EECS 583 – Winter 2024 – Midterm Exam

Wednesday, Mar 20, 2024
Exam duration: 1 hr 20 min
Open book, open notes

Name: _____

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 10 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: Long Answer
5 questions, 75 pts total Score:_____

Total (100 pts): _____

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

1) Is branch profile information useful to consider when performing LICM? Answer Yes or No and **briefly explain**. (5 pts)

2) Which two phrases below describe Available Expressions analysis? Circle the **2 best** answers below. (5 pts)

Bottom Up

Top Down

Any Path

All Path

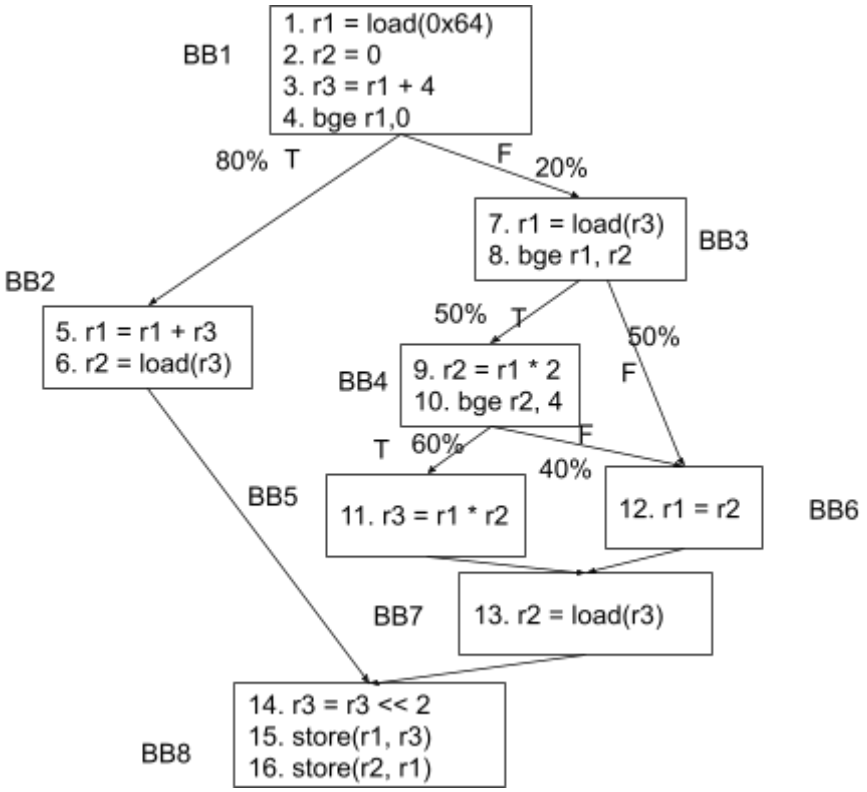
3) Suppose basic block X dominates basic block Y in function run(). When run() is executed, is it ever possible to visit X *after* visiting Y? Answer Yes or No, and **briefly explain**. (5 pts)

4) Using Homework 2 as inspiration, would frequent-path constant propagation provide any advantages over traditional constant propagation? **Briefly explain with a small code example.** (5 pts)

5) Name **one benefit and one cost** of compile time loop unrolling? (5 pts)

Part II. Longer Problems (Questions 6–10) (75 pts)

6) For the following control flow graph (CFG): (15 pts)



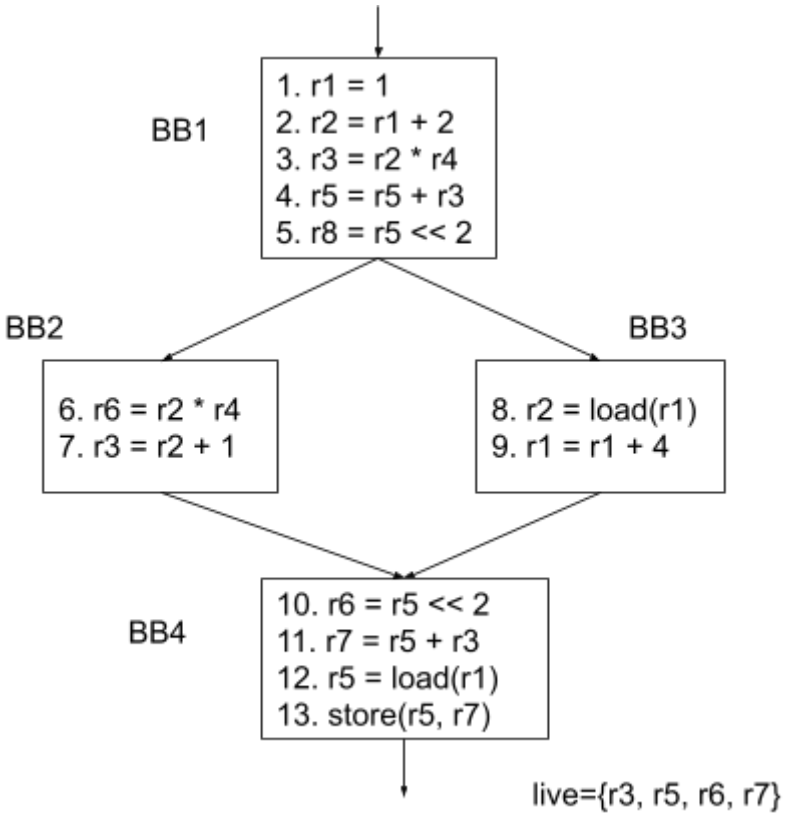
a. Compute the minimum number of predicates needed to if-convert the code. (5 pts)

