EECS 583 – Winter 2023 – Midterm Exam

Wednesday, March 15, 2023
Time constraint: 1 hr 20 min
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 11 questions divided into 2 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.
The rubrics used to grade each question will be on Gradescope.

Part I: Short Answer
7 questions, 40 pts total Score:_____

Part II: Long Answer
4 questions, 60 pts total Score:_____

Total (100 possible): _______

Part I. Short Answer (Questions 1-7) (40 pts)

1) In register allocation, live ranges are constructed by taking the intersection of 2 dataflow analyses. Name the 2 dataflow analyses used. (5 pts)

Reaching definitions & Liveness.

2) In liveness analysis, how would the analysis results be changed if the meet function was modified from using union of live variables (IN sets) of the successor blocks to *intersection* over the same sets of variables? Briefly explain your answer. (5 pts)

Must consume / Used on All paths

3) Calculate the *liveness IN set* for the following basic block, where the OUT set is calculated as {r3,r4}. (5 pts)

```
1. r5 = &A + 12
2. r1 = load(r5)
3. r3 = r1 + r2
```

OUT = {r3,r4}

IN = {r2, r4}

4) We know from class that common subexpression analysis (CSE) creates opportunities to apply copy propagation because it introduces moves. Is the reverse true, where copy propagation creates opportunities for CSE? If yes, show an example with at most 3 instructions. If not, explain why. (5 pts).

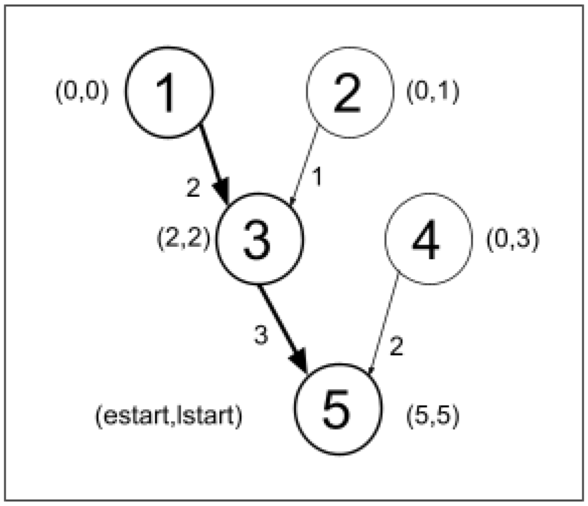
Yes. One possible example:
r1 = r2 + 10
r5 = r2
r4 = r5 + 10

5) Using profile data is generally viewed as a way to improve the effectiveness of compilers. Are there any *potential pitfalls* (e.g., making application performance worse) using profile data? Briefly explain why or why not. (5 pts)

Yes. The tests/inputs used for profiling may not match the actual use case of an application. In which case, any optimizations made using it may make the performance worse.
Profiling can lead to an overhead during compilation, which may not be desirable.

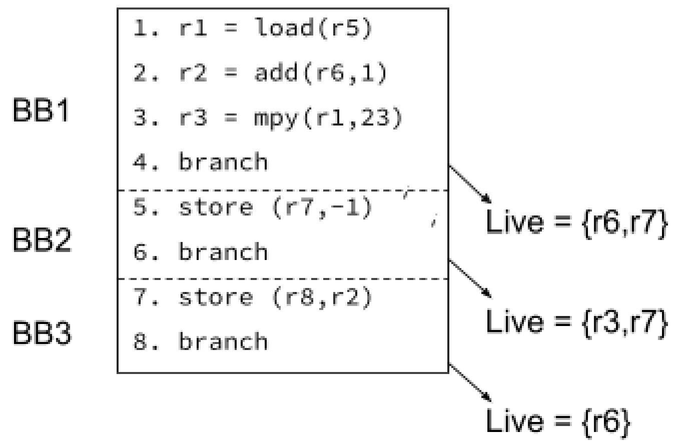
6) Calculate the *critical path length* of the following basic block. Assume the following instruction latencies: load = 2, add = 1, mpy = 3. (5 pts)

```
1. r1 = load(r6)
2. r2 = add(r7,3)
3. r3 = mpy(r1,r2)
4. r4 = load(r8)
5. r5 = add(r3,r4)
```



Length = 5

7) Given the following *scheduled* superblock with associated liveness information for each exit point, determine the basic blocks that instructions 1, 2, and 3 could have originated from. Note that instructions may have originated from multiple basic blocks. You should make **no assumptions** about the addresses used by the load and store instructions. Stores are formatted as store (address, data). (10 pts)



```
1. r1 = load(r5)
2. r2 = add(r6,1)
3. r3 = mpy(r1,23)
4. branch
-------------------
5. store (r7,-1)
6. branch
-------------------
7. store (r8,r2)
8. branch
```

Circle the basic blocks that the instruction could have originated from:

Instruction 1:	[✓ BB1]	BB2	BB3
Instruction 2:	[✓ BB1]	[✓ BB2]	[✓ BB3]
Instruction 3:	[✓ BB1]	[✓ BB2]	BB3

Instruction 1 is a load, and hence cannot be safely moved across either branch since we do not know anything about its address, which could be illegal. Hence, Restriction 2 for upward code motion is violated across both branches 4 and 6.

