ECE 468 & 573 — Midterm 2 (Key and Rubric)

November 4, 2011

Name: _____

Purdue email: _____

Please sign the following:
I affirm that the answers given on this test are mine and mine alone. I did not receive help from any person or material (other than those explicitly allowed).

X _____

Part	Points	Score
1	10	
2	30 [+10]	
3	40	
4	20 [+10]	
Total	100 [+20]	

Part 1: Common subexpression elimination (10 pts)

For the next questions, consider the following piece of code:

```
1: A = B + C;
2: B = A + C;
3: Q = A + C;
4: A = A + C;
5: P = B + C;
```

1) Assume there is no aliasing between variables. For each statement, list which expressions are "available" *after* the statement executes (5 pts)

1 point per entry

1	B+C
2	A+C
3	A+C
4	-----
5	B+C

2) What does the code look like after performing CSE (when eliminating a redundant expression, replace it with the variable that holds the calculated value of the expression) (5 pts)

One point per line of code.

Part 2: Register allocation (30 pts [+10 for ECE 573])

For the next 3 problems, consider the following code (assume this is the full program):

```
1:  A = 7
2:  B = 8
3:  A = A + B
4:  C = A + B
5:  D = B + C
6:  E = D + C
7:  D = A + E
8:  F = E + B
9:  B = A + F
10: WRITE(B) //this counts as a use of B
```

1) Show which variables are live *after* each instruction (10 pts) **(1 point per entry)**

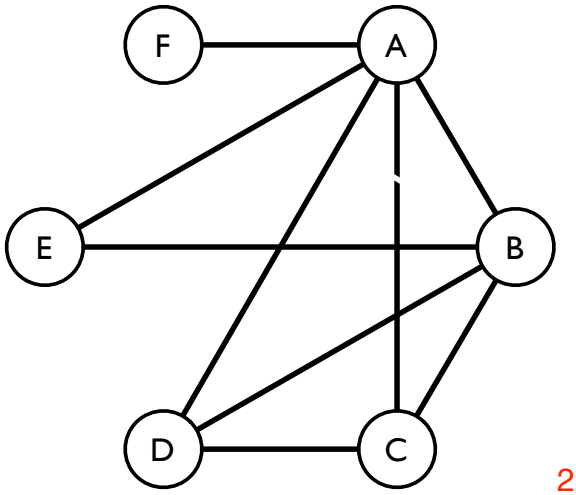
1	A
2	A, B
3	A, B
4	A, B, C
5	A, B, C, D
6	A, B, E
7	A, B, E
8	A, F
9	B

If you shifted your answers (e.g., if you showed live-in instead of live-out), I took off half points for the shifted answers.

2) How many registers are needed to perform register allocation on this code without spills? (3 pts)

no partial credit

3) Draw the interference graph for the code (7 pts)



2 points per missing/extra line

Consider performing bottom up register allocation. For the following scenarios, show what code needs to be generated for the given three-address-code instruction and give the state of the registers after code generation (if a value in a register is dirty, mark it with a *) If variables need to be spilled, always spill the variable in the numerically lowest register first.

4) Before this instruction, the state of the registers is as follows:

R1	R2	R3
A	B*	C

a) What code is generated for `A = A + B` where A, B and C are live after this instruction (3 points)?

```
R1 = R1 + R2
```

b) What is the state of the registers *after* this code (2 points)?

R1	R2	R3
A*	B*	C

5) Before the instruction, the state of the registers is as follows:

R1	R2	R3
D*	B	E

a) What code is generated for `A = B + C` **where A, C, D and E are live after this instruction (3 points)?**

```
ST R1, D //spill D
LD C, R1
//Free B -- no code because B is not dirty
R2 = R2 + R1
```

```
1 point per code mistake
```

b) What is the state of the registers *after* this code (2 points)?

R1	R2	R3
C	A*	E

6) [ECE 573 only, bonus points for ECE 468] Show the results of coloring the interference graph you generated in problem 3, assuming 3 registers.

a) Give the stack generated during the simplification phase. Mark potentially spilled variables with a *. When simplifying the graph, always choose the alphabetically-earliest legal variable to remove from the graph first; if there are no legal variables, choose the alphabetically earliest variable to spill. (5 points)

E, F, A*, B, C, D

ECE 468 bonus points counted half.

b) Show which variables are assigned to which registers. If a variable has to be spilled, indicate so. When assigning registers to variables during the coloring phase, choose the numerically-earliest legal register whenever you have a choice. You *do not* have to do any code rewriting; just mark spilled variables and continue assigning registers (5 points)

R1: D, F, E **R2: C** **R3: B**

ECE 468 bonus points counted half.

Part 3: Instruction Scheduling (40 pts)

For the following problems, assume a machine that has 2 ALUs that can execute adds and shifts, with a single-cycle latency, 1 MU that can do integer multiplication with a two-cycle latency and adds with a two cycle latency, and 1 LS unit that can execute loads with a two-cycle latency and stores with a one-cycle latency.

