EECS 583 – Fall 2022 – Midterm Exam

Wednesday, November 2, 2022
Time constraint: 1hr 45min
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: __reaching definitions, available definitions, or available expressions

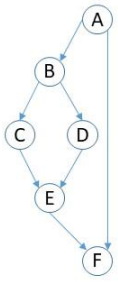
Backward analysis: ____liveness_____

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)

Several answers are possible here: increase sequential instructions by making the fall through paths more likely, better instruction cache behavior, reduced instruction working set size

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)

Yes, in this example, blocks B and E have the same CD set and are not empty. Note blocks A and F have the same sets, but they are both NULL.



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

Reduces dynamic instruction count by moving an instruction from the loop body to the preheader.

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)

To make sure that the destination register is not live and thus would overwrite a live value that is used before being redefined when the branch is taken. Remember, the speculated instruction is executed more often than in the original program, so you have to make sure that when the instruction is executed more often, it does not change program behavior.

6) Optimize the following code by applying common sub-expression elimination. (5pts)

Original

```
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
```

Optimized (according to lecture slides)

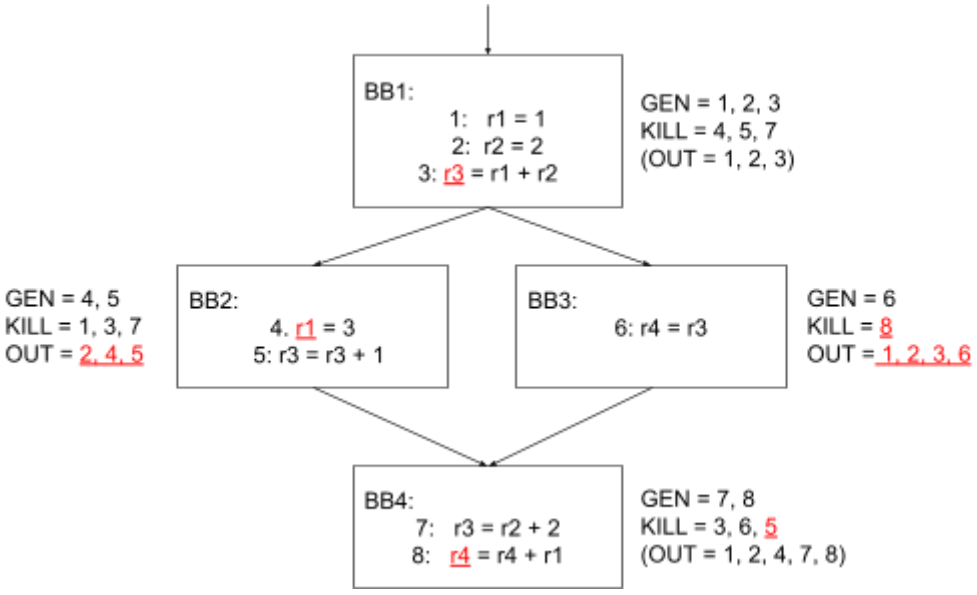
```
1. r1 = 2
2. r2 = r1 + 2
9: r100 = r2
3. r3 = r1 + r2
4. r4 = r100
5. r2 = load(r5)
6. r7 = r100
7. r1 = r4 * r3
8. r8 = r1 + 2
```

Or the following optimized code is also accepted

4. r4 = r2
6. r7 = r4

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

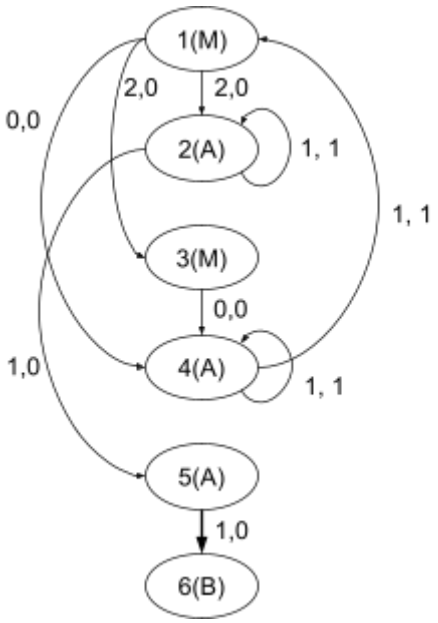
7) Given the following control flow graph and Reaching Definition(Rdef) for BB1, BB2, BB3 and BB4, fill in the missing operands (short underline _) to satisfy the Rdef Analysis result, use register from r1, r2, r3, or r4 at most once for specifying the missing source/destination operands. And complete in the KILL sets for BB3 and BB4, each blank (short underline _) should fill in one number from 1 to 8. Finally, compute the OUT sets for BB2, BB3 and BB4. Each blank (long underline _____) could fill in more than one numbers from 1 to 8. (15 pts)



