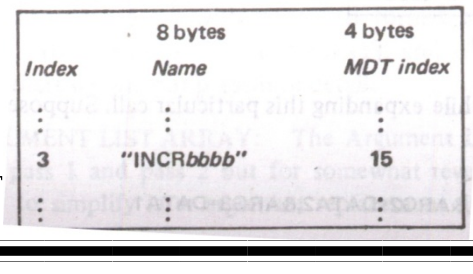
there is macro expansion the ALA fills the arguments of the corresponding index with its appropriate argument in the call. This will clear from the example. Here when LOOP INCR DATA1, DATA2, DATA3 is executed the ALA fills the argument fields of the corresponding index #0, #1, #2, #3 with LOOP, DATA1, DATA2 and DATA3.



Macro Definition Table

Index		Card (80 bytes per entry)	
⋮		⋮	
15	&LAB	INCR	&ARG1,&ARG2,&ARG3
16	#0	A	1, #1
17		A	2, #2
18		A	3, #3
19		MEND	
⋮		⋮	

MACRO DEFINTION TABLE: It is table of text lines. It consists of two fields, index that keep track of line numbers of the macro definition and the card that is 80 bytes of size and is responsible for storing the macro definition. Everything except the pseudo code MACRO is



	8 bytes	4 bytes
Index	*Name*	*MDT index*
⋮	⋮	⋮
3	"INCR*bbbb*"	15
⋮	⋮	⋮

inserted into the MDT. MEND pseudo code marks the end of the macro definition.

MACRO NAME TABLE: It is similar to out MOT and POT in assembler. It has got three field, Index field that keep track of various macro that are defined, the Name field that keep track of names of the macros and the MDT index is a pointer to the entry in MDT.

8) Write short notes on any two: 5x2

(i) General machine structure

Ans

All the conventional modern computers are based upon the concept of *stored program computer*, the model that was proposed by John von Neumann.
The components of a general machine are as follows:
Instruction interpreter: group of electronic circuits that performs the intent of instruction of fetched from memory.
Location counter: LC otherwise called as *program counter PC* or *instruction counter IC*, is a hardware memory device which denotes the location of the current instruction being executed.
Instruction register: A copy of the content of the LC is stored in IR.
Working register: are the memory devices that serve as "scratch pad" for the instruction interpreter.
General register: are used by programmers as storage locations and for special functions.