Instruction 2 is not restricted by any rule, hence it could originate from any basic block.

Instruction 3 cannot be from BB3 since it is live out in the exit branch from BB2. Hence, Restriction 1 is violated for branch 6. But there is no violation for branch 4.

Part II. Longer Problems (Questions 8-11) (60 pts)

8) For the following code:

a. Draw the control flow graph (CFG) (6 points)

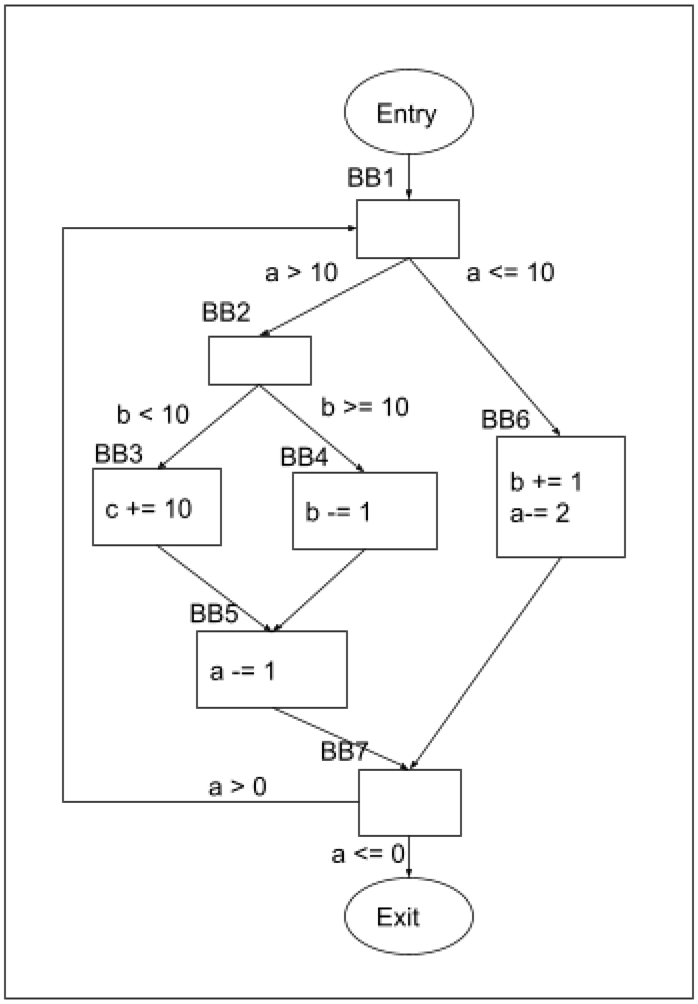
b. Compute the CD (Control Dependence) sets for each Basic Block (BB) (6 points).

c. Using the CDs, determine the minimum number of predicates required to if-convert the code. (3 points)

Hint: The CFG should contain 7 basic blocks, not including pseudo entry/exit basic blocks. Some basic blocks may be empty.

```
do {
    if (a > 10){
        if (b < 10) {
            c += 10;
        } else {
            b -= 1;
        }
        a -= 1;
    }
    else {
        b += 1;
        a -= 2;
    }
} while (a > 0);
```

