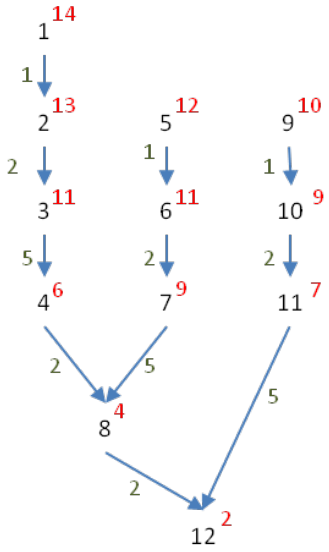
CS4130/CS5130 Homework Assignment #5 Solutions

1. (15 points) Apply list scheduling on the following code, assuming two integer units and one memory unit. The integer operation, loadi, takes one cycle. Integer operations add and mul take 2 cycles. The memory operation, load, takes 5 cycles. Your answer should include

```
1.  loadi   0          ⇒ r1
2.  add     r_arp, r1  ⇒ r1
3.  load    r1         ⇒ r1
4.  add     r1, r1     ⇒ r1
5.  loadi   4          ⇒ r2
6.  add     r_arp, r2  ⇒ r2
7.  load    r2         ⇒ r2
8.  mul     r1, r2     ⇒ r1
9.  loadi   8          ⇒ r3
10. add     r_arp, r3  ⇒ r3
11. load    r3         ⇒ r3
12. mul     r1, r3     ⇒ r1
```

   a. (6 points) a data dependence DDD (To make your DDD easy to read, you can just show the flow/true dependences and the scheduling priority (length of the longest path) of each node),

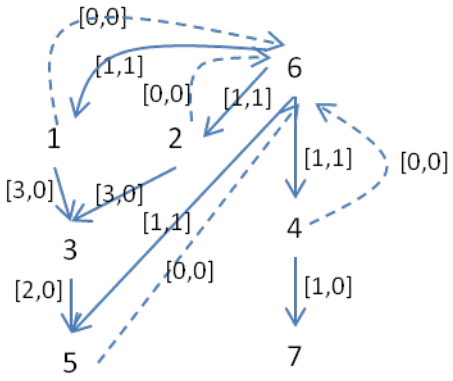


   b. (6 points) a schedule based on your DDD.

cycle	INT	INT	MEM
0	1	5	
1	2	6	
2	9		
3	10		3
4			7
5			11
6			
7			
8	4		
9			
10	8		
11			
12	12		

c. (3 points) Is your schedule optimal? Why?
Yes. The longest path is 14 cycles which is the best we can do. The schedule is 14 cycles.

2. (12 points, CS5130 students only) Problem 9 on Page 676. (Note that $r_{@z}$ should be r_{ctr} at line 6)



In the dependence graph, the solid edges are true dependences and the dashed ones are anti-dependences. Each edge is annotated with the delay and the number of cross iterations.

Seven instruction for 2 units make RC $\left\lceil \frac{7}{2} \right\rceil = 4$ which also satisfies the constraint that the three memory instructions must be on Unit 0.

Based on the dependence graph, the longest recurrence circuit is 1-3-4-6-1 with delay of 6 cycles and total number of cross iterations of 1. So the DC is 6. When picking II=6, scheduling fails and II=7 also fails. The final scheduling is as below while all instructions are at iteration offset 0. No prolog and epilog are needed.

	Unit 0	Unit 1
0	1	4
1	2	
2		
3		
4		3
5		
6	5	6
7	7	

3. (12 points) Apply local bottom-up register allocation on the following code. Assume that K=3 in addition to the register reserved for r_{arp}. Show your work. (Note that "store vr6 ⇒ vr5" means store the value of vr6 into the memory location with address vr5. Both vr5 and vr6 in this instruction are reads).

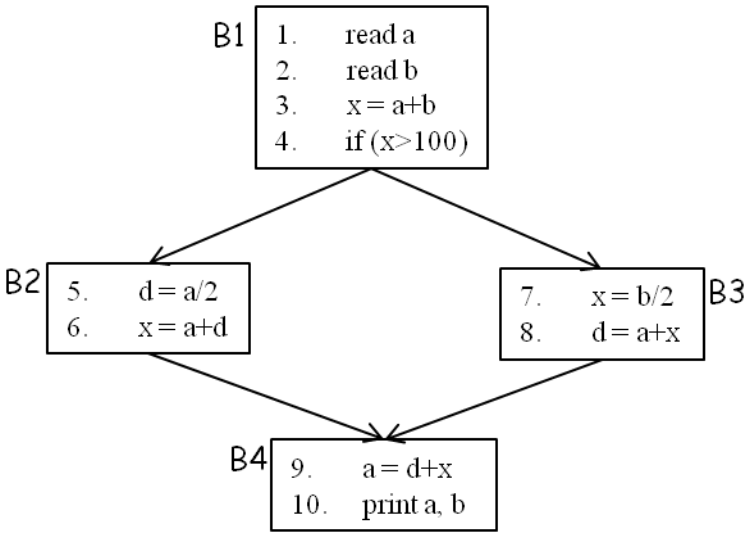
1. loadAI r_{arp}, -4 ⇒ vr1
2. loadAI r_{arp}, -8 ⇒ vr2
3. loadi 4 ⇒ vr3
4. mult vr2, vr3 ⇒ vr4
5. add vr1, vr4 ⇒ vr5
6. loadi 10 ⇒ vr6
7. store vr6 ⇒ vr5
8. loadAI r_{arp}, -12 ⇒ vr7
9. add vr7, vr2 ⇒ vr8
10. mult v8, vr3 ⇒ vr9
11. store vr3 ⇒ vr9