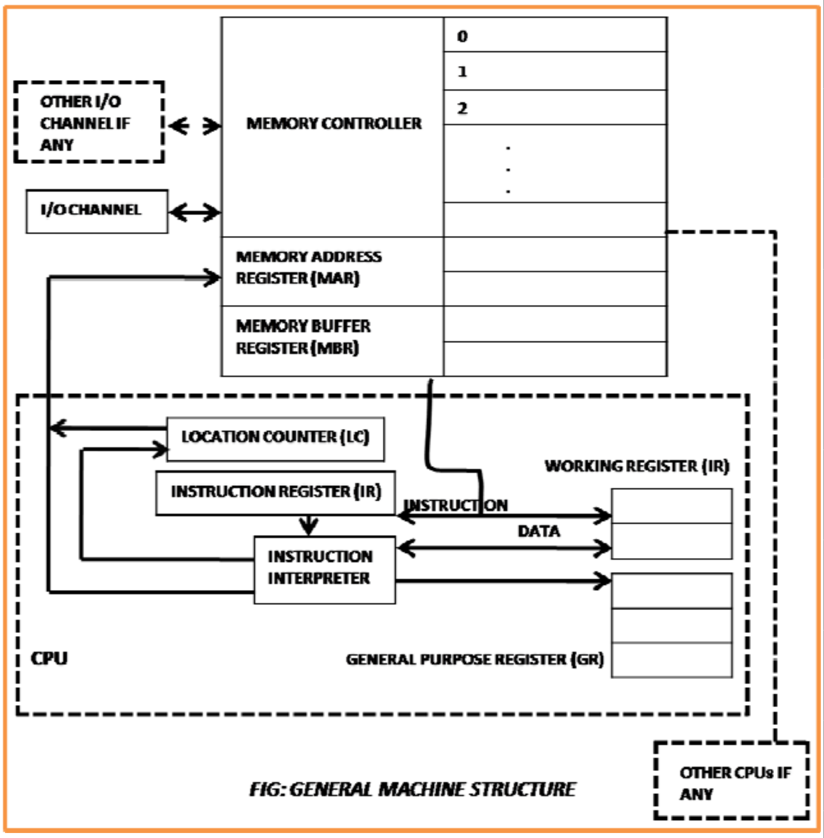
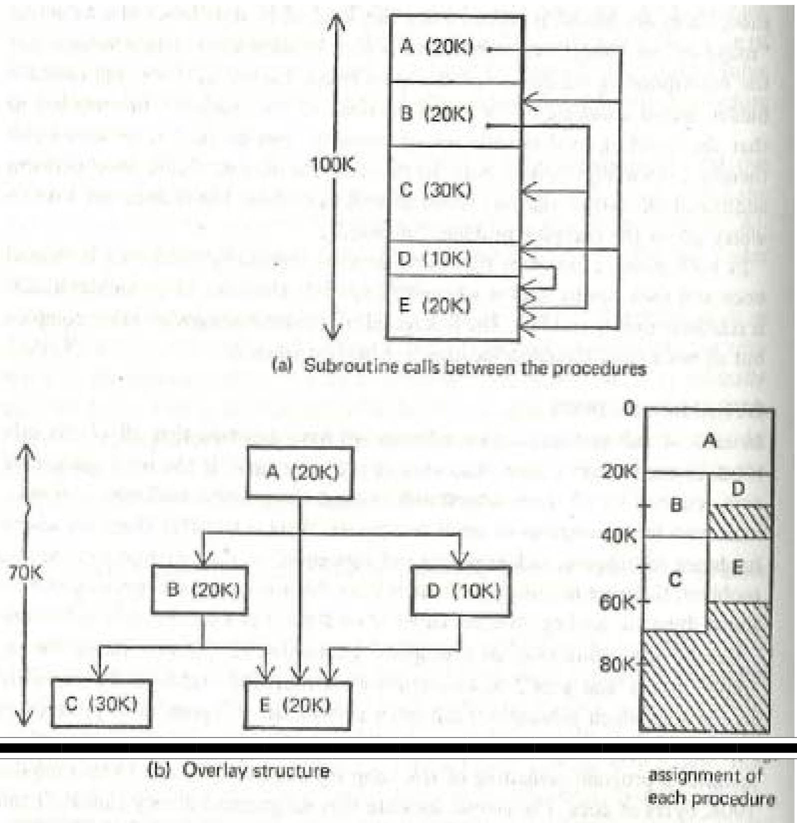


FIG: GENERAL MACHINE STRUCTURE

(j) Overlays

Ans Usually the subroutines of a program are needed at different times: for example, pass) and pass 2 of an assembler are mutually exclusive. By explicitly recognizing which subroutines call other subroutines it is possible to produce an *overlay structure* that identifies mutually exclusive subroutines. Figure illustrates a program consisting of *five* subprograms (A,B,C,D and E) that require 100K bytes of core. The arrows indicate that subprogram A only calls B, D and



E; subprogram B only calls C and E; subprogram D only calls E; and subprograms C and E do not call any other routines. Figure 2 highlights the interdependencies between the procedures. Note that procedures Band D are never in use at the same time; neither are C and E. If we load only those procedures that are actually to be used at any particular time, the amount of core needed is equal to the longest path of the overlay structure. This happens to be 70K for the example in Figure 2 procedures A, B, and C. Figure 3 illustrates a storage assignment for each procedure consistent with the overlay structure. In order for the overlay structure to work it is necessary for the module loader to load the various procedures as they are needed. We will not go into their specific details, but there are many binders capable of processing and allocating an overlay structure. The portion of the loader that actually intercepts the "call" and loads the necessary procedure is called the *overlay supervisor* or simply the *flipper.* The overall scheme is called *dynamic loading* or *load-on-call*(LOCAL).

(k) Storage classes.

Ans Today's computer user is starting to demand more flexible types of storage than those provided by FORTRAN, ALGOL, and COBOL. In particular, he would like to be able to allocate and free his storage as needed. While many people do not feel that PL/I is a good language, much of their criticism is levelled at it's syntactical ambiguities. In the area of storage allocation, PL/I's powerful facilities are generally endorsed since they appear to fulfil the user's expanding requirements. We will discuss different types of storage allocation and their use. The next section presents their implementation. Because of their generally, we have chosen the types of storage PL/I offers the user; ALGOL 68 and many of the new languages provide somewhat similar facilities. We will discuss the following types of storage:

I. Internal static	3. Automatic	5. External controlled
2. External static	4. Internal controlled	6. Based