Control Flow Graph

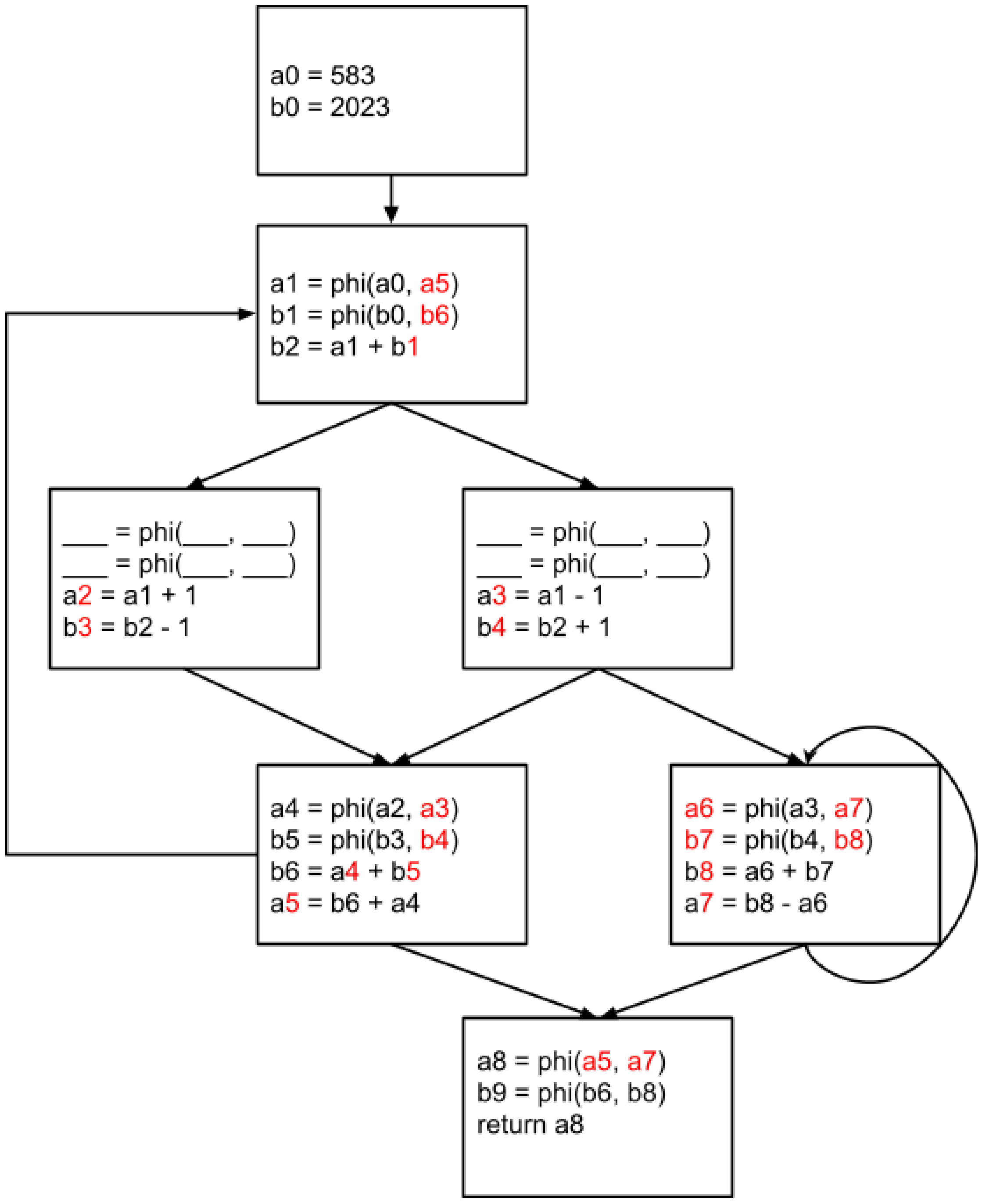


BB	CD set
1	ϕ
2	{-1}
3	{-2}
4	{+2}
5	{-1}
6	{+1}
7	ϕ

The minimum number of predicates required to if-convert the code is ___4_______.

9) Satisfy static single assignment (SSA) form by filling in the blanks in the code segment below. (15 points)

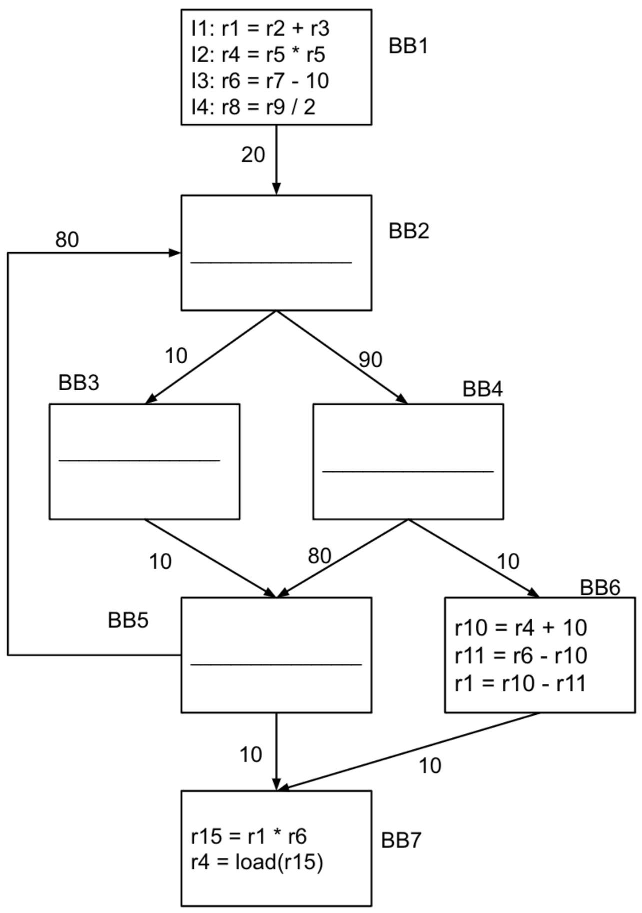
- The result and arguments of a phi node must be different instances of the same variable (e.g. y1 = phi(y2, y3)).
- Some phi nodes may be unnecessary and should be left empty.
- Choose operands between a0 and a8, and b0 and b8. Repetition is allowed.



