6.a. SIMPLE CODE GENERATOR ALGORITHM

- **Explain code generation phase with simple code generation Algorithm.[Nov/Dec 2014]**
- **Explain in detail about the simple code generator. [Nov/Dec 2011]**
- **Explain code Generation Algorithm in detail.[May/Jun 2012]**
- **Write note on simple code generator. [May /June 2016]**

A SIMPLE CODE GENERATOR

- A code generator generates target code for a sequence of three- address statements and effectively uses registers to store operands of thestatements.
- For example: consider the three-addressstatement **a := b+c** It can have the following sequence of codes:

```
ADD Rj, Ri   Cost = 1 // if Ri contains b and Rj contains c

             (or)

ADD c, Ri    Cost = 2          // if c is in a memory location

             (or)
```

MOV c, Rj Cost = 3 // move c from memory to Rj and add

ADD Rj, Ri

Register and Address Descriptors:

- A register descriptor is used to keep track of what is currently in each registers. The register descriptors show that initially all the registers are empty.

- An address descriptor stores the location where the current value of the name can be found at run time.

A code-generation algorithm:

The algorithm takes as input a sequence of three -address statements constituting a basic block. For each three-address statement of the form x : = y op z, perform the following actions:

1. Invoke a function *getreg* to determine the location L where the result of the computation y op z should be stored.
2. Consult the address descriptor for y to determine y', the current location of y. Prefer the register for y' if the value of y is currently both in memory and a register. If the value of y is not already in L, generate the instruction **MOV y" , L** to place a copy of y in L.

3. Generate the instruction **OP z" , L** where z' is a current location of z. Prefer a register to a memory location if z is in both. Update the address descriptor of x to indicate that x is in location L. If x is in L, update its descriptor and remove x from all other descriptors.
4. If the current values of y or z have no next uses, are not live on exit from the block, and are in registers, alter the register descriptor to indicate that, after execution of x : = y op z , those registers will no longer contain y or z.

Generating Code for Assignment Statements:

- The assignment d : = (a-b) + (a-c) + (a-c) might be translated into thefollowing three-address code sequence:

t : = a - b u : = a - c v : = t + u d : = v + u
with d live at the end.

Code sequence for the example is:

Statements	Code Generated	Register descriptor	Address descriptor
		Register empty	
t : = a - b	MOV a, R0 SUB b, R0	R0 contains t	t in R0
u : = a - c	MOV a , R1 SUB c , R1	R0 contains t R1 contains u	t in R0 u in R1
v : =t + u	ADD R1, R0	R0 contains v R1 contains u	u in R1 v in R0
d : = v + u	ADD R1, R0 MOV R0, d	R0 contains d	d in R0 d in R0 and memory

Generating Code for Indexed Assignments

The table shows the code sequences generated for the indexed assignment statements **a : = b [i]** and **a [i] : = b**

Statements	Code Generated	Cost
a : = b[i]	MOV b(Ri), R	2
a[i] : = b	MOV b, a(Ri)	3

Generating Code for Pointer Assignments

The table shows the code sequences generated for the pointer assignments **a : = *p** and ***p : = a**

Generating Code for Conditional Statements

Statement	Code
if x < y goto z	CMP x, y CJ< z /* jump to z if condition code is negative */
x : = y +z if x <0 goto z	MOV y, R0 ADD z, R0 MOV R0,x CJ< z

Statements	Code Generated	Cost
a : = *p	MOV *Rp, a	2
*p : = a	MOV a, *Rp	2