EECS 583 – Fall 2019 – Midterm Exam

Wednesday, November 13, 2019; 10:40am-12:30pm
Friday, November 15, 2019; 10:40am-12:30pm
Open book, open notes

Name: _____Key_(Answers in red)_____

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.

Part I: Short Answer
5 questions, 25 pts total Score:_____

Part II: Medium Problems
6 questions, 75 pts total Score:_____

Total (100 possible): _____

Part I. Short Answer (Questions 1-5) (25 pts)

1) Name a way that a compiler can use profiling information to improve performance. (5 pts)

(many answers were accepted for this question, here are a few)
Find hot code and more aggressively optimize in those code regions.
Forming traces/superblocks.
Laying out hot code blocks in sequence to improve Icache performance
Optimize for the common case and repair for the uncommon case such as the frequent path LICM in homework 2.

2) When is a control flow edge from basic block A to B (A→B) a backedge? (5 pts)

When B dominates A.

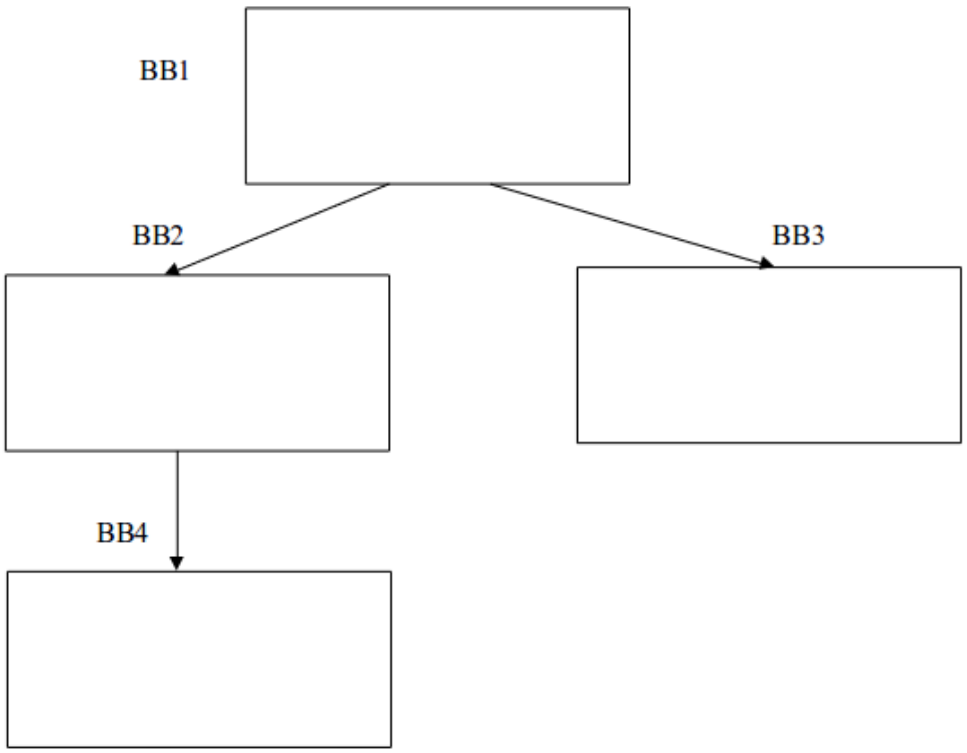
3) Why is it necessary to prove that an instruction will not cause an exception (Restriction 2) when performing upward code motion? (5 pts)

Speculative execution occurs when instructions are moved upwards, hence instructions execute more often than in the original program. It is necessary to prove an instruction will not cause exceptions to prevent spurious exceptions from occurring when an instruction would not have executed in the original program. These spurious exceptions break otherwise working code.

4) If any instruction in a basic block is scheduled later than its Lstart, then the schedule length for the basic block will be longer than the critical path length. Is the preceding statement True or False? Explain your answer. (5 pts)

True. By definition, Lstart is the latest an instruction can be scheduled such that the schedule length is not increased beyond the infinite resource schedule length (i.e., critical path length). Scheduling an instruction after its Lstart will lengthen the critical path.

