EECS 583 – Fall 2021 – Midterm Exam

Wednesday, November 3, 2021
Time constraint: 1hr 45min
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.

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) What is the main difference between reaching definitions and available definitions? (5 pts)

Reaching definitions is any-path analysis thus uses union for the Meet function, whereas available definition is all-path analysis thus it uses intersect for the Meet function.

2) Does the order you process instructions in a basic block to compute GEN/KILL matter for a given dataflow analysis? Yes/No and briefly explain. (5 pts)

Yes, there are forward analyses (e.g., reaching definitions) where GEN/KILL are computed sequentially from the first instruction to the last instruction in each BB and backward analyses (e.g., liveness) where GEN/KILL are computed sequentially from the last instruction to the first instruction in each BB. Note there is no ordering among the BBs, GEN/KILL can be computed for BBs in arbitrary order.

3) Is it possible to unroll a loop with a statically (compile-time) unknown number of iterations? Yes/No and briefly explain. (5 pts)

Yes. There are 2 options. First, just insert conditional exit branches (breaks) at the end of each unrolled body, which is the reverse condition of the original loop back branch, i.e., i >= 100 for the original loop back branch of i < 100. Second, and the preferred method, is to insert a remainder loop to execute the iterations that are not a multiple of the unroll factor, then the loop back branches can be removed from the unrolled body for all except the last iteration. Note that you can only use the remainder loop approach when the number of iterations is known at run-time just before the loop is entered, i.e., for (i=0; i<N; i++). The remainder loop approach will not work for a pointer chasing loop, i.e., (while ptr!=NULL) ptr = ptr->next.

4) Can performing LICM on an instruction ever hurt performance? Yes/No and briefly explain. (5 pts)

Yes, there are 2 options. First, if the preheader executes more frequently than the basic block in the loop body where the instruction is hoisted from, then LICM will increase dynamic instruction count which would likely hurt performance. Second, LICM may also increase the register pressure by requiring the destination register of the hoisted instruction to be live/occupied for the entire duration of the loop. In loops that require large numbers of registers, then LICM again would likely hurt performance.
(Note that it is not FPLICM)

5) The 90/10 rule says that 90% of an application's execution is spent in 10% of the code. Give an example of how a compiler can exploit this rule to improve performance. (5 pts)

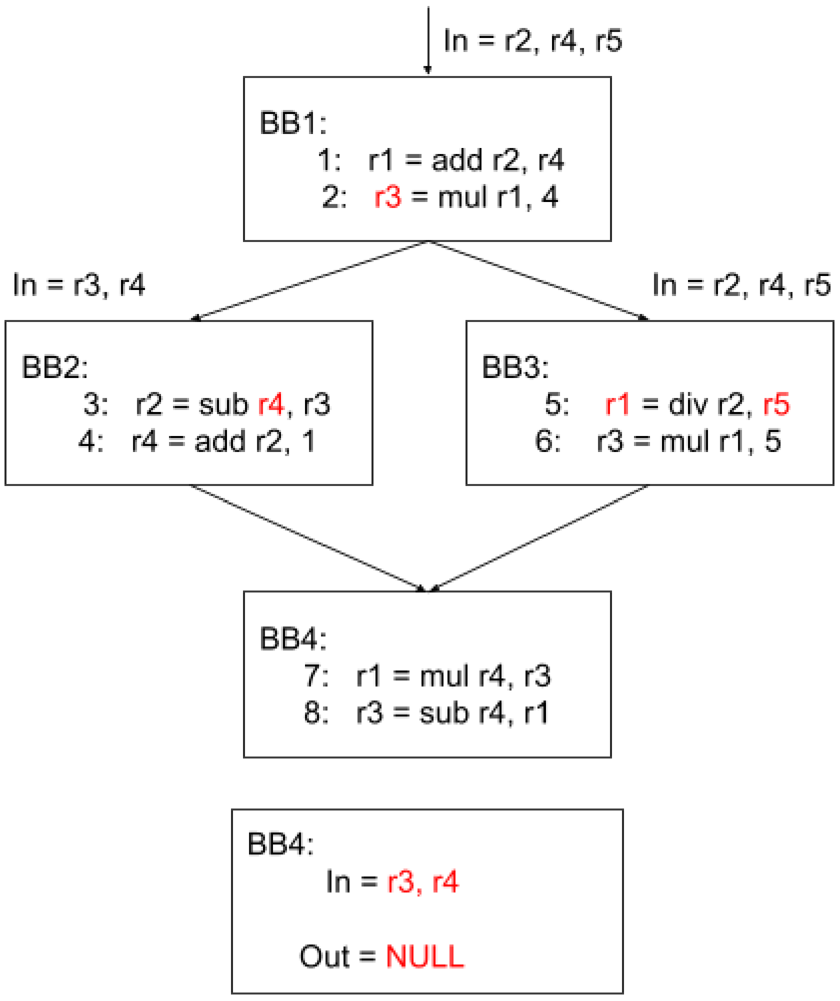
Many answers are possible here with the common theme of enabling better optimization of hot code at the possible expense of cold code. Use profile information to form traces so that hot paths are placed sequentially in memory to optimize instruction cache behavior. Cold code can also be placed separately from hot code to optimize the instruction cache. Superblocks can be formed and scheduled so the hot code gets better instruction overlap than the cold code. Similar to homework 2, profile information can be used to optimize frequent paths by performing optimizations (i.e., LICM) when the optimization conditions are only satisfied along the hot paths and not along the cold paths thereby improving performance of the hot code.

