

# Assignment 1

## Instruction Selection and Register Allocation

15-411: Compiler Design

Rob Arnold (rdarnold@andrew) and Eugene Marinelli (emarinel@andrew)

Due: Tuesday, September 9, 2008 (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 9. 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.

In this assignment you will generate machine code to evaluate a simple arithmetic expression, carrying out some non-trivial compilation steps. The expression is

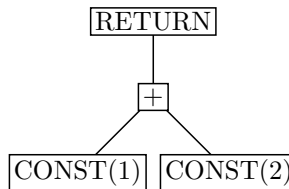
```
{ return (7 + 3 * (8 - 2)) - 4 * (9 + 5); }
```

### Problem 1 (20 points)

- (a) Construct an abstract syntax tree (AST) corresponding to the expression. For example, the AST for the expression

```
{ return 1 + 2; }
```


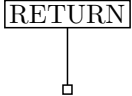
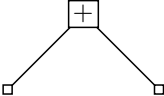
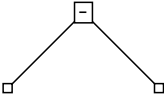
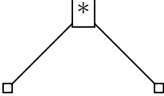
would be represented as<sup>1</sup>:



- (b) Translate the AST from problem 1(a) into linear three-address form by applying maximal munch using the tiles in the table below. Temporaries should be called `t0`, `t1`, ..., `tn`.

---

<sup>1</sup>The L<sup>A</sup>T<sub>E</sub>X graph library used to generate these trees can be found at <http://www.cs.umu.se/~drewes/graphs/>. The library apparently doesn't work with `pdflatex`. You can work around this using `latex` and `dvipdf`, for example.

Tile	IR
	<code>ti &lt;- n;</code>
	<code>return ti;</code>
	<code>ti &lt;- tj + tk;</code>
	<code>ti &lt;- tj - tk;</code>
	<code>ti &lt;- tj * tk;</code>

For example, the AST from part (a) would be translated to

```

t0 <- 1;
t1 <- 2;
t2 <- t0 + t1;
return t2;

```

## Problem 2 (20 points)

- Compute the live variables after each statement in the program generated in problem 1b.
- Construct the interference graph. Is it chordal<sup>2</sup>?
- Reorder the statements so that the program gives the same result, but there is less interference (i.e., fewer registers are needed), or state that the instruction order is already optimal in this sense. If it is already optimal, you only need to show one answer each for parts (d) and (e).
- Construct elimination orders using maximal cardinality search for each of the two interference graphs. If the algorithm has a choice, list the lowest-numbered temporary first.
- Use greedy coloring to allocate registers for both the original program and the reordered program. Label the registers `r0`, `r1`, ..., `rn`.

---

<sup>2</sup>Refer to the lecture notes for Lecture 3 at <http://www.cs.cmu.edu/~fp/courses/15411-f08/lectures/03-regalloc.pdf>.

### Problem 3 (20 points)

- (a) Translate the program from the AST in problem 1(a) into an ABI-conformant<sup>3</sup> x86-64 assembly routine, selecting instructions and allocating the registers using an algorithm of your choosing. Describe briefly which algorithm you used. Use the general-purpose x86-64 registers<sup>4</sup> and beware the calling conventions. Assume that all arithmetic is done on signed 64-bit integers. It should be possible to assemble your functions using the GNU assembler. The function should be callable using the following C prototype:

```
extern long ass1a();
```

- (b) Handwrite assembly code using only `mov`, `addq`, `mulq`, `subq`, and `ret` instructions which uses as few instructions as possible and carries out the required operations. You should not use any laws of modular arithmetic such as commutativity, associativity, or distributivity to optimize the computation, and you should not use constant folding, that is, all operations should be performed at runtime. It should be callable as

```
extern long ass1b();
```

- (c) How many instructions appear in the systematically constructed assembly code, and how many instructions does your hand-crafted code have? Explain why you believe your code to be optimal, under the given constraints. Do you believe your code also uses the minimal number of necessary registers? Explain why or why not.

---

<sup>3</sup>Refer to <http://www.x86-64.org/documentation/abi.pdf>.

<sup>4</sup>Refer to page 7 of <http://www.cs.cmu.edu/~fp/courses/15411-f08/misc/asm64-handout.pdf>.