Problem Set 4: Local Optimizations
Due October 26, 2020, 5PM EDT (9PM UTC)

For Parts I, II, and III, consider the following piece of three-address code:

```
1.  READ(A)
2.  READ(B)
3.  C = A + B
4.  A = A + B
5.  B = C * D
6.  T1 = C * D
7.  T2 = T1 + C
8.  F = A + B
9.  C = F + B
10. G = A + B
11. T3 = F + B
12. T4 = C * D
13. T5 = T3 + T4
12. WRITE(T5)
```

Part I: Common Subexpression Elimination

1. Show the result of performing Common Subexpression Elimination (CSE) on the above code. Rather than generating assembly, just give new 3AC. If you're reusing the results of an expression, just generate code that looks like X = Y, where Y is the variable/temporary that held the result of the original calculation.

   Answer:

   Let's begin by computing available expressions after each statement, written in the format `[expression, temp/var where expression is stored]`. If an expression is already available, we won't add it to our set of available expressions (though you could, which can give you some more optimization opportunities).

```
1.  READ(A)
2.  READ(B)
3.  C = A + B [A + B, C]
4.  A = A + B
5.  B = C * D [C * D, B]
6.  T1 = C * D [C * D, B]
7.  T2 = T1 + C [C * D, B], [T1 + C, T2]
8.  F = A + B [C * D, B], [T1 + C, T2], [A + B, F]
```

```
9.  C = F + B [A + B, F], [F + B, C]
10. G = A + B [A + B, F], [F + B, C]
11. T3 = F + B [A + B, F], [F + B, C]
12. T4 = C * D [A + B, F], [F + B, C], [C * D, T4]
13. T5 = T3 + T4 [A + B, F], [F + B, C], [C * D, T4], [T3 + T4, T5]
12. WRITE(T5) [A + B, F], [F + B, C], [C * D, T4], [T3 + T4, T5]
```

And now we'll rewrite instructions that are recomputing available expressions:

```
1.  READ(A)
2.  READ(B)
3.  C = A + B [A + B, C]
4.  A = C
5.  B = C * D [C * D, B]
6.  T1 = B [C * D, B]
7.  T2 = T1 + C [C * D, B], [T1 + C, T2]
8.  F = A + B [C * D, B], [T1 + C, T2], [A + B, F]
9.  C = F + B [A + B, F], [F + B, C]
10. G = F [A + B, F], [F + B, C]
11. T3 = C [A + B, F], [F + B, C]
12. T4 = C * D [A + B, F], [F + B, C], [C * D, T4]
13. T5 = T3 + T4 [A + B, F], [F + B, C], [C * D, T4], [T3 + T4, T5]
12. WRITE(T5) [A + B, F], [F + B, C], [C * D, T4], [T3 + T4, T5]
```

2. Suppose F and A were aliased. How would that change the results of CSE?

   Answer: If F & A were aliased, then after line 8, `A + B` would no longer be available. While it might seem like that would then mean we couldn't rewrite line 10 anymore, the compiler can also realize that `A + B` and `F + B` are equivalent expressions. So instead, it can rewrite line 10 to `G = C`.

Part II: Liveness Analysis

1. Perform liveness analysis for the original code. Show which variables are *live out* for each statement (i.e., which variables are live immediately after each statement). Assume that this code represents the full program.

   Answer: Live sets *after* each line has executed shown on each line:

```
1.  READ(A) {A, D}
```

```
2.  READ(B) {A, B, D}
3.  C = A + B {A, B, C, D}
4.  A = A + B {A, C, D}
5.  B = C * D {A, B, C, D}
6.  T1 = C * D {A, B, C, D, T1}
7.  T2 = T1 + C {A, B, D}
8.  F = A + B {A, B, D, F}
9.  C = F + B {A, B, C, D, F}
10. G = A + B {B, C, D, F}
11. T3 = F + B {T3, C, D}
12. T4 = C * D {T3, T4}
13. T5 = T3 + T4 {T5}
12. WRITE(T5) {}
```

2. Perform liveness analysis for the code that you generated by performing CSE (Part I, problem 1). Comment on any differences between these liveness results and the liveness results in Part II, problem 2.