Minimum Predicates Needed = _____

b. We profiled the code and found that BB1 was executed **100** times and recorded the edge probabilities of conditional branches ('bge' instruction) and labeled the edges with 'T' for true (branch taken) and 'F' for false (branch not taken). In modern CPUs, a branch predictor is a unit that will decide which path to follow when encountering a conditional branch. Assuming each non-branch instruction (including CMPP) takes **1** unit of time to execute. Each branch ("bge") instruction takes **1** unit of time when the branch predictor is correct, and takes **12** units of time if it predicts incorrectly. Assume CMPP instructions can only compute 1 predicate at a time (CMPPs do not have 2 destinations). Ignore instructions for predicate initialization and unconditional jumps that are not shown in the CFG.

If our branch predictor **always predicts not taken ('F')**, would if-converting the entire CFG offer performance benefits? Justify your answer **numerically** (10 pts)

7) Consider optimizing the code segment below. Assume all registers are initialized before entering BB1 and the liveness after exiting BB4 is listed below BB4.



a. Which instructions can be optimized by Common Subexpression Elimination (CSE)? List the instruction numbers. Do not apply other optimizations (12 pts)

b. Name one more optimization that can be applied beyond the CSE done in part (a). (3 pts)

8) There are 9 instructions in a basic block (BB) for which we calculated the Estart and Lstart values using the instruction latencies specified below. However, due to a glitch, the original instruction order got lost and now we want to piece it back together.

a. We remember that the **store instruction** was the last instruction of the BB and has the largest Estart and Lstart values that are specified in the table.
b. Assume all operands used as addresses are valid, but all registers must be defined before being used to ensure correctness (e.g., no registers are live into the BB).
c. 6 of the 9 instructions have slack 0, while the others have slack > 0.

Determine the missing Estart/Lstart values in the table below for the original ordering. Remember, the numbers in the table do not represent the original order. Hint: Try finding the critical path and drawing the dependence graph. (15 pts)

