EECS 583 – Fall 2013 – Midterm Exam

Wednesday, November 20, 2013; 7:00-9:00pm
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 12 questions divided into 3 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, 15 pts total Score:_____

Part II: Medium Problems
5 questions, 55 pts total Score:_____

Part II: Longer Problems
2 questions, 30 pts total Score:_____

Total (100 possible): _____

Part I. Short Answer (Questions 1-5) (15 pts)

1) Name a dataflow analysis discussed in class that is *forward* and *any* path (3 pts)

Reaching definitions

2) Name one advantage that superblocks provide a compiler over traditional basic blocks (3 pts)

Several answers are possible here: Larger scope for scheduling to find more ILP, Isolate hot paths from cold paths for increased optimization opportunities, Improved icache performance due to sequential layout of hot path.

3) Name two optimizations that exhibit *constructive interference*, i.e., the application of one optimization enables additional opportunities for the other to be applied. (3 pts)

Many answers were accepted for this question: constant propagation & constant folding, CSE & copy propagation, constant/copy prop & induction variable strength reduction, copy propagation & dead code elimination

4) Give a reason why it may be *undesirable* to apply common subexpression elimination (CSE) to an instruction that can be *legally* optimized? (3 pts)

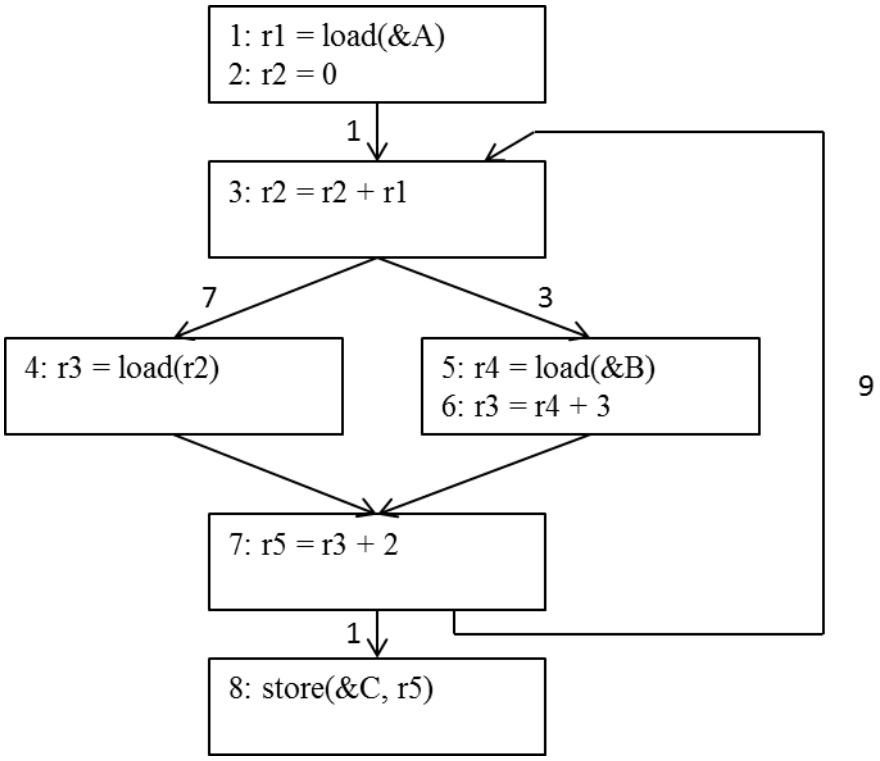
There are several answers for this question as well: increase register pressure as CSE requires preserving the destination of the first instruction for a longer period of time which in turn may lead to additional spill code during register allocation, increased number of dependences as the second instruction is made dependent on the first (or the copy) to remove the redundancy, if the second instruction takes no more time than a copy (say an add), then you eliminate an add for 2 copies.

