EECS 583 – Fall 2012 – Midterm Exam

Monday, November 19, 2012
10:40-12:30: open book, open notes

Name: KEY

Please sign indicating that you have upheld the Engineering Honor Code at the University of Michigan.

"I have neither given nor received aid on this examination."

Signature: ______________________

There are 12 questions divided into 3 sections. The point value for each question is specified with that question. Please show your work unless the answer is obvious. If you need more space, use the back side of the exam sheets.

Part I: Short Answer
4 questions, 12 pts total — Score:_____

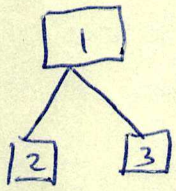
Part II: Short Problems
6 questions, 58 pts total — Score:_____

Part II: Longer Problems
2 questions, 30 pts total — Score:_____

Total (100 possible): _______

Part I. Short Answer (Questions 1-4) (12 pts)

1) A basic block can be control dependent on an edge coming from a *dominating* basic block? Is this statement True or False, briefly explain. (3 pts)



True - BB2 is control dependent on BB1, while BB1 dominates BB2

2) Compilers often provide warnings if local variables can be used without first being defined in a function. Circle the standard dataflow analysis provides the best information to issue such warnings. (3 pts)

[✓ Liveness]   Reaching defs   Available defs   Available exprs

3) When scheduling a basic block, all instructions must be scheduled at their Estart time to achieve the *minimum* schedule length. Is this statement True or False, briefly explain. (3 pts)

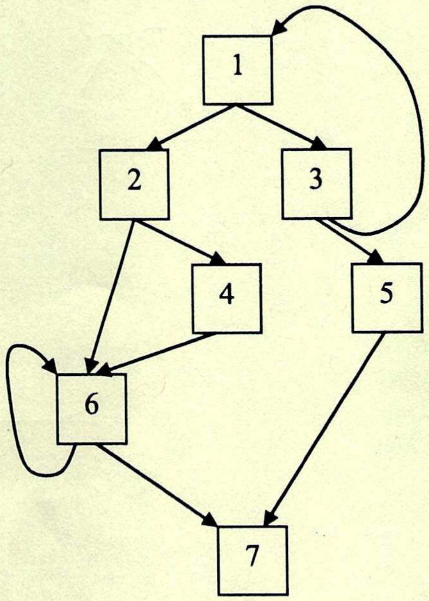
False - Instructions can be scheduled in the range of Estart to Lstart, and still finish the basic block with minimum length.

4) Compute the *ResMII* for a loop with 8 memory and 20 arithmetic instructions on a processor with 2 fully pipelined MEM units and 3 fully pipelined ALUs. (3 pts)

$$ResMII = \max\left(\left\lceil \frac{8}{2} \right\rceil, \left\lceil \frac{20}{3} \right\rceil\right) = \underline{7}$$

Part II. Short Problems (Questions 5-10) (58 pts)

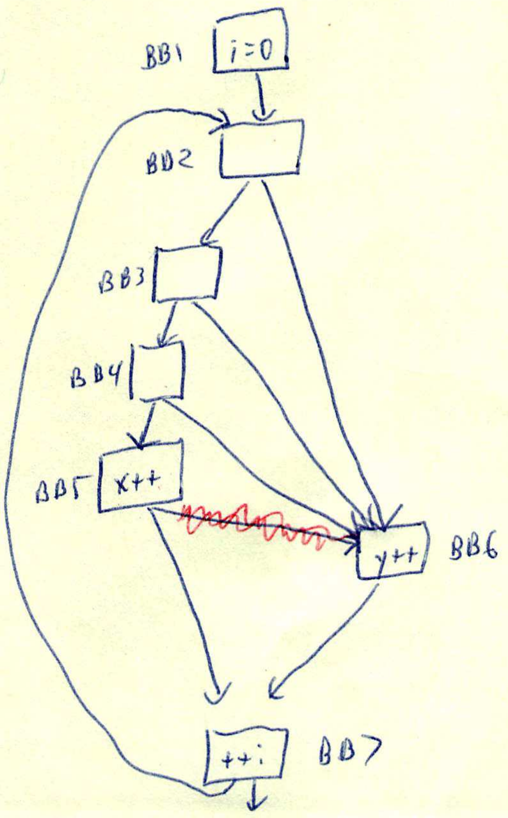
5) Compute the post dominator (PDOM) set for each basic block in the following control flow graph. (5 pts)