Instruction Latencies

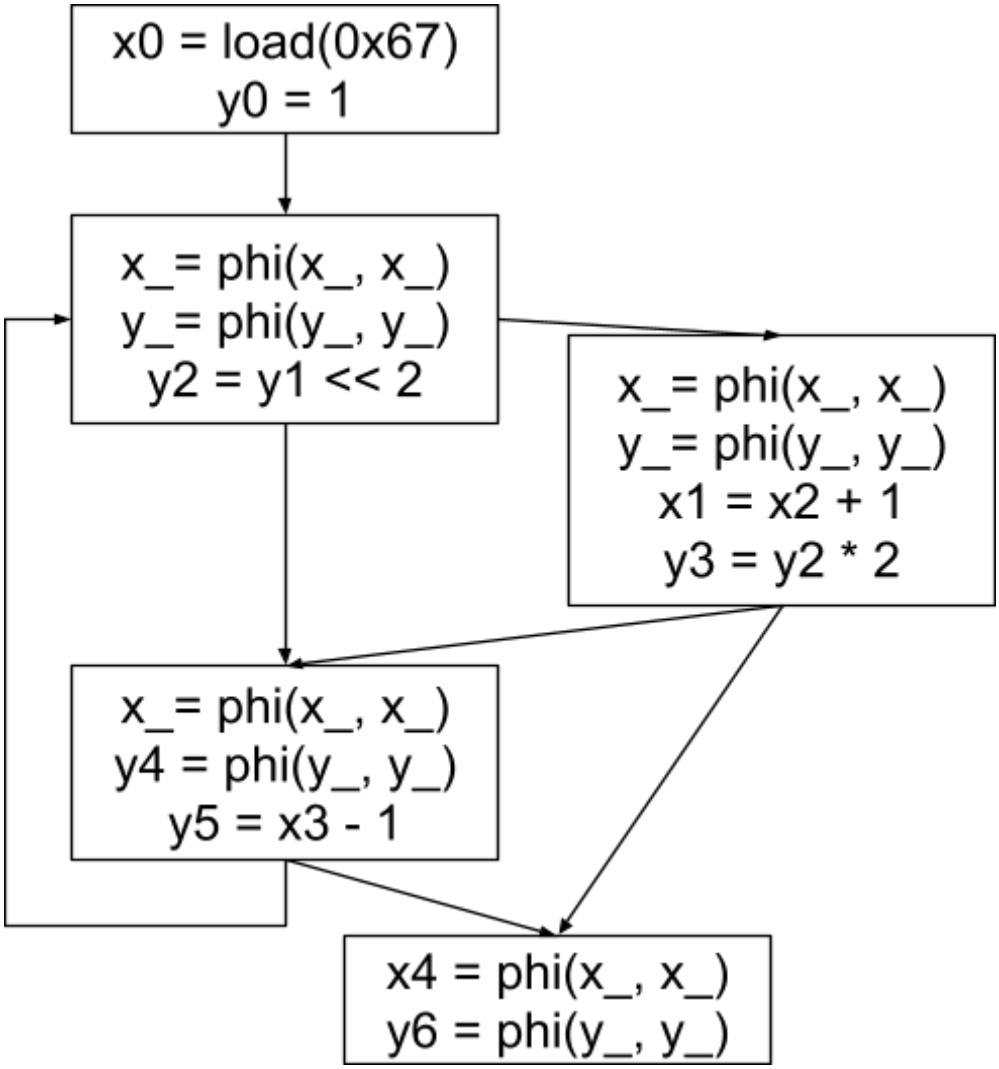
Load/Store: 2
Add: 1
Multiply: 3

No.	Instruction	EStart	LStart
1	r2 = load(0xDF2333)		
2	r6 = load(0xBF4363)		
3	r3 = load(0xCF3537)		
4	r9 = r2 + r3		
5	r2 = r5 + r6		
6	r4 = r3 + 6		
7	r3 = r5 * 3		
8	r5 = r2 * r4		
9	store(0xA23455, r9)	**10**	**10**

.

.

9) Fill in the blanks below to put the code into the static single assignment (SSA) form. Leave phi nodes blank when they are unnecessary. Choose operands sequentially from the lists x0, x1, … xn, and y0, y1, … ym, using each operand only after all its preceding operands in the sequence have been used. Repetition of registers used is permitted. For phi nodes, place the operand in the **first** position if it originates from the **left** edge, and in the **second** position if it originates from the **right** edge. (15 pts)



10) 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 unit and the ALU and branch instructions use the ALU units. (15 points)

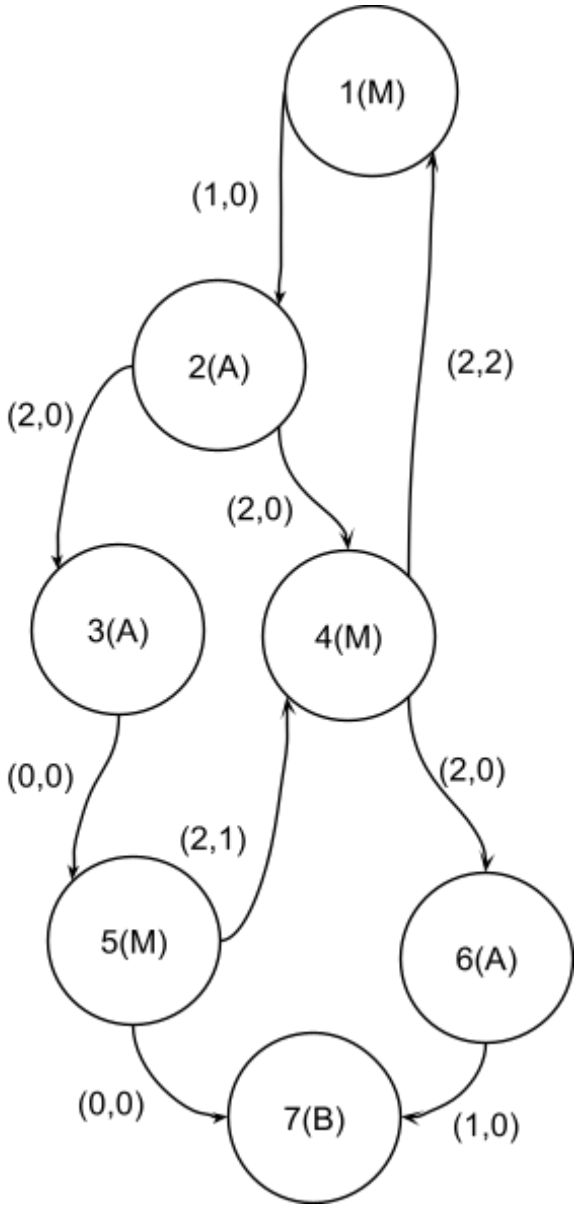
a. Determine the MII. **Show your work and fill out the following:** (5 pts)

Cycles (for RecMII): ResMII = _____

RecMII = _____ MII = _____

b. Generate the rolled and unrolled schedules using this MII. Assume **lower instruction numbers have a higher priority, i.e. instruction 1 has the highest priority**. (10 pts)

Processor Model
3 fully pipelined functional units
– 2 ALUs
– 1 MEM unit



Unrolled Schedule (may contain extra rows)

	ALU0	ALU1	MEM
0			
1			
2			
3			
4			
5			
6			
7			
8			

Rolled Schedule (may contain extra rows)

	ALU0	ALU1	MEM
0			
1			
2			
3			