5) A basic block can only be control dependent on an immediate predecessor block. Is the preceding statement True or False? Explain why True or give counter example if False (5 pts)



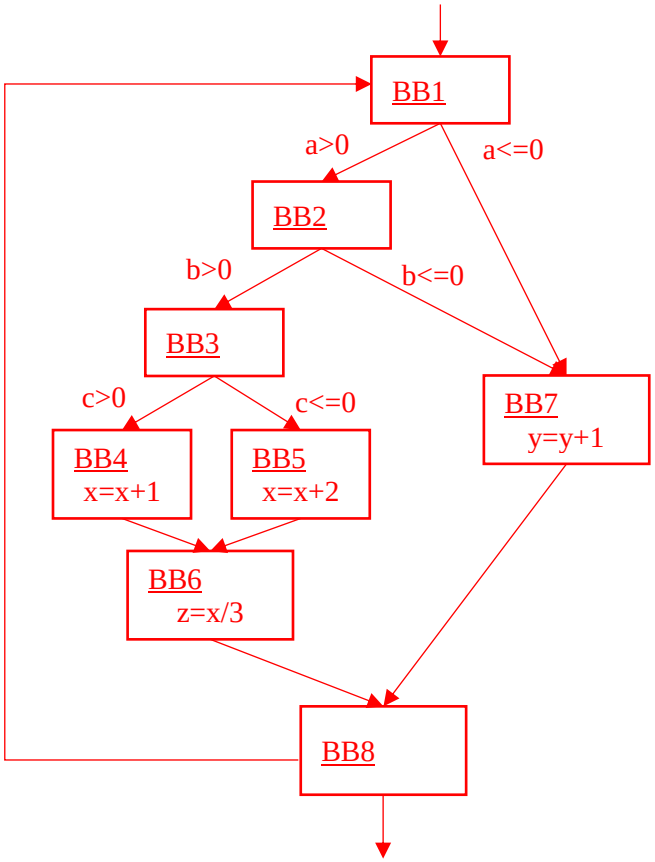
False. Many counter examples are possible and were accepted, the above is 1 example. BB4 is CD on BB1 and is not its immediate predecessor. BB4 is CD on BB1 because BB4 is only executed if the correct branch from BB1 is taken.

Part II. Medium Problems (Questions 6-11) (75 pts)

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

```
do{
   if (a>0 && b>0){
      if (c>0)
         x+=1;
      else
         x+=2;
      z=x/3;
   }else{
      y+=1;
   }
}while(z<100)
```

CFG

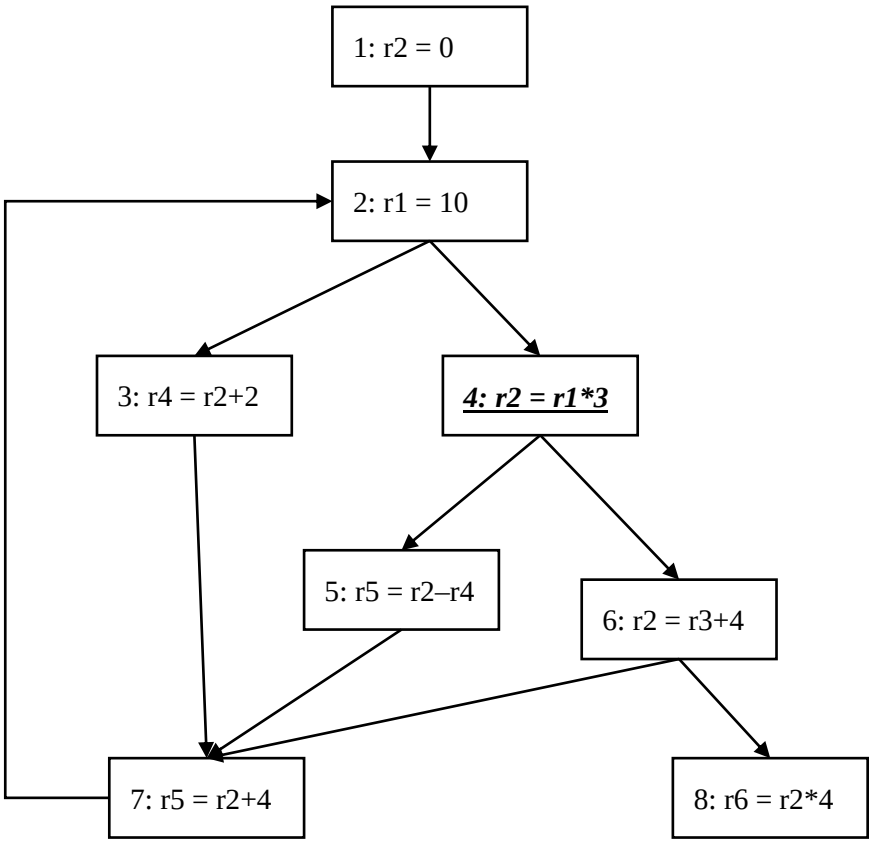


BB	Post dom	Imm Post dom	CD
1	1,8	8	-
2	2,8	8	-1
3	3,6,8	6	-2
4	4,6,8	6	-3
5	5,6,8	6	+3
6	6,8	8	-2
7	7,8	8	+1, +2
8	8	-	-

(Remember, first step in CD calculation is to nuke backedges)

5 unique CD sets. Thus, at least 5 predicates are necessary.

