Code generation

Will out a assembly or machine code from quadruples.

Store every computation until block is exited

some optimization can be achieved in this phase also this level

Code generation involves examination of a large no of cases, e.g,

for the computation of A=B op C

if B and C are in registers Ri and Rj, code could be

ADD Ri Rj , with A in Rj

this is possible only if C is not *live*

Further, if B is in memory and C is in register Rj, then

MOV B Ri Rj

ADD Ri Rj

Again C should not be live

Problems with code generation

- What type of code should we generate
- What should be the order of computation
- Which registers are used, and
- How to use registers

Code generation (through function GETREG)

Register descriptor(RD)

To keep track of what is currently in each register.

It will be consulted when a new register is required

Address descriptor(AD)

To keep track of the location(s) where the current value of the name can be found(a name can be in register or in memory location).

It is consulted every time the location(address) of a name is required

Algorithm

For every expression A=BopC

1. Invoke the function GETREG( ) to determine the location L where the computation BopC should be performed (L could be a register or memory location)
2. Consult AD to find the current location B` of B . If it is in register as well as in memory location consult register value of B
3. Generate code **move B` L**
4. Consult AD to find the current location C` of C
5. Generate code **op C` L**
6. Update the address descriptor for A (as A is in L), Also update RD if L is a register
7. Further, Alter the RD to indicate that no registers have the values of B or C, if they are not live after the exit from the block.

Getting a suitable register -Function GETREG( )

Returns the location L to hold the value A

STEPS

1. If the name B is in a register and it is not Live and has no next use after the execution of A=B op C, return the register of B for L
2. Update the AD to indicate that B is no longer in L
3. Find an empty register for L if step (1) fails
4. If (3) fails, and if A has a next use in the block/live, choose an occupied register by moving its contents in a memory location M, ie, generate

   MOV R, M

   An occupied register could be one whose

   1. Data is referenced furthest in the future, or

   2. Value is also in memory
5. If A has no next use in the block, and all above steps fail, choose the memory location of A as L

CODE GENERATION THROUGH DAG

By rearranging the nodes in a DAG (rearranging the order of computation) code generation can optimized

Algorithm (rearranging the DAG order)

```
(Gives the order in reverse)
  While unlisted interior nodes remain do (initially all nodes unlisted)
    begin
        Select an unlisted node n all of whose parents have been listed;
        list n;
         While the left most child m of n has no unlisted presents and is not a leaf do
          begin
                  list m;
                   n=m
           end
end
```

Algorithm [for labeling nodes of tree]

Post order traversal of nodes

```
If n is a leaf then
    If n is leftmost child  Label(n) = 1
    else Label(n) = 0
Else
   begin
        let $n_{1,}$, $n_2$ ….., $n_k$ be the children of node n ordered by Label
        so, Label($n_1$) >= Label($n_2$) >= … >= Label($n_k$) ;
            Label(n) = $\underset{1<=i<=k}{\text{MAX}}$ { Label($n_i$ ) +i +1}
```