6) Can the Estart for an instruction in a basic block ever be larger than its Lstart? Yes/No and briefly explain. (5 pts)

No, the Estart is the earliest start time you can schedule an instruction ignoring resource constraints and reflects the maximal dependence distance from the start of the BB, while the Lstart is the latest time you can schedule an instruction ignoring resource constraints such that the basic block can finish by its maximal Estart time. Since Lstart is computed from the bottom of the dependence graph using the maximal Estart time, the smallest Lstart for any instruction is it's Estart, and of course Lstart can be larger than Estart for instructions not on the critical path.

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

7) Given the following control flow graph and liveness information for BB1, BB2, and BB3, compute the Liveness IN/OUT sets for BB4, and fill in the missing operands to satisfy the Liveness analysis result. You should use each register r1, r2, r3, r4, or r5 **at most** once for specifying the missing source/destination operands. (15 pts)



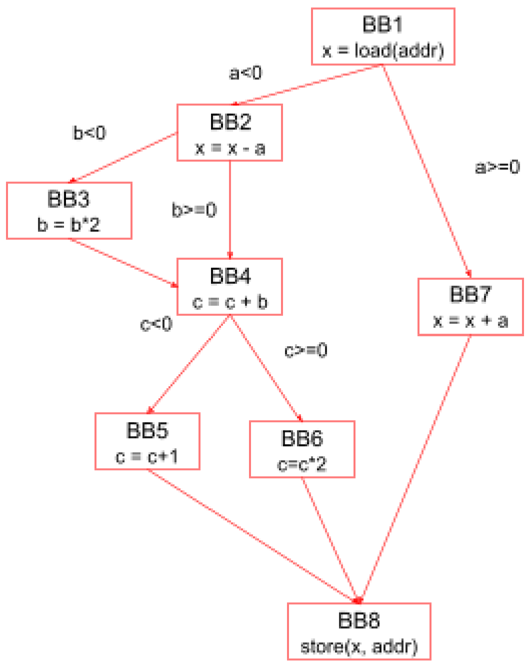
8) Given the following if-converted code, draw the original CFG graph indicating the home location of all arithmetic/load/store instructions. Hint: the original CFG should have 8 BBs. (10 pts)

Recall that the format for cmpp instruction is as follows:

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

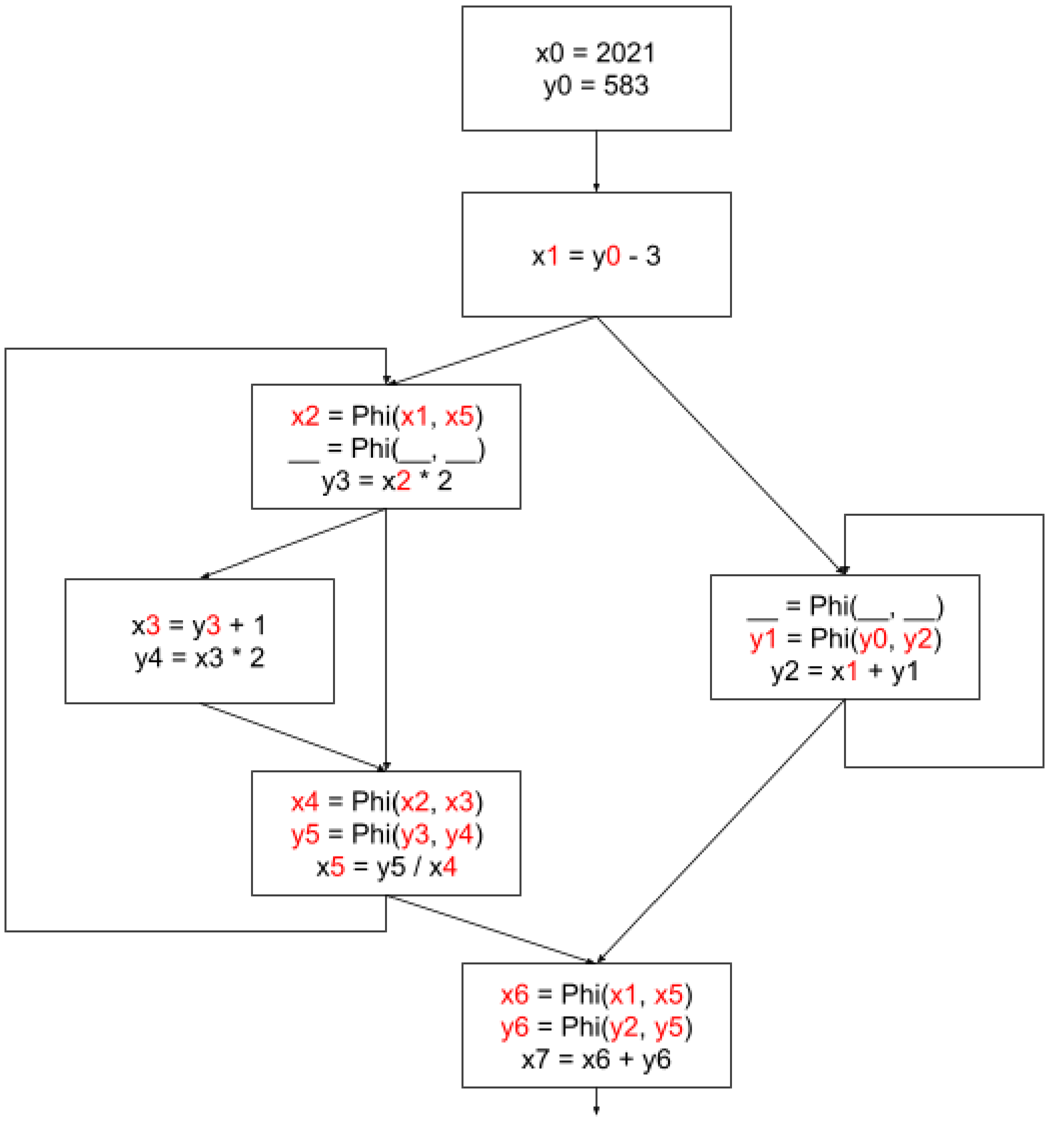
p1 = first destination predicate
p2 = second destination predicate
D1a = action specifier for first destination
D2a = action specifier for second destination
cond = compare condition
p3 = guarding predicate

```
x = load(addr)
p1, p2 = cmpp.UN.UC(a<0) if T
p3 = cmpp.UN(b<0) if p1
x = x-a if p1
x = x+a if p2
b = b*2 if p3
c = c+b if p1
p4,p5 = cmpp.UN, UC(c<0) if p1
c = c*2 if p5
c = c+1 if p4
store(x, addr) if T
```



