ECE 468 & 573 — Midterm 2

October 31, 2013

Name: __KEY______________________________

Purdue email: ______________________________________

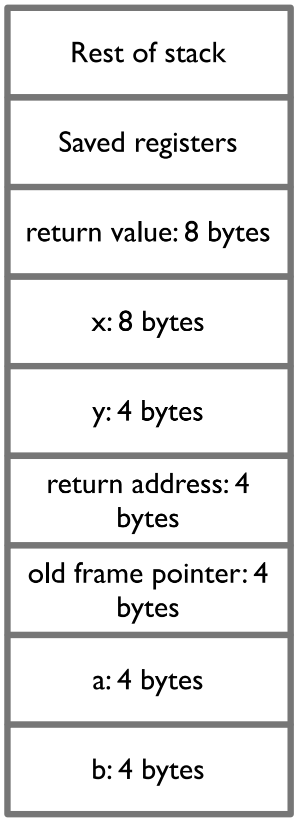
Please sign the following:
I affirm that the answers given on this test are mine and mine alone. I did not receive help from any person or material (other than those explicitly allowed).

X ____

Part	Points	Score
1	10	
2	10	
3	25 [+10]	
4	35	
5	20 [+10]	
Total	100 [+20]	

Part 1: Semantic actions (10 pts)

1) Consider the following activation record after calling a function, foo. Give a possible function *declaration* for foo, showing the arguments and return values, including types. Assume the language supports ints (4 bytes) and doubles (8 bytes). Say whether the code that generated this activation record used caller- or callee-saves registers. (6 points)



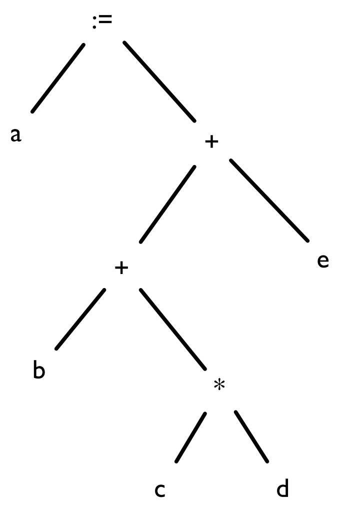
This uses caller-saves registers (the registers are in the part of the stack managed by the caller)

function declaration matching stack:
double foo (double x, int y)

The *definition* of foo would also have two local ints, a and b

2) Draw the abstract syntax tree for the following piece of code (4 pts):

a := b + c * d + e



Part 2: Common subexpression elimination (10 pts)

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

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

2) What does the code look like after performing CSE? Leave the code in IR form. When eliminating a redundant expression, replace it with the variable that holds the previous result of computing the expression (3 pts)

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

3) How would your response to part 2 change if A and B were aliased? (2 pts)

Statement 2 would no longer be redundant (statement 1 would immediately kill "B + C")

Part 3: Register allocation (25 pts [+10 for 573])

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

```
1:  A = 7
2:  B = 8
3:  T1 = A + B
4:  T2 = T1 + C
5:  C = A + B
6:  T3 = T2 + C
7:  A = C + T3
8:  T4 = A + B
9:  B = A + T4
10: WRITE(B) //this counts as a use of B
```

1) Show which variables and temporaries are live *after* each instruction (assume no aliasing) (10 pts)

1	A C
2	A B C
3	A B C T1
4	A B T2
5	B C T2
6	B C T3
7	A B
8	A T4
9	B
10	

2) Consider Instruction 7 in the above code. What variables would be live after instruction 7 if A *may be aliased* to B and C (5 points)

A, B and C (instruction 8 uses A and B, and *may* use C because A may be aliased to C)

For the following scenarios (Not the same code as the previous page!), show what code needs to be generated for the given three-address-code instruction using bottom-up register allocation, 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. Assembly should take the form [Rx = Rb op Ry], [LD var, Rx] or [ST Rx, var]

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

R1	R2	R3
A	B*	C*

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

```
ST R2, B
LD D, R2
R1 = R1 + R2
```