5) For a processor with infinite resources, a naïve heuristic that *maximally if-converts* acyclic control flow graphs would be seemingly effective. In what situation would this strategy not be effective? Briefly explain your answer (3 pts)

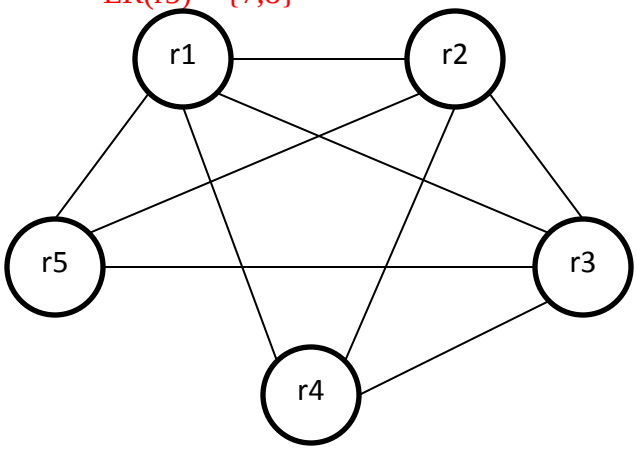
There are 2 possible answers for this one: Imbalanced dependence heights of if-converted blocks increases the dependence height of the shorter path independent of the number of resources, presence of a hazard in one of the blocks that is if-converted can slow down the other paths that are if-converted.

Part II. Medium Problems (Questions 6-10) (55 pts)

6) Consider the following code segment. If *2 physical registers* are available, *how many spills* will occur when virtual registers r1, r2, r3, r4, and r5 are allocated? Assume that &A, &B, and &C are compile time constants and do not require registers. The edges of the control flow graph have been annotated with the profile execution counts. Justify your answer. (10 pts)



LR(r1) = {1,2,3,4,5,6,7}
LR(r2) = {2,3,4,5,6,7}
LR(r3) = {4,6,7}
LR(r4) = {5,6}
LR(r5) = {7,8}



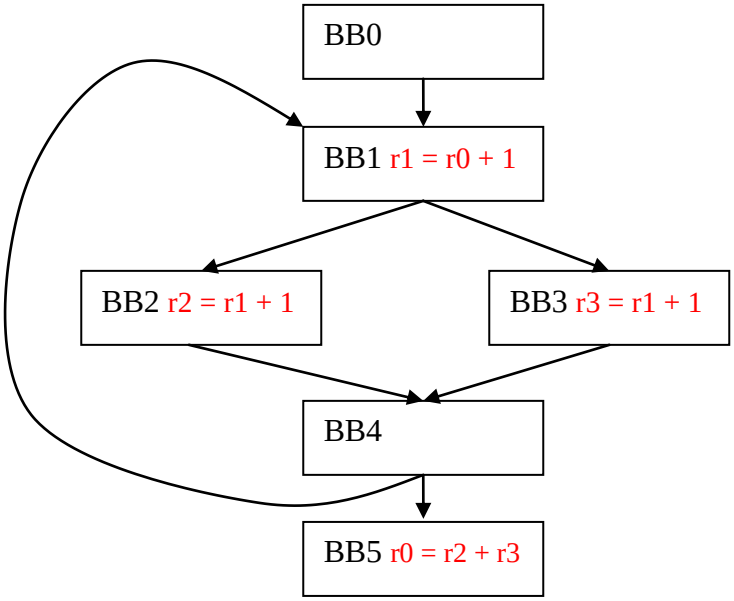
stack: r3 (removed)
r2 (removed)
r5 (spilled)
r4 (spilled)
r1 (spilled)

