27/3/24

Unit - IV

Code generation

Issues in design of code generator:-

1. Input to the code generator
2. Target programs
3. Memory management
4. Instruction selection
5. Register allocation
6. Choice of evaluation order
7. Approaches to code generation.

Input to the code generator:-

* Here the input is intermediate go code. the intermediate code is may be in any forms of such as three address code, quadruples, prix notation or else it may be represented using graphical representation such as syntax trees or DAG.

* The front end of compiler necessary type checking and type conversion needs to be done.

* The code generation phase requires the complete error free intermediate code has input.

Target programs:-

* 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 fixed memory location and then can be executed immediately.

* The benefit of such target code is that small programs can be quickly compiled.

* eg: WATFIV and PL/C are the compilers it produce the absolute code as output.

* The advantage of producing assembly code has output make the code generation process easier.

* If the target code cannot handle the relocation automatically then the compiler must provide explicit relocation information to the loader in order to link the seperately compiled subroutine segments.

Memory Management:-

* Both the front end and code generator performs the task of mapping the name in the source program to address the data object in runtime memory.

* The type in the declaration statement determines the amount of storage (memory) needed to store the declared identifier.

* eg: For incidence of a reference 'goto s' is encounted in three address code then in appropriate jump instruction can be generated by computing the memory address for label s.

Instruction selection:-

* Important factor for the code generator. the selection of instruction depends upon the instruction set of target machine.

* Two important factor for instruction selection are
  1. The speed of instruction.
  2. machine idioms

* For each type of three address code, the code skeleton can be prepare which ultimately gives the target code for the corresponding code construct

* eg: x = a+b => MOV a, R0 (load a to register R0)

MOV b, R1
ADD R1, R2
STOR x, R2

ADD b,R0 (Addition of b to R0)

MOV R0,x (stores the content of register R0 to x)

* In the above example the code is generated line by line which a line by line code generation process generates the poor code because the redundancy is achieved by subsequent lines and those redundancies cannot be consider in the process of line by line code generation.

* eg: $x = y + z$
$a = x + t$

* The code for above statement can be generated as follows:

```
MOV y,R0
ADD z,R0
MOV R0,x
MOV x,R0
ADD t,R0
MOV R0,a
```

* In this code is a poor code because MOV R0,a is not used and statement MOV a,R0 is redundant hence the efficient code can be:

```
MOV y,R0
ADD z,R0
ADD t,R0
MOV R0,a
```

* The quality of generated code is decided by its speed and size. line by line translation of three address code into target code leads to correct code but it can generate unacceptable or non efficient target code.

Register allocation:-

* There are two important activities is done while using registers,

1. Register allocation - During register allocation select appropriate set of variables that will reside in register
2. Register assignment: During register assignment pick up the specific registers in which corresponding variables will reside. Obtain the minimum optimal assignment of registers to variable is difficult - memory

* eg: IBM systems, integer multiplication requires register pairs

* consider the three address code $t_1 = a + b$
$t_1 = t_1 * c$
$t_1 = t_1 / d$

The efficient machine code will be

```
MOV a,R0
ADD b,R0
MUL c,R0
DIV d,R0
MOV R0,t
```

Choice of evaluation orders:

* some order requires less number of register to hold the intermediate result then the others.

* picking up the best order is one of the difficulty in code generator mostly we can avoide them problem by reversing the order in which the three address code is generated by semantic action.

Approaches to code generation:-

* whereas algorithms are used to produce the correct code.

DAG (Directed Acyclic graph):-

* DAG is (Directed Acyclic graph), DAG is used to apply transformation on the basic block.

Applications of DAG:-

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

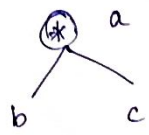
---

1. Construct the DAG for the following block:

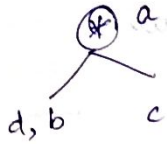
$a := b * c$
$d := b$
$e := d * c$
$b := e$
$f := b + c$
$g := f + d$

Solution:-

step 1:



step 2:

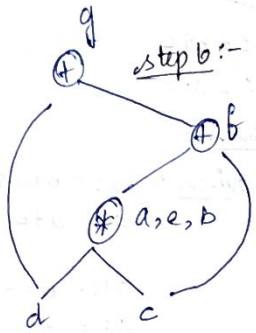


step 3:

step 4:

step 5:

step 6:-



Code generator:-

1. Generate the code for the following statements for target machine.

a) $x := x + 1$

Solution:-

Given: $x := x + 1$

$t_1 := x + 1$

$x := t_1$

The code will be,

```
MOV x, R0
ADD #1, R0
STOR R0, x.
```

2. b) $x := a + b + c$

Solution:-

Given:- $x := a + b + c$

$t_1 := b + c$

$t_2 := a + t_1$

$x := t_2$

The code will be,

```
MOV b, R0
ADD c, R0
ADD a, R0
STOR R0, x.
```

3. i) a := b+c
d := a+e

Solution :-

Given :- a := b+c
d := a+e

t1 := b+c
a := t1
t2 := a+c
d := t2

The code will be, MOV b,R0
ADD c,R0
MOV R0,a
MOV a,R0
ADD e,R0
MOV R0,d

MOV R0,a and MOV a,R0 the instructions will be redundant so the final code is,

MOV b,R0
ADD c,R0
ADD e,R0
MOV R0,d

ii. x := a/(b-c) - d*(e+f)

Solution :

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

t1 := b-c
t2 := a/t1
t3 := t2-d
t4 := e+f
t5 := t3*t4
x := t5

The code will be,

MOV b,R0
SUB c,R0
MOV a,R1
DIV R0,R1
SUB d,R1
MOV f,R2
ADD e,R2
MUL R1,R2
MOV R2, x

v) W := (A+B) + (A+C) + (A+C)

Solution :-

Given :-

W := (A+B)+(A+C)+(A+C)

t1 := A+B

t2 := A+C

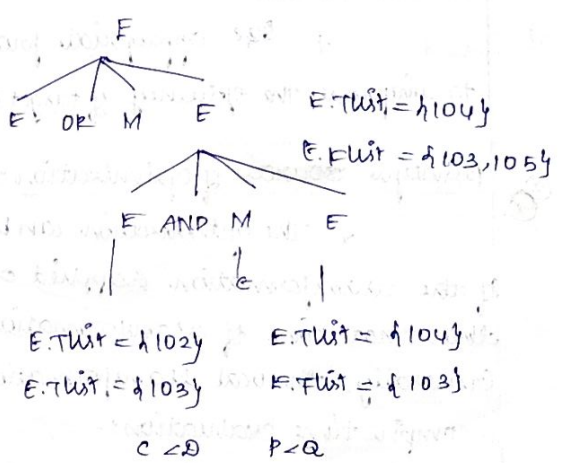
t3 := t1+t2

Backpatching :-

i. A<B OR C<D AND P<Q and show annotated parse tree.

Solution :-

100 : if A<B goto ___
101 : goto ___
102 : if C<D goto ___
103 : goto ___
104 : if P<Q goto ___
105 : goto ___



backpatching functions