Assignment 1:
Instruction Selection and Register Allocation

15-411: Compiler Design
David McWherter (cache@cs) and Noam Zeilberger (noam@cs)

Due: Tuesday, September 11, 2007 (1:30 pm)

Problem 1

[20 points]

Consider the arithmetic and memory instructions in Figure 9.1 in the textbook, and the IR tree depicted in Figure 9.2.

Assume that all instructions take 2 cycles.

(a) Using the patterns in Figure 9.1, trace the dynamic programming instruction selection algorithm when applied to the IR tree in Figure 9.2. Assume that all instructions take 2 cycles, and that TEMP nodes are available in registers.

We expect your trace as a sequence of lines of the following format:

$$NodeNum\text{:} \quad Pattern_1 = Cost_1, \ldots, Pattern_i = Cost_i$$

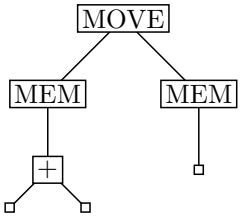
Enumerate each node of the IR tree in postorder (FP is 1, CONST(a) is 2, + is 3, MEM is 4, etc.). Each $Cost_i$ should reflect the total cost (in cycles) of the pattern and all its children. Only present patterns which match at the given node. For example, the first three lines should read:

Node 1: (-)=0
Node 2: ADDI(0,CONST)=2
Node 3: ADDI(t,CONST)=2, ADD(t,t)=4
...

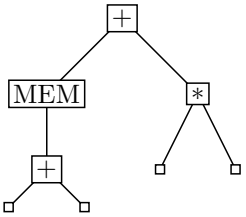
(b) What parts of the trace in part (a) change if the MOVEM instruction takes only 1 cycle? What if it takes 4 cycles?

(c) Assume we add two new instructions and their associated tree patterns:

```
BIGMEM  M[r1 + r2] = M[r3]
```



```
INDEXOP  r1 = M[r2+r3] + r4*r5
```



Assume BIGMEM takes 2 cycles and INDEXOP takes 1 cycle.

Clearly, use of BIGMEM will save two cycles in certain situations, such as our IR tree. As such, we should add it to the set of patterns used by Maximal Munch. Assume we do so.

Clearly, the use of INDEXOP can save three cycles. As such we should also add it to the patterns used by Maximal Munch. Using INDEXOP in our tree is also clearly preferable to using BIGMEM.

Will Maximal Munch ever emit the INDEXOP instruction instead of BIGMEM? If so, illustrate the final tiling. Otherwise, how must Maximal Munch be changed to ensure INDEXOP will be emitted (without removing patterns for BIGMEM or any other instructions).

Will INDEXOP be generated by Dynamic Programming instruction selection? If not, what would need to be done to ensure it?

Problem 2

[20 points]
Consider the following L1 program, which is already in 3-address form:

```
{
 a = 3;
 b = 5;
 c = 7;
 d = b-a;
 e = a*c;
 f = d+e;
 g = e*f;
 h = b*g;
 return h;
}
```

(a) Compute the live variables at each program point.

(b) Construct the interference graph.

(c) What is the minimum number of colors needed to color this graph? Rewrite the program using this many variables.

(d) Is it possible to reduce the number of colors needed by reordering the statements?

Problem 3

[20 points]
The program in Question 2 can be considered as computing a single expression:

```
{ return 5*((3*7)*((5-3)+(3*7))); }
```

(a) Simulate the simple register allocation algorithm for trees (Appel Algorithm 11.9) on the expression to assign registers to the intermediate values. (Assume that constants are non-trivial tiles, requiring a single move instruction.) Give the resulting straight-line code.

(b) Simulate the Sethi-Ullman algorithm on the expression and give the resulting straight-line code.

(c) Compare your answers in 3a and 3b. Does one use fewer registers? Why?

(d) One feature of the original L1 code not captured by the expression tree is the reuse of intermediate values. For example, in the original code the expression `3*7` is computed only once and stored into the variable `e`. In what kind of situation could reuse of intermediate values actually result in *slower* code?