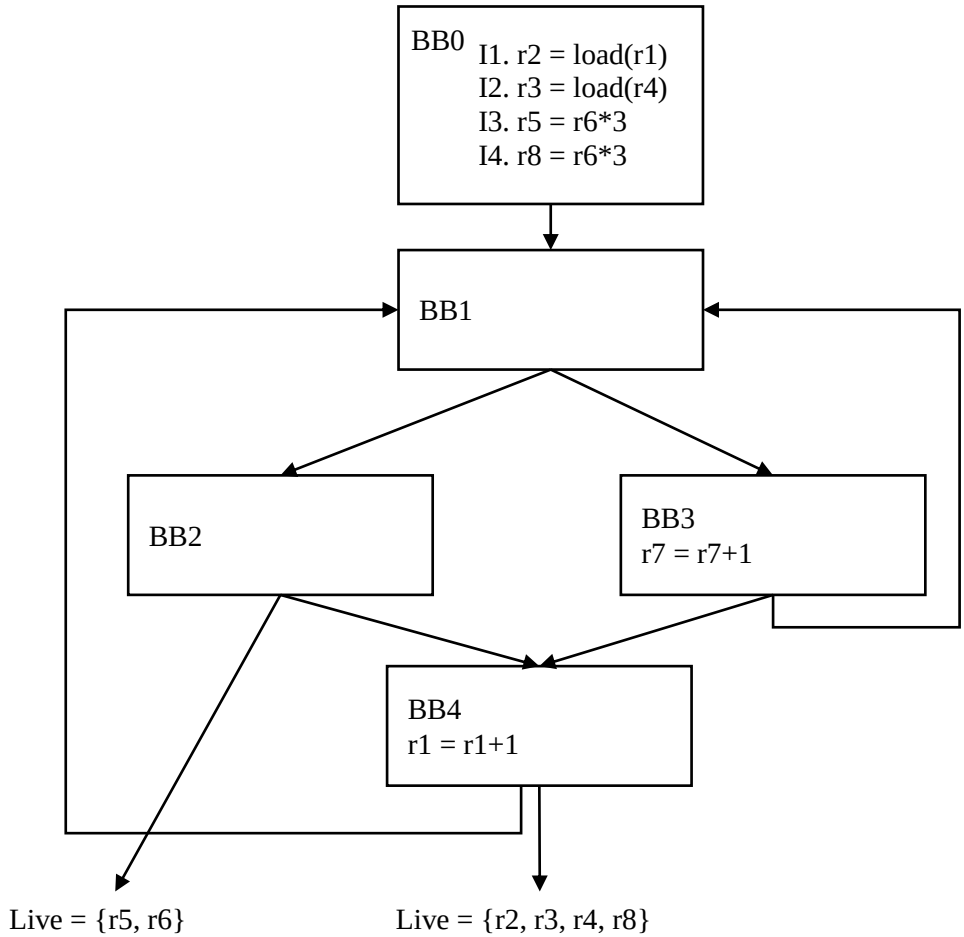
7) Find *all uses* of r2 defined by instruction 4 (i.e., DU chain for instruction 4). You may solve by inspection. (10 pts)



3, 5, 7

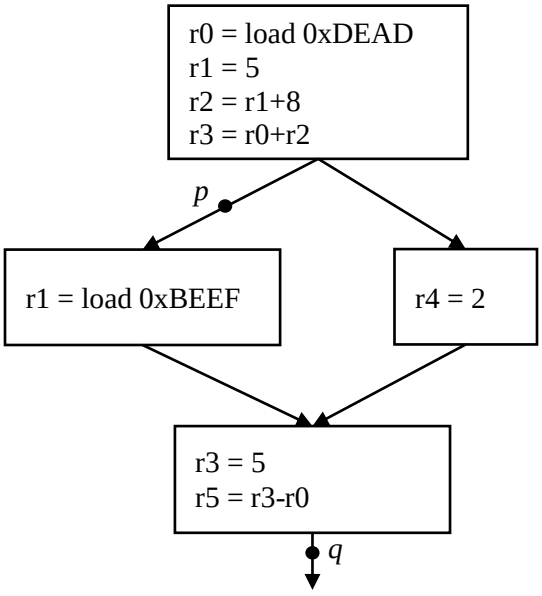
8) You are trying to reverse engineer some optimized assembly code to determine the original locations of instructions before optimization. In the following loop consisting of 4 basic blocks (BB1-BB4), the preheader (BB0) contains 4 instructions (I1, I2, I3, I4) that were possibly removed from the loop using LICM. For each instruction, determine whether LICM could have been legally applied and if so, which basic block(s) the instruction could have originally resided. Circle the correct answers. (10 pts)