8) Given the loop in DSA form and dependence graph, answer the following questions related to modulo scheduling. Store instruction follows store(addr, val). The processing model contains 2 ALU and 1 MEM, all are fully pipelined. Inst 1, 3 use MEM; Inst 2, 4, 5 and 6 use ALU (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.

```
Loop:
  1: r2[-1] = load(r1[0])
  2: r3[-1] = r2[-1] + r3[0]
  3: store(r1[0], r2[-1])
  4: r1[-1] = r1[0] + 1
  5: p1[-1] = cmpp (r3[-1] < 10)
  remap r1,r2,r3,…
  6: brlc p1[-1] Loop
```



Unrolled Schedule (may contain extra rows):

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

Rolled Schedule:

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

(a) 1->3->4->1 cycleRecII = (2 + 0 + 1) / (0 + 0 + 1) = 3;
ResII = max (4/2, 2/1) = 2;
MII = max(RecII, ResII) = 3; So it is recurrence constrained.
(b) shown in the table (it's ok to swap ALU0 and ALU1)

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.

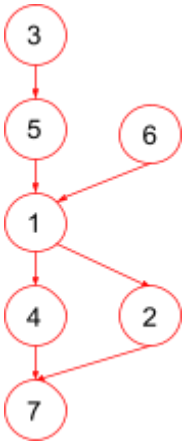
Determine the original order of the instructions using the partial Estart/Lstart values and complete the missing Estart/Lstart values in the table below for the original ordering. Remember, the instruction numbers do not represent the original order. (15pts)

Instruction latencies

add: 1
mul: 3
load/store: 2

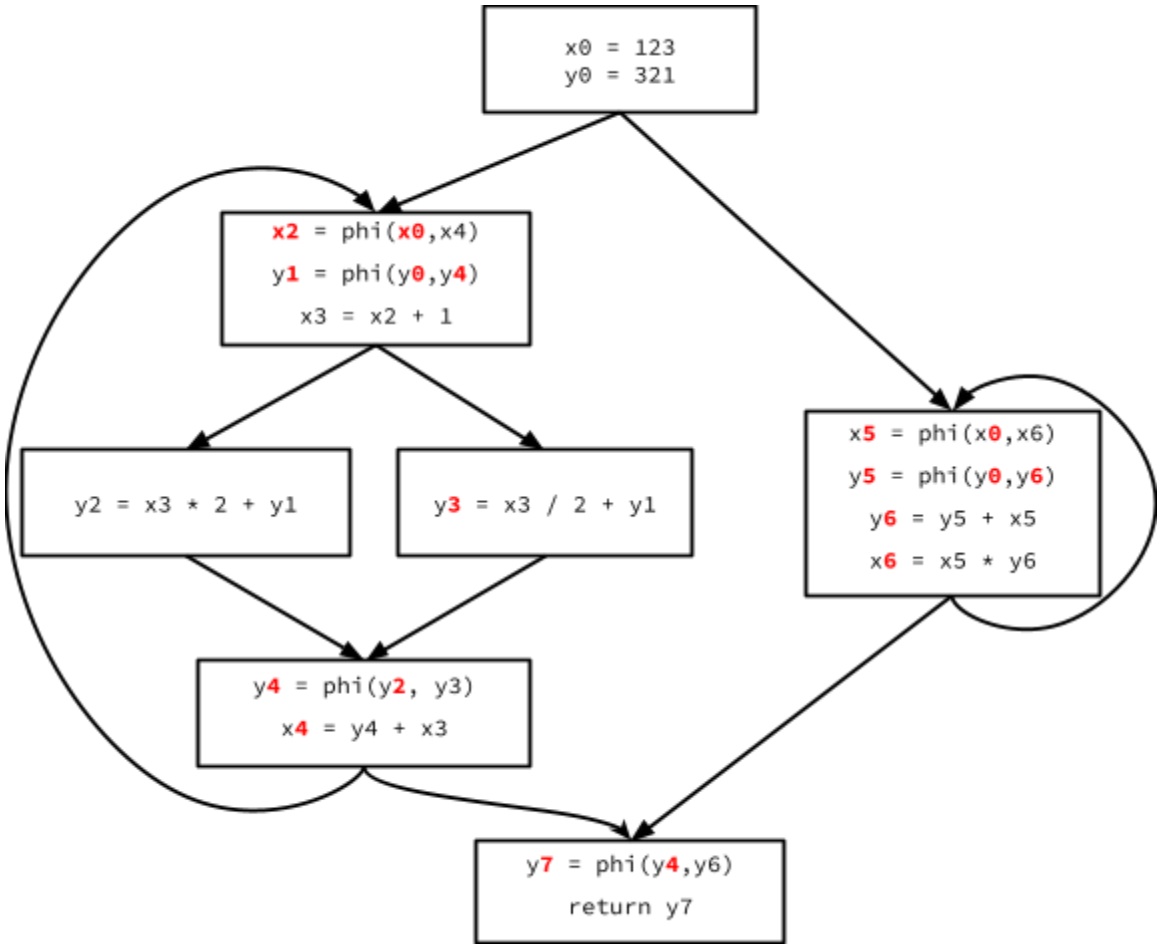
#	Instruction	Estart	Lstart
1	r3 = r5 * r1	3	3
2	r2 = r3 + 1	6	7
3	r5 = load(r5)	0	0
4	r6 = load(r3)	6	6
5	r5 = r5+1	2	2
6	r1 = load(r4)	0	1
7	r2 = r6 * 2	8	8

The dependency graph should look like



Multiple valid answers exist: (Note that 3, 5, 6 … is not valid because Lstart of 6 < Estart of 5)
3, 6, 5, 1, 2, 4, 7;
6, 3, 5, 1, 2, 4, 7;
3, 6, 5, 1, 4, 2, 7;
6, 3, 5, 1, 4, 2, 7;

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)). For your answers, choose from x0 to x6 and y0 to y6. Note that the variable x1 is skipped in this problem.



