CS 415 Compilers: Problem Set 2
Due date: Friday, February 16, 11:59pm

Problem 1 – Bottom-up register allocation

Source code

```
program main;
    var a, b, c, d, e, f, g, h: integer;
    begin
      a := 1;
      b := 2;
      c := b - 4;
      d := a + b;
      e := d + 1;
      f := e - c * e;
      g := (d + e) + f;
      h := g + a;
      writeln(h)
    end.
```

Assume that the following ILOC code is passed on to the register allocator.

```
loadI 1024 => r0
loadI 1 => r1        // a := 1
loadI 2 => r2        // b := 2
subI r2, 4 => r3     // c := b - 4
add r1, r2 => r4     // d := a + b
addI r4, 1 => r5     // e := d + 1
mult r3, r5 => r6    // c * e
sub r5, r6 => r7     // f := e - (c * e)
add r4, r5 => r8     // d + e
add r8, r7 => r9     // g := (d + e) + f
add r9, r1 => r10    // h := g + a
storeAI r10 => r0, 4 // printing requires value to be in memory
outputAI r0, 4      // print @h = 4 , h is only value in memory
```

Show the ILOC code that would be generated by the bottom-up algorithm discussed in class for 1.) (MAX_LIVE - 1) = 3 and 2.) (MAX_LIVE - 2) = 2 available registers for allocation. Note: For bottom-up register allocation, you don't need any feasible registers since we can use loadAI and storeAI instructions to perform spilling of registers to and from memory.

Problem 2 – Instruction scheduling

Perform forward list scheduling for the following ILOC code:

```
a   loadI 1024  => r0
b   loadI 0  => r1
c   storeAI r1  => r0, 0
d   loadI 63  => r3
e   storeAI r3  => r0, 4
f   loadI 5  => r5
g   loadAI r0, 0  => r6
h   add r5, r6  => r7
i   storeAI r7  => r0, 8
j   loadAI r0, 8  => r3
k   loadI 9  => r10
l   sub r3, r10  => r11
m   storeAI r11  => r0, 12
n   loadAI r0, 4  => r13
o   loadI 3  => r14
p   mult r13, r14  => r15
q   storeAI r15  => r0, 16
r   loadAI r0, 16  => r3
s   loadI 7  => r18
t   mult r3, r18  => r4
u   storeAI r4  => r0, 20
v   loadAI r0, 12  => r21
w   loadAI r0, 20  => r22
x    add r21, r22  => r23
y   storeAI r23  => r0, 24
z   loadAI r0, 24  => r25
aa   storeAI r25  => r0, 28
bb   outputAI r0, 28
```

There are many possible variants of the basic forward list scheduling algorithm.

1. Show the dependence graph for the basic block. All true, anti, and output dependences needed to ensure the correct order of execution. You may omit dependences that are "covered" by other dependences in the graph.
2. Label the nodes in the dependence graph based on the longest latency-weighted path (see our class notes). Use the latencies as we discussed in class (anti-dependencies have full latency). Show the result of forward list scheduling using the longest latency-weighted path heuristic.
3. Instead of the longest latency-weighted path, use a selection heuristic that prefers nodes with the highest latency. Use the latencies as we discussed in class. Show the result of this scheduling heuristic.

4. (OPTIONAL) Come up with our own heuristic for selecting nodes (instructions) in the forward list scheduling algorithm. Show the resulting reordered instructions.