ECE 573 — Midterm 2
November 3, 2009

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 ____

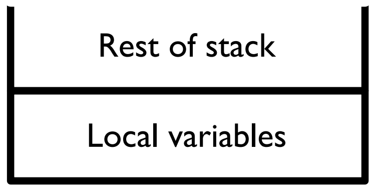
Part	Points	Score
1	20	
2	12	
3	40	
4	38	
5	30	
Total	140	

Part 1: Semantic actions and functions (20 pts)

1) Why is it useful to pass arguments as read-only? (2 pts)

2) Draw the AST that would be generated for the expression `*(p) + x` and for each AST node, show the data object that would be generated. For each data object, show the code that would be generated (if any), the temporary the result will be stored in, and whether the temporary is an R-value or an L-value. (8 pts)

3) Here is a partial stack of a method being executed (the stack grows down). Show the stack after calling `int foo(int a, int[] b)`. Show the frame pointer, and note how much space each part of the stack occupies (32-bit ints and pointers). Assume that we are using a callee-saves convention, and that the machine has 4 registers (plus FP & SP registers). Assume that there are no nested scopes or local variables in foo. (10 pts)



Part 2: Common subexpression elimination (12 pts)

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

```
1: A = B + C;
2: D = A - C;
3: E = B + C;
4: B = C + E;
5: F = A - C;
6: G = B + C;
```

1) Assume there is no aliasing between variables. For each statement, list which expressions are "available" *after* the statement executes (6 pts)

1	
2	
3	
4	
5	
6	

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) (3 pts)

3) Now assume that B and C are aliased. How does that change the results of CSE? Show the code that would be generated in this case. (3 pts)

Part 3: Register allocation (40 pts)

1) **Consider the following code which performs A = A + B; C = B; If A and B are aliased, this code is broken. Indicate what code needs to be inserted to make the code perform correctly. (2 pts)**

```
1: LD A, R1

2: LD B, R2

3: R1 = R1 + R2

4: ST A, R1

5: ST C, R2
```

For the next problems, consider the following code:

```
1: LD A, T1
2: LD B, T2
3: T3 = T1 * T2
4: T4 = T1 + T2
5: LD C, T5
6: T6 = T5 + T4;
7: T7 = T6 + T3;
8: T8 = T1 + T7;
9: ST T8, A
```

2) **Show which temporaries are live *after* each instruction (9 pts)**

1	
2	
3	
4	
5	
6	
7	
8	
9	

3) **How many registers are required to allocate every temporary to a register without spilling? (1 pt)**

4) **Assume we have a machine with 3 registers available (not including spill registers). What temporaries get assigned to registers if we perform *top-down* register allocation. If there is a tie in the algorithm, choose the lowest numbered temporary. (3 pts)**

5) **Perform bottom-up register allocation on this piece of code. At each instruction, show which temporary is assigned to which register *after* the instruction is executed (if a register is freed, mark it as such even if it still holds a value). When a register needs to be spilled, pick the one whose value is next used the farthest away. If there is a tie, pick the lowest numbered register. If multiple registers are free when allocating registers, choose the lowest numbered one. Indicate where loads and stores due to spills happen (12 pts)**

Inst	R1	R2	R3	Loads/Stores due to spills
1				
2				
3				
4				
5				
6				
7				
8				
9				

6) Draw the interference graph for this piece of code (5 pts)

7) What order will nodes be colored if you perform register allocation using graph coloring? Simplify nodes with the most edges first. Whenever you are presented with a choice (for simplification or spilling), choose the lowest numbered temporary. (4 pts)

8) Given your answer to the previous question, which temporaries get assigned to which registers? Whenever you are presented with a choice of registers, choose the lower numbered register. (4 pts)

Part 4: Instruction scheduling (38 pts)