1) Draw the reservation tables for the following instructions: LD, ST, ADD, MUL, SHIFT (8 pts):

1 point per reservation table

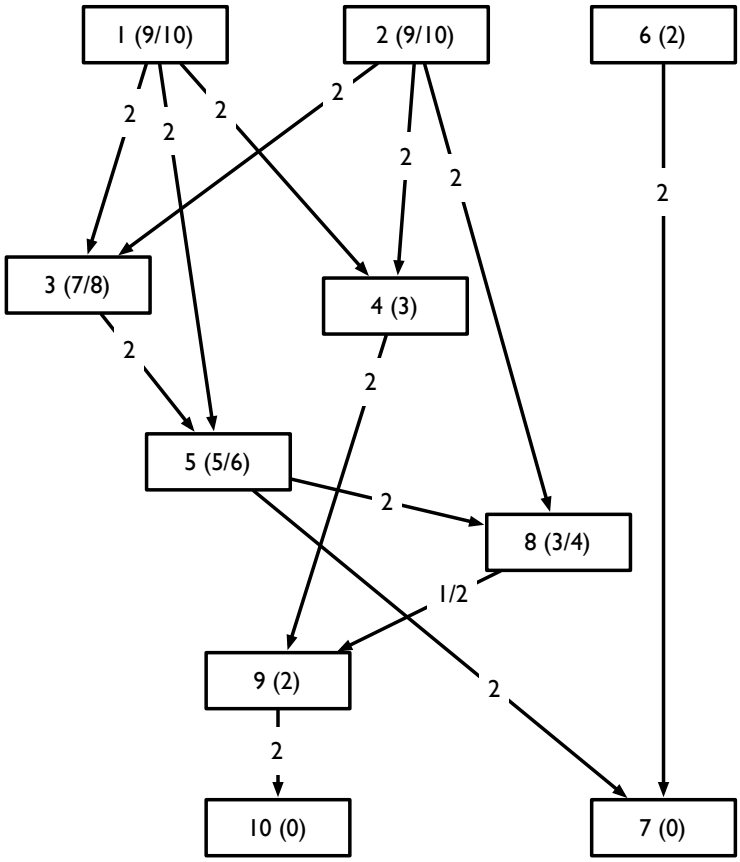
LD: reservation table with LD/ST unit occupied for two cycles
ST: reservation table with LD/ST unit occupied for one cycle
MUL: reservation table with MU occupied for two cycles
SHIFT: two reservation tables, each with a different ALU occupied for one cycle.
ADD: three reservation tables: two with a different ALU occupied for one cycle, one with MU occupied for 2 cycles

2) Draw the data-dependence graph for the following piece of code, including latencies. Show the *heights* of each node in the graph (12 pts):

```
1:  LD A, R1; //Load A into R1
2:  LD B, R2;
3:  R3 = R2 * R1;
4:  R4 = R1 << R2;
5:  R5 = R1 * R3;
6:  LD C, R6;
7:  R7 = R5 + R6;
8:  R8 = R2 + R5;
9:  R9 = R8 * R4;
10: ST(R9), D; //Store R9 into D
```

-2 points for height errors (max of -4)
-2 points for latency errors (max of -4)
    Latency of 1 or 2 for ADD ok, as long as consistent.
-2 points for dependence errors (no max)

heights in parentheses (if two heights listed, second is height if adds have latency 2)



3) For each instruction above, show in which cycle it will be executed if we use height-based list scheduling. If there is a tie in heights, give priority to the instruction earlier in program order. Show your work in the table below (20 points)

Cycle	ALU1	ALU2	MU	LD/ST	Inst(s) scheduled
1				1	1
2				1	1 (cont)
3				2	2
4				2	2 (cont)
5	4		3	6	3, 4, 6
6			3	6	3 (cont), 6 (cont)
7			5		5
8			5		5 (cont)
9	8	7			7, 8
10			9		9
11			9		9 (cont)
12				10	10
					two points per incorrectly scheduled instruction

Part 4: Loop optimizations (20 pts [+10 for ECE 573])

For the next 2 problems, consider the following code:

```
1:  A = 2
2:  B = 10
3:  if (A < B) goto 9
4:  C = B * A + 2
5:  D = 2 * A + 3
6:  if (C < E) goto 9
7:  A = A + 1;
8:  goto 3
9:  A = E + B
```

1) Give the code that would be produced after performing strength reduction (10 points). For partial credit, identify any loop induction variable(s) and mutual induction variable(s).

induction variable: A (1 point for identifying)
mutual induction variables: C, D (1 point for identifying, 3 pts for right code for D, 4 pts for right code for C)

```
1:  A = 2
2:  B = 10; C' = B * A + 2; D' = 2 * A + 3
3:  if (A < B) goto 9
4:  C = C'
5:  D = D'
6:  if (C < E) goto 9
7:  A = A + 1; C' = C' + B; D' = D' + 2
8:  goto 3
9:  A = E + B
```

3) Suppose you are porting code from a machine with a very large L2 cache to a machine that has a very small L2 cache. Will loop tiling and loop interchange be *more* or *less* useful optimizations on the second machine compared to the first? Explain why (10 points):

For a large cache, it's possible that the code's working set will already fit in cache, so the optimizations won't be useful anyway, whereas that's less likely for the small cache.

General principle: smaller caches means more misses. More misses means that anything you can do to improve locality will be good. Thus, both optimizations are more useful.

4) [ECE 573 only, bonus points for ECE 468] Give an example of a piece of code where loop fusion is illegal *that does not use arrays* (10 points):

Many possibilities. The key is that iterations of the second loop needs to depend on results from the first loop, but results from *later* iterations of the first loop. The easiest way to do this is to update a (single) variable in each iteration of the first loop and then read that variable in the second loop.

ECE 468 bonus points counted half.