EECS 583 – Fall 2022 – Midterm Exam

Wednesday, November 2, 2022
Time constraint: 1hr 30min
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 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
6 questions, 30 pts total Score:_____

Part II: Medium Problems
5 questions, 70 pts total Score:_____

Total (100 possible): _______

Part I. Short Answer (Questions 1-6) (30 pts)

1) Name a forward and backward dataflow analysis that we discussed in class. (5 pts)

Forward analysis: __________________

Backward analysis: __________________

2) What is the main purpose of a compiler identifying and co-locating hot blocks of code together as done with trace selection? (5 pts)

3) Can 2 different basic blocks in a function have the same control dependence sets (CD sets) that are not null/empty? If yes, draw a simple CFG to illustrate. If no, briefly explain why not. (5 pts)

4) Why does loop invariant code motion (LICM) generally improve performance? (5 pts)

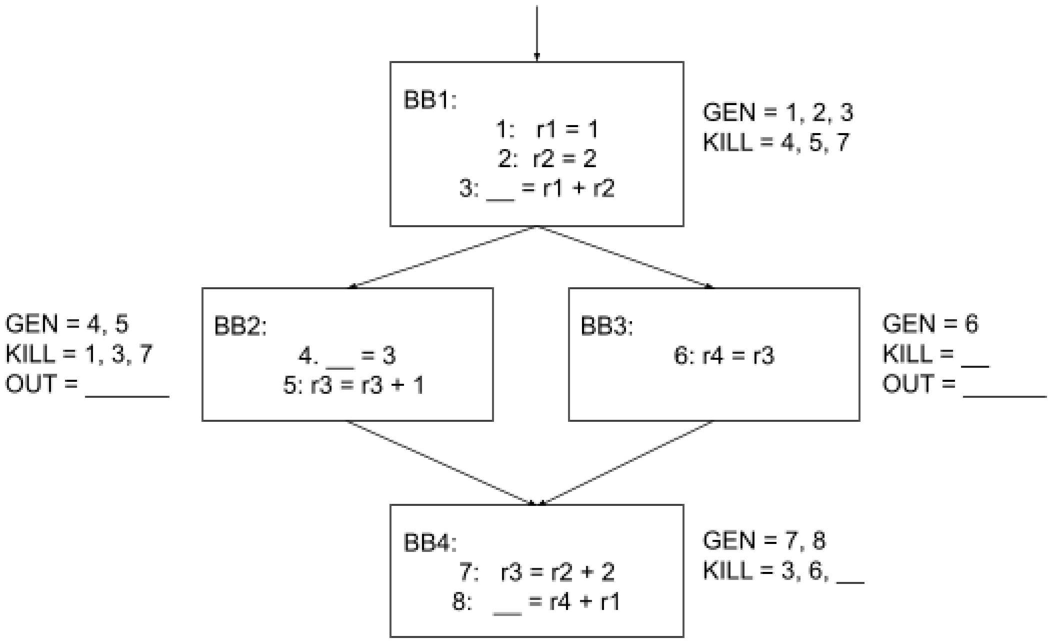
5) When the compiler wants to move an instruction above a branch (e.g., speculate), why is it important to check the liveness constraint? (5 pts)

6) Optimize the following basic block by applying **common sub-expression elimination (CSE)**. Don't apply other optimizations. Write down only the modified instructions with their corresponding instruction number in the same format as the following basic block (5pts)

```
1. r1 = 2
2. r2 = r1 + 2
3. r3 = r1 + r2
4. r4 = r1 + 2
5. r2 = load(r5)
6. r7 = r1 + 2
7. r1 = r4 * r3
8. r8 = r1 + 2
```

Part II. Medium Problems (Questions 7-11) (70 pts)