10) The following profile augmented CFG shows the code after optimizing it using LICM. Four instructions (I1 to I4) were hoisted, **one each** from BB2, BB3, BB4, and BB5. You can assume that none of the instructions cause exceptions. The number nearest the branch arrow indicates the number of times that branch was taken.

a. Find a mapping of instructions to basic blocks such that each instruction (I1, I2, I3, I4) is mapped to a **different** basic block (BB2, BB3, BB4, BB5) such that the instruction could have been hoisted from that basic block. (10 points)

b. Calculate the savings, in terms of the number of **dynamic** instructions reduced, by this optimization. (5 points)



r6 is used in BB6 and BB7, hence I3 can go only in BB2.

r4 is used in BB6, but killed in BB7, and hence I2 cannot go in BB3 or BB5, and must go in BB4. (because I3 is in BB2)

r1 is used in BB7, but killed in BB6, and hence I1 cannot go in BB3 or BB4, hence must be in BB5. (because I)

r8 has no restrictions, hence, it can be in any of the BBs. But I4 must be in BB3, as that is the only remaining option.

Block counts of BB1, BB2, BB3, BB4, BB5 are 20, 100, 10, 90, 90, respectively. Hence, the number of instructions saved is: (BB2+BB3+BB4+BB5) - 4*(BB1) = 290 - 80 = 210.

Instruction	I1	I2	I3	I4
Original Basic Block	BB5	BB4	BB2	BB3

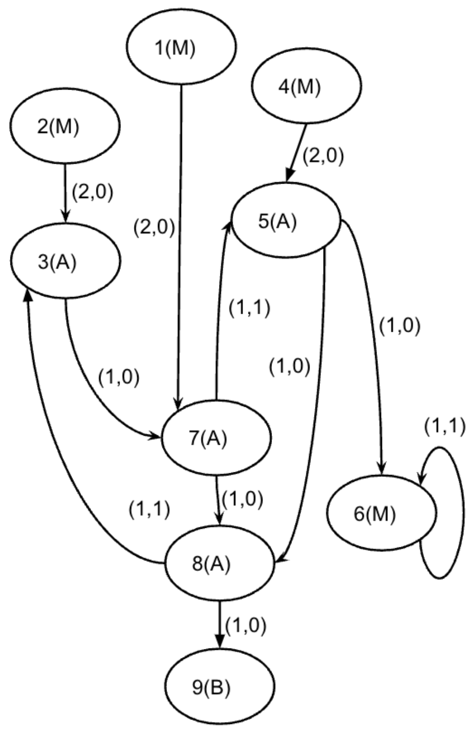
Savings = #instructions before LICM - #instructions after LICM = __210____

11) Given below is a loop dependence graph and a processor model. (M), (A), and (B) refer to memory, ALU, and branch instructions respectively. The memory instructions use the memory units and the ALU and branch instructions use the ALU units.

a. Determine the MII. Show your working. (5 points)

b. Generate the rolled and unrolled schedules. Lower instruction numbers will have a higher priority, i.e. instruction 1 has the highest priority. (10 points)

Processor Model
4 fully pipelined function units
– 2 ALU units
– 2 MEM units



Unrolled Schedule (may contain extra rows)

	ALU1	ALU2	MEM1	MEM2
0			1	2
1			4	
2		3		
3	5	7		
4	8			6
5	9			
6				
7				
8				

Rolled Schedule (may contain extra rows)

	ALU1	ALU2	MEM1	MEM2
0	5	7	1	2
1	8		4	6
2	9	3		
3				

ResMII = max(ceil(5/2), 4/2) = 3

Cycles: (3,7,8,3), (6,6), (7,5,8,3,7)

RecMII = max((1+1+1)/(0+0+1), 1/1, (1+1+1+1)/(1+0+1+0)) = 3

MII = max(3,3) = 3