3 spills

	r1	r2	r3	r4	r5
Cost	11	18	20	6	11
Neighbors	4	4	4	3	3
c/n	2.75	4.5	5	2	3.67

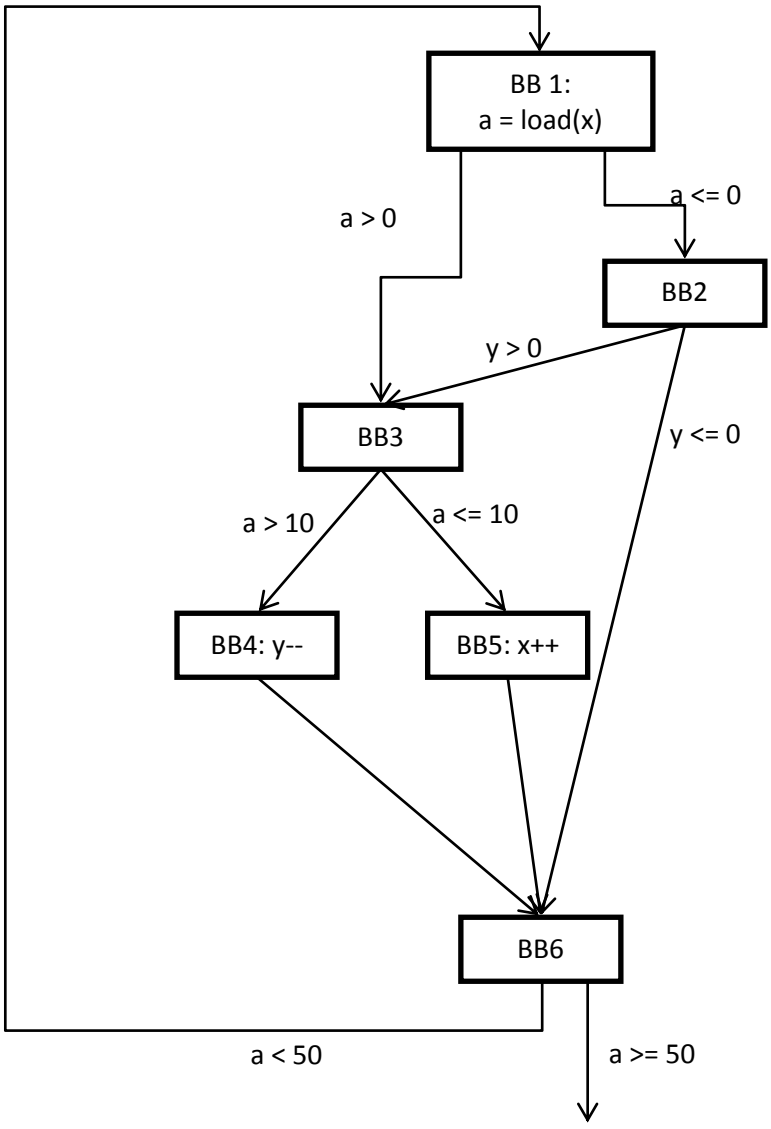
7) In the following control flow graph (CFG), add 4 instructions such that the following conditions are met: each instruction is in a different basic block, each instruction sources the destination register of one of the instructions (possibly itself) but all destination registers must be sourced at least once, at least 3 of the instructions can be removed from the loop by LICM. You may assume any relevant registers are properly initialized. (10 pts)

Below is one possible answer – more may be possible. Note, inserting dead instructions of the form rx = rx + 0 into the loop is a creative but not a correct answer as LICM itself should not be assumed to also do dead code elimination.



