

Tutorial 4: Solutions

CSC 467 Compilers and Interpreters
Fall Semester, 2005

1. Since we want to use as few registers as possible, the instructions will look like:

op R_i, R_i, R_j

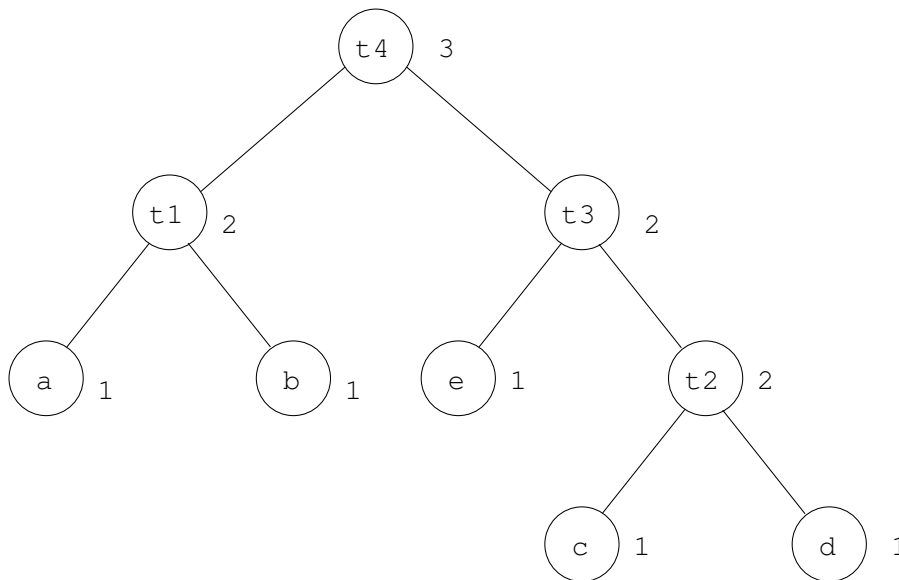
First, we must modify the labeling algorithm. It cannot be the same as before because now we keep all operands in registers and we need a *load* for each one (therefore, all leaves should have label 1).

```

if  $n$  is a leaf then  $label(n) := 1$ 
else begin /*  $n$  is an interior node */
    let  $n_1, n_2$  be  $n$ 's left and right child respectively
    if  $label(n_1) = label(n_2)$  then  $label(n) = label(n_1) + 1$ 
    else  $label(n) = \max\{label(n_1), label(n_2)\}$ 
end

```

The labeled tree will be:



```

procedure gencode(n);
begin
if n is a leaf representing operand name then
    print 'MOV' || name || ',' || top(rstack)
else if n is an interior node with left child n1 and right child n2 then
    if  $1 \leq \text{label}(n_1) < \text{label}(n_2)$  and  $\text{label}(n_1) < r$  then begin
        swap(rstack);
        gencode(n2);
        R := pop(rstack);
        gencode(n1);
        print op || top(rstack) || ',' || top(rstack) || R;
        push(rstack, R);
        swap(rstack);
    end
    else if  $1 \leq \text{label}(n_2) \leq \text{label}(n_1)$  and  $\text{label}(n_2) < r$  then begin
        gencode(n1);
        R := pop(rstack);
        gencode(n2);
        print op || R || ',' || R || ',' || top(rstack);
        push(rstack, R);
    end
    else begin /* both labels geq r */
        gencode(n2);
        T := pop(tstack);
        print 'MOV' || top(rstack) || ',' || T;
        gencode(n1);
        R := pop(rstack);
        print 'MOV' || T || ',' || top(rstack);
        print op || R || ',' || R || ',' || top(rstack);
        push(rstack, R);
        push(tstack, T);
    end
end

```

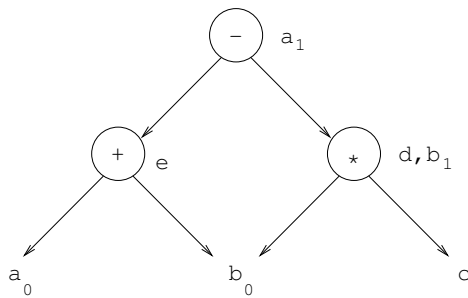
The code generated for the labeled tree is shown on the next page. Also shown are the sequence of calls to *gencode* and the contents of *rstack* at the time of each call (in brackets, top at left end).

```

gencode(t4)    [R0R1R2]
  gencode(t1)  [R0R1R2]
    gencode(a)  [R0R1R2]
      MOV a, R0
    gencode(b)  [R1R2]
      MOV b, R1
      ADD R0, R0, R1
  gencode(t3)  [R1R2]
    gencode(t2) [R2R1]
      gencode(c) [R2R1]
        MOV c, R2
      gencode(d) [R1]
        MOV d, R1
        ADD R2, R2, R1
    gencode(e)  [R1]
      MOV e, R1
      SUB R1, R1, R2
  SUB R0, R0, R1

```

2.



3. Let's number the statements:

- (1) $d := b * c$
- (2) $e := a + b$
- (3) $b := b * c$
- (4) $a := e - d$

- (a) Since a is live at the end, (1) and (2) have to go before (4).
Similarly, (3) must go after (1) and (2).

Therefore, the valid evaluation orders are:

- (1)(2)(3)(4)
- (1)(2)(4)(3)
- (2)(1)(3)(4)
- (2)(1)(4)(3)

Another solution suggested by one student says that since d is not alive, one can eliminate (1) all together and replace d by b (since they compute the same value) in (4), call

it (4'). This also gives us one legal evaluation order:

(2)(3)(4')

- (b) Since (a) is live at the end, (1) and (2) have to go before (4).
Statement (3) can be dropped.

Therefore, the valid evaluation orders are:

(1)(2)(4)

(2)(1)(4)