I1	LICM legal?	Yes	No	Possible original blocks: BB1 BB2 BB3 BB4
I2	LICM legal?	Yes	No	Possible original blocks: BB1 BB2 BB3 BB4
I3	LICM legal?	Yes	No	Possible original blocks: BB1 BB2 BB3 BB4
I4	LICM legal?	Yes	No	Possible original blocks: BB1 BB2 BB3 BB4



I1: r1 is not loop invariant => no LICM
I2: There is use of r3 after BB4, so def must be available there => only BB1,4
I3: There is use of r5 at left branch out of BB2, so def must be available there => only BB1,2
I4: There is use of r8 after BB4, so def must be available there => only BB1,4

9) You are building a new dataflow analysis that can identify trusted variables. To be conservative, we assume all values loaded from memory to be untrusted and any variables consumed in a block that are not defined in the current basic block to also be untrusted. Constants are assumed trusted. For other types of instructions, the destination register is trusted if all of its source operands contain known trusted values. *For your compiler, build a dataflow analysis pass that identifies trusted variables at a certain point p.* (15 pts)



For example,

At point *p*, {r1,r2} are trusted while {r2,r3} are trusted at point *q*.

(a) Is this a forward or backward data flow analysis problem?

(b) Is this an all-path or any-path dataflow analysis problem?

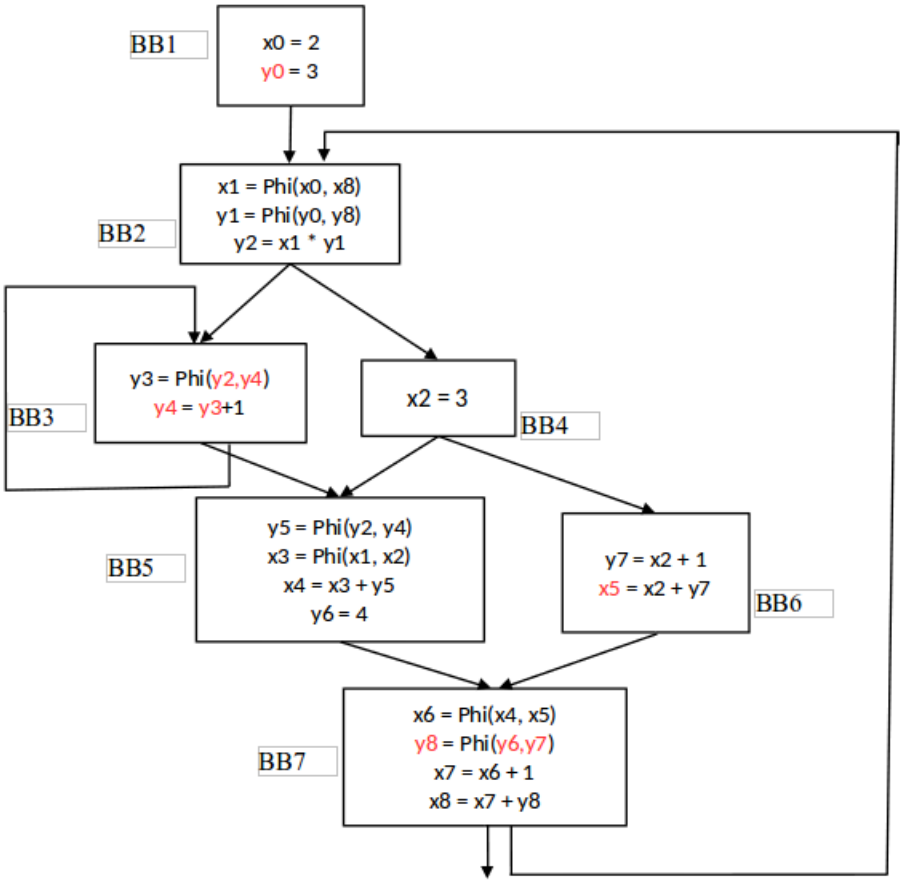
(c) Define GEN and KILL sets to identify trusted variables.