The table blow shows the status of each instruction before and after allocation.

	r1	r2	r3	stack
Name	⊥	⊥	⊥	r3
Next	∞	∞	∞	r2
Free	T	T	T	r1
loadAI rarp, -4 => r3				
	r1	r2	r3	stack
Name	⊥	⊥	vr1	
Next	∞	∞	5	r2
Free	T	T	F	r1
loadAI rarp, -8 => r2				
	r1	r2	r3	stack
Name	⊥	vr2	vr1	
Next	∞	4	5	
Free	T	F	F	r1
loadi 4 => r1				
	r1	r2	r3	stack
Name	vr3	vr2	vr1	
Next	4	4	5	
Free	F	F	F	
store r1 => rarp, off //spill vr3 (r1) mult r2, r1 => r1				
	r1	r2	r3	stack
Name	vr4	vr2	vr1	
Next	5	9	5	
Free	F	F	F	
add r3, r1 => r1				
	r1	r2	r3	stack
Name	vr5	vr2	⊥	
Next	7	9	∞	
Free	F	F	T	r3
loadi 10 => r3				
	r1	r2	r3	stack
Name	vr5	vr2	vr6	
Next	7	9	7	
Free	F	F	F	
store r3 => r1				
	r1	r2	r3	stack
Name	⊥	vr2	⊥	
Next	∞	9	∞	r1
Free	T	F	T	r3
loadAI rarp, -12 => r1				
	r1	r2	r3	stack
Name	vr7	vr2	⊥	
Next	9	9	∞	
Free	F	F	T	r3
add r1, r2 => r2				

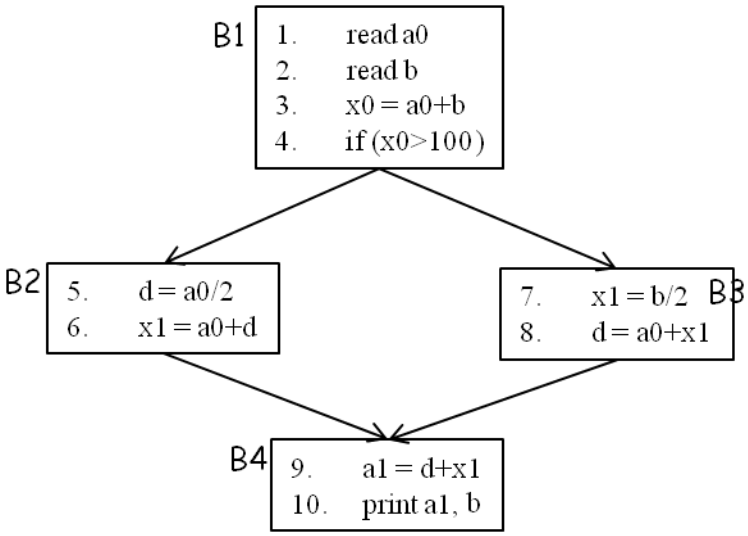
	r1	r2	r3	stack
Name	⊥	vr8	⊥	
Next	∞	10	∞	r1
Free	T	F	T	r3
load rarp, off => r1 // load the spilled vr3 mult r2, r1 => r2				
	r1	r2	r3	stack
Name	vr3	vr9	⊥	
Next	11	11	∞	
Free	F	F	T	r3
store r1 => r2				
	r1	r2	r3	stack
Name	⊥	⊥	⊥	r2
Next	∞	∞	∞	r1
Free	T	T	T	r3

4. (20 points) Consider the control flow graph below. Each statement is labeled by its statement number and each basic block is also labeled as B#.



