Lab 8: Code Generation and Register Allocation

In this lab, you will work on a set of code-generation and register-allocation exercises. Download and unzip the file `lab8.zip`. You'll see three sets of programs, each consists of an IR program (`exampleX.ir`), an (naive) assembly program (`exampleX-naive.s`), and a driver program (`testX.c`).

Problem 1

Consider an IR program (`example1.ir`) and its naive target code (`example1-naive.s`):

```
_f ()
(a,b,c)
{
   a = 0
L:
   b = a + 1
   c = c + b
   a = b * 2
   if a < 1000 goto L
   return c
}
```

```
    .text
         # _f () (a, b, c)
    .p2align 4,0x90
    .globl _f
_f:
    subq $40,%rsp
         #  a = 0
    movq $0,%r10
    movl %r10d,(%rsp)
         # L:
_f_L:
         #  b = a + 1
    movslq (%rsp),%r10
    movq $1,%r11
    addq %r11,%r10
    movl %r10d,4(%rsp)
         #  c = c + b
    movslq 8(%rsp),%r10
```

```
    movslq 4(%rsp),%r11
    addq %r11,%r10
    movl %r10d,8(%rsp)
         #  a = b * 2
    movslq 4(%rsp),%r10
    movq $2,%r11
    imulq %r11,%r10
    movl %r10d,(%rsp)
         #  if a < 1000 goto L
    movslq (%rsp),%r10
    movq $1000,%r11
    cmpq %r11,%r10
    jl _f_L
         #  return c
    movslq 8(%rsp),%eax
    addq $40,%rsp
    ret
         # Total inst cnt: 25
```

1. Compile and run this program:

   ```
   linux> gcc -o test1 test1.c example1-naive.s
   linux> ./test1
   ```

2. Try to improve the target code by keeping values in registers, hence reducing the number of memory loads and stores. Do this in any way you see fit. Call your new program `example1-reg.s`, and compile and run it.

3. How many loads and stores are you able to elimiate?

Problem 2

Consider the following IR program (`example2.ir`):

```
_f ()
(a,b,c)
{
   a = 2
L:
   c = 1
   b = a + 1
   a = c * b
   if a < b goto L
   return b
}
```

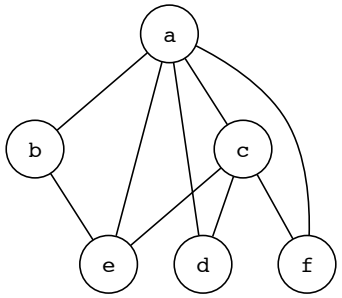
1. Find liveness information for every instruction.
2. What's the maximum number of variables that are simultaneous live at a point in the program? Base on this number, can you decide how many registers are needed?
3. Draw an interference graph, and color it. How many colors do you need? Does it match the number from the previous question?
4. Use the coloring result to write a new program, `example2-reg.s`, and compile and run it.

Problem 3

Consider the third IR program (`example3.ir`) and it's interference graph:

```
_f (a)
(b,c,d,e,f)
{
    e = 42
    b = a * 4
    goto L3
L0:
    c = a + b
    if c < 100 goto L1
    d = 10 + c
    e = d + d
    goto L2
L1:
    f = c / 10
    e = f - 40
L2:
    b = e - c
L3:
    if e > 0 goto L0

    return e
}
```



1. From the interference graph, we can see that variable `a` interferes with all other variables. Find evidence from the program that variable `a` is indeed simultaneously live with each of the other variables. (Drawing a control-flow graph may help.)
2. Follow the coloring algorithm to Color the interference graph.

Problem 4

Write an example IR program whose interference graph requires five colors. Design your program so that it can be easily extended to induce a graph requiring an arbitrarily large number of colors.