11) Given the following if-converted code, convert it to a corresponding CFG with 9 basic blocks. (10 pts)

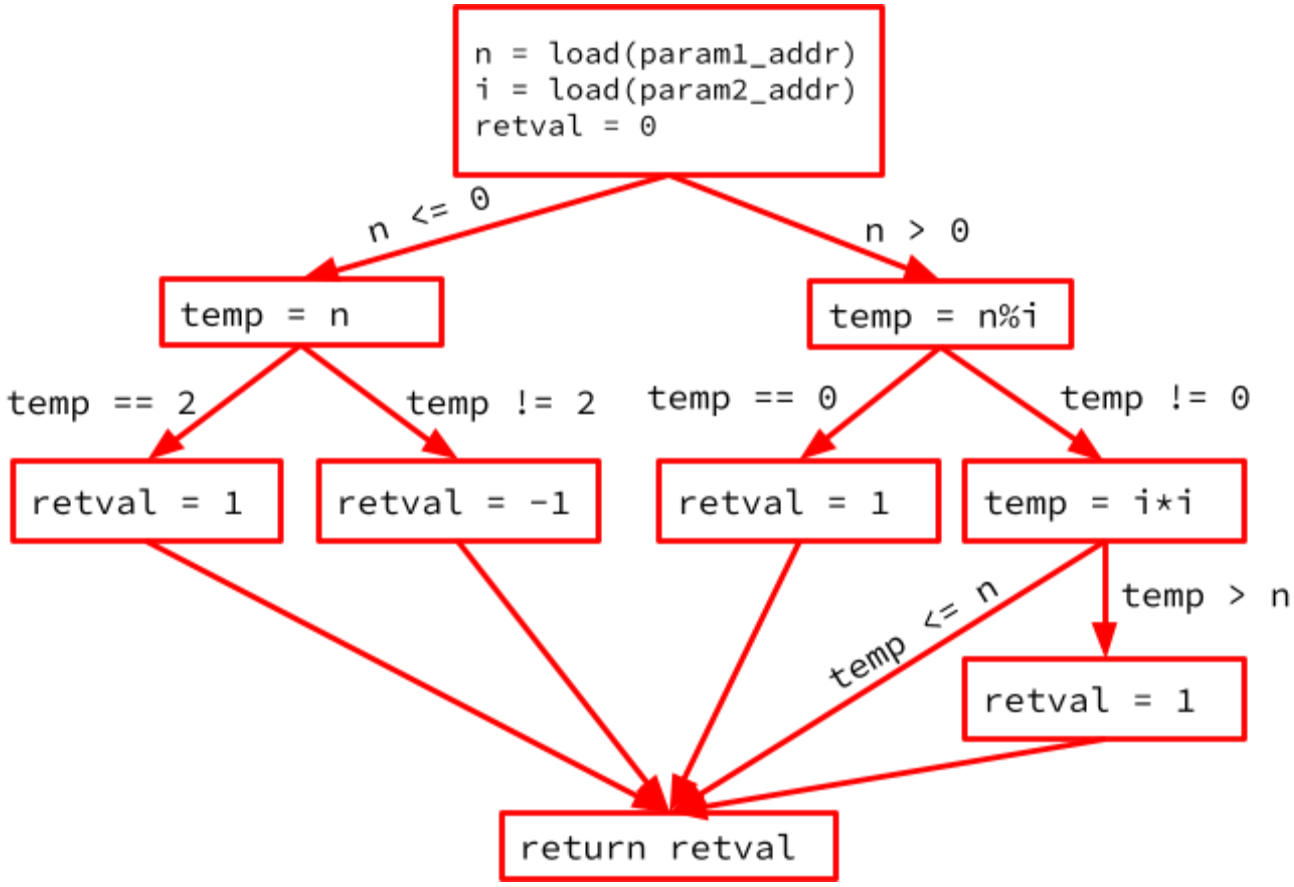
Recall that the format for cmpp instruction is as follows:

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

```
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
```

Solution:


n = load(param1_addr)
i = load(param2_addr)
retval = 0
n <= 0
n > 0
temp = n
temp = n%i
temp == 2
temp != 2
temp == 0
temp != 0
retval = 1
retval = -1
retval = 1
temp = i*i
temp <= n
temp > n
retval = 1
return retval