BB	PDOM
1	1, 7
2	2, 6, 7
3	3, ~~5~~, 7
4	4, 6, 7
5	5, 7
6	6, 7
7	7

6) Draw the control flow graph (CFG) and determine the *minimum* number of predicates required to if-convert the code. (10 pts)

```
i = 0;
do {
   if (A[i] && A[i]->b && A[i]->b->c)
      x++;
   else
      y++;
} while (++i < 100);
```

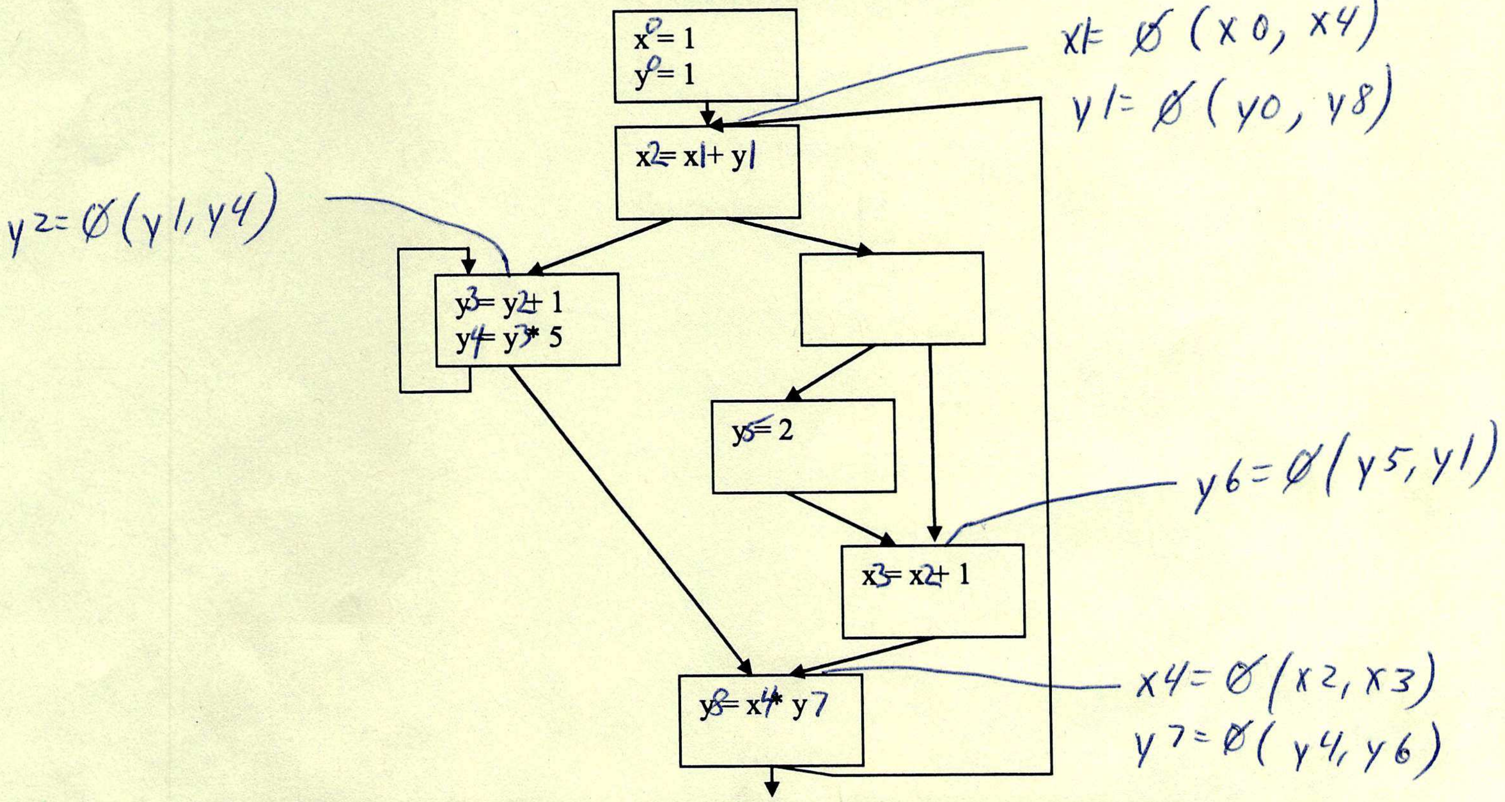


BB	CD
1	—
2	—
3	-2
4	-3
5	-4
6	2, 3, 4
7	—

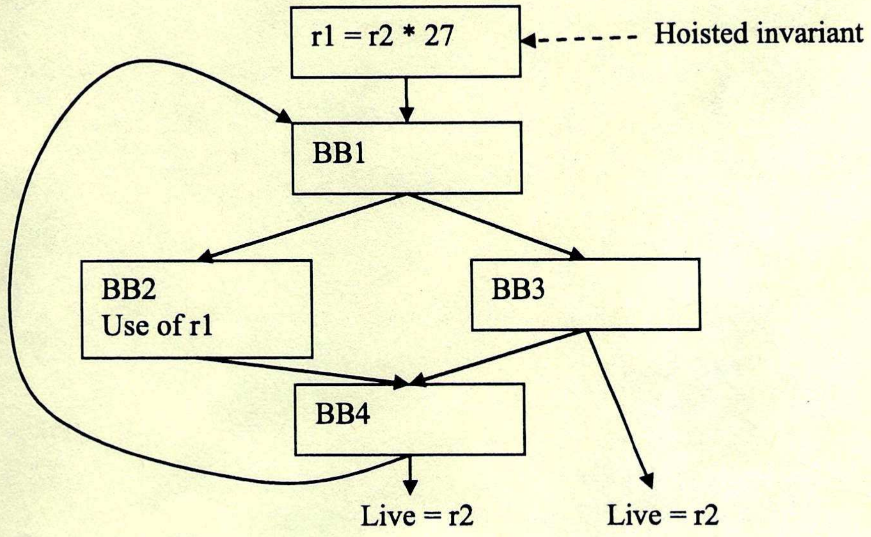
> 4 unique CD sets

=> 4 predicates

7) Convert the following program segment into static single assignment (SSA) form. You should perform the necessary renames and show the Phi nodes. Solving by inspection is fine and you can put your solution directly on the diagram. (10 pts)

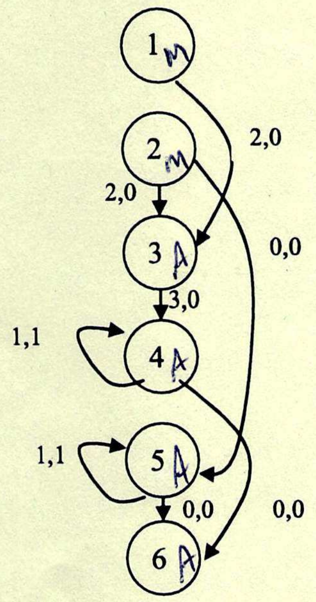


8) In the following loop consisting of basic blocks 1-4 with 2 exit points, LICM has been applied to the instruction "r1 = r2 * 27". Assuming the optimization was legal, which block(s) could *not* be the original home of the invariant instruction? Explain. (8 pts)



Could not come from: BB3, BB4 > A def of r1 in these blocks is not guaranteed to reach the use in BB2

9) Generate the II=4 modulo schedule for the following dependence graph and processor model. Show the unrolled and rolled schedules for your answer. You can assume that instruction 1 is the highest priority, 2 is second highest, etc. and you do not need to assign staging predicates. (15 pts)



Processor model
2 fully pipelined function units
1 ALU, 1 MEM

Instructions 1 and 2 are memory
Instructions 3, 4, 5 and 6 use the ALU
Instruction 6 is the branch