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

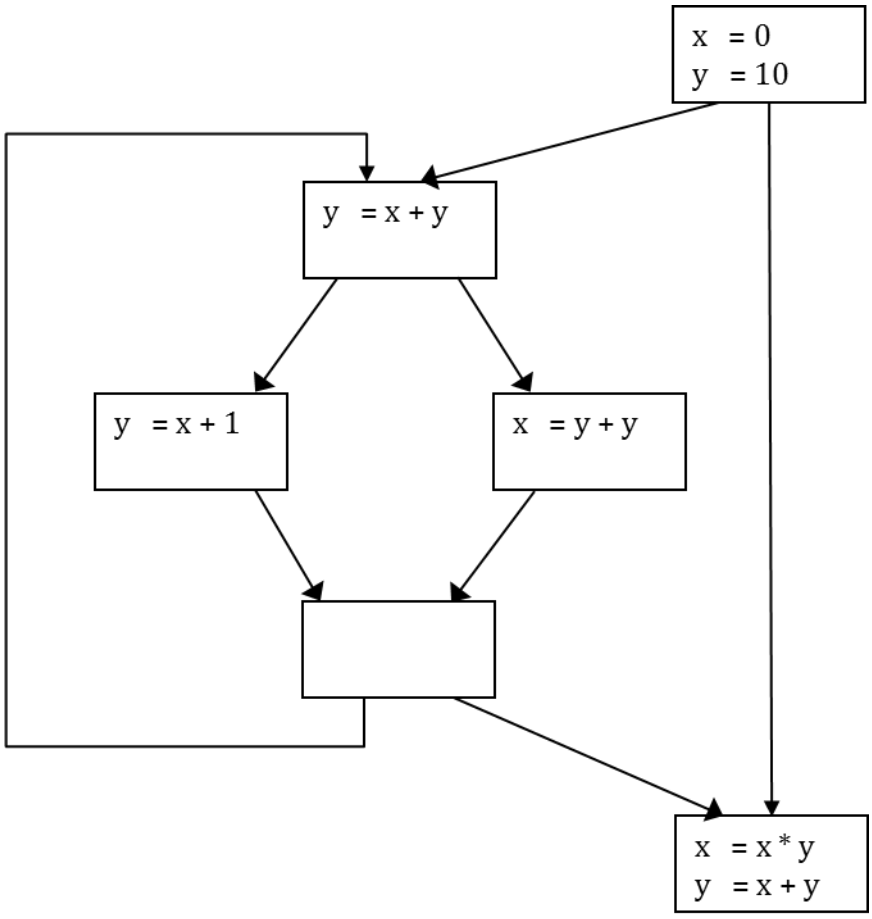
```
do {
  a = load(x)
  if ((a > 0) || (y > 0)) {
        if (a > 10)
           y--;
        else
           x++;
  }
} while (a < 50)
```

