ECE 468 & 573

Problem Set 4: Instruction scheduling & loop transformations **Solutions**

Instruction scheduling

1. Assume that your machine has two ALUs, which can execute ADDs in a single cycle and *one* of the two ALUs can execute MULs in two cycles, but the ALUs are fully pipelined (a new instruction can be issued in each cycle). There is also a single LD/ST unit. LOADs take up an ALU for one cycle and the LD/ST for two cycles, while STs take up an ALU for one cycle and the LD/ST for one cycle. The LD/ST unit is *not* fully pipelined (new instructions cannot be scheduled on the unit until the previous instructions complete). What are the latencies for each instruction? Draw the reservation tables for this machine.

   Latencies: ADDs take one cycle, MULs take two cycles, LOADs take three cycles and STs take two cycles.

   Reservation tables (note that because the ALUs are fully pipelined, the reservation table for multiply doesn't span two cycles, even though MULs have a two-cycle latency):

   ADD1:

ALU0	ALU1	LD/ST
x		

   ADD2:

ALU0	ALU1	LD/ST
	x	

   MUL:

ALU0	ALU1	LD/ST
x		

   LD1:

ALU0	ALU1	LD/ST
x		
		x
		x

   LD2:

ALU0	ALU1	LD/ST
	x	
		x
		x

ST1:

ALU0	ALU1	LD/ST
x		
		x

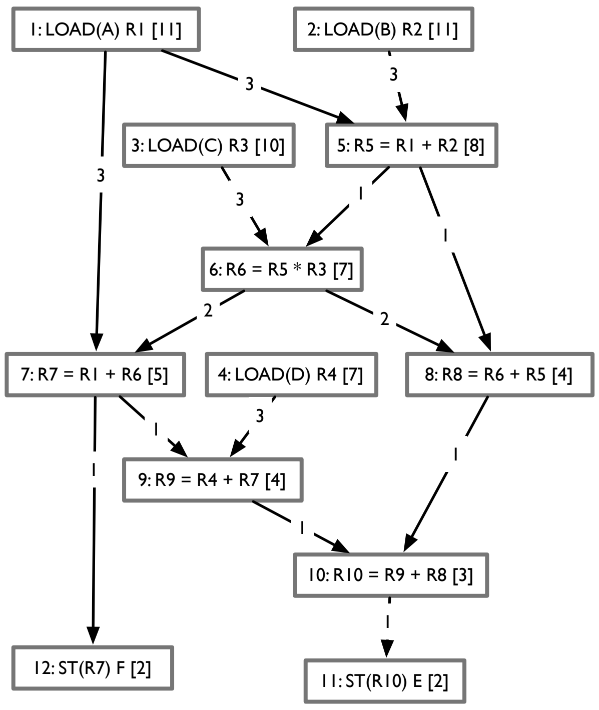
ST2:

ALU0	ALU1	LD/ST
	x	
		x

2. Draw the scheduling DAG for the following program:

```
1.  LOAD(A) R1
2.  LOAD(B) R2
3.  LOAD(C) R3
4.  LOAD(D) R4
5.  R5 = R1 + R2
6.  R6 = R5 * R3
7.  R7 = R1 + R6
8.  R8 = R6 + R5
9.  R9 = R4 + R7
10. R10 = R9 + R8
11. ST(R10) E;
12. ST(R7) F;
```

Answer: Given below. Latencies listed on edges, heights in brackets.



3. Give the schedule you obtain after performing height-based list scheduling on the above code.

Answer:

Cycle	ALU0	ALU1	LD/ST	Instructions scheduled
1	1			1
2			1	1 (cont)
3	2		1	1 (cont), 2
4			2	2 (cont)
5	3		2	2 (cont), 3
6	5		3	3 (cont), 5
7	4		3	3 (cont), 4
8	6		4	4 (cont), 6
9			4	4 (cont), 6 (cont)[1]
10	7	8		7, 8
11	9	12		9, 12
12	10		12	10, 12 (cont)
13	11			11
14			11	11 (cont)

[1] Note that 6 does not take up space in the ALU anymore, since the ALU is fully pipelined