(a) Forward
(b) All path
(c) Next page

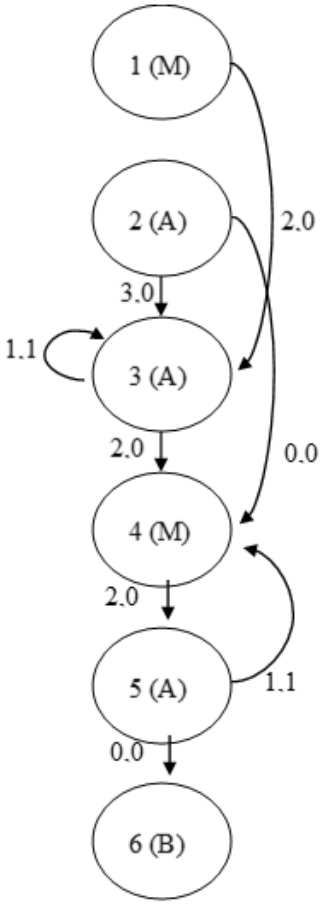
```
For each basic block
    GEN = none
    KILL = none
    for dest in each instruction
        if load instr then
            GEN -= dest
            KILL+=dest
        else if const move (const assignment) then
            GEN += dest
            KILL -= dest
        else
            isTrusted = True
            for each operand in instruction
                if operand not in GEN
                    isTrusted = False
                    break
            if isTrusted then
                GEN += dest
                KILL -= dest
            else
                GEN -= dest
                KILL += dest
```

10) Satisfy static single assignment (SSA) form by **filling in the blanks with variables from the below pool**. The result and arguments of a Phi node must either be all x's or all y's (i.e. x5 = Phi(x0, x4)). Solving by inspection is fine. (15 pts)

Variable Pool (each can be used more than once):
x5, y0, y2, y3, y4, y6, y7, y8



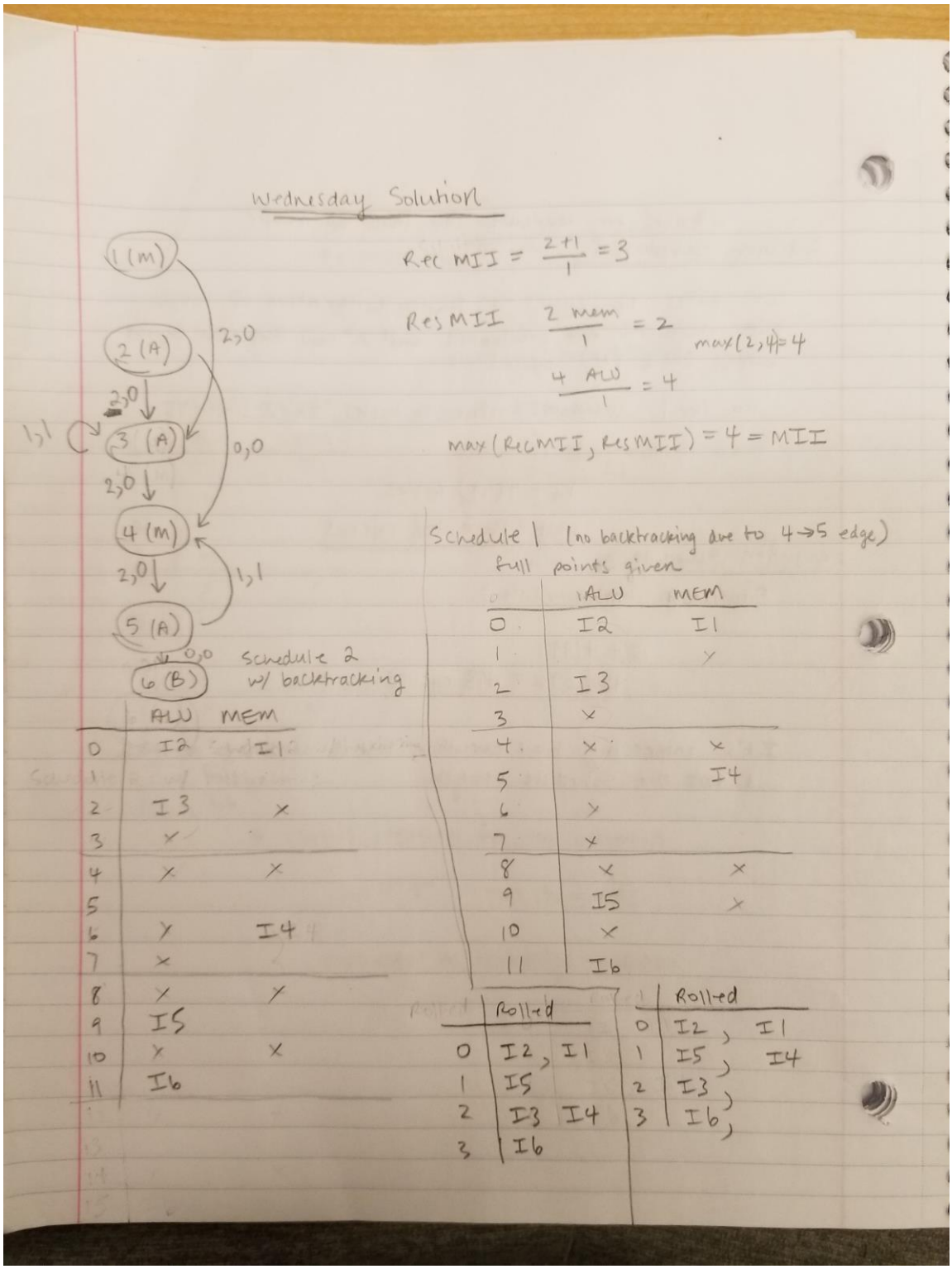
11) Compute the ResMII, RecMII, and MII for following dependence graph and processor model. Then, **generate the MII modulo schedule. Show the unrolled and rolled schedules for your answer.** You can assume that instruction 1 is the highest priority, 2 is second, etc. You do not need to assign staging predicates. **Also, calculate how many cycles it takes to execute 100 iterations of the loop.** (15 pts)