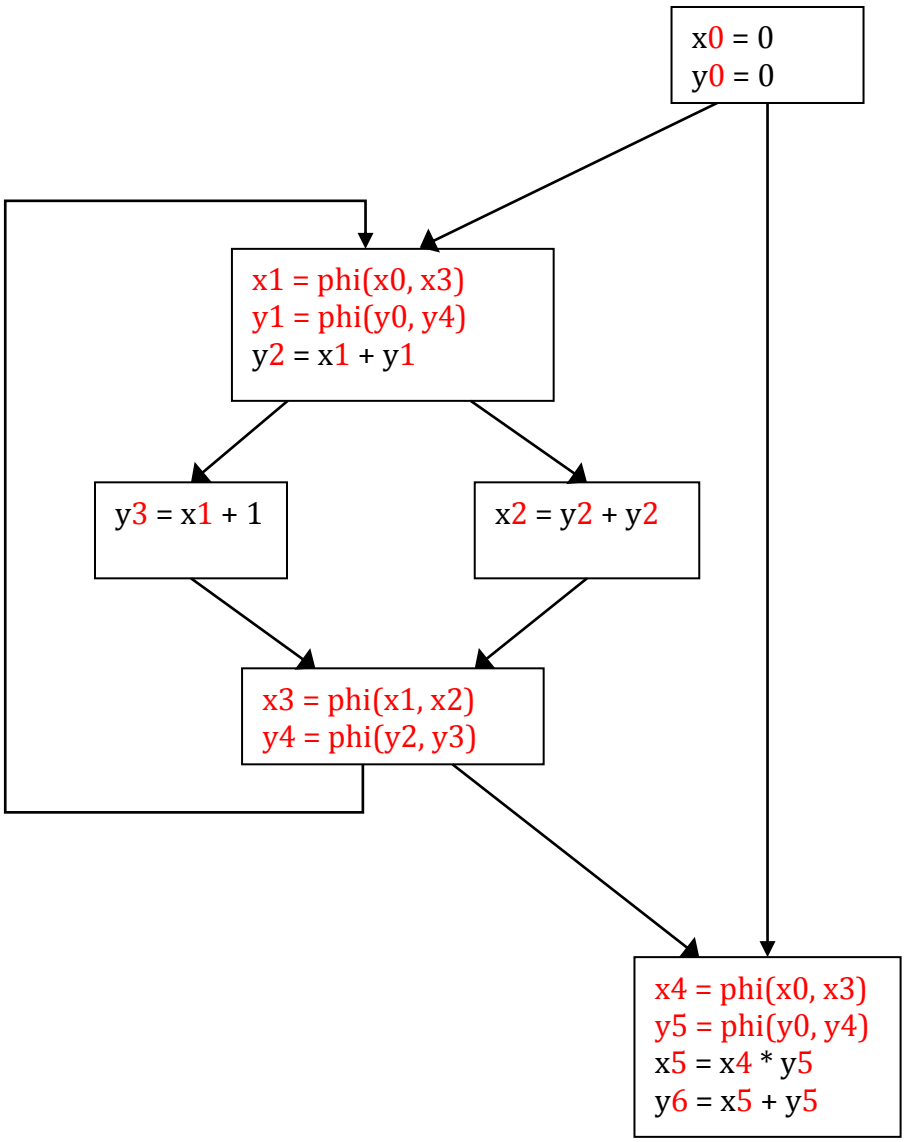


BB	CD
1	0
2	+1
3	-1, -2
4	-3
5	+3
6	0

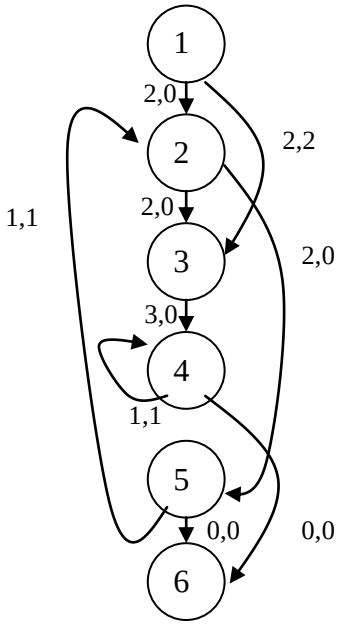
4 unique sets ➔
4 predicates
minimum

8) Convert the following program segment into static single assignment (SSA) form. You should perform the necessary renames and show the Phi nodes. Solving by inspection is fine and you can put your solution directly on the diagram. (10 pts)




x0 = 0
y0 = 0
x1 = phi(x0, x3)
y1 = phi(y0, y4)
y2 = x1 + y1
y3 = x1 + 1
x2 = y2 + y2
x3 = phi(x1, x2)
y4 = phi(y2, y3)
x4 = phi(x0, x3)
y5 = phi(y0, y4)
x5 = x4 * y5
y6 = x5 + y5

9) 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. (15 pts)



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

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

ResMII
For the ALU: 4 operations / 2 units = 2
For the MEM: 2 operations / 1 unit = 2
ResMII = MAX(2, 2) = 2

RecMII
2→5→2: (2+1)/(0+1) = 3
4→4: 1/1 = 1
RecMII = MAX(3,1) = 3

MII = MAX(ResMII, RecMII) = MAX(2,3) = 3

2 schedules were accepted for this problem. The correct schedule is actually difficult to generate as it requires backtracking when instruction 5 is scheduled (assuming instructions 1-4 have already been scheduled) to remove instructions 3&4 which conflict with it. Both schedules are for II=3 and consist of 3 stages.