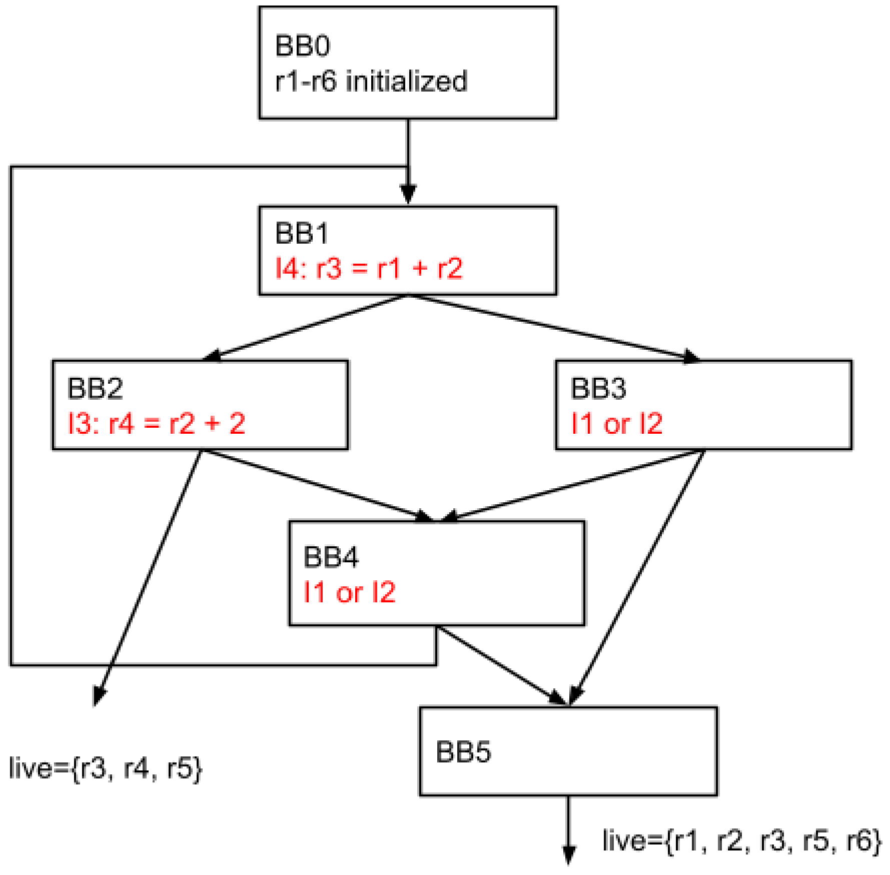
9) Satisfy static single assignment (SSA) form by filling in the blanks in the code segment below. Remember, the result and arguments of a Phi node must be different instances of the same variable (i.e., x1 = Phi(x2, x3)). Note that some Phi nodes may be unnecessary and should be left empty. For your answers, choose from x0 to x6 and y0 to y6. (15 pts)

1



10) You want to apply Loop Invariant Code Motion (LICM) to the CFG below. Insert the following instructions I1-I4 into BB1-BB4 with a maximum of 1 instruction added to each BB (i.e., one instruction in BB1, one in BB2, etc.) so that LICM can hoist as many instructions as possible. Just mark on the CFG below where the instructions should be placed and indicate whether they can be hoisted to the preheader. For those instructions that could not be hoisted, specify a reason. (15 pts)

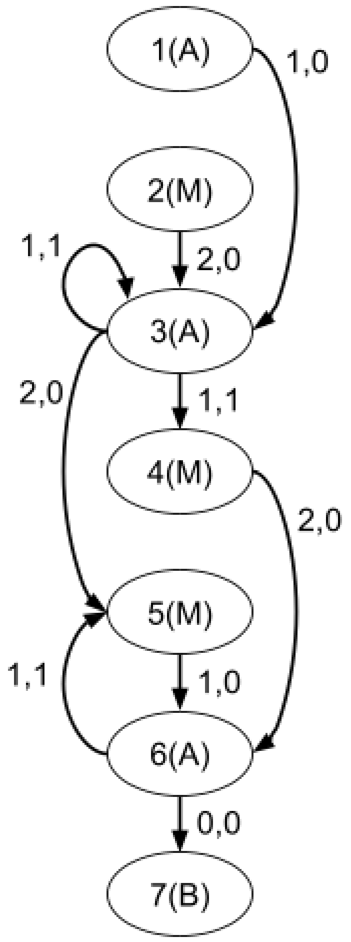
I1: r6 = r6 + 1
I2: r5 = r3 * r1
I3: r4 = r2 + 2
I4: r3 = r1 + r2



I3 and I4 can be hoisted by LICM.
I1 cannot be hoisted because it is a self-increment and makes it not a loop invariant
I2 cannot be hoisted because r5 is live in both exits "for all exit BB, if dest(X) is live on the exit edge, X is in the available defs set on the edge" according to lecture
Some will think about putting I2 in BB1 so it can be hoisted, but one of the src operands of I2 is r3. If and only if I2 is in BB1 could make I2 loop invariant. Since we have the constraint that BB1-BB4 has one instruction each, I2 cannot be hoisted.
So the best solution that could hoist as many instructions as possible is to put I4 in BB1 and put I3 in BB2.

11) Given the dependence graph and the processor model below, answer the following questions related to modulo scheduling. (15 pts)
(a) Is the graph 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 2, 4, and 5 are memory
Instructions 1, 3, 6, and 7 use the ALU
Instruction 7 is a branch

Unrolled Schedule (may contain extra rows):

	ALU1	ALU2	MEM
0	1		2
1			4
2		3	
3			
4			
5			5
6		6	
7			
8	7		
9			
10			
11			

Rolled Schedule:

	ALU1	ALU2	MEM
0	1	6	2
1			4
2	7	3	5

ResMII = max(4/2, 3/1) = 3
RecMII = max(1/1, (1+1)/(1+0)) = 2
MII = max(3, 2) = 3 $\rightarrow$ resource constraint