CD Unit-IV - Compiler design

Compiler Design (Jawaharlal Nehru Technological University, Hyderabad)

UNIT - IV

4 Run-Time Environment and Code Generation

Part I : Run-Time Environment

4.1 : Storage Organization

Q.1 Explain the run time storage organization of a program. ☞ [JNTU : Part B, Jan.-12, Marks 8]

Ans. : • The compiler demands for a block of memory to operating system. The compiler utilizes this block of memory for running (executing) the compiled program. This block of memory is called **run time storage.**

- The run time storage is subdivided to hold code and data such as :
  - i) The generated target code
  - ii) Data objects
  - iii) Information which keeps track of procedure activations.
- The size of generated code is fixed. Hence the target code occupies the statically determined area of the memory. Compiler places the target code at the lower end of the memory.
- The amount of memory required by the data objects is known at the compiled time and hence data objects also can be placed at the statically determined area of the memory. Compiler prefers to place the data objects in the statically determined area because these data objects then can be compiled into target code. For example in FORTRAN all the data objects are allocated statically. Hence the static data area is on the top of code area. The subdivision of run time memory as shown by following Fig. Q.1.1.
- The counterpart of control stack is used to manage the active procedures. Managing of active procedures means that when a call occurs then execution of activation is interrupted and information about status of the stack is saved on the stack. When the control returns from the call this suspended activation is resumed after storing the values of relevant registers. Also the program counter is set to the point immediately after the call. This information is stored in the stack area of run time storage. Some data objects which are contained in this activation can be allocated on the stack along with the relevant information.

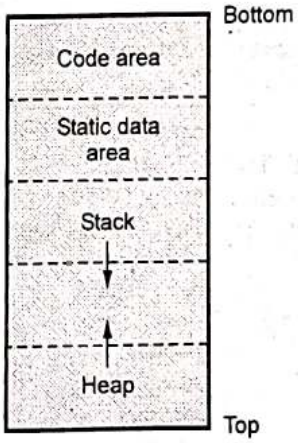


Fig. Q.1.1 Run time storage organization

- The heap area is the area of run time storage in which the other information is stored. For example memory for some data items is allocated under the program control. Memory required for these data items is obtained from this heap area. Memory for some activation is also allocated from heap area.
- The size of stack and heap is not fixed it may grow or shrink interchangeably during the program execution.
- Pascal and C need the run time stack.

Q.2 Explain various storage allocation strategies with its merits and demerits. ☞ [JNTU : Part B, Nov.-16, Marks 10]

OR Compare different storage allocation strategies. ☞ [JNTU : Part B, Marks 5]

Ans. :

Sr. No.	Static allocation	Stack allocation	Heap allocation
1.	Static allocation is done for all data objects at compile time.	In stack allocation, stack is used to manage runtime storage.	In heap allocation, heap is used to manage dynamic memory allocation.
2.	Data structures can not be created dynamically because in static allocation compiler can determine the amount of storage required by each data object.	Data structures and data objects can be created dynamically.	Data structures and data objects can be created dynamically.
3.	**Memory allocation :** The names of data objects are bound to storage at compile time.	**Memory allocation :** Using Last In First Out (LIFO) activation records and data objects are pushed onto the stack. The memory addressing can be done using index and registers.	**Memory allocation :** A contiguous block of memory from heap is allocated for activation record or data object. A linked list is maintained for free blocks.
4.	**Merits and limitations :** This allocation strategy is simple to implement but supports static allocation only. Similarly recursive procedures are not supported by static allocation strategy.	**Merits and limitations :** It supports dynamic memory allocation but it is slower than static allocation strategy. Supports recursive procedures but references to non local variables after activation record can not be retained.	**Merits and limitations :** Efficient memory management is done using linked list. The deallocated space can be reused. But since memory block is allocated using best fit, holes may get introduced in the memory.

Q.3 Explain in brief about heap storage allocation Strategy. ☞ [JNTU : Part B, Mar.-16, Marks 5]

Ans. :

a. If the values of non local variables must be retained even after the activation record then such a retaining is not possible by stack allocation. This limitation of stack allocation is because of its Last In First Out nature. For retaining of such local variables heap allocation strategy is used.

b. The heap allocation allocates the continuous block of memory when required for storage of activation records or other data object. This allocated memory can be deallocated when activation ends. This deallocated (free) space can be further reused by heap manager.

c. The efficient heap management can be done by

i) Creating a linked list for the free blocks and when any memory is deallocated that block of memory is appended in the linked list.

ii) Allocate the most suitable block of memory from the linked list. i.e. use best fit technique for allocation of block.

Q.4 What are the advantages of heap storage allocation ? ☞ [JNTU : Part A, Dec.-17, Marks 2]

Ans. : The advantages of heap storage allocation are as follows -

1) There is no limit on memory size. One can use as much memory as required.

2) Any object created in the heap space has global access and can be referenced from anywhere of the application.

3) It is a flexible storage allocation strategy.

4) Automatic garbage management technique can be used for heap storage allocation.

Q.5 Explain in brief about stack storage allocation strategy. ☞ [JNTU : Part B, Nov.-17, Marks 5]

Ans. : Refer Q.2.

4.2 : Stack Allocation of Space

Q.6 Define activation record. Explain in brief about the fields in activation record.

☞ [JNTU : Part A, Mar.-17, Marks 2
Part B, Nov 17, Marks 5]

Ans. : • The activation record is a block of memory used for managing information needed by a single execution of a procedure.

• FORTRAN uses the static data area to store the activation record where as in PASCAL and C the activation record is situated in stack area. The contents of activation record are as shown in the following Fig. Q.6.1.

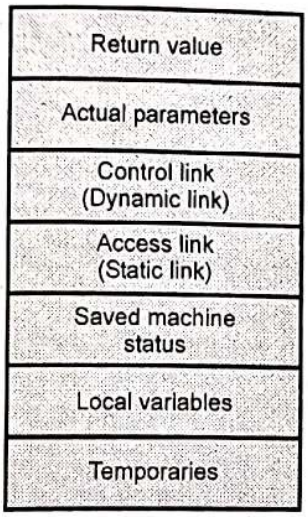


Fig. Q.6.1 Model of activation record

• Various fields of activation record are as follows -

1. **Temporary values** - The temporary variables are needed during the evaluation of expressions. Such variables are stored in the temporary field of activation record.
2. **Local variables** - The local data is a data that is local to the execution of procedure is stored in this field of activation record.
3. **Saved machine registers** - This field holds the information regarding the status of machine just before the procedure is called. This field contains the machine registers and program counter.
4. **Control link** - This field is optional. It points to the activation record of the calling procedure. This link is also called dynamic link.
5. **Access link** - This field is also optional. It refers to the non local data in other activation record. This field is also called static link field.
6. **Actual parameters** - This field holds the information about the actual parameters. These actual parameters are passed to the called procedure.
7. **Return values** - This field is used to store the result of a function call.

• The size of each field of activation record is determined at the time when a procedure is called.

Q.7 What is activation tree ?

☞ **[JNTU : Part A, Marks 2]**

Ans. : Activation tree is a data structure used for specifying the flow of control among the procedures. In an activation tree -

i) Each node shows an activation of procedure.

ii) For showing activation of main program root is used.

iii) If lifetime of 'x' occurs before the lifetime of 'y' then x becomes left child of y.

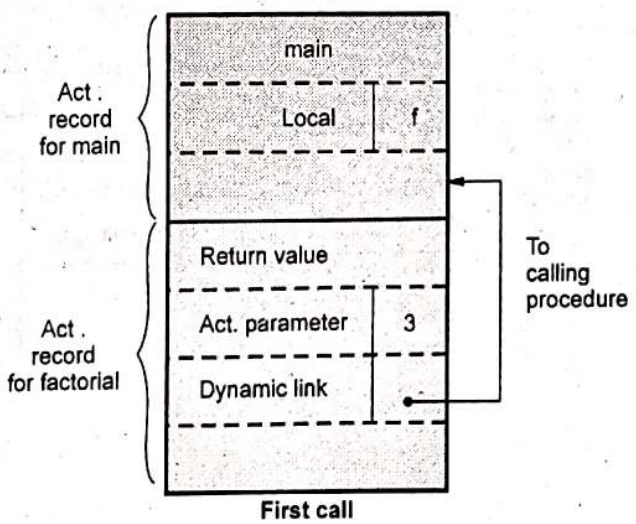
iv) If control flows from x to y then 'x' is parent node of 'y'.

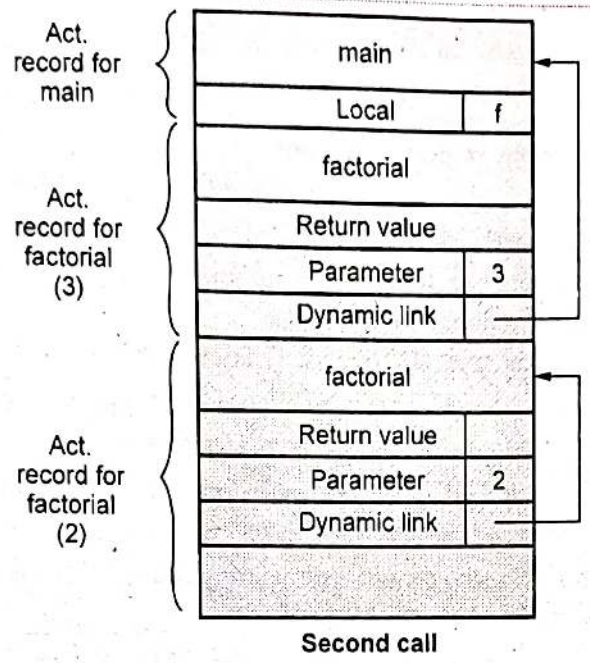
Q.8 By taking the example of factorial program explain how activation record will look like for every recursive call in case of factorial (3).

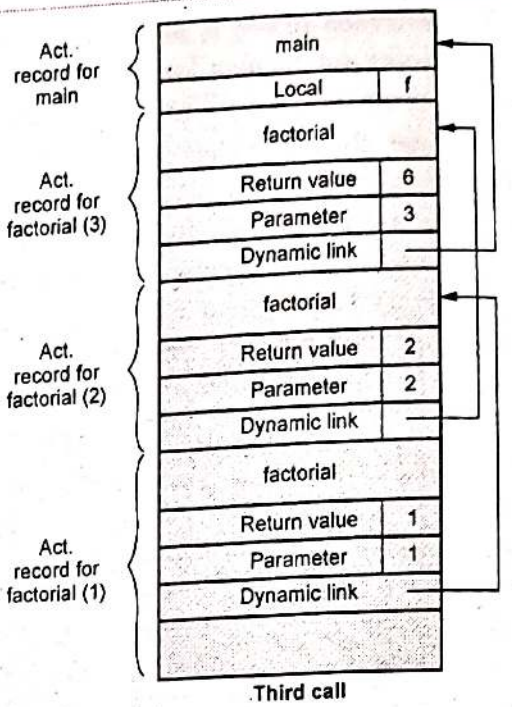
☞ **[Part B, Marks 5]**

Ans. :

```
main()
{
   int f;
   f = factorial (3);
}
int factorial (int n)
{
   if (n==1)
         return 1;
   use
         return (n*factorial (n–1));
}
```





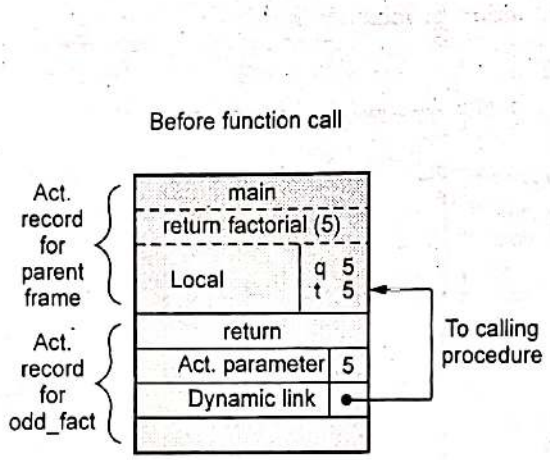


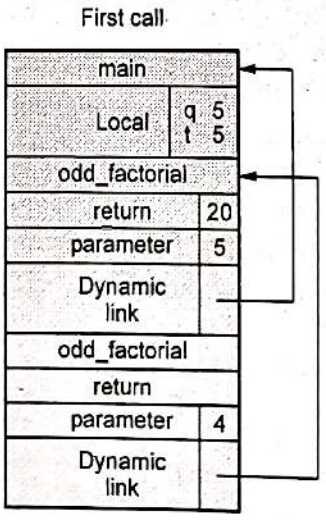
Q.9 Explain what information are stored in stack during function calls. Consider the following code :