7) Given the following control flow graph and partial reaching definition (Rdef) analysis results for BB1, BB2, BB3 and BB4, fill in the missing operands and KILL/OUT sets to satisfy the Rdef result. For missing source/destination operands, use registers r1, r2, r3, or r4 **at most once**. For missing KILL/OUT sets, use instructions 1 to 8, repetitions are possible. (15 pts)

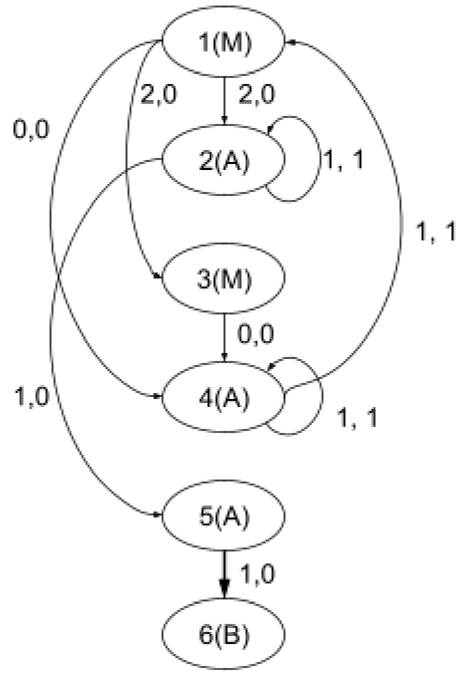


8) Given the loop dependence graph and the processor model below, answer the following questions related to modulo scheduling. (15 pts)
(a) Is the loop resource or recurrence constrained? Justify your answer. (5 pts)
(b) Generate both unrolled and rolled schedules for MII = 3. (10 pts)

For scheduling, you can assume instruction 1 is the highest priority, 2 is the second highest priority, etc. You do not need to assign staging predicates.

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

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



Unrolled Schedule (may contain extra rows):

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

Rolled Schedule:

	ALU0	ALU1	MEM
0			
1			
2			

9) There are 7 instructions in a basic block (BB) and a student has computed the Estart and Lstart values for a subset of instructions using the instruction latencies specified below. Due to a corrupt disk, the original order of the instructions was lost and the instructions got randomly ordered. The student reassigns the number of each instruction and knows the corresponding partial Estart and Lstart values (see Table below). It is also known that Instruction 7 (r2 = r6*2) is the last instruction of the BB and has the largest Estart and Lstart values.

Determine the original order of the instructions using the partial Estart/Lstart values and complete the missing Estart/Lstart values in the table below for the original ordering. Remember, the instruction numbers do not represent the original order. (15pts)

