Code generation using DAG

Labelling algorithm

- Simple algorithm to determine optimal order of evaluation of statements in a basic block.
- Optimal order = shortest instruction sequence.
- Algorithm divided into two parts:-

Part I:-

Label each node of a tree in bottom up order with number denoting no of registers require to evaluate the tree without intermediate storage.

Part II:-

Tree traversal algorithm whose order governed by computed node labels. Optimal output code is generated during tree traversal.

Labelling algorithm

Labelling algorithm

{

If n is the leaf node then

if n is the leftmost child of its parents then

Label(n) =1

else

Label(n) =0

else

Label(n) = max[Label (ni) + (i-1)] // for i=1 to k

// where n1, n2,……nk are children of n.

If L1==L2 Label(n) L1+1 if L1# L2 Lebel(n) =Max(L1, L2)

Labelling algorithm example -1

t1= a+b

t2= c+d

t3=e-t2

t4=t1-t3

Labelling algorithm example - 2

Consider the expression

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

*t1= c*d*

t2=b-t1

t3=f+g

t4=e/T3

a=t2+t4

Code generation algorithm

1. The code generation algorithm is represented as a function *gencode*(*n*), which produces code to evaluate the node labeled *n*.

2. Register allocation is done from a stack of register names *rstack* , initially containing r_0, r_1, . . . , r_k (with r_0 on top of the stack).

3. *gencode*(*n*) evaluates *n* in the register on the top of the stack.

4. Temporary allocation is done from a stack of temporary names *tstack*, initially containing t_0, t_1, . . . , t_k (with t_0 on top of the stack).

5. *swap*(*rstack*) swaps the top two registers on the stack.

Code generation from labelled tree

Procedure gencode(n)

{

CASE 0:

If n is a leaf node and the leftmost child of its parents then generate instruction

move name, RSTACK[top]

CASE 1:

If n is an interior node with children n1 and n2 then

if label(n2)=0 then

{

Let name be the operand represented by n2;

gencode(n1)

generate op name, RSTACK[top]

}

Code generation from labelled tree

CASE 2:

If n is an interior node with children n1 and n2 then

if label(n2)> label(n1) and label(n1)< r then

{

Swap top wo registers of RSTACK gencode(n2)

R=pop(RSTACK)

gencode(n2)

generate OP, R, RSTACK[top]

PUSH(R, RSTACK)

SWAP top two registers of RSTACK

}

Code generation from labelled tree

CASE 3:

If n is an interior node with children n1 and n2 then

label(n2)<= label(n1) and label(n1)<r and label(n2)> r then

{

gencode(n2)

T= pop(TSTACK)

generate MOV, RSTACK[top], T

gencode(n1)

generate OP RSTACK[top]

PUSH(T, TSTACK)

}

Code generation from labelled tree

CASE 4:

If n is an interior node with children n1 and n2 then

label(n2)<= label(n1) and label(n2)< r then

```
{
    gencode(n1)
    R= pop(RSTACK)
    gencode(n2)
    Generate op, RSTACK[top] , R
    PUSH (R, RSTACK)
}
```

Call to *gencode()*	Condition satisfied and case	Applied procedure	RSTACK contains top two registers	Code generation

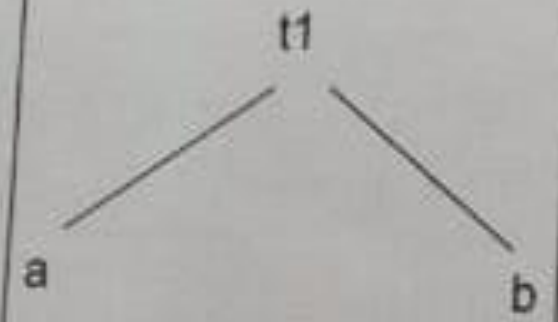
gencode(t4)	t4
	t1 t3
	(1) (2)
	n1=1, n2=2
	n2>n1, n1<r
	→ CASE2
		1) Swap top two registers
		2) Call *gencode (t3)*
		12) R = POP R1
		13) Call *gencode(t1)*
		17) Generate SUB R1,R0
		18) PUSH R1
		19) Swap top two registers
			R1,R0
			R1,R0
			R0
			R0
			
			R1,R0
			R0,R1
				MOV c, R1
				MOV c, R0
				ADD D,
				R0
				SUB
				R0,R1
				MOV a, R0
				ADD b, R0
				SUB
				R1,R0

gencode(t3)	t3
	e t2
	(1) (1)
	n1=1, n2=1
	n2<=n1, n2<r
	→ CASE3
		3) Call *gencode (e)*
		5) R=POP R1
		6) Call *gencode(t2)*
		10) Generate SUB R0,R1
		11) PUSH R1
			R1,R0
			R0
			
			R1,R0
				MOV e, R1
				MOV c, R0
				ADD D,
				R0
				SUB
				RO,R1

gencode(e)	Leaf node → Case 0	4) MOV e,R1	R1,R0	MOV e, R1
gencode(t2)	t2 c d (1) (0) n1=1, n2=0 n2=0 → CASE1	7) Call *gencode (c)* 9) Generate ADD D, R0	R0	MOV c, R0 ADD D, R0

(Cont'd)

Table 7.9 (Continued)

Call to *gencode()*	Condition satisfied and case	Applied procedure	RSTACK contains top two registers	Code generation
gencode(c)	Leaf node → Case 0	8) Generate MOV c, R0	R0	MOV c, R0
gencode(t1)	 n1=1, n2=0 n2=0 → CASE1	14) Call *gencode (a)* 16) Generate ADD b, R0	R0	MOV a, R0 ADD b, R0
gencode(a)	Leaf node → Case 0	15) Generate MOV a, R0	R0	MOV a, R0

Generated code

```
MOV e, R1
MOV c, Ro
ADD d, R0
SUB R0, R1
MOV a, R0
ADD b, R0
SUB R1, Ro
```