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

R1	R2	R3
A*	D	C*

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

R1	R2	R3
D*	B	E

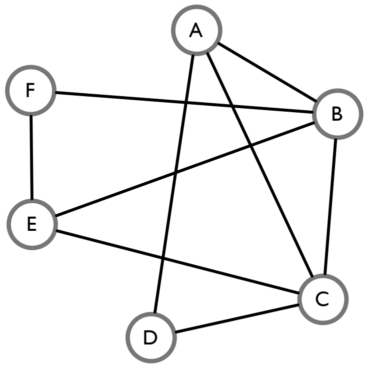
a) What code is generated for `A = F + G` **where A, B, D, and E are live after this instruction (3 points)?**

```
ST R1, D
LD F, R1
LD G, R2 //Free R1 and R2
R1 = R1 + R2
```

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

R1	R2	R3
A*		E

5) [ECE 573 only] Consider the following interference graph (10 points):



a) Give the stack generated during the simplification phase, assuming a machine with 3 registers. 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)

D, A, C, B, E, F (Note that once D is removed from the graph, A can be removed, so it is removed next)

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: F, C
R2: E, A
R3: B, D

Part 4: Instruction Scheduling (35 pts)

For the following problems, consider a machine which has 2 ALUs, a MU and a LD/ST unit. The architecture has five instructions: ADD, SUB, MUL, LD and ST. ADD and ST take one cycle each, SUB, MUL and LD take two cycles each. The reservation tables for the instructions are given below (note that the MU is fully pipelined, so a new MUL instruction can be issued in every cycle).

ADD:

ALU0	ALU1	MU	LD/ST
	X		

ALU0	ALU1	MU	LD/ST
X			

SUB:

ALU0	ALU1	MU	LD/ST
X			
X			

ALU0	ALU1	MU	LD/ST
	X		
	X		

MUL:

ALU0	ALU1	MU	LD/ST
		X	

ST:

ALU0	ALU1	MU	LD/ST
			X

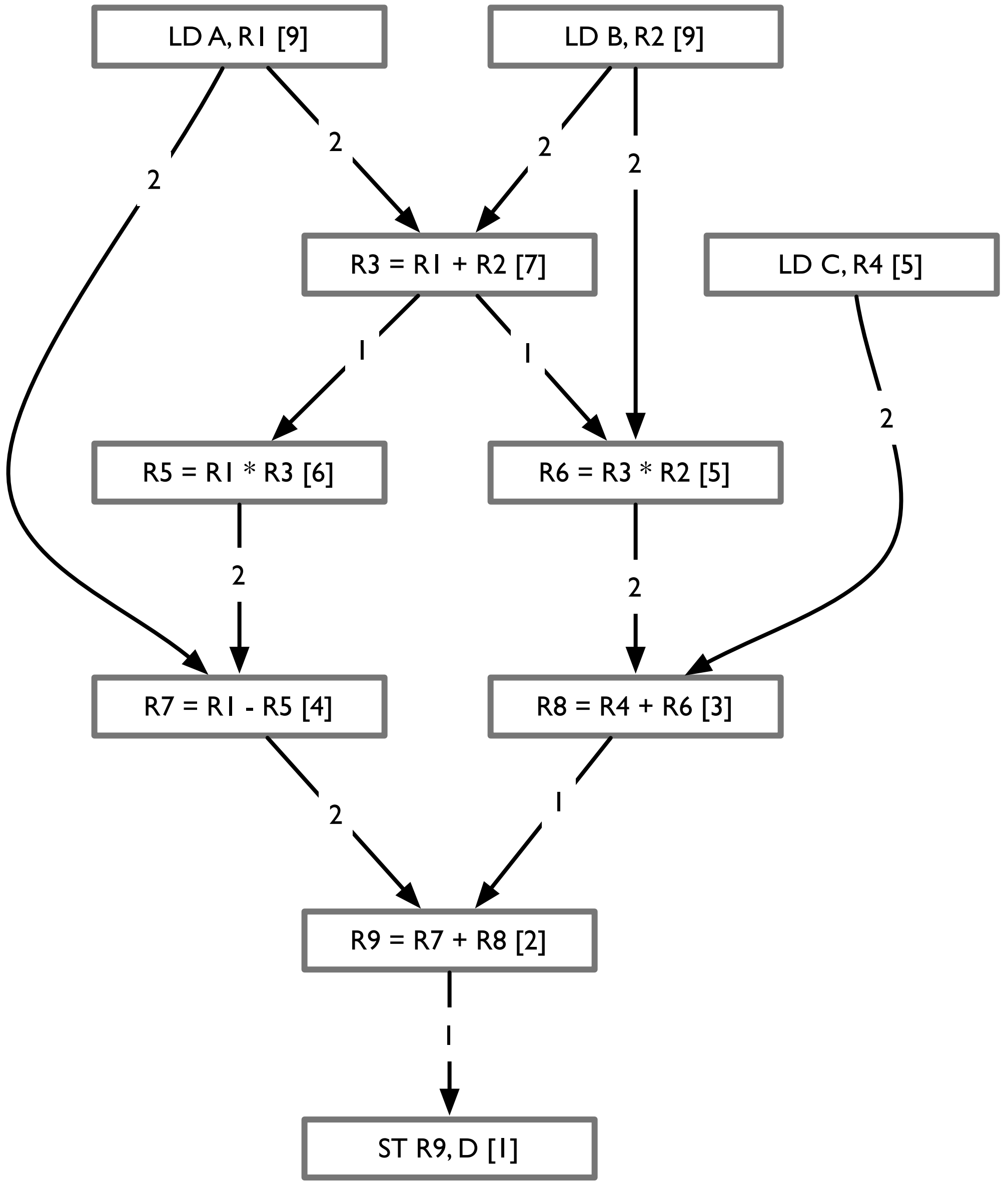
LD:

ALU0	ALU1	MU	LD/ST
X			
			X

ALU0	ALU1	MU	LD/ST
	X		
			X

1) **Draw the data-dependence graph for the following piece of code, including latencies. Show the *heights* of each node in the graph (15 pts):**

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



2) 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 (you may not use all the slots) (20 points)

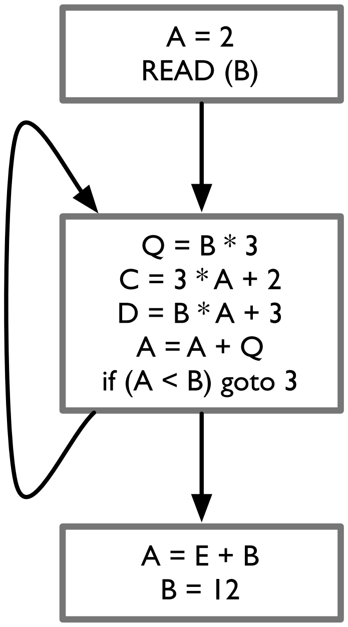
Cycle	ALU1	ALU2	MU	LD/ST	Inst(s) scheduled
1	1				1
2	2			1	2, 1 (cont)
3	4			2	4, 2 (cont)
4	3			4	3, 4 (cont)
5			5		5
6			6		6, 5 (MU free, but 5 not done)
7	7				7, 6 (MU free, but 6 not done)
8	7	8			8, 7 (cont)
9	9				9
10				10	10

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

For the next 2 problems, consider the following code:

```
1:  A = 2
2:  READ(B) //B = value provided by user
3:  Q = B * 3
4:  C = 3 * A + 2
5:  D = B * A + 3
6:  A = A + Q
7:  if (A < B) goto 3
8:  A = E + B
9:  B = 12
```

1) Draw the control flow graph for this code (one CFG node for each basic block). (7 points)



2) Identify any loop-invariant statements in the program. Can they be moved outside the loop? (3 points)

Q = B * 3 is loop invariant, and can be moved

3) 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). ***If you identified any loop invariant code in the previous question, do not move it outside the loop.***

```
1:   A = 2
2:   READ(B) //B = value provided by user
2′: C′ = 3 * A + 2
2′′:D′ = B * A + 3
3:   Q = B * 3
4:   C = C′
5:   D = D′
6:   A = A + Q
6′: C′ = C′ + Q * 3
6′′:D′ = D′ + B * Q
7:   if (A < B) goto 3
8:   A = E + B
9:   B = 12
```

induction variable: A
mutual induction variables: C, D

4) [573 only] Apply linear test replacement to the code in the previous part. (10 points)

```
1:   A = 2
2:   READ(B) //B = value provided by user
2′: C′ = 3 * A + 2
2′′:D′ = B * A + 3
3:   Q = B * 3
4:   C = C′
5:   D = D′
6:   //A = A + Q
6′: C′ = C′ + Q * 3
6′′:D′ = D′ + B * Q
7:   if (C < 3 * B + 2) goto 3
8:   A = E + B
9:   B = 12
```