Correct answer

Rolled schedule

Cycle	Instructions scheduled
0	1
1	3,5
2	2,4,6

Unrolled schedule

Cycle	Instructions scheduled
0	1
1	
2	2
3	
4	3,5
5	
6	
7	
8	4,6

Alternate answer – This answer is not technically correct because it does not properly honor the cross-iteration dependence between instruction 5 and 2. Instruction 5 is issued too late to produce a value for instruction 2 in the next iteration. However, this is the most natural schedule to obtain if you follow the modulo scheduling algorithm presented in class, thus it was accepted as a correct answer.

Rolled schedule

Cycle	Instructions scheduled
0	1
1	3,4
2	2,5,6

Unrolled schedule

Cycle	Instructions scheduled
0	1
1	
2	2
3	
4	3
5	5
6	
7	4
8	6

Part III. Longer Problems (Questions 11-12) (30 pts)

10) Given the following definition of anticipated: *An expression E is anticipated at a point p if every path from p to Exit contains an instruction that evaluates E and is not preceded on that path by an instruction that might kill E*. The idea is to determine how early one could compute an expression in the program before it actually needs to be used.

So for example, at the top of the left block, the expressions r2+r3 and r7-r8 are anticipated, but at the top of the right block only r7-r8 is anticipated.

r4 = r7 - r8 r1 = r2 + r3	r4 = r7 - r8 r6 = r4 + r5

Define the set of dataflow equations to solve for anticipated expressions. You should define GEN, KILL, IN, and OUT. (14 pts)

This problem is equivalent to reverse available expressions, thus it is bottom up and all paths.

GEN/KILL calculation

Initialize GEN/KILL = 0 for all BB
For each basic block, BB
  For each instruction in BB from bottom to top, X
    G = op
    K = {all instructions which source a destination operand of X} – X
    GEN(BB) = (GEN(BB)-K) + G
    KILL(BB) = (KILL(BB)-G)+ K

IN/OUT calculation

Initialize OUT=0 for all BB; IN(BB)={all instructions} – KILL(BB)
While (change)
  For each basic block, BB
    OUT(BB) = Intersect IN(successors of BB)
    IN(BB) = (OUT(BB) – KILL(BB)) + GEN(BB)
    Change = true if IN(BB) was changed from its old value

Your boss has tasked you with reverse engineering a competitor's VLIW processor. You have been provided an application and its resulting compiler schedule. Through some preliminary analysis, some of the machine characteristics have been determined, as provided in the table on the left. But they have not yet determined the complete resource usage of each instruction, which is where you come in. (Note: "?" can be between 0 and the total resources the associated type.) (16 pts)

Opcode	Latency	Resources
Add	1	1X, 0Y, 0Z
Mpy	2	?X, ?Y, 2Z
Load	3	1X, ?Y, 1Z
Store	1	?X, 2Y, ?Z

Total resources: 2X, 3Y, 3Z

Application
1: r1 = load (A)
2: r2 = load (B)
3: r3 = load (C)
4: r4 = r1 + r2
5: r5 = load (r4)
6: r6 = r4 + 1
7: r7 = r6 × r5
8: r8 = r3 + r5
9: r9 = r7 × r6
10: store(r4, r8)

Best schedule	
time	instructions
0	1
1	2
2	3
3	
4	4
5	5,6
6	
7	
8	7
9	8
10	10
11	9

a) Draw the data dependence graph for the application labeling each edge with the latency. You can assume that each source operand is read at time 0 and each destination operand is written at the latency. Note, instructions 1, 2 and 3 do not alias with instruction 10.