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

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

******This dependency graph is for the FRIDAY version, so look at the corresponding FRIDAY answer. For the WEDNESDAY version, the edge from 2->3 was 2,0, which makes the problem more difficult. The Wednesday graph is re-drawn in the Wednesday solution for clarity.



Wednesday Solution

Rec MII = $\frac{2+1}{1} = 3$

Res MII $\frac{2 \text{ mem}}{1} = 2$ $\quad \max(2,4) = 4$

$\frac{4 \text{ ALU}}{1} = 4$

$\max(\text{RecMII}, \text{ResMII}) = 4 = \text{MII}$

Schedule 2 w/ backtracking

	ALU	MEM
0	I2	I1
1		
2	I3	x
3	x	
4	x	x
5		
6	x	I4
7	x	
8	x	x
9	I5	
10	x	x
11	I6	

Schedule 1 (no backtracking due to 4→5 edge) full points given

	ALU	MEM
0	I2	I1
1		x
2	I3	
3	x	
4	x	x
5		I4
6	x	
7	x	
8	x	x
9	I5	x
10	x	
11	I6	

	Rolled	
0	I2,	I1
1	I5	
2	I3	I4
3	I6	

	Rolled	
0	I2,	I1
1	I5,	I4
2	I3,	
3	I6,	

cycles to execute 100 iters of loop?

Solution Schedule 1

It takes 12 cycles to finish 1 iteration of loop. But, iterations are overlapped and a new iteration begins every MII (4) cycles

So iter 1 completes after 12 cycles. Iter 2 after 16....

so iter 100 finishes after

$12 + 4(99)$ cycles

$12 + 396 = 408$ cycles

Solution Schedule 2

Similarly for schedule 2

$12 + 4(99)$

$12 + 396 = 408$ cycles

It's interesting how execution time is sensitive to MII and not the schedule length

Friday Solution

$$ResMII = \max\left(\frac{4\ ALU}{1}, \frac{2\ MEM}{1}\right) = 4$$

$$RecMII = \frac{2+1}{1} = 3$$

$$\max(4,3) = 4 = MII$$

unrolled schedule

	ALU	MEM
0	I2	I1
1	x	
2		
3	x	
4	x	x
5	I3	
6		
7	x	I4
8	x	x
9	x	
10	I5	
11	I6	

rolled schedule

0	I2	I1
1	I3	
2	I5	
3	I6	I4

cycles required for 100 iterations

$$12 + 4(99) = 12 + 396 = 408$$

↑ explanation on Wednesday solution