   Answer: Note that B is live a lot less than it used to be, because many of its uses are removed due to CSE. Also, A is not live after line 8 executes, whereas in the original program it is alive until line 10. Again, this is because CSE eliminates a use of A.

```
1.  READ(A) {A, D}
2.  READ(B) {A, B, D}
3.  C = A + B {C, D}
4.  A = C {A, C, D}
5.  B = C * D {A, B, C, D}
6.  T1 = B {A, B, C, D, T1}
7.  T2 = T1 + C {A, B, D}
8.  F = A + B {B, D, F}
9.  C = F + B {C, D, F}
10. G = F {C, D}
11. T3 = C {T3, C, D}
12. T4 = C * D  {T3, T4}
13. T5 = T3 + T4 {T5}
12. WRITE(T5) {}
```

Part III: Dead code elimination

1. What statements can be eliminated by dead code elimination for the original code?

Answer: Statement 10 is dead, statement 7 is dead, statement 6 is dead.

2. What statements can be eliminated by dead code elimination for the code post-CSE (i.e., the code you generated in Part I, problem 1)?

   Answer: No change.

Part IV: Register allocation

For the following problems, consider this code:

```
1.  READ(A)
2.  READ(B)
3.  C = A + B
4.  A = A + B
5.  B = C * D
6.  T1 = C * D
7.  T2 = T1 + C
8.  F = A + B
13. T3 = F + T2
12. WRITE(T3)
```

Assume this is the entire program, and hence that no variables are live after execution. Assume you have only three registers.

1. Perform register allocation using the greedy register allocation algorithm from lecture. If you have to allocate a register, *always allocate the lowest number register that is free.* If you have to spill a register, *always spill the lowest-numbered register you can.*

   Answer:

   Let's start by performing liveness analysis on this code:

```
1.  READ(A) {A, D}
2.  READ(B) {A, B, D}
3.  C = A + B {A, B, C, D}
4.  A = A + B {A, C, D}
5.  B = C * D {A, B, C, D}
6.  T1 = C * D {A, B, C, T1}
7.  T2 = T1 + C {A, B, T2}
8.  F = A + B {F, T2}
13. T3 = F + T2 {T3}
12. WRITE(T3) {}
```

And now we give the code generated for each 3AC instruction. After each statement, we will indicate the temporary/variable in each register, and use a * to indicate if the value is dirty.

```
# READ(A)
READ r1 #allocate r1 to a
# r1: A*

#READ(B)
READ r2 #allocate r2 to B
# r1: A*, r2: B*

#C = A + B
ADD r3, r1, r2 #allocate r3 to C
# r1: A*, r2: B*, r3: C*

#A = A + B
#free B, no write because it’s dead
ADD r1, r1, r2
# r1: A*, r2: empty, r3: C*

#B = C * D
LD R2, D #load D from memory
SW R1, A #spill r1 to stack to make room for B
MUL R1, R3, R2
# r1: B*, r2: D, r3: C*

#T1 = C * D
#free D, no write
MUL R2, R3, R2 #allocate r2 to T1
# r1: B*, r2: T1*, r3: C*

#T2 = T1 + C
#free T1, no write because it’s dead
#free C, no write because it’s dead
ADD R2, R2, R3 #allocate r2 to T2
# r1: B*, r2: T2*, r3: free

#F = A + B
LD R3, A #load A back from memory
#free A because it’s dead
```

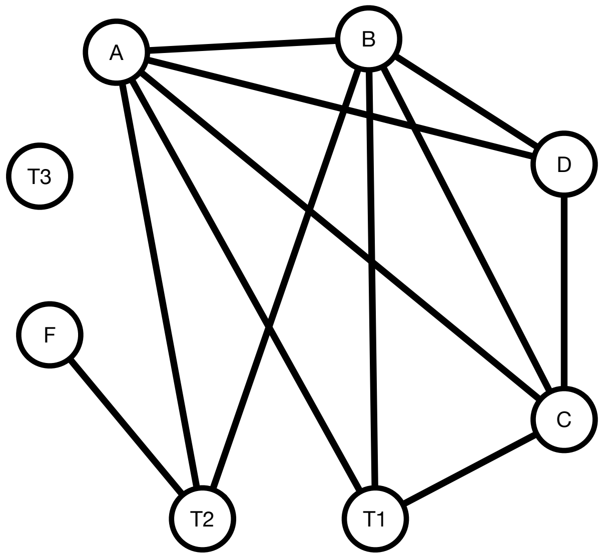
```
#free B because it's dead
ADD R1, R3, R1 #allocate r1 to F
# r1: F*, r2: T2*, r3: free

#T3 = F + T2
#free T2 because it's dead
#free F because it's dead
ADD R1, R1, R2 #allocate r1 to T3
# r1: T3*, r2: free, r3: free

#WRITE(T3)
WRITE r1
```

2. Draw the interference graph for this code.

Answer


A
B
D
T3
F
C
T2
T1