Assignment 5: Optimizations

15-411: Compiler Design
Josiah Boning (jboning@andrew) and Ryan Pearl (rpearl@andrew)

Due: Friday, December 9, 2011

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 in person at Josiah's office hour on Friday, or send a typeset solution by email to 15411@symbolaris.com. If you decide not to typeset your answers, make sure the text and pictures are legible and clear.

Problem 1: Jump Tables (20 points)

The switch statement is a language construct found in C but not C0. One way of compiling a switch statement is to treat it as a cascade of if-else statements (with some extra considerations made for letting control flow fall through to the next case if the "break" statement is missing from a case). However, there is a more efficient implementation that can often be used: a jump table. In a jump table, the value being switched on is used as an index into a table of jump statements that each lead to the instructions for a particular case.

(a) Write down an assembly representation of the switch statement below that uses a jump table. We are mostly interested in seeing the concept of a jump table, so feel free to use "abstract" assembly (with variable names, for example) so long as all of your instructions have a clear counterpart on a real processor.

```
unsigned int x, y, z;
// Other code
switch (x) {
case 0: y = 1; break;
case 1: y = 8; z = 9; break;
case 2: y = 3; break;
default: y = 0;
}
// More code
```

(b) What condition does a switch statement need to meet for a jump table to be used?

(c) Describe how a compiler could transform a switch statement which doesnt meet the necessary condition into one that does.

(d) When would applying your transformation from part (c) not be a good idea?

Problem 2: Instruction Scheduling (20 points)

One of the techniques used in processor architecture to improve performance is *pipelining*. A pipelined processor divides instruction execution into stages, allowing several instructions to be executed at the same time.

For this problem, we will assume that the processor begin executing one instruction per cycle, and each instruction takes three cycles to complete.[1] We will also assume that the processor reads any registers required by an instruction at the beginning of the first cycle, and that the results of an instruction are visible registers at the end of the third cycle. If an instruction would be pipelined, but depends on the value of a register which is currently being computed, the processor will *stall* until the value is ready.

For example, suppose we want to run the following small assembly snippet:

```
r0 <- r0 * 2
r1 <- r1 + 4
r2 <- r1 / 3
```

The instructions passing through the processor's pipeline would look as follows:

cycle	**stage 1**	**stage 2**	**stage 3**
0	`r0 <- r0 * 2`		
1	`r1 <- r1 + 4`	`r0 <- r0 * 2`	
2		`r1 <- r1 + 4`	`r0 <- r0 * 2`
3			`r1 <- r1 + 4`
4	`r2 <- r1 / 3`		
5		`r2 <- r1 / 3`	
6			`r2 <- r1 / 3`

Note that the processor could not begin executing the third instruction until the second instruction was complete. Often, a compiler can mitigate this delay by (carefully) changing the order of the instructions it emits, called *instruction scheduling*. In this case, swapping the first and second lines of the assembly allows the processor to execute the instructions in one fewer cycle without changing the result of the program:

cycle	**stage 1**	**stage 2**	**stage 3**
0	`r1 <- r1 + 4`		
1	`r0 <- r0 * 2`	`r1 <- r1 + 4`	
2		`r0 <- r0 * 2`	`r1 <- r1 + 4`
3	`r2 <- r1 / 3`		`r0 <- r0 * 2`
4		`r2 <- r1 / 3`	
5			`r2 <- r1 / 3`

[1] On modern processors, this is a gross oversimplification. Superscalar execution, multiple dispatch, and other architectural features make the situation quite complicated. Out-of-order execution even to some extend supersedes the sort of instruction scheduling discussed here. See for example the Intel optimization manual: `http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf`

We will be working with the following abstract assembly:

```
r1 <- r0
r2 <- r1 + 4
r3 <- r2 - 5
r4 <- 5
r5 <- r3 * 2
r6 <- 8
r7 <- r6 / r1
r6 <- 10
r8 <- r6 % r1
```

(a) On our hypothetical processor, how many cycles does it take to execute the code as given? You should show *some* supporting work rather than simply giving a final answer, but it does not need to be in the table format we used above (though this may be helpful to you!).

(b) Write down all dependencies between instructions. You may wish to refer to the November 11 lecture notes on data dependencies. (Arrows between lines of code is an acceptable format for your answer.)

(c) Give a version of the program which schedules instructions to reduce execution time, making sure not to change the result of the program. How many cycles does your version of the program take to run? Again, provide some support for this number.

(d) Pipelining speeds up execution by taking advantage of parallelism in the program. What is the theoretical limit of this speedup? To answer this, imagine that rather than having one of each pipeline stage, our processor can begin executing arbitrarily many instructions in the same cycle, and find the minimum number of cycles required to execute the program.[2]

Problem 3: Inlining (20 points)

Inlining is the technique of replacing a call to some function *foo* by inserting the body of *foo* into the calling function.

(a) What are the advantages and disadvantages of inlining?

(b) Given your answer to part (a), what criteria would you use to decide which call sites to inline in a program? Your criteria should be precise enough that a compiler (as opposed to a programmer) would be able to check them.

(c) An important consideration for implementing optimizations is when to apply them in the compilation process. If you were implementing inlining, would you apply it at the AST, the IR, or the post-code-generation instructions level? Explain your decision, and also list any advantages of the other options that you considered.

[2]One tool designed for a very similar sort of problem is the Gantt Chart: `http://en.wikipedia.org/wiki/Gantt_chart`