MRT

	ALU	MEM
0	X	X
1	X	X
2	X	
3	X	

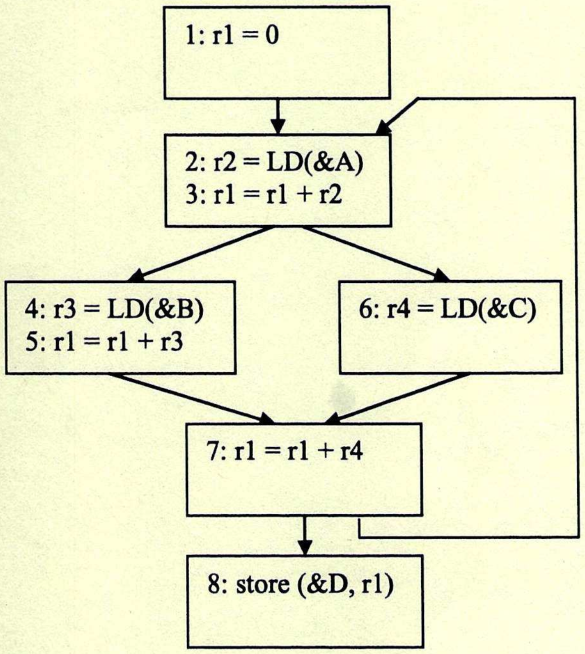
Unrolled Schedule

time	instruction
0	1
1	2
2	5
3	
4	3
5	
6	
7	
8	
9	4
10	
11	6

Rolled Schedule

time	instruction
0	1,3
1	2,4
2	5
3	6

10) Consider the following code segment. How many physical registers are necessary to allocate virtual registers r1, r2, r3, r4 *without spilling*? Assume that &A, &B, &C, and &D are compile time constants and do not require registers. Justify your answer. (10 pts)

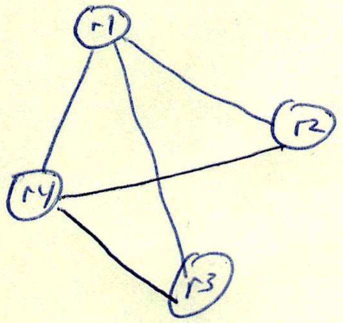


LR(r1) = 1, 2, 3, 4, 5, 6, 7, 8

LR(r2) = 2, 3

LR(r3) = 4, 5

LR(r4) = 6, 7, 2, 3, 4, 5

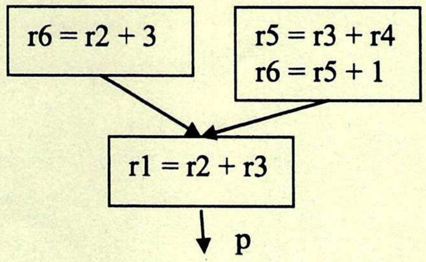


3 registers are required

Part III. Longer Problems (Questions 11-12) (30 pts)

11) Suppose that you are given a new dataflow problem, MustDef. At a point p in a program, MustDef is defined as the set of variables that are guaranteed to be defined along all paths from ENTRY to p.

So for example, at point p, MustDef contains r1 and r6, but not r5.



Define the set of dataflow equations to solve for MustDef. You should define GEN, KILL, IN, and OUT. (15 pts)

This analysis is similiar to available definitions, but of the variables rather than the actual instructions.

GEN/KILL calculation

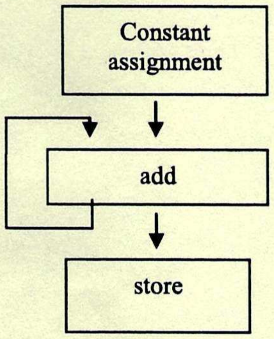
```
GEN(X) = 0
KILL(X) = 0

for each basic block, X
    for each instruction in X, I
        for each dest operand of I, dest
            GEN += dest
            KILL += $\emptyset$        ( no KILL in this analysis )
```

IN/OUT calculation

```
while (change)
    for each basic block, X
        $IN(X) = \cap (OUT(Y))$, for all predecessors Y of X
        $OUT(X) = GEN(X) + (IN(X) - KILL(X))$
```

12) You are the professor in a compiler construction class and have to write an exam question on optimization. Create a small pseudo assembly function which reduces to three instructions: a constant assignment, an add, and a store, each in its own basic block with the implied control flow as shown below. **Note: you do not need to include a branch instruction**. You should use the optimizations listed below, **they must be applied once each (exhaustively), in an order selected by you.**

Create the answer key by showing the code before and after each optimization. Assume that each optimization must fire at least one time. You may use all or a subset of {r1, r2, r3, r4, r5} as extra registers, and your starting code *should not contain any dead code.* (15 pts)