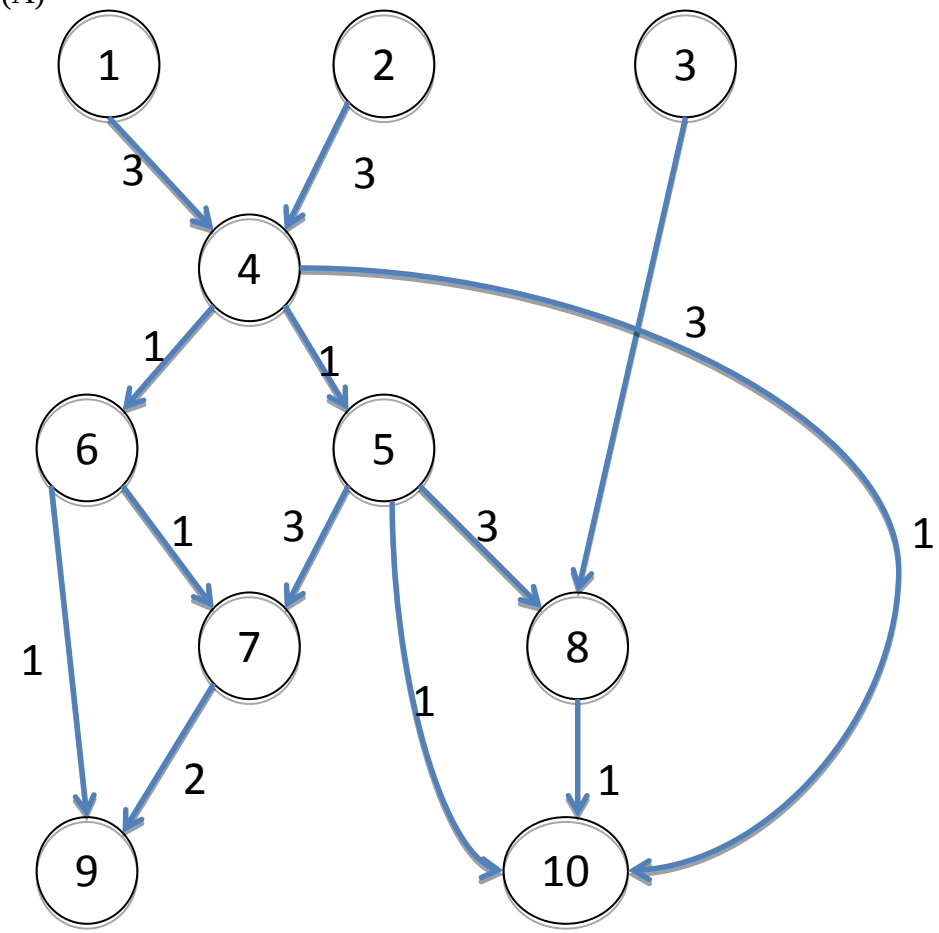
b) What operations are on the critical path? How many cycles does the application require if there were infinite resources?

c) Suppose the schedule on the right is the best that can be achieved for the application on the processor. What can you conclude about the resource utilization of each operation type? Explain your answer.

d) Suppose you can add 1X resource to the processor or reduce the multiply latency to 1 cycle. Which change will increase the performance of the application the most? Explain your answer.

Use the next page for your answers

Answer 12)
(A)



(B) Operations on critical path: 1, 2, 4, 5, 7, 9.
9 cycles with infinite resources.

(C)

- add: all resources are known
- mpy: instructions 7 (mult) and 8 (add) cannot issue at the same time, so there must be a resource constraint → the multiply requires at least 2 X to issue
- load: since two loads can't issue simultaneously (like instructions 1,2), each load must require at least 2 Y resources to issue.
- store: 10 (store) and 9 (mult) cannot issue at the same time. If the multiply requires 2 or more X resources to issue, the store must require at least 1 X resources as well in order to prevent these from issuing simultaneously.

(D) Reducing mpy latency may allow mpys to finish faster, but it does nothing to reduce resource contention. 7 & 8 would still issue at the same times as in the schedule, and while 7 may finish 1 cycle earlier, 9 would not be able to issue any sooner due to 9 & 10's resource contention.

Adding 1X will allow 7 & 8 to be issued at t=8. At t=9, the 10 can issue since 8 is done. The 8 will finish at t=10, allowing 9 to issue at t=10, one cycle earlier. So, adding 1X will have the most impact on application performance.