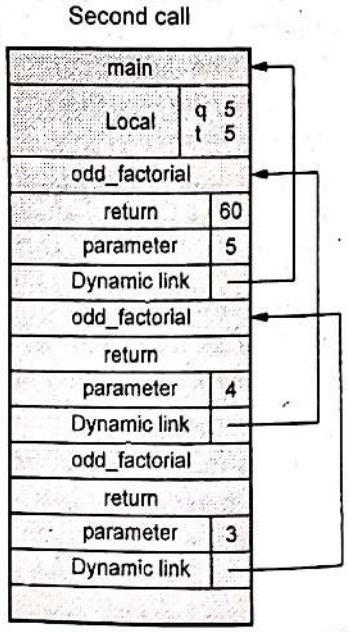
```
Line No's
1. int odd_factorial (int q) {
2. int factorial (int n) {
3. if (n == 1) return q;
4. int t = factorial (n-1);
5. return n * t;
6. }
7. if (q % 2 == 0)
8. return q;
9. return factorial (t);
10. }
```

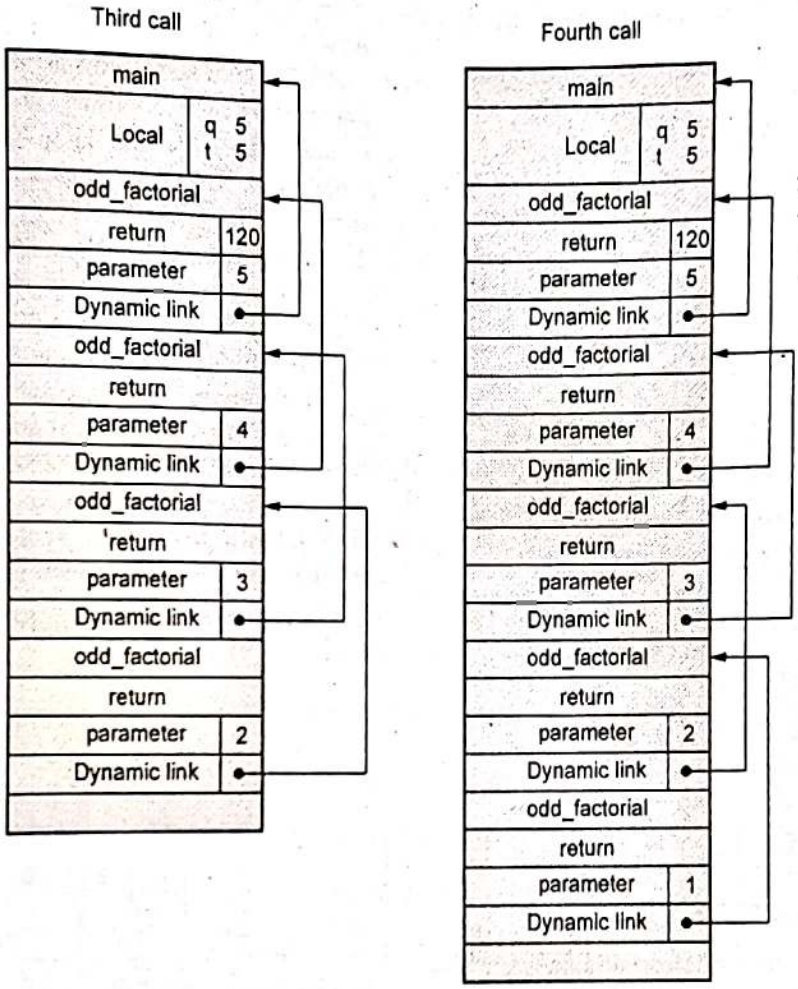
Consider calling the function odd_factorial (5). Show the state of the stack, including parent frame pointers, during the execution of the return statement in line 3. ☞ **[JNTU : Part B, May-12, Marks 15]**

Ans. :









Finally 120 will be returned to main.

4.3 : Access to Non Local Data on the Stack

Q.10 What are local and non local variables ?

Ans. : **Local variable** : A variable declared in block B is called local variable of block B.

Non Local variable : A variable of enclosing block which is accessible within block B is called non-local variable of block B. To understand the local and nonlocal variables let us see one example.

Q.11 Obtain the static scope of the declarations made in the following piece of code.

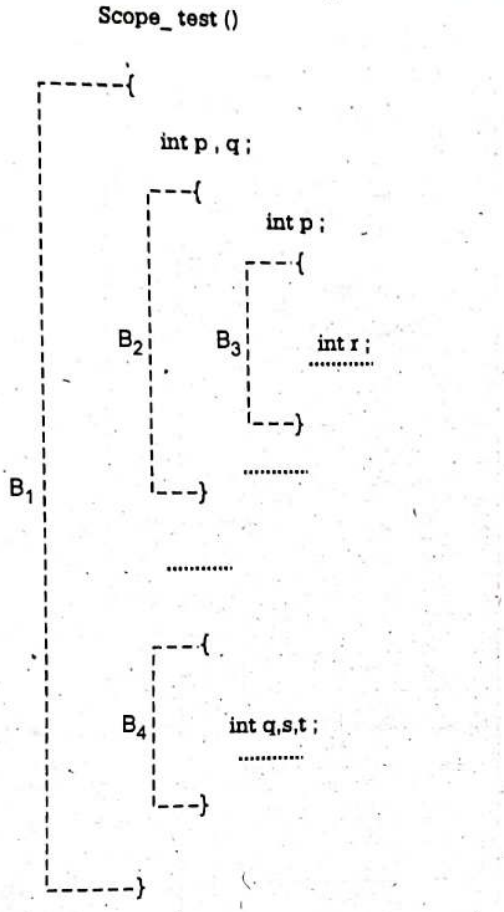


Fig. Q.11.1

Ans. : To denote the scope of the variable we will use name of the varibale followed by the block name.

Block	Local variable	Non Local variable
B_1	pB_1, qB_1	-
B_2	pB_2	pB_1, qB_1
B_3	rB_3	pB_2, qB_1
B_4	qB_4, sB_4, tB_4	pB_1, qB_1

Q.12 What is static and dynamic scope rule ?

Ans. : 1. Static scope rule : The static scope rule is also called as lexical scope. In this type the scope is determined by examining the program text. PASCAL, C and ADA are the languages that use the static scope rule. These languages are also called as block structured languages.

2. **Dynamic scope rule :** For non block structured languages this dynamic scope allocation rules are used.

   The dynamic scope rule determines the scope of declaration of the names at run time by considering the current activation.

   LISP and SNOBOL are the languages which use dynamic scope rule.

Q.13 Obtain the static scope of the declarations made in the following piece of code. ☞**[JNTU : Part B]**

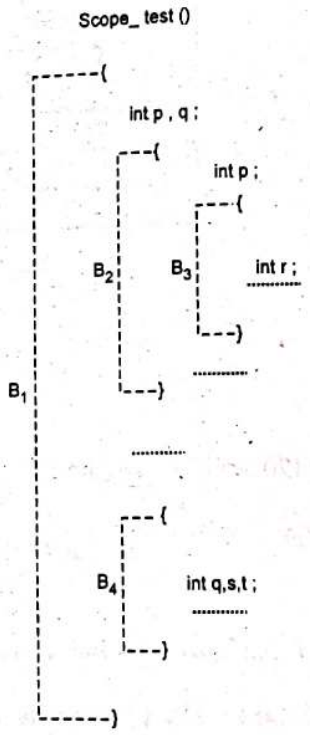


Fig. Q.13.1

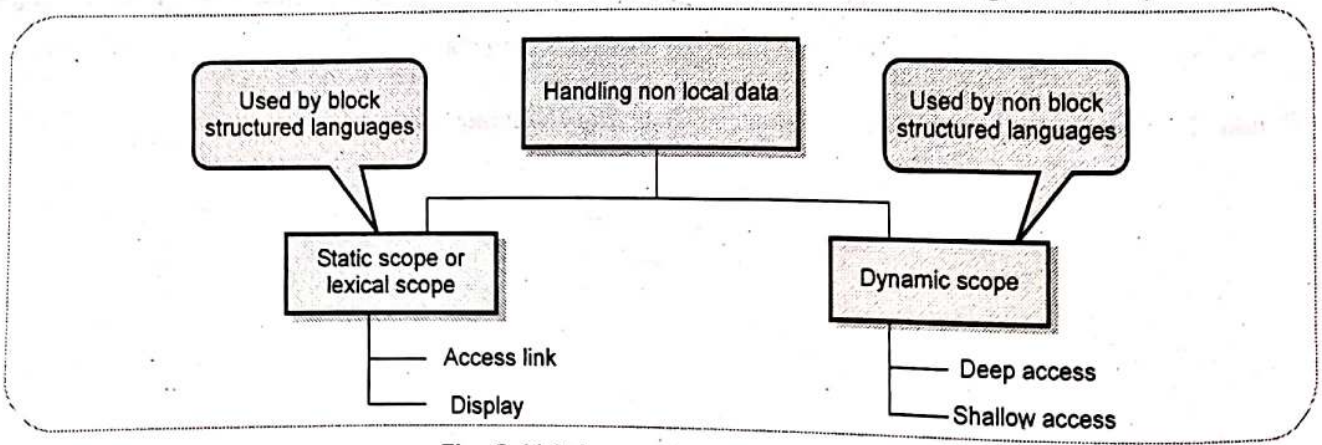
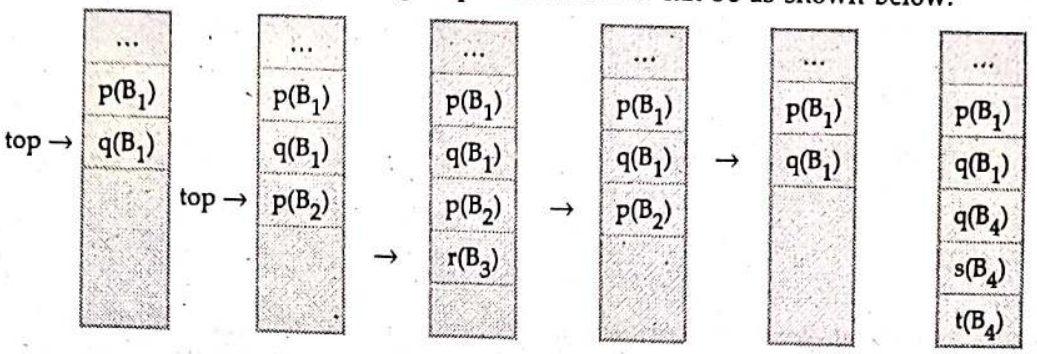


Fig. Q.11.2 Access to non local data

Ans. : The storage can be allocated for a complete procedure body at one time.

The storage for the names corresponding to particular block can be as shown below.



Q.14 Explain the behaviour of stack, used for allocation of storage, for the execution of each statement of the block structured program :

☞ [JNTU : Part B]

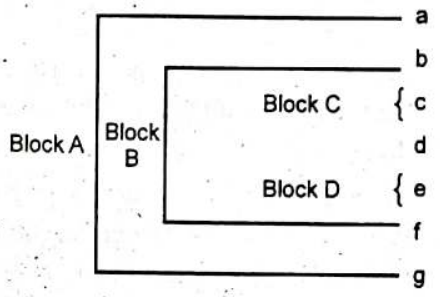


Fig. Q.14.1

Ans. : The block structure storage allocation which can be done using stack is as given below.

Thus block structure storage allocation can be done by stack.