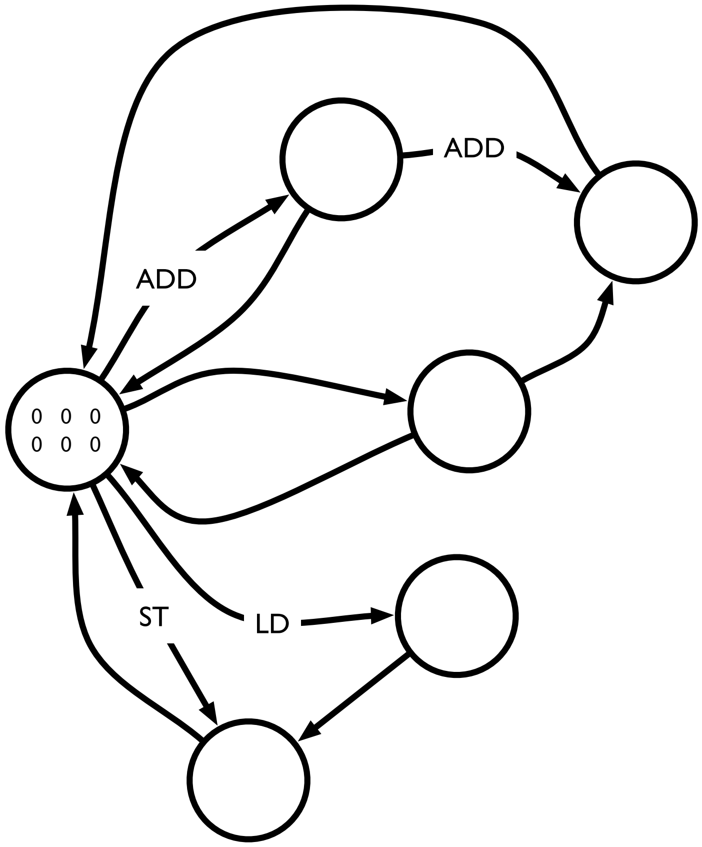
For the following problems, consider the following piece of code:

```
1: LD X, T1
2: LD Y, T2
3: T3 = T1 + T2
4: T4 = T3 + T3
5: LD Z, T5
6: T6 = T5 + T3
7: T7 = T1 + T4
8: T8 = T6 + T7
```

1) Assuming no aliasing, draw the data dependence graph for this piece of code. Assume loads take 2 cycles and all other instructions take 1 cycle (6 pts)

2) Consider an architecture with 2 ALUs and 1 MEM unit. An ADD takes up an ALU for one cycle, a LOAD takes up a MEM unit for 2 cycles and a STORE takes up a MEM unit for 1 cycle. Draw the reservation tables for the three instructions (5 pts)

3) Below is a partial resource constraint FA. Label the unlabeled edges, and fill in the blank states (you don't need to add any of the missing states or edges!) (8.5 pts)



4) How many states would the full FA have? (Hint: you don't need to draw out the whole FA to figure this out) (2.5 pts)

5) What are the *heights* of the instructions in this program? (8 pts)

Instruction	Height
1	
2	
3	
4	
5	
6	
7	
8	

6) Fill in the schedule for this program using height based scheduling. For each cycle, write which instructions are newly scheduled in which functional units. Place an "X" if a functional unit is still in use from a previous cycle, and write nothing if the unit is idle. You may not use all the rows in the table! (8 pts)

Cycle	ALU0	ALU1	MEM
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

Part 5: Loop optimizations (30 pts)

1) Why is loop interchange useful? (2 pts)

2) Why is loop fusion useful? (2 pts)

For the next questions, consider this piece of code:

```
1:        x = a * a;
2:        c = -1 * x;
3:   L1:  if (x >= 100) goto L2;
4:        z = 2 * c; //z is negative!
5:        y = z * x;
6:        x = x + 1;
7:        goto L1;
8:   L2:
```

3) Which statements are loop invariant? Can they be moved outside the loop? (4 pts)

4) Which variables are induction variables ? (1pt) Which variables are mutual induction variables? (2pt)

5) What does this code look like after performing code motion *and* strength reduction? (10 pts)

6) What does the code look like after performing linear test replacement? Remove any unnecessary instructions (10 pts)