Assignment 1
Instruction Selection and Register Allocation

15-411: Compiler Design
Nathan Snyder (npsnyder@andrew) and Anand Subramanian(asubrama@andrew)

Due: Tuesday, September 7, 2010 (1:30 pm)

Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own.

You may hand in a handwritten solution or a printout of a typeset solution at the beginning of lecture on Tuesday, September 7. Please read the late policy for written assignments on the course web page. If you decide not to typeset your answers, make sure the text and pictures are legible and clear.

Problem 1 (30 points)

(a) Consecutive statements in a program can be represented in an AST by a `SEQ` node that has two statements (possibly other `SEQ`s) as children. For example, the program

```
x = 5 + 3;
return x;
```

would be represented in an AST as

```
SEQ(ASSIGN(VAR(x), PLUS(CONST(5), CONST(3))), RETURN(VAR(x)))
```

Using this type of AST, write down (either as in the example or by drawing a real tree) the AST for the following program

```
x = 4 - (2 + 5) * 8;
y = (x + 1) * 3;
return x - y;
```

(b) When we expand the capabilities of a programming language, we also need to extend the AST to represent the new features. Write down the AST for the following program, choosing a reasonable AST representation of the "if" and "==" constructs.

```
x = 2;
if (x == 0)
    return 4;
else
{
    x = 5;
    return x;
}
```

(c) Now you will perform instruction selection on the AST you created in part (a) into three-operand assembly language by using the patterns in the table below. As a sample, the example AST from part (a) would be translated to

```
t0 <- 5
t1 <- 3
t2 <- t0 + t1
x  <- t2
t3 <- x
return t3
```

We aren't performing register allocation yet (that's for problem 2), so we will continue to refer to variables by their names and generate new temp variables $(t_0, ..., t_n)$ as necessary. S_1 and S_2 refer to the first and second subtrees of an AST node. $S_n$instrs refers to the instructions generated for S_n, and $S_n$temp refers to a new temp variable created to hold the result of S_n in cases where S_n has one. Lastly, r is the temp where the result of an expression should be placed.

Pattern	Assembly
CONST(n)	r <- n
VAR(x)	r <- x
PLUS(S_1, S_2)	S_1instrs, S_2instrs, r <- S_1temp + S_2temp
MINUS(S_1, S_2)	S_1instrs, S_2instrs, r <- S_1temp - S_2temp
TIMES(S_1, S_2)	S_1instrs, S_2instrs, r <- S_1temp * S_2temp
ASSIGN(VAR(x), S_2)	S_2instrs, x <- S_2result
RETURN(S_1)	S_1instrs, return S_1result
SEQ(S_1, S_2)	S_1instrs, S_2instrs

(d) Now perform instruction selection on the AST you created in part (b). To accomplish this, we will need to introduce "cmpeq r x y", label l", "jmp l", and "jmpnzero l x" into our assembly language, where l is always a number that identifies a label. "cmpeq r x y" assigns 1 to r if the values of x and y are equal and 0 to r otherwise. In the case of jmpnzero, control flow jumps to label l if the value of x is not 0. You will need to come up with your own patterns for generating instructions for "if" and "==" , and you must write down these patterns in addition to the specific program.

Problem 2 (30 points)

In this question you will perform the register allocation algorithm discussed in class on a small (and rather bizarre) assembly program. The registers to be used are r_1,...,r_n so for the purposes of this question you have as many registers as you need (though the algorithm will still be trying to use as few as possible). The language used is the assembly from problem 1 with an additional division instruction, used as in "$t_i \leftarrow t_j$ / t_k". As in x86 assembly, the division instruction has some special conditions associated with it; specifically, t_j must be assigned to register r_0 and t_k must be assigned to r_1. A final consideration when allocating registers is that in the instruction "return t_i", t_i must always be assigned to register r_0.

```
t0 <- 2
t3 <- 4
t4 <- t3 - 2
t5 <- 6
t2 <- t5 / t4
t1 <- t2 * t3
label 1
t0 <- t0 * t1
t1 <- t1 - t2
t6 <- 9
jmpnzero 1 t1
return t0
```

(a) Compute the live variables at each instruction in the above program.

(b) Construct the interference graph for the program. If you don't want to actually draw a graph, you can just list the variables that each variable interferes with. You should also state whether the graph is chordal.

(c) What problem does the current program have for allocating registers? Give a modified version of the program that does not have this problem (you're probably yearning to make the whole program less hideous, but try to make the smallest change that allows register allocation).

(d) Use the chordal graph coloring algorithm discussed in class to allocate registers for all the temps in the modified program. If you did part (c) correctly then your liveness analysis and interference graph should still be usable with slight modification.