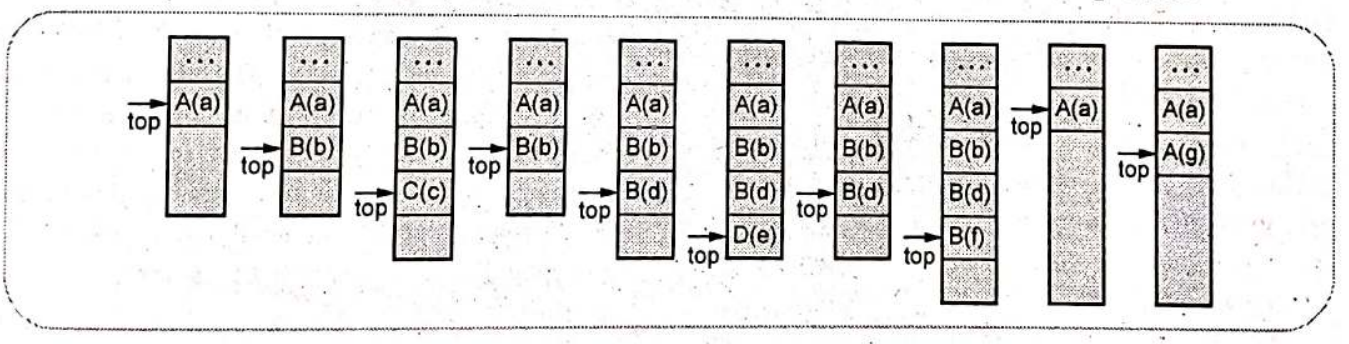


Fig. Q.14.1 (a)

4.4 : Heap Management

Q.15 What is heap management ?

☞ [JNTU : Part A, Marks 2]

Ans. : Heap is a portion of memory that can remain allocated for indefinite time. The memory created using dynamic allocation technique makes use of heap memory. For example in C++ or Java we can make use.

Q.16 What are the tasks of memory manager ?

☞ [JNTU : Part A, Marks 3]

Ans. : • The task of memory manager is to keep track of all the free space that can be available in the heap. Following are the two important functions of memory manager -

1. **Allocation :** When a program requests for allocation of memory for some object or variable then the memory manager allocates a chunk of memory to the object/variable. Sometimes the memory manager makes use of the free space available in the heap memory in order to satisfy the request. If still the required free space is not available then it increases the size of heap storage by getting some bytes of memory from virtual memory of operating system.

2. **Deallocation :** The memory manager returns the deallocated memory to the pool of free storage maintained in the heap memory. This free space is used by the memory manager to satisfy the request for memory allocation.

Q.17 What are desired properties of memory manager ? ☞ **[JNTU : Part A, Marks 3]**

Ans. : The desired properties of memory manager are :

1. **Space efficiency :** The memory manager must minimize the requirement of total heap space required by the program. For that matter, the larger programs are allowed to run in fixed virtual address space. The reduction in fragmentation also helps in achieving the space efficiency.
2. **Program efficiency :** By using the heap memory appropriately the program efficiency can be achieved. The object that are required more frequently must be placed in the accessible region, this will help to run the program faster. The principle of locality is used to achieve the program efficiency.
3. **Low overhead :** Memory allocation and deallocation are the most frequently performed operations in the program. If these operations are executed efficiently then it reduces the overhead in program execution.

Q.18 Explain memory hierarchy in detail. ☞ **[JNTU : Part B, Marks 5]**

Ans. : • Compiler optimization is very much dependent upon the memory management technique.

- The efficiency of program is determined by - 1) Number of instructions getting executed and 2) The amount time required by each instruction for execution. The time taken by the instruction to execute greatly depends upon how much time it takes to access different parts of the memory.
- The significant difference in memory access time is due to **limitations of hardware technologies.** Quite often there are small storage with fast access and large storage with slow access. To make use of the efficient memory access all the modern computers arrange in memory in **hierarchy.** Fig. Q.18.1 represents the memory hierarchy.

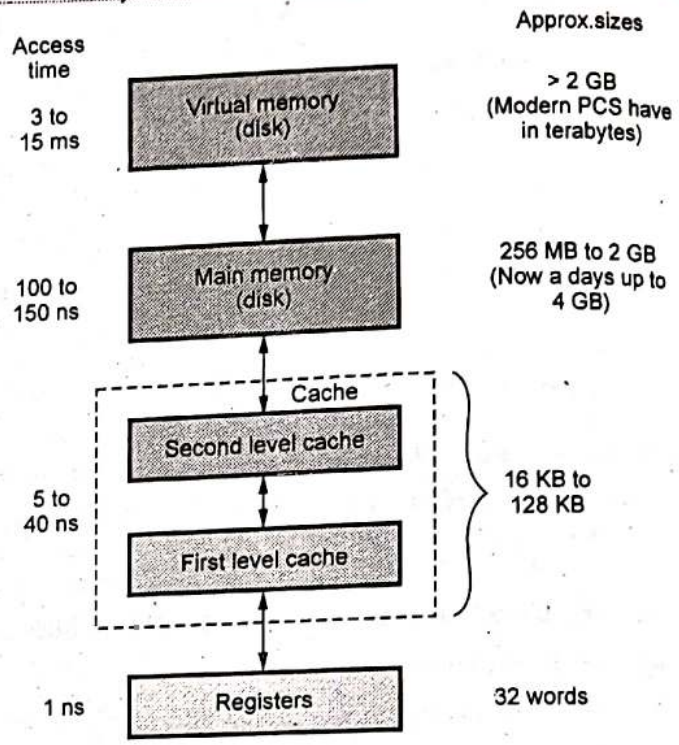


Fig. Q.18.1 Memory hierarchy

- The processors has several registers which can be controlled by the software. Then the levels of cache memory come. It is there in kilobytes to several megabytes.The next level has the physical main memory which is made up of hundreds of megabytes to gigabytes of dynamic RAM. Finally the virtual memory which has several gigabytes of memory.
- During the memory access, the data is looked in the lower level of memory and if it is not found there then the next level is accessed and so on. This simplifies the programming task and the same program can work effectively on various machines with different memory configurations.

Q.19 Write short note on - Locality in programs. ☞ **[JNTU : Part B, Marks 5]**

Ans. : • Many programs spent lot of time in executing small amount of code by accessing very little amount of memory. Thus exhibits the high **degree of locality.**

- Programs has two types of localities - **temporal locality** and **spatial locality.**
- The program has **temporal locality** if the locations it access are likely to accessed again within in short period of time. The program has the **spatial locality**

if the locations close to the locations it access are likely to be accessed within a short time.

- Following are some **reasons why the programs spent most of its time in executing very small amount of code -**

1. Most of the programs contain the instructions that are never executed. Programs make use of libraries that use small amount of provided functionalities. Many times there are changes in the requirements of the systems or the legacy systems get evolved.
2. Typically, small amount of code is often executed. For example the code for performing input and output operations or exceptional cases.
3. Many programs spend a lot of time in executing the inner loops or the recursive calls.

- The locality makes use of **memory hierarchy** for increasing the efficiency in program execution. For instance by placing the most common data in fast but small storage and less frequently needed data in slow and large storage the memory access time can be reduced.
- We can improve the temporal and special locality of data access in the program by changing the order of computation. For example we can bring the data from slow level(main memory) to fast level(disk) and perform the computation on this data. This makes the execution faster.

Q.20 What is fragmentation ? Explain.

[JNTU : Part A, Marks 3]

Ans. : • Due to frequent memory allocation and deallocation the heap memory becomes fragmented. That is the memory contains many free slots as well as many chunks of allocated memory. The **free slots** in the memory are called **holes.**

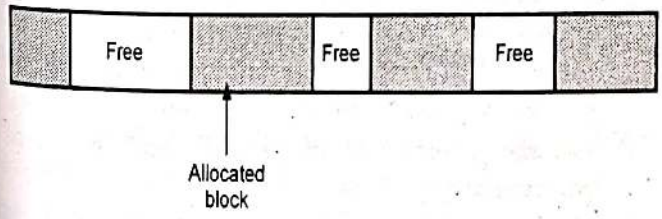


Fig. Q.20.1 Heap memory (Fragmented)

- When a memory allocation request comes, the memory manager places the request into large enough hole. This will satisfy the request but at the same time it creates one more small hole.
- With the deallocation request, the freed space is returned to the pool of free space. If memory allocation of deallocation is not done carefully many small noncontiguous holes in the memory may get created. This leads to fragmentation.

Q.21 Explain in detail, the strategy for reducing fragmentation in heap memory.

[JNTU : Part B, Marks 5]

Ans. : Two strategies are used for reducing fragmentation :

1) Best and next fit

- **Best fit :**
  - In the best fit strategy, the smallest hole which is large enough for the object that demands for allocation of memory is used.
  - In this method the memory manager searches the entire list of free slots and finds the hole which is the best possible that can be used for allocating the memory for the target object.
  - Although this strategy produces best allocation space, this method is less efficient as it requires more time in searching the appropriate hole and create many small holes.
- **Next fit :**
  - In this strategy, the free list of holes is simply scanned to find the large enough hole to hold the object.
  - This strategy is more efficient than the best fit. However due to this strategy the small holes tend to accumulate at the beginning of the free list.
  - The problem of small holes accumulating is solved with next fit strategy which starts each search where the last one left off. It wraps around to the beginning when the end of the list is reached.

2) Managing and coalescing free space

- Due allocation of memory randomly many holes are created in the memory.

- Coalescing is the technique by which many free spaces are combined together to form a large chunk of free space.
- The **advantage** of this strategy is that the fragmentation gets reduced and a large free chunk is available for allocating the space for the object of large size(since many small chunks can not hold the large object).

4.5 : Introduction to Garbage Collection

Q.22 Explain the term garbage collection

☞ [JNTU : Part A, Marks 2]

Ans. : • Garbage is a substantial amount of memory which gets allocated dynamically but is unreachable memory. If this memory is unreachable then it simply becomes the wastage of memory. Hence a new technique called garbage collector is introduced.

- Garbage collection is a technique in which the chunks memory holding unreachable objects are reclaimed.

Q.23 What is mutator in garbage collection ?

Ans. : Mutator is a user program which **creates the objects** using the heap memory. These mutators may refer or drop the references to another objects. The objects become garbage when the mutator programs can not reach them. Then the job of garbage collector is to reclaim such objects and handing them to memory manager.

Q.24 Explain the concept reachable objects in fragmentation.

Ans. : • The data that can be accessed by the program directly without using the reference to pointer is called root set.

- In Java, the root set consists of all the static field members and all the variables on its stack. Thus the program can access the members of root set any time. The objects that are references to these data members are called reachable objects.

Q.25 What are the design goals of garbage collection ? ☞ [JNTU : Part B, Marks 5]

Ans. : 1) Type safety

- The automatic garbage collection is not possible with all the languages. There are some languages the type of data components can be determined. Such languages are said to be type safe. For example ML.
- There are some languages whose type cannot be determined at the compile time but can be determined at run time. Such languages are called dynamically typed languages. For example Java.
- If the languages are not statically or dynamically type safe then such languages are called unsafe. For example C,C++.
- The unsafe languages are bad candidates of garbage collection mechanism. In unsafe languages the memory addresses are manipulated arbitrarily and no storage can be reclaimed safely.

2) Performance metrics

- Following are some performance metrics that must be considered while designing the garbage collector

i) **Space usage :** The garbage collection must avoid fragmentation and should make best use of memory.

ii) **Overall execution time :** The garbage collection is slow. It should not increase total run time of application. Its performance is determined by how it accesses the memory subsystem.

iii) **Program locality :** Simply the running time can not determine the speed of the garbage collection. The garbage collection controls the placement of data. Due to this locality of data gets affected. The locality of data plays an important role in determining the total required running time of the program.

iv) **Pause time :** Garbage collectors cause the programs (mutators) to pause suddenly for arbitrarily long time. Hence instead of minimizing overall execution time it is essential to minimize the pause time. In real time applications certain computations must be completed in some specific time. In this case the maximum pause time must be reduced. In fact, Garbage collection is occasionally used in case of real time applications.

Q.26 Explain the four operations that can be performed to change the set of reachable objects.

☞ [JNTU : Part A, Marks 4]

Ans. : Following are four basic operations that a program(mutator) can perform to change the set of reachable objects -

1. **Object allocation** : Memory manager allocates the chunk of memory for the objects. Due to this new objects are getting added in the list of reachable objects.
2. **Parameter passing and return** : Reference of objects can be passed as parameter and some results are returned back. The objects that are denoted by the references remain reachable.
3. **Reference assignments** : When we write the assignment statement a=b where a and b are both references then the object referred by b is also referred by a. Another consequence of this is that the original reference of a gets lost and now it points to the object pointed by b. If the object to which the b is referring becomes unreachable then all the objects that are reachable through b also becomes unreachable.
4. **Procedure returns** : The local variables in a the procedure have the limited scope. If this procedure holds the reachable references to the objects then those objects become unreachable when the procedure returns back to the caller.

Q.27 What is reference counter used in garbage collection ? ☞ [JNTU : Part A, Marks 2]

Ans. : Reference count is a special counter used during implicit memory allocation. With reference counting mechanism in garbage collector every object must have a field for the reference count.

