Assignment 1: Register Allocation and Introduction to SSA

15-411/611: Course Staff

Due Thursday, September 17, 2020 (11:59pm)

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

You should submit your assignment as a PDF on Gradescope. If the code does not work for you, or if you have any other trouble enrolling on Gradescope, please contact the course staff on piazza. Please read the late policy for written assignments on the course web page.

Problem 1 (20 points)

In this question you will perform the register allocation algorithm discussed in class on a small assembly program which computes $\log_2(6x - 2) + 1$ (in the code given, the input x is hardcoded to be 42).

```
t0 <- 42 // "input"
t1 <- 6
t2 <- t0 * t1
t3 <- 2
t4 <- t2 - t3
t5 <- 1
t6 <- 0
t7 <- 1

label .loop
t4 <- t4 >> t5
t6 <- t6 + t7
branch t4 .loop .exit

label .exit
ret t6
```

The target of your compilation will be a three-address machine with as many registers as you need (though the algorithm will still be trying to use as few as possible). The registers are named $r_0, \ldots, r_n$. The language also has a right shift instruction $d \leftarrow s_1 >> s_2$.

(a) Compute the live-out sets for 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) Use the maximum cardinality search algorithm we described in lecture, starting from `t7`, to construct a simplicial elimination ordering. Then, using this ordering, use the greedy graph coloring described in class to assign registers $r_0, \ldots, r_n$ to temps.

Now we will add a restriction to our three-address assembly language: the register r_0 *must* be used as the return register (in other words, the operand of `ret` must be `r0`). Similarly, in the shift instruction $d \leftarrow s_1$ `>>` s_2, the same register r_0 *must* be to hold s_2, the magnitude of the shift.

(d) Why does this represent a problem for our sample program? Give a slightly modified but equivalent version of the program that does not have this problem.

(e) Do the graph coloring algorithm like in 1(c) on this modified program. This time, allocate your registers in a way such that t_7 is assigned to the register with the highest possible number, and explain how you did this.

Problem 2 (10 points)

Recall that a *chordal* graph is a graph where every cycle of length 4 or larger contains a chord (an edge that connects to vertices on the cycle but is not part of the cycle).

(a) Write a program in three-address assembly that has a non-chordal interference graph and uses not more than 4 temps. Draw the interference graph.

(b) Rename the temps in your program so that you get an equivalent program that assigns every temp at most once (the resulting program is in SSA form). Draw the interference graph of the modified program. Is the graph chordal?

Problem 3: Stuck in the Middle (20 points)

We generate a *Collatz sequence* c_i, starting from some positive integer n, with the following mathematical definition:

$$a_0 = n$$

$$a_{i+1} = \begin{cases} a_i/2 & \text{if } a_i \text{ is even} \\ 3a_i + 1 & \text{otherwise} \end{cases}$$

The *stopping time* of a Collatz sequence is the smallest index i such that $a_i = 1$. It is currently not known if every Collatz sequence reaches 1 (and thereby stops). The following C0 function is intended to compute the *maximum number* in the Collatz sequence for n before it stops.

```
int collatz(int n)
//@requires n >= 1;
{
 int r = n;
 while (n > 1) {
   if (n > r) r = n;
   if (n % 2 == 0)
     n = n / 2;
   else
     n = 3*n + 1;
 }
 return r;
}
```

The following is a valid three-address abstract assembly translation:

```
collatz(n):
    r <- n
    goto .loop
.loop:
    if (n > 1) then .body else .done
.body:
    if (n > r) then .l1 else .l2
.l1:
    r <- n
    goto .l2
.l2:
    m <- n % 2
    if (m == 0) then .l3 else .l4
.l3:
    n <- n / 2
    goto .loop
.l4:
    n <- n * 3
    n <- n + 1
    goto .loop
.done:
    ret r
```

(a) Show the control flow graph of the program pictorially, carefully encapsulating each basic block. Label each basic block with the label from the abstract assembly code.

(b) Show the dominator tree of the program, again labeling each block with the label from the abstract assembly code.

(c) For each block, list the blocks in its dominance frontier.

(d) Convert the abstract assembly program into SSA, using ϕ functions.

(e) Transform your SSA code into minimal SSA.

Problem 4 (Book Prize)

This is a really hard problem. Feel free to think about this and email the instructor a proof if you think you've got a good solution. This problem will probably *not* help you with your compiler implementation or future written assignments. So it's absolutely okay to ignore it. However, the 411/611 student with the first correct solution will receive the instructor's favorite math book as a prize.

Problem

Reproduce the following definitions from lecture: perfect elimination ordering; maximum cardinality search; chordal graph. Prove the following theorem.

> Performing a maximum cardinality search on a graph G produces a perfect elimination ordering for G if and only if G is chordal.

Rules

- The first correct solution emailed to the instructor wins
- There is no time limit
- The work must be your own
- You must prove all theorems (from lecture or elsewhere) that you use
- A solution is considered correct if you can convince two or more participants in the instructor's group meeting by giving a talk