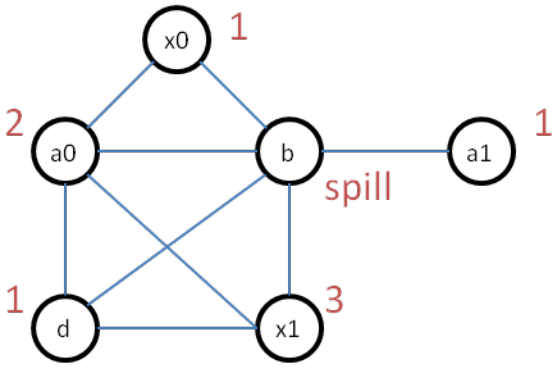
a. What are the live ranges for the bottom-up allocator? Draw the interference graph for bottom-up allocation.

Note that a@1, a@3, a@5, a@6 are connected through DU chains and form one live range. a@9 and a@10 form another live range. Similarly, x@3 and x@4 form one live range and the rest x's form the other. All other variables form their own live ranges. The CFG below renames a and x so each live range has a unique name.



LR(a0) = {2, 3, 4, 5, 6, 7, 8}
LR(a1) = {10}
LR(b) = {3, 4, 5, 6, 7, 8, 9, 10}
LR(d) = {6, 9}
LR(x0) = {4}
LR{x1} = {8, 9}

The interference graph is as below.



b. Show the process of the bottom-up allocation assuming that 3 registers, r1, r2, and r3, are available in addition to 2 reserved registers, r4 and r5, for spills.

The spill cost of each node is as below:

a0	a1	b	d	x0	x1
5/4	2/1	4/5	4/3	2/2	4/3

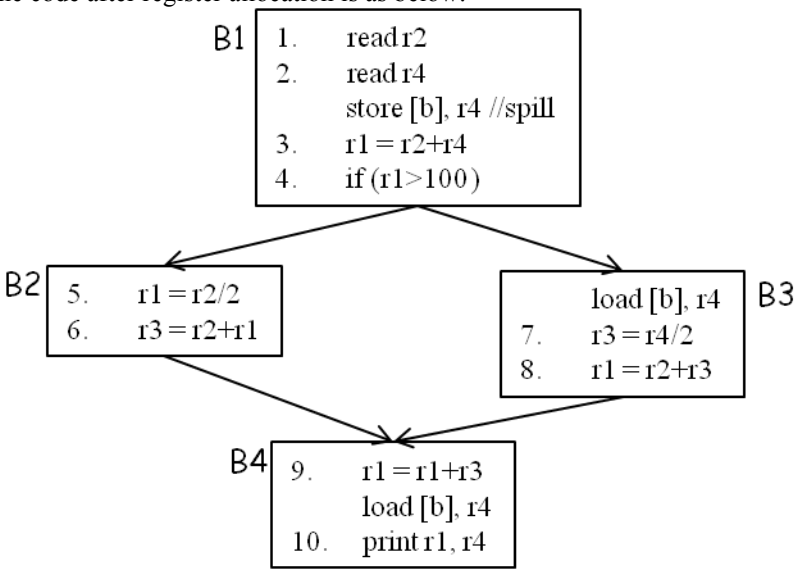
cost(b) < cost(x0) = cost (a0) < cost(x1) = cost(d) < cost(a1).
In the first phase of the bottom-up allocation algorithm, a1, x0 are pushed into the stack because their degrees are less than 3. Now b is pushed into the stack because its cost is smallest among the remaining nodes. Then the rest nodes, x1, a0, d are pushed into the stack (the order does not matter) because all their degrees are less than 3. The stack below is the status after this phase.

d
a0
x1
b
x0
a1

In the coloring phase, nodes are popped off the stack and colored. Nodes d, a0, and x1 receive 1, 2, and 3 respectively. Then b is spilled. Nodes x0 and a1 both receive 1. The allocated registers are annotated in the interference graph.

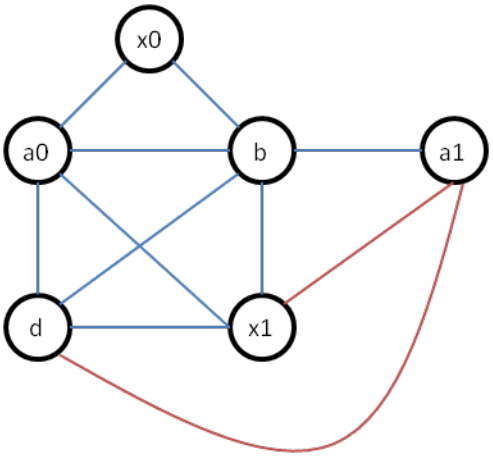
c. Rewrite the code based on your allocation and insert appropriate spill code if needed.

The code after register allocation is as below.



d. What are the live ranges for the top-down allocator? Draw the interference graph.

LR(a0) = {B1, B2, B3}
LR(a1) = {B4}
LR(b) = {B1, B2, B3, B4}
LR(d) = {B2, B3, B4}
LR(x0) = {B1}
LR{x1} = {B2, B3, B4}


x0
a0
b
a1
d
x1