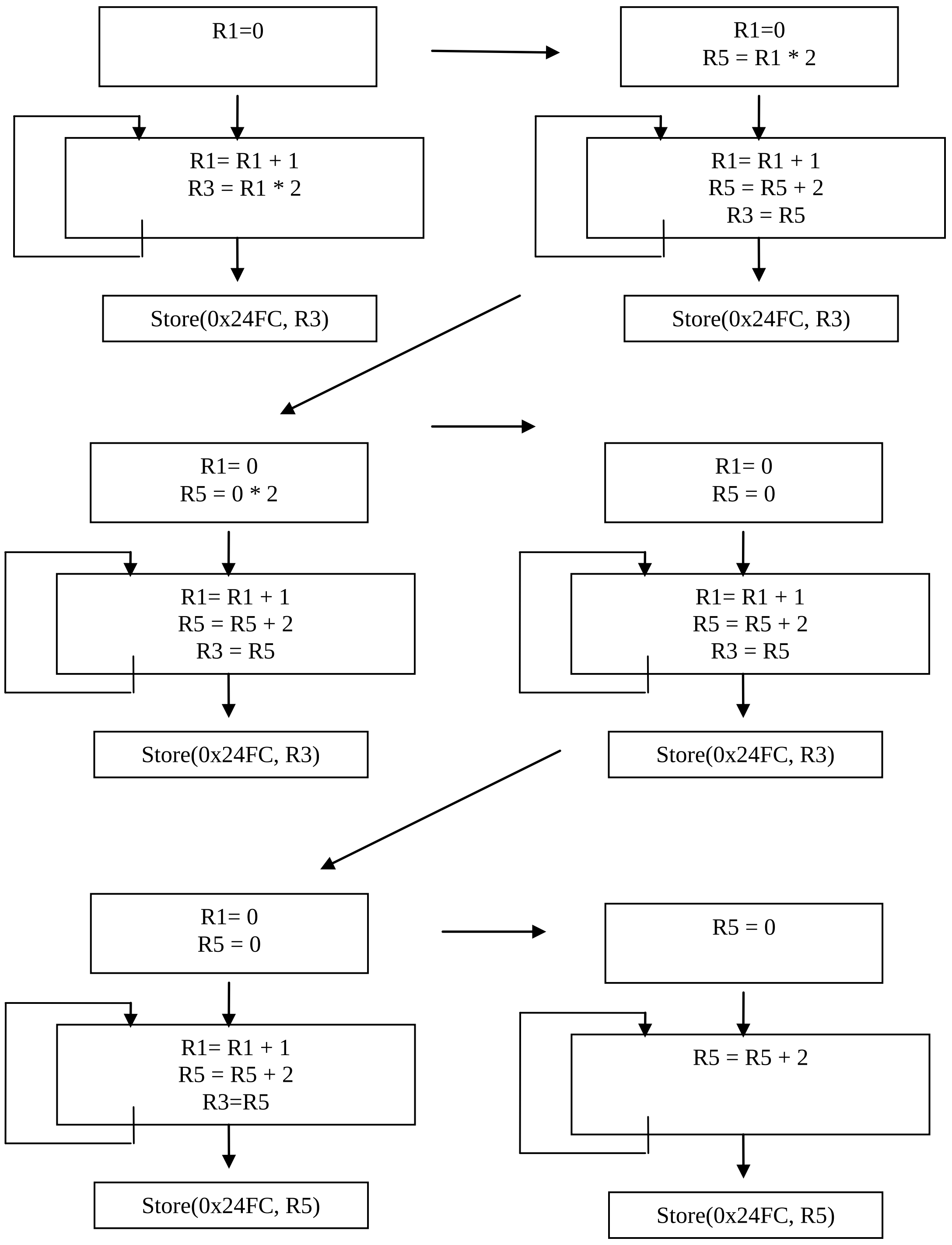
Code template after all optimizations

Optimizations:

Constant folding
Dead code elimination
Constant propagation
Induction variable strength reduction
Copy propagation

Many possible answers… Induction variable strength reduction is a good candidate to go first as it creates additional instructions.

Order: Strength reduction, constant prop, constant folding, copy prop, dead code elimination



```
R1=0

R1= R1 + 1
R3 = R1 * 2

Store(0x24FC, R3)
```

```
R1=0
R5 = R1 * 2

R1= R1 + 1
R5 = R5 + 2
R3 = R5

Store(0x24FC, R3)
```

```
R1= 0
R5 = 0 * 2

R1= R1 + 1
R5 = R5 + 2
R3 = R5

Store(0x24FC, R3)
```

```
R1= 0
R5 = 0

R1= R1 + 1
R5 = R5 + 2
R3 = R5

Store(0x24FC, R3)
```

```
R1= 0
R5 = 0

R1= R1 + 1
R5 = R5 + 2
R3=R5

Store(0x24FC, R5)
```

```
R5 = 0

R5 = R5 + 2

Store(0x24FC, R5)
```