4.6 : Introduction to Trace Based Collection

Q.28 What are different states that may occur during trace based algorithms ?

Ans. : Following are the states in trace based algorithm -

1) **Free** : A chunk is in free state if it is ready to get allocated. A free chunk does not hold a reachable object.
2) **Unreached** : Initially chunks are supposed to be unreachable. Whenever a chunk is allocated by memory manager its state is set to unreachable state. After the round of garbage collection, the state of reachable object is reset to unreached to get ready for next round.
3) **Unscanned** : The reachable chunks can be either in unscanned or scanned states. If the chunk is a reachable chunk and but its pointers have not been scanned, then that chunk is in unscanned state.
4) **Scanned** : Every unscanned object will lead to a scanned state. To scan an object, we examine each of the pointers within it and follow those pointers to the objects to which they refer. If a reference is to an unreached object, then that object is put in the unscanned state. When the scan of an object is completed, that object is placed in the Scanned state. A scanned object can only contain references to other scanned or unscanned objects and never to unreached objects.

Q.29 Explain basic marks and sweep algorithm. ☞ [JNTU : Part B, Marks 5]

Ans. : The mark and sweep algorithm is a garbage collection algorithm in which two major operations are performed - marks and sweep.

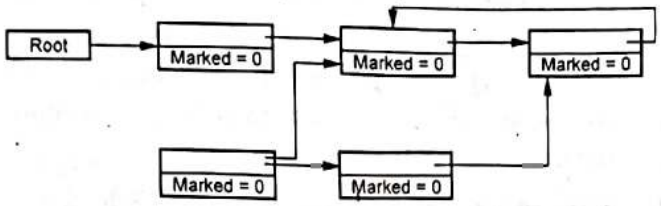
Mark phase : Before entering the object into the mark phase, there is a mark bit associated with each object which is set to false(0). In the mark stage, start from the root node in your application and set the marked bit for all the reachable items to 1(true). Now to perform this operation we simply need to do a graph traversal, a depth first search. In this traversing, each object is considered as a node. Then all the nodes that are reachable from this node or root note are considered as reachable nodes and set their mark bit to true.

Sweep phase : In this phase, all the unreachable objects are cleared. That means it clears the heap memory for all the unreachable objects. All those objects whose marked value is set to false are cleared from the heap memory. The unreachable objects are thus reclaimed.

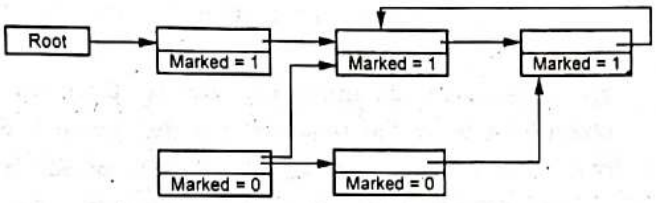
Now the mark value for all the reachable objects is set to false again since we will run the algorithm and again we will go through the mark phase to mark all the reachable objects.

This algorithm is also called **tracing algorithm.**

Before garbage collection



Stage : After mark phase



Stage : After sweep phase

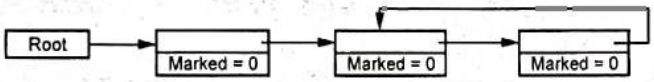


Fig. Q.29.1 Stages in mark-sweep algorithm

Part II : Code Generation

4.7 : Issues in Design of Code Generator

Q.30 Mention the properties that the code generator should posses.

☞ **[JNTU : Part A, May-18, Marks 2, Nov.-15, Marks 3]**

Ans. : Various properties desired by an object code generation phase are

- **Correctness** - It should produce a correct code and do not alter the purpose of source code.
- **High quality** - It should produce a high quality object code.
- **Efficient use of resources of the target machine** - While generating the code it is necessary to know the target machine on which it is going to get generated. By this the code generation phase can make an efficient use of resources of the target machine. For instance memory utilization while allocating the registers or utilization of arithmetic logic unit while performing the arithmetic operations.
- **Quick code generation** - This is a most desired feature of code generation phase. It is necessary that the code generation phase should produce the code quickly after compiling the source program.

Q.31 Describe code generator design issues.

☞ **[JNTU : Part B, Marks 5]**

Ans. : Input to the code generator : The code generation phase takes intermediate code as input. This intermediate code along with the symbol table information is used to determine the runtime addresses of the data objects. The code generation phase requires the complete error free intermediate code as input for generating the target code.

Target programs : The output of code generator is target code. Typically, the target code comes in three forms such as : Absolute machine language, relocatable machine language and assembly language.The advantage of producing target code in absolute machine form is that it can be placed directly at the fixed memory location and then can be executed immediately.

Memory management : Both the front end and code generator performs the task of mapping the names in the source program to addresses to the data objects in run time memory. The names used in the three address code refer to the entries in the symbol table. The type in a declaration statement determines the amount of storage(memory) needed to store the declared identifier. Thus using the symbol table information about memory requirements code generator determines the addresses in the target code.

Register allocation : If the instruction contains register operands then such a use becomes shorter and faster than that of using operands in the memory. Hence while generating a good code efficient utilization of register is an important factor.

Choice of evaluation order : The evaluation order is an important factor in generating an efficient target code. Some orders require less number of registers to hold the intermediate results than the others. Picking up the best order is one of the difficulty in code generation.

Q.32 Discuss about instruction selection and register allocation. [JNTU : Part B, Nov.-17, Marks 5]

Ans. : Refer Q.31.

4.8 : The Target Language

Q.33 What are the forms of a target program ? [JNTU : Part A, Dec.-17, Marks 2]

OR What are various possible outputs of the code generator. [JNTU : Part A, Dec.-16, Marks 2]

Ans. : • The output of code generation is an object code or machine code. This code normally comes in following forms

1. Absolute code 2. Relocatable machine code
3. Assembler code

• Let us discuss each in detail.

1. **Absolute code** - Absolute code is a machine code that contains reference to actual addresses within program's address space. The generated code can be placed directly in the memory and execution starts immediately. Generally small programs can be compiled and executed quickly because of absolute code generation. A number of "student-job" compilers such as WATFIV and PL/C produce absolute code.
2. **Relocatable code** - Producing a relocatable machine language program as output allows subprograms to be compiled separately. A set of relocatable object modules can be linked together and loaded for execution with the help of a *linking loader*. The advantage of generating relocatable machine language code is that we gain a great deal of flexibility in being able to compile subroutines separately and to call other previously compiled programs from an object module.

   If the target machine does not handle relocation automatically, the compiler must provide explicit relocation information to the loader to link the separately compiled program segments.
3. **Assembler code** - Producing an assembly language program as output makes the process of code generation somewhat easier. We can generate symbolic instructions and use the macro facilities of the assembler to help in generation of code. But generating assembler code as an output makes code generation process slower.(because assembling, linking and loading is required.)

Q.34 What are the object code forms ? Explain the issues in code generation. [JNTU : Part B, Mar.-16, Marks 5]

Ans. : Refer Q.33 and Q.31.

4.9 : Addresses in the Target Code

Q.35 Explain the form of target machine code in detail. [JNTU : Part B, Marks 5]

Ans. : • We will assume that in the target computer addresses are given in bytes and four bytes form a word.

• There are n general purpose registers R0,R1,....Rn-1.

• The two address instruction is of the form.

op source destination

where op is an opcode and source and destination are data fields.

For instance :

MOV - moves from source to destination.

ADD - add source to destination.

SUB - substracts source from destination.

The source and destination are specified by registers and memory locations

• The addressing modes used are as follows

Addressing mode	Form	Address	Added cost
absolute	M	M	1
register	R	R	0
indexed	c(R)	c + contents(R)	1
indirect register	*R	contents(R)	0
indirect indexed	*c(R)	contents(c+contents(R))	1
literal	#c	c	1

• If we have absolute or resister addressing mode we can use M or register for source or destination.

• For example, the instruction MOV R1,M stores the contents of register R_1 into memory location M.

- For the indexed addressing mode the address offset c from the value of register R0 can be written as

MOV 7(R1), M

- Means it strores the value contents (7+ contents(R1) to the memory location M.

The last two addressing modes represent the indirect addresses indicated by *.

- For the instruction MOV *7(R0),M

It stores the value contents(contents (7+ contents(R0)) to the memory location M.

- In the literal addressing mode the source becomes constant.
- For example MOV #5,R0

By this instruction we can store the constant 5 into register R0.

4.10 : Basic Blocks and Flow Graphs

Q.36 Define basic blocks.
[JNTU : Part A, March-16, Marks 3, Nov.-15, Marks 2]

Ans. : The basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching.

An example of the basic block is as shown below.

$t_1 := a * 5$
$t_2 := t_1 + 7$
$t_3 := t_2 - 5$
$t_4 := t_1 + t_3$
$t_5 := t_2 + b$

Q.37 List the terminologies used in basic blocks.
[JNTU : Part A, May-18, Marks 2]

Ans. : • **Define and use** - The three address statement a := b+c is said to define a and to use b and c.

- **Live and dead** - The name in the basic block is said to be live at a given point if its value is used after that point in the program. And the name (variable) in the basic block is said to be dead at a given point if its value is never used after that point in the program.

Q.38 How can you identify the leader in a basic block ? **[JNTU : Part A, Mar.-16, Marks 3]**

Ans. : Following rules are used to identify leader in a basic block -

a) The **first statement** is a leader.
b) Any target statement of conditional or unconditional **goto** is a leader.
c) Any statement that immediately **follow a goto** or unconditional goto is a leader.

Q.39 Write procedure to identify basic blocks.
[JNTU : Part B, March-17, Marks 5]

OR Write an algorithm for constructing basic block. **[JNTU : Part A, Dec.-16, Marks 3]**

Ans. : Any given program can be partitioned into basic blocks by using following algorithm. We assume that an intermediate code is already generated for the given program.

1. First determine the **leaders** by using following rules.
   a) The **first statement** is a leader.
   b) Any target statement of conditional or unconditional **goto** is a leader.
   c) Any statement that immediately **follow a goto** or unconditional goto is a leader.
2. The basic block is formed starting at the leader statement and ending just before the next leader statement appearing.

Q.40 Consider the following program code for computing dot product of two vectors a and b of length 10 and partition it into basic blocks.

```
prod = 0;
i = 1;
do
{
prod = prod + a[i] * b[i];
i=i+1;
}while(i<=10);
```
[JNTU : Part B, May-09, Marks 16]

Ans. : First we will write the equivalent three address code for the above program.

1. prod := 0
2. i := 1
3. $t_1 := 4 * i$
4. $t_2 := a[t_1]$ /* computation of a[i] */

5. $t_3 := 4 * i$
6. $t_4 := b[t_3]$ /* computation of b[i] */
7. $t_5 := t_2 * t_4$
8. $t_6 := prod+t_5$
9. $prod := t_6$
10. $t_7 := i+1$
11. $i := t_7$
12. if i< = 10 goto (3)

According to the algorithm

Statement 1 is a leader by rule 1(a).

Statement 3 is also leader by rule 1(b).

Hence, statement 1 and 2 form the basic block. Similarly statement 3 to 12 form another basic block.

Block 1

1. prod := 0
2. i := 1

B_1

Block 2

3. $t_1 := 4 * i$
4. $t_2 := a[t_1]$
5. $t_3 := 4 * i$
6. $t_4 := b[t_3]$
7. $t_5 := t_2 * t_4$
8. $t_6 := prod+t_5$
9. $prod := t_6$
10. $t_7 := i + 1$
11. $i := t_7$
12. if i<=10 goto (B_2)

Q.41 What is a flow graph ?

[JNTU : Part A, May-18, Dec.-17, Marks 3, Dec.-16, Marks 2]

Ans. : A flow graph is a directed graph in which the flow control information is added to the basic blocks.

For example, consider the three address code as

1. prod := 0
2. i := 1
3. $t_1 := 4 * i$
4. $t_2 := a[t_1]$ /* computation of a[i] */
5. $t_3 := 4 * i$
6. $t_4 := b[t_3]$ /* computation of b[i] */
7. $t_5 := t_2 * t_4$
8. $t_6 := prod+t_5$
9. $prod := t_6$
10. $t_7 := i+1$
11. $i := t_7$
12. if i <= 10 goto (3)

The flow graph for the above code can be drawn as follows.

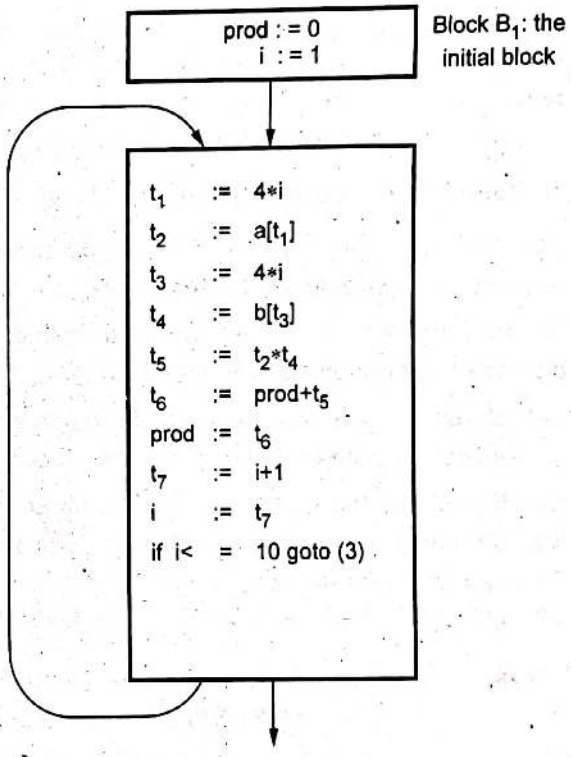


Fig. Q.41.1 Flow graph

Q.42 Define flow graph. Explain how a given program can be converted to a flow graph ?

[JNTU : Part B, Mar.-16, Marks 5]

Ans. : Refer Q.41.

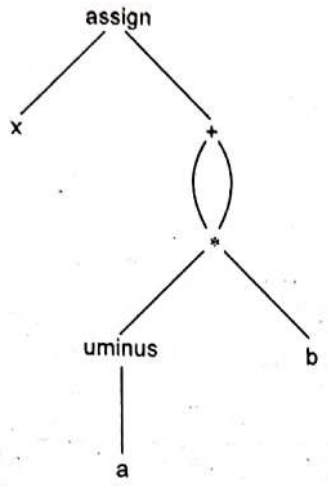
4.11 : Optimization of Basic Blocks

Q.43 What is DAG ? Mention its applications.

[JNTU : Part A, May-18, Nov.-15, Marks 3]

Ans. : DAG : DAG stands for directed acyclic graph. It is a graph that is directed and without cycles connecting the other edges.

For example -



DAG

Fig. Q.43.1 DAG

Applications : The DAG is used in

1. Determining the common sub-expressions (expressions computed more than once).
2. Determining which names are used inside the block and computed outside the block.
3. Determining which statements of the block could have their computed value outside the block.
4. Simplifying the list of quadruples by eliminating the common sub-expressions and not performing the assignment of the form x:=y unless and until it is a must.

Q.44 Write an algorithm for construction of DAG

☞[JNTU : Part B, Marks 5]

Ans. : We assume the three address statement could of following types

Case (i) x:=y op z

Case (ii)x:=op y

Case (iii) x:=y

With the help of following steps the DAG can be constructed

Step 1 : If y is undefined then create node(y). Similarly if z is undefined create a node(z).

Step 2 : For the case(i) create a node(op) whose left child is node(y) and node(z) will be the right child. Also check for any common subexpressions. For the case(ii) determine whether is a node labeled op,such node will have a child node(y). In case(iii) node n will be node(y).

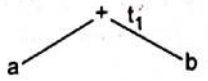
Step 3 : Delete x from list of identifiers for node(x). Append x to the list of attached identifiers for node n found in 2.

Q.45 For the following statements draw the DAG.

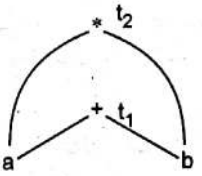
$t_1 = a + b;$
$t_2 = a * b;$
$t_3 = t_1 * b;$
$c = t_3 + t_2;$

☞[JNTU : Part B, Jan.-12, Marks 5]

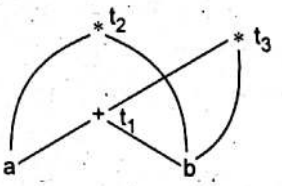
Ans. : Step 1 :



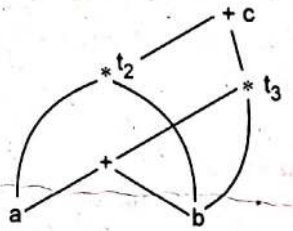
Step 2 :



Step 3 :



Step 4 :

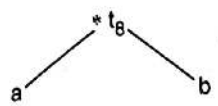


Q.46 For the following statements draw the DAG.

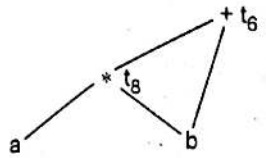
$t_8 = a*b$
$t_6 = t_8 + b$
$t_5 = a - b$
$t_4 = t_6 * t_5$
$t_3 = t_4 - e$
$t_2 = t_8 * t_4$
$t_1 = t_2 * t_3$

☞[JNTU : Part B, Jan.-12, Marks 5]

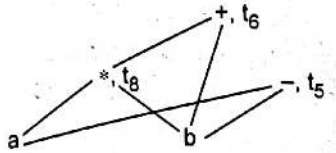
Ans. : Step 1 :



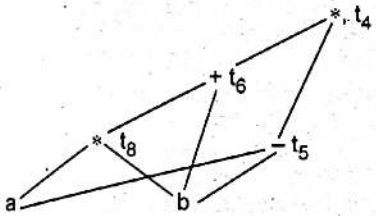
Step 2 :



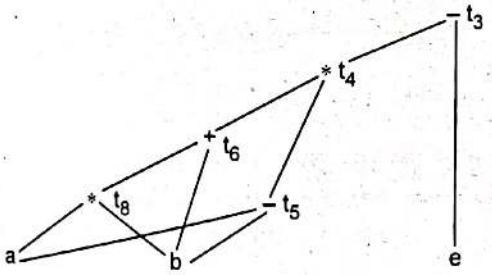
Step 3 :



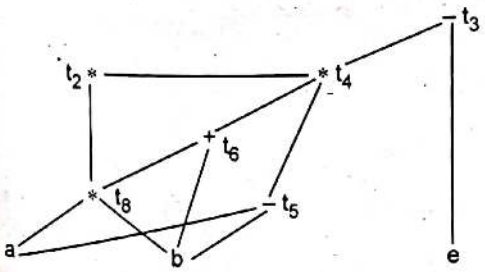
Step 4 :



Step 5 :



Step 6 :



Step 7 :

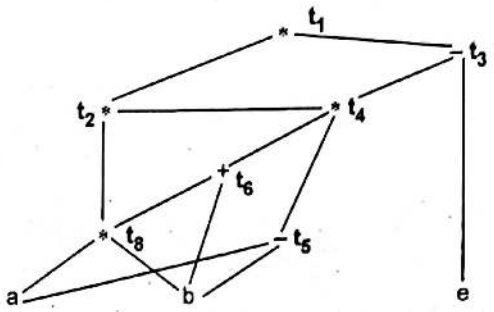
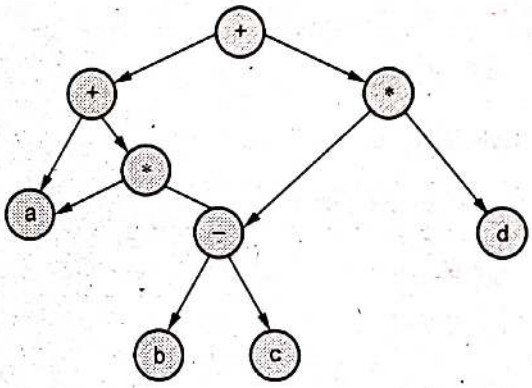


Fig. Q.46.1 DAG

Q.47 Construct DAG for the expression a + a * (b – c) + (b – c) * d

☞ [JNTU : Part B, Dec.-12, Marks 5]

Ans. : The DAG will be



Q.48 Construct DAG for the following basic block :

T1=A+B
T2=C+D
T2=E–T2
T4=T1–T3

☞ [JNTU : Part A, Dec.-16, Marks 3]

Ans. : We will construct a DAG for

$t_1 = a + b$

$t_2 = c + d$

$t_3 = e - t_2$

$t_4 = t_1 - t_3$

For the above order of evaluation we get the code as [Assuming two registers R_0 and R_1 are available]

MOV a, R_0

ADD b, R_0

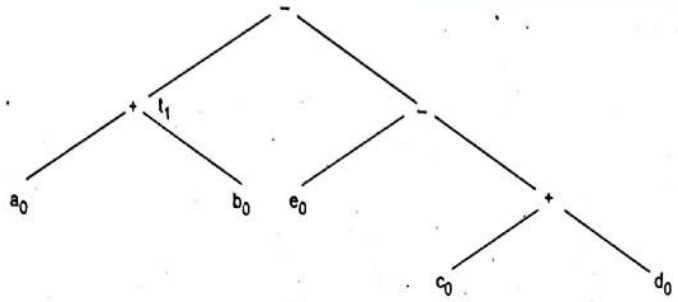


Fig. Q.48.1

MOV c, R_1

ADD d, R_1

MOV R_0, t_1

MOV e, R_0

SUB R_1, R_0

MOV t_1, R_1

SUB R_0, R_1

MOV R_1, t_4

If we change the sequence of computation as :

$t_2 = c + d$

$t_3 = e - t_2$

$t_1 = a + b$

$t_4 = t_1 - t_3$

Then the generated code will be

MOV c, R_0

ADD d, R_0

MOV e, R_1

SUB R_0, R_1

MOV a, R_0

ADD b, R_0

SUB R_1, R_0

MOV R_0, t_4

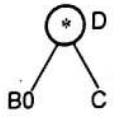
Thus by rearranging the order the code can be generated.

Q.49 What is DAG ? Construct DAG for the following basic block ?
D=B*C; E=A+B; B=B*C; A=E-D;
[JNTU : Part B, Nov.-17,Mar.-16, Marks 10]

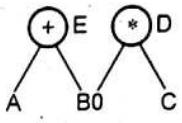
Ans. : Definition of DAG : Refer Q.43.

DAG for basic block :

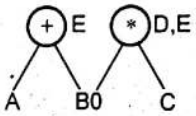
Step 1 :



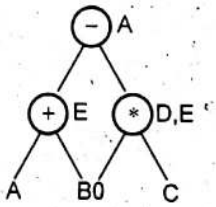
Step 2 :



Step 3 :



Step 4 : Finally we get the following DAG



Q.50 Construct DAG for a + a * (b – c) + (b – c)* d. Also generate three address code for same.
[JNTU : Part B, Marks 5]

Ans. : The three address code is

$t_1 = b - c$

$t_2 = a * t_1$

$t_3 = a + t_3$

$t_4 = t_1 * d$

$t_5 = t_3 + t_4$

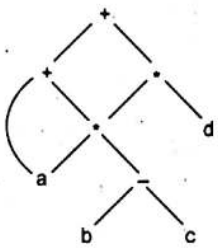


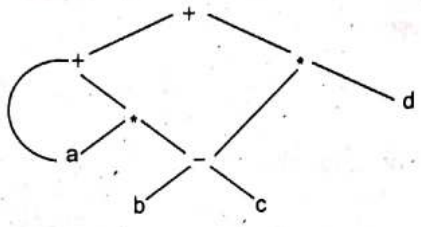
Fig. Q.50.1 DAG for a + a * (b – c) + c) * d

Q.51 Describe how DAG can be used in register allocation process ? Give examples.
[JNTU : Part B, Nov.-15, May-18, Marks 10]

Ans. : The DAG can be used for code generation. In this process the three address code is generated and DAG is constructed for this three address code. Sometimes we may need to rearrange the order of three address code for efficient usage of register.

Q.52 Draw a DAG for expression a + a * (b – c) + (b – c) * d ☞ **[JNTU : Part B, Marks 5]**

Ans. :



Q.53 Explain how type checking and error reporting is performed in a compiler. Draw syntax tree and DAG for following statement. Write three address codes from both. a = (a + b * c) ^ (b * c) + b * c. ☞ **[JNTU : Part B, Marks 5]**

Ans. : The type checker is a **translation scheme** in which the type of each expression from the types of subexpressions is obtained. The type checker can decide the types for arrays, pointers, statements and functions.

The type checker ensures following things -

i) Each identifier must be declared **before the use.**

ii) The use of identifier must be **within the scope.**

iii) An identifier must **not** have **multiple definitions** at a time within the same scope.

The given grammar has basic data types as char, int, float and for reporting the errors **type_error.**

We assume that the index of the array start at 0. For example if

```
int arr[100];
```

leads to **type expression** as array(0,1...99,int) similarly

```
int *ptr;
```

The **type expression** for the above statement can be pointer(int)

Let us write the translation scheme for the above grammar.

Production rule	Type expression
S → D;E	This means all declarations before expressions
D → T LIST	LIST.type:=T.type
LIST → id	{addtype(id.entry,LIST.type)}
T → char	{T.type:=char}
T → int	{T.type:=int}
T → float	{T.type:=float}
LIST → *L_1	{L_1.type := pointer(LIST.type)}
LIST → L_1[num]	{L_1.type:=array(0...num.val–1,LIST.type)}

} Type checker

After obtaining the type expression it becomes convenient to write the semantic rules. These are the semantic rules for type checking of expression.

For example : while performing binary operation the data types are decided as follows

$E \rightarrow E_1$ op E_2 {E.type:= if E_1.type:= int and E_2.type:= int

then int

else if E_1.type:= float and E_2.type:= float

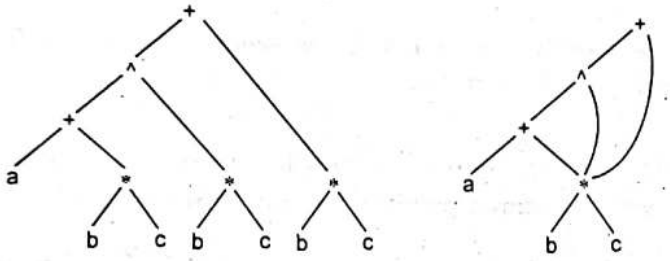
then **float**

else if E_1.type:= char and E_2.type:= char

then **char**

else type_error

The **type_error** routine is used to report the error that occur during type checking.



Syntax tree

DAG

Three address code for syntax tree	Three address code for DAG
t_1:= b*c	t_1:= b*c
t_2:= a+t_1	t_2:= a + t_1
t_3:= b*c	t_3:= t_2 ^ t_1
t_4:= t_2 ^ t_3	t_4:= t_3 + t_1
t_5:= b*c	
t_6:= t_4 + t_5	

Q.54 Construct DAG of basic blocks after converting the code in 3-address representation :

```
i = 1 ;
j = 2 ;
repeat
A[i] = j
j = j * 2 ;
i = i + 1 ;
until (i > 10).
```

☞[JNTU : Part B, Marks 5]

Ans. : 1) i : = 1

2) j : = 2

3) t1 : = 4 * i / * 4 is a width * /

4) t2 : = Addr (A) - 4

5) t2 [t1] : = j

6) j : = j * 2

7) i : = i +1

8) if i < 10 goto (3)

DAG can be

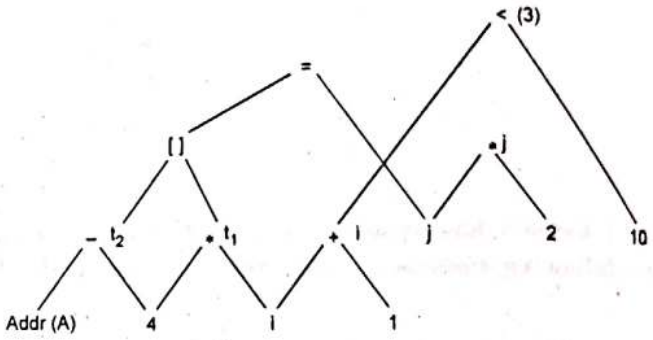
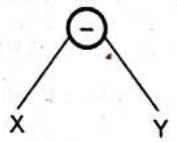


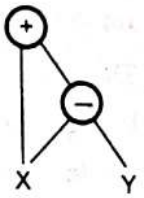
Fig. Q.54.1

Q.55 Show the DAG for statement Z = X - Y + X * Y * U - V/W + X + V. ☞[JNTU : Part B, Marks 5]

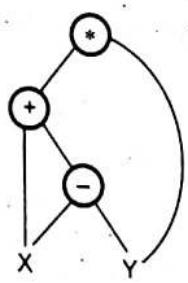
Ans. : Step 1



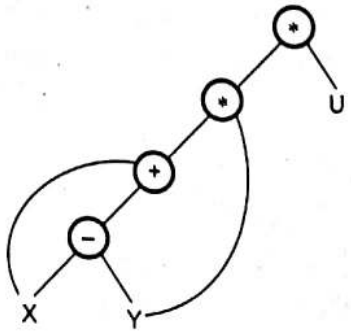
Step 2 : X - Y + X



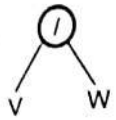
Step 3 : X - Y + X * Y



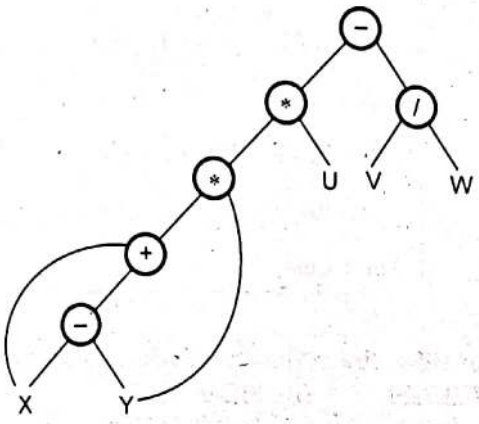
Step 4 : X - Y + X * Y * U



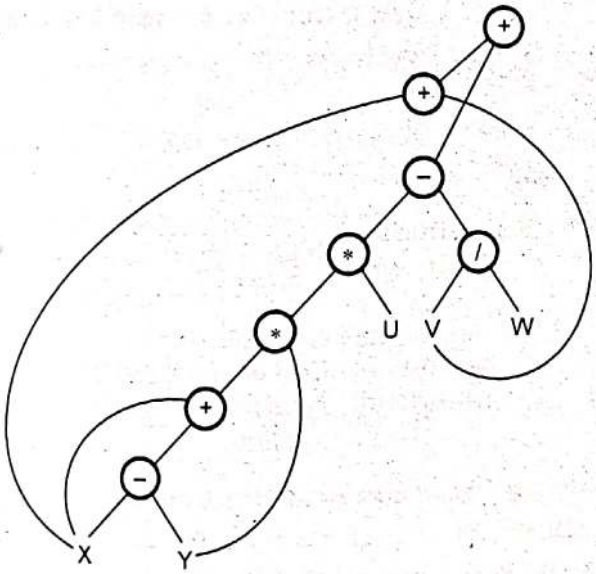
Step 5 : V / W



Step 6 : X – Y + X * Y * U – V/ W



Step 7 : X – Y + X * Y * U – V/ W + X + V



Q.56 Draw DAG for given block :

a = b + c
b = a – d
c = b + c
d = a – d

☞ **[JNTU : Part B, Marks 5]**

Ans. :

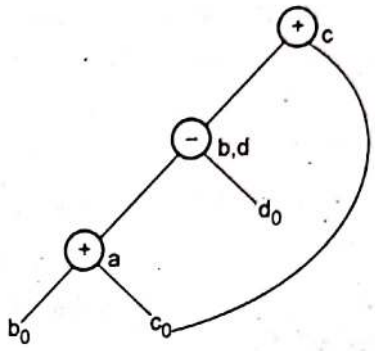


Fig. Q.56.1

4.12 : A Simple Code Generator

Q.57 What is meant by register descriptor and address descriptor ?

☞ **[JNTU : Part A, Mar.-17, Marks 2]**

Ans. :

1. A **register descriptor** is used to keep track of what is currently in each register. The register descriptors show that initially all the registers are empty. As the code generation for the block progresses the registers will hold the values of computations.

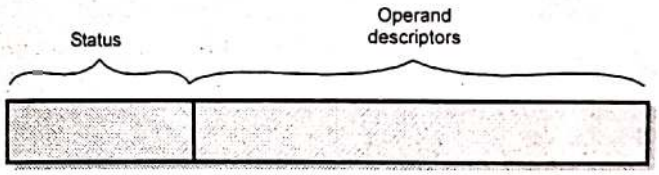


Fig. Q.57.1 Register descriptor

2. The **address descriptor** stores the location where the current value of the name can be found at run time. The information about locations can be stored in the symbol table and is used to access the variables.

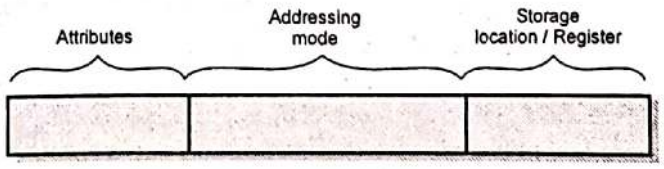


Fig. Q.57.2 Address descriptor

- The address descriptor has following fields

- The **attributes** mean type of the operand. It generally refers to the name of temporary variables.
- The **addressing mode** indicates whether the addresses are of type 'S', 'R', 'IS', 'IR'.

  S is used to indicate value of operand in storage.

  R is used to indicate value of operand in register.

  IS indicates that the address of operand is stored in storage i.e. indirect accessing.

  IR indicates that the address of operand is stored in register i.e. indirect accessing.
- The third field is **location field** which indicates whether the address is in storage location or in register.

Q.58 Generate the code for the following expression : x=(a+b) – ((c+d) –e). Also compute its cost. ☞ [JNTU : Part B, Mar.-17, Marks 10]

Ans. : We will write the three address code for the given expression

T1 = a + b

T2 = c + d

T3 = T2 – e

T4 = T1 – T3

The code can be generated as follows

MOV a, R0

ADD b,R0 /* R0=a+b*/

MOV d, R1

ADD c, R1 /*R1=c+d*/

SUB e, R1

SUB R1, R0

MOV R0, x

The cost of above instructions can be computed as -

Instruction	Cost
MOV a, R0	1 + 1 + 0 = 2
ADD b,R0	1 + 1 + 0 = 2
MOV d, R1	1 + 1 + 0 = 2
ADD c, R1	1 + 1 + 0 = 2
SUB e, R1	1 + 1 + 0 = 2
SUB R1, R0	1 + 0 + 0 = 1
MOV R0, x	1 + 0 + 1 = 2
Total Cost	13

Q.59 Consider the following code sequence

(i) MOV B, R0
ADD C, R0
MOV R0, A

(ii) MOV B, A
ADD C, A

Calculate cost of above instructions.

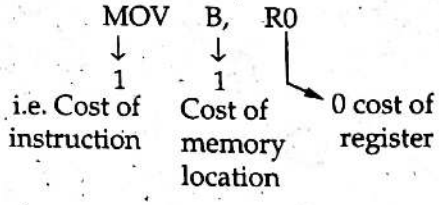
☞ [JNTU : Part B, Aug.-07, Marks 10]

Ans. : MOV B, R0

ADD C, R0

MOV R0, A

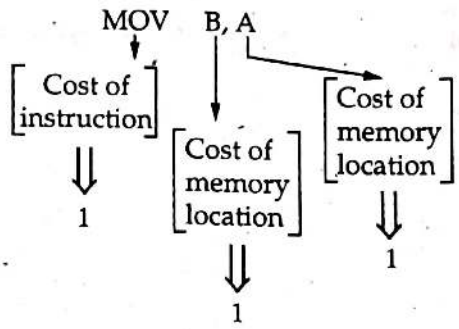
For instruction



∴ MOV B, R0 → cost = 1 + 1 + 0 = 2
ADD C, R0 → cost = 1 + 1 + 0 = 2
MOV R0, A → cost = 1 + 0 + 1 = 2

Total cost = 6

ii)



∴ MOV B, A → cost = 1 + 1 + 1 = 3
ADD C, A → cost = 1 + 1 + 1 = 3

Total cost = 6

Q.60 Write an algorithm for code generation using GETREG. [JNTU : Part B, Marks 5]

Ans. : Read the expression in the form of Operator, operand1 and operand2 and generate code using following algorithm -

```
Gen_Code(operator,operand1,operand2)
{
  if (operand1.addressmode = 'R')
  {
      if (operator = '+')
      Generate('ADD operand2,R0');
      else if(operator = '-')
          Generate('SUB operand2,R0');
      else if(operator = '*')
          Generate('MUL operand2,R0');
      else if(operator = '/')
          Generate('DIV operand2,R0');
  }
      else    if (operand2.addressmode = 'R')
  {
      if (operator = '+')
      Generate('ADD operand1,R0');
      else if(operator = '-')
          Generate('SUB operand1,R0');
      else if(operator = '*')
          Generate('MUL operand1,R0');
      else if(operator = '/')
          Generate('DIV operand1,R0');
  }
      else
  {
      Generate('MOV operand2,R0);
          if (operator = '+')
      Generate('ADD operand2,R0');
      else if(operator = '-')
          Generate('SUB operand2,R0');
      else if(operator = '*')
          Generate('MUL operand2,R0');
      else if(operator = '/')
          Generate('DIV operand2,R0');
  }
```

Q.61 Generate the code sequence using code generation algorithm for the following expression W : = (A – B) + (A – C) + (A – C). [JNTU : Part B, Aug.-08, Marks 8]

Ans. : We will write the 3 address code for given expression

$t_1 := A - B$

$t_2 := A - C$

$t_3 := t_1 + t_2$

$t_4 := t_3 + t_2$

$W := t_4$

Three address code	Target code sequence	Register descriptor	Operand/ address descriptor
$t_1 := A - B$	MOV A, R_0 SUB B, R_0	Empty R_0 contains t_1	t_1 is in R_0
$t_2 := A - C$	MOV A, R_1 SUB C, R_1	R_1 contains A R_1 contains t_2	t_2 is in R_1
$t_3 := t_1 + t_2$	ADD R_1, R_0	R_0 contains t_3	t_3 in R_0
$t_4 := t_3 + t_2$ $W := t_4$	ADD R_1, R_0 MOV R_0, W	R_0 contains t_4 R_0 contains W	W is in R_0

Q.62 Generate code for the following C program using any code generation algorithm.

```
main()
{
  int i;
  int a[10];
  while(i<=10)
   a[i]=0;
}
```

[JNTU : Part B, May-08, Marks 10]

Ans. : First of all we will generate three address codes for the given C code.

Label 1 : if i<=10 goto Label 2

goto Label 3

Label 2 : t_1 := i*4

t_2 := addr(a)

$t_2[t_1]$:= 0

goto Label1

Label 3 : stop

The code generation using simple code generation algorithm for the corresponding code is

```
MOV i,R0
Label 1 : CMP R0,#10
        JLE Label2
        JMP Label3
Label 2 : MOV #0,a(R0)
        JMP Label1
Label 3 : EXIT
```

Q.63 Generate code for following c statements

(i) x = f(a) + f(a) + f(a) **(ii) x = f(a) / g (b, c)**

(iii) x = f(f(a)) **(iv) x = + + f(a)**

☞ [JNTU : Part B, May-08, Marks 8]

Ans. : i) f(a) denotes a call to the function. This call is made thrice. And them addition of their return values is done. The result is finally stored in variable x.

We will first write a three address code for given statement.

```
        t1 : = 0
        t2 : = 1
        t3 : = 3
L1 :    if t2 <= t3 goto Loop
        goto End
Loop :  parama
        call f,1
        return t4
        t1 : = t1 + t4
        t2 : = t2 + 1
        goto L1
End :
```

The code will be

(10) MOV #0, R_0 (11) MOV #1, R_1

(12) MOV #3, R_2 (13) CMP R_1, R_2

(14) CJ < = (16) (15) H A L T

(16) MOV a, R_3 /* parameter a is in register R3 */

(17) GOTO 100 /* At location 100 procedure f is defined */.

(18) MOV AH, R_4/* return val from function f is in AH */

(19) ADD R_4, R_0 /* it is stored in R4 then added to R0 *\

(20) MOV R_0, x /* R0 value is stored in x */

(21) INC R_1

(22) Loop (13)

⋮

(100) /* procedure for f*/

ii) x = f(a) / g (b, c)

Param a

call f, 1

return t_1

Param b

Param c

call g, 2 /* 2 parameter to g */

return t_2

$x = t_1 / t_2$.

The code will be

(1) MOV a, R_0

(2) GOTO 100

(3) MOV AX, R_1 /* return value of f(a) in R_1 */

(4) MOV b, R_2

(5) MOV c, R_3

(6) GOTO 200

(7) MOV AX, R_4 /* return value of g(b, c) in R_4 */

(8) DIV R_1, R_4 /* f(a)/g(b ,c)*/

(9) MOV R_4 x. /* store result in x */

⋮

(100) /* procedure for f */

⋮

(200) /* procedure for g */

⋮

iii) Three address code for x=f(f(a))

param a

call f, 1

return t_1

param t_1

call f, 1

return t_2

x : = t_2

The code will be

(1) MOV a, R_0 /* parameter a in R_0 */

(2) GOTO 100 /* at location 100 f(a) is defined */

(3) MOV AX, R_1 /* return addres of AX is stored in R */

(4) MOV R_1, R_2 /* Take parameter R_1 in register R_2 */

(5) GOTO 200 /* at 200 location procedure f(f(a)) */

(6) MOV AX, R_3 /* return value in R_3 */

(7) MOV R_3, x /* finally in x */

⋮

(100) /* procedure for f(a) */

⋮

(200) /* f(f(a)) */

⋮

iv) x = + + f(a)

param a

call f, 1

return t_1

$t_1 = t_1 + 1$

x : = t_1

The code will be

(1) MOV a, R_0 /* parameter a is in R_0 */

(2) GOTO 100 /* At location 100 f(a) is defined */

(3) MOV AX, R_1 /* MOV return value in R1 */

(4) ADD # 1, R_1

(5) MOV R_1, x /* Final result in variable x */

⋮

(100) /* procedure for f(a) */

Q.64 Generate code for following statements for the target machine (target machine is a byte addressable machine with 4 bytes to a word and N general purpose registers). Assuming all variables are static. Assume 3 registers are available.

(a) x = a [I + 1]
(b) a [I] = b [c [I]]
(c) a [I][J] = b [I][k]* c [k][J]
(d) a [I] = a [I]+ b [J]

☞ **[JNTU : Part B, May-05, Marks 8]**

Ans. :

(a) x = a [I]+1

MOV I, R_0

MOV a(R_0), R_1

ADD #1, R_1

MOV R_1, x.

(b) a [I] = b [c [I]]

MOV I, R_0

MOV c(R_0), R_1 /* c [I] */

MOV b (R_1), R_2 /* b [c [I]] */

MOV R_2, a (R_0)

(c) a [I][J] = b [I][k]* c [k][J]

The two dimensional array can be represented by one dimensional array.

For instance : Consider elements 10, 20, 30, 40, 50, 60, 70, 80

	1	2	3	4
1	10	20	30	40
2	50	60	70	80

Two dimensional arra

1	2	3	4	5	6	7	8	
10	20	30	40	50	60	70	80	

↑

In 2D array i = 2, j = 3

In 2D array i = 2, j = 3

Hence in row major representation

A [i] [j] = Base address + (j * row-size + i)
× Element size

Element size = 4 bytes.

Row-size = 2

Then A [i] [j] = Base address + (2 * j+ i) * 4.

The code will be

MOV J, R_0

MUL #2, R_0

ADD k, R_0

MUL #4, R_0

MOV c(R_0), R_1 /* c[k] [J] */

MOV k, R_0

MUL #2, R_0

ADD I, R_0

MUL #4, R_0

MUL b (R_0), R_1 /* R_1 = b [I] [k] * c[k] [J] */

MOV J, R_0

MUL #2, R_0

ADD I, R_0

MUL #4, R_0

MOV R_1, a(R_0)

(d) a [I] = a [I]+b [J]

MOV I, R_0

MOV a(R_0) , R_1

MOV J, R_0

ADD b(R_0), R_1

MOV I, R_0

MOV a(R_0), R_2

MOV R_1, R_2

Q.65 Generate the code for the following C statements using its equivalent three address code.
a) a = b + c
b) x = a/(b + c) d*(e + f)
c) *A = p
d) A = B + C **☞[JNTU : April-11, Marks 16]**

Ans. : (a) a = b + c

MOV b, R0

ADD c, R0

MOV R0, a

(b) x = a/(b + c) + d*(e + f)

MOV b, R0

ADD c, R0

ADD d, R0

MOV e, R1

ADD f, R1

MUL R1, R0

MOV a, R1

DIV R1, R0

MOV R0, x

(c) *A = P

MOV P, R0

MOV R0, *A

(d) MOV C R0

ADD B, R0

MOV R0, A

Q.66 Generate object code for the following statements a= b+c; d=a+e;
☞[JNTU : Part A, Nov.-15, Marks 2]

Ans. : The object code for **a = b + c** is

MOV b, R0

ADD c, R0

MOV R0, a

The object code for **d = a + e** is

MOV a, R0

ADD e, R0

MOV R0, d

4.13 : Peephole Optimization

Q.67 Explain various method to handle peephole optimization. **☞[JNTU : Part B, Mar.-17, Marks 10]**

Ans. : Definition

Peephole optimization is a simple and effective technique for locally improving target code. This technique is applied to improve the performance of the target program by examining the short sequence of target instructions and replacing these instructions by shorter or faster sequence.

The peephole optimization can be applied on the target code using following characteristic.

1. Redundant instruction elimination

- Especially the redundant loads and stores can be eliminated in this type of transformations.

For example :

MOV R0,x
MOV x,R0

We can eliminate the second instruction since x is already in R0. But if (MOV x, R0) is a label statement then we cannot remove it.

- We can eliminate the unreachable instructions. For example, following is a piece of C code.

```
sum=0
if(sum)
 printf("%d",sum);
```

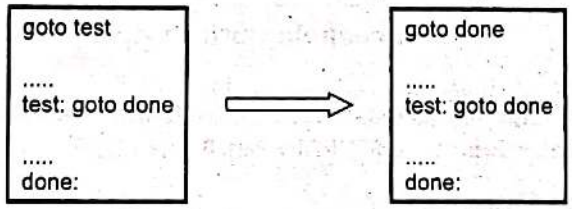
Now this if statement will never get executed hence we can eliminate such a unreachable code. Similarly

```
int fun(int a,int b)
{
c=a+b;
return c;
printf("%d",c);  /* unreachable code and hence can be
eliminated */
}
```

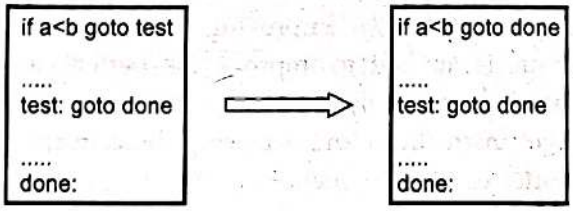
2. Flow of control optimization

Using peephole optimization unnecessary jumps on jumps can be eliminated.

For example,



Thus we reduce one jump by this transformation.



Another example could be

3. Algebraic simplification

Peephole optimization is an effective technique for algebraic simplification.

The statements such as

```
x := x + 0
```

or

```
x := x * 1
```

can be eliminated by peephole optimization.

4. Reduction in strength

Certain machine instructions are cheaper than the other. In order to improve the performance of the intermediate code we can replace these instructions by equivalent cheaper instruction. For example, x^2 is cheaper than x * x. Similarly addition and subtraction is cheaper than multiplication and division. So we can effectively use equivalent addition and subtraction for multiplication and division.

5. Machine idioms

The target instructions have equivalent machine instructions for performing some operations. Hence we can replace these target instructions by equivalent machine instructions in order to improve the efficiency. For example, some machines have auto-increment or auto-decrement addressing modes that are used to perform add or subtract operations.

Q.68 What is algebraic transformation ?

☞ [JNTU : Part A, Dec.-17, Marks 2]

Ans. : • Algebraic transformations are kind of transformations that can be applied for algebraic expressions. This transformation is basically obtained for code optimization.

- The typical transformations that come under category of algebraic transformation are -
  1) Removing of operations like -
     i) Multiplication by 1
     ii) Multiplication by 0
     iii) Addition by 0
  2) Reorder instructions based on **commutative properties of operators.** For example x + y is same y + x.

Q.69 Explain the following peephole optimization technique :

a) Elimination of redundant code.

b) Elimination of unreachable code.

☞ [JNTU : Part B, Nov.-16, Marks 10]

Ans. : Refer Q.67.

4.14 : Register Allocation and Assignment

Q.70 How to allocate registers to instruction?

☞ [JNTU : Part A, March-17, Marks 3]

OR What is meant register allocation and assignment ? **☞ [JNTU : Part A, Mar.-16, Marks 3]**

Ans. : • The most commonly used strategy to register allocation and assignment is to assign specific values to specific registers. For instance for base addresses separate set of registers can be used. Similarly for storing the stack pointers again a separate set of registers can be assigned; arithmetic computations can be done using separate set of registers. Remaining set of registers are used by the compiler for suitable purposes.

- The advantage of this approach is that the design process of compiler for code generation becomes simplified.
- The disadvantage of this method is that the design of compiler becomes complicated because of restrictive use of registers. At the same time certain set of registers remain totally unused over substantial portions of code and some set of registers get overloaded. But this disadvantage can be tolerable by most of the compiler and this approach can be adopted in most of the computing environment.
- Various strategies used in register allocation and assignment
  1. Global register allocation.
  2. Usage count.
  3. Register assignment for outer loop.
  4. Graph coloring for register assignment.

4.15 : Dynamic Programming Method of Code Generation

Q.71 Explain the working principle of dynamic programming of code generation with suitable example [JNTU : Part B, Marks 10]

Ans. : The dynamic programming works on following principle -

Suppose that an expression E is given. Then by dynamic programming algorithm the expression E is broken into subexpressions. The optimal code for these subexpressions is obtained. Then using the given operator of E the subexpressions are evaluated on "contiguously".

To evaluate a tree T the T_1 is evaluated first, then T_2 gets evaluated and then we proceed for root or it will be: evaluation of T_2 then T_1 and finally root.

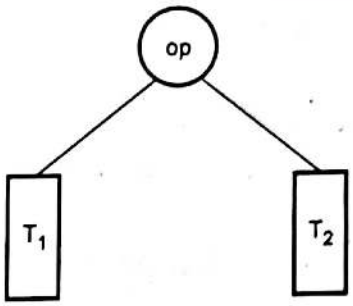


Fig. Q.71.1 Tree T

If T_1 gets evaluated completely then the computation is stored in some register. Thereafter T_2 is evaluated. Thus the order of evaluation is in contiguous fashion.

Example : By considering following instruction set each of unit cost and two registers R_0 and R_1 are available; obtain the cost vector for the expression.

$(a+b)*(c/d)$

$R_i := M$

$R_i := R_i$ op R_j

$R_i := R_i$ op M

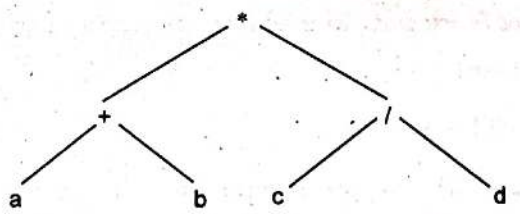
$R_i := R_j$

$M := R_i$

Sol. : Here M refer to memory location and R_i and R_j refers to R_0 or R_1. For the expression $(a+b)*(c/d)$ first we will construct an expression tree or syntax tree as follows.

Now we will compute cost vectors in bottom-up fashion.

For node labelled as 'a'



c[0] = 0 This is the cost of computing a in memory. As a is already in memory location the cost turns out to be 0.

c[1] = 1 This is the cost of computing a in register. As we can load a into register i.e. R_0 :=a hence the cost is 1.

c[2] = 1 The cost of computing a into two available registers. We can utilize only one. once the cost of

two registers available will be same as cost of one register available.

Hence the cost vector of leaf node a is (0, 1, 1). In the same manner cost vector for node b is (0, 1, 1).

We will now compute the cost vector for right subtree (Note : The computation of cost vector is to be done in postorder manner).

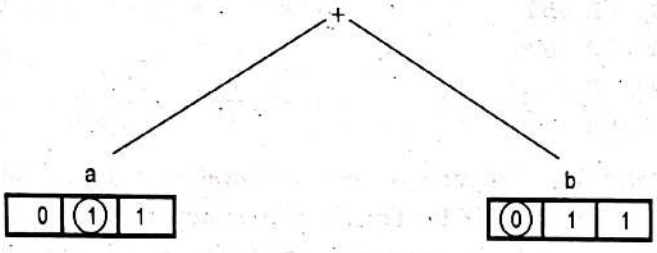
The cost vector for c is (0, 1, 1).

The cost vector for d is (0, 1, 1).

Consider node +

c[0] = 3 This is the cost of computing + into memory.

We will match $R_0 := R_0 + M$. The cost of one register available from left subtree and cost of computing right subtree from memory location.



As the node is operator we can add 1 to the total cost. Hence

$R_0 = 1+0+1 = 2$. But this is the cost computing available in

R_0 = [as $R_0 := R_0 + M$] We can store R_0 at memory because we are computing c[0] i.e. for memory location. Storing R_0 to memory again costs 1.

$\therefore$ Total cost + 1= 2 + 1 = 3

Hence c[0] = 3.

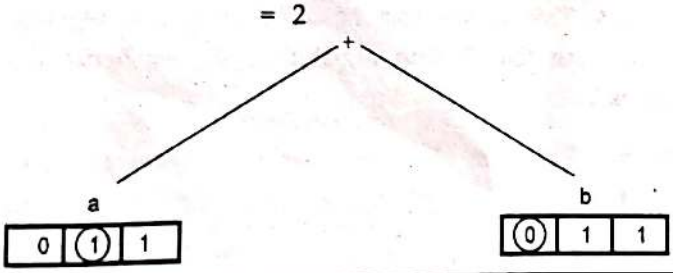
c[1] = 2 The cost of computing + in one register available.

We will match $R_0 := R_0 + M$

plus 1 for cost of instruction

$= 1 + 0 + 1$

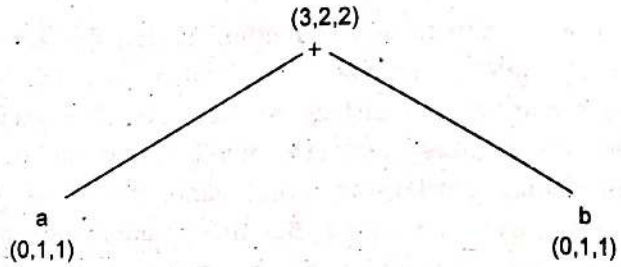
$= 2$



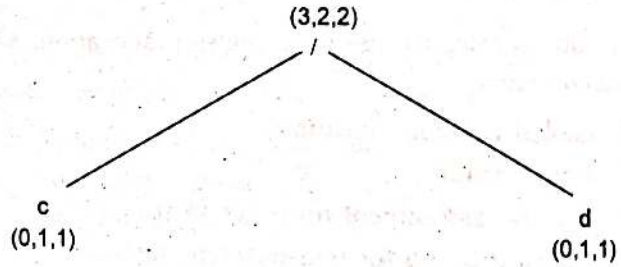
c[2] = 2 The cost of computing + in two registers available.

For this computation we can match it with two instructions. $R_0 := R_0 + R_1$ and $R_0 := R_0 + M$. If $R_0 := R_0 + R_1$ is matched then we get 1 + 1 + 1 = 3. But as both the left and right nodes are operands we can keep one operand in memory and one in register and there by obtain the optimum cost. Hence we will match $R_0 := R_0 + M$ even if cost is computed for two register available.

Hence , c[2] = 1 + 0 + 1 = 2.



Continuing in this fashion we can obtains the cost for node/as follows.

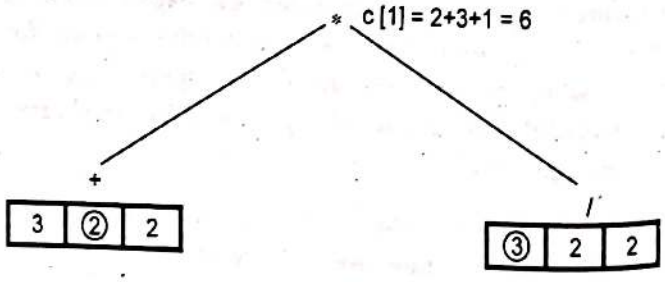


For the root node *

We will first determine the cost of computing root with one register available.

c[1] = 6

As we can match $R_0 := R_0 * M$. That means minimum cost of evaluating left subtree into one register, plus minimum cost of evaluating right subtree into memory, plus 1 for cost of instruction.

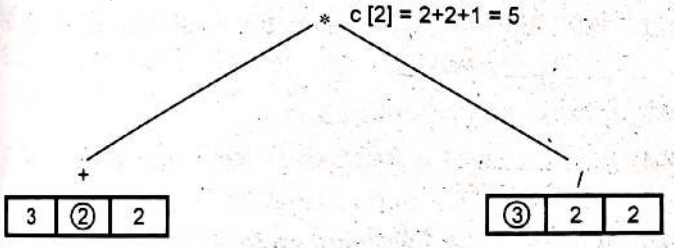


Now let us compute the minimum cost for * node with two registers available. There are 3 cases to compute the root node. These cases are also useful to decide the order of evaluation of subtrees.

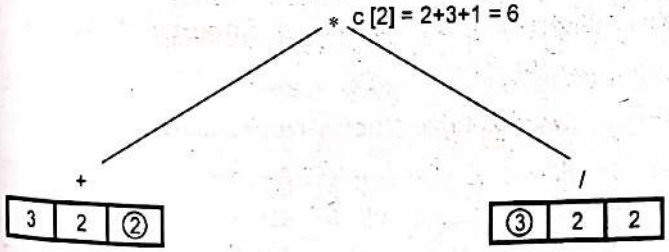
Case 1 : Compute using instruction $R_0 := R_0 + R_1$. The left subtree can be computed with two registers available and right subtree can be computed with one register. Therefore cost can be

c[2] = Cost of left subtree with 2 register
+ Cost of right subtree with 1 register
+ 1 for instruction
= 2 + 2 + 1 = 5

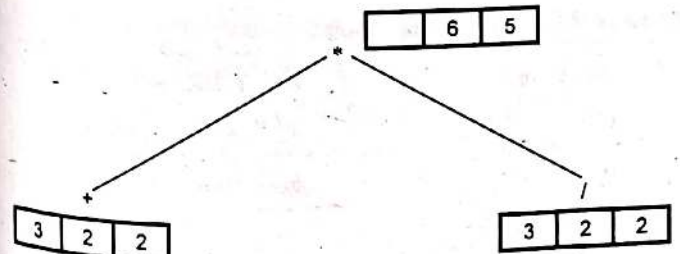
Case 2 : Compute using match for instruction $R_0 := R_0 + R_1$. By considering left subtree with one register in R_0 and right subtree with two registers in R_1 we get,



Case 3 : Compute using match for instruction $R_0 := R_0 + M$. Assume right subtree zin memory and left subtree with two registers plus one for instruction we get,

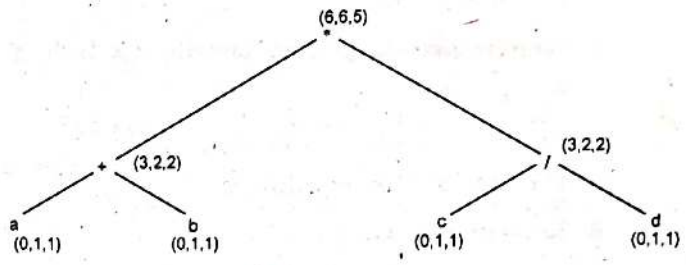


From above three cases, case 1 and case 2 gives optimum cost for c[2]. That means we can consider the evaluation order as mentioned in case 1 or from case 2. And cost of two registers available is 5.



Let us compute c[0].

The optimum cost with all registers available is 5. [As we have calculated in case 1 or case2]. We want c[0] means minimum cost of computing root into memory. We will then use $M = R_i$. Hence to store the value into memory location form register we will add 1 to 5. Hence c[0] = 6. Finally the syntax tree with cost vector defined will be,



By traversing this tree we can obtain the code as,

R_0 :=a or	R_0 := c
R_0 :=R_0 + b	R_0 := R_0/d
R_1 :=c	R_1 := a
R_1 :=R_1/d	R_1 := R_1 + b
R_1 :=R_1*R_0	R_0 := R_1 + R_0

Multiple Choice Questions for Mid Term Exam

Q.1 Compiler places target code at _____.

- a lower end of memory
- b upper end of memory
- c anywhere in the memory depending upon the code
- d none of the above

Q.2 Control stack in run time environment is used to manage _____.

- a data object
- b active procedures
- c target code
- d none of the above

Q.3 Recursive procedures are not supported by _____.

- a stack allocation
- b heap allocation
- c static allocation
- d code area