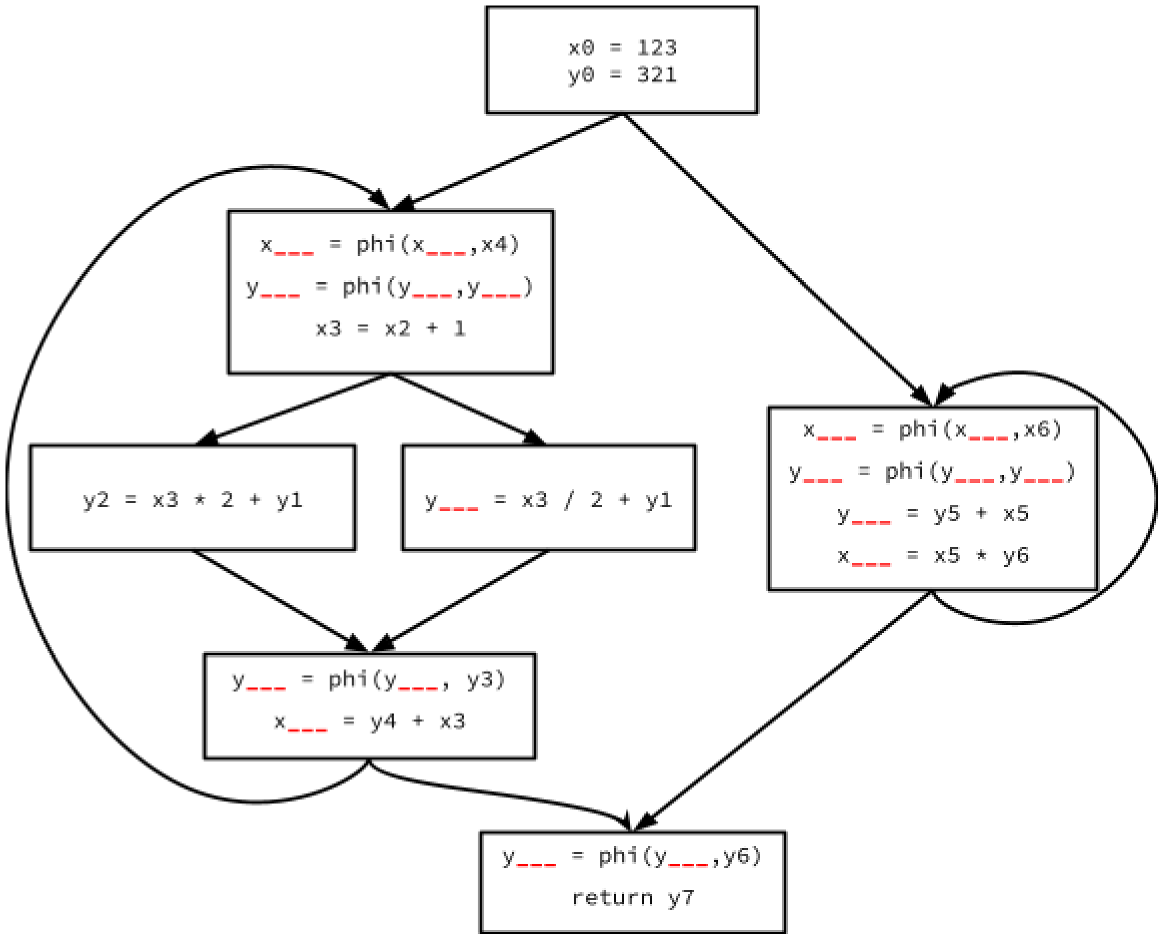
Instruction latencies

add: 1
mul: 3
load: 2

#	Instruction	Estart	Lstart
1	r3 = r5 * r1	3	3
2	r2 = r3 + 1	6	7
3	r5 = load(r5)		
4	r6 = load(r3)		
5	r5 = r5+1	2	2
6	r1 = load(r4)		1
7	r2 = r6 * 2	8	8

Original instruction order:

10) Satisfy static single assignment (SSA) form by filling in the blanks in the code segment below. Remember, the result and arguments of a Phi node must be different instances of the same variable (i.e., x1 = Phi(x2, x3)). Note that some Phi nodes may be unnecessary and should be left empty. For your answers, choose operands from x0 to x6 and y0 to y7. Note that operand names may be used multiple times. (15 pts)



11) Given the following if-converted code, convert it to a corresponding control flow graph (CFG). Hint: your answer should have 9 basic blocks. (10pts)

Recall that the format for cmpp instruction is as follows:

p1, p2 = CMPP.D1a.D2a(cond) if p3, where

p1 = first destination predicate
p2 = second destination predicate
D1a = action specifier for first destination
D2a = action specifier for second destination
cond = compare condition
p3 = guarding predicate

```
n = load(param1_addr) if T
i = load(param2_addr) if T
retval = 0 if T
p1, p2 = cmpp.UN.UC(n<=0) if T
temp = n if p1
p3, p4 = cmpp.UN.UC(temp==2) if p1
retval = 1 if p3
retval = -1 if p4
temp = n % i if p2
p5, p6 = cmpp.UN.UC(temp == 0) if p2
retval = 1 if p5
temp = i * i if p6
p7 = cmpp.UN(temp > n) if p6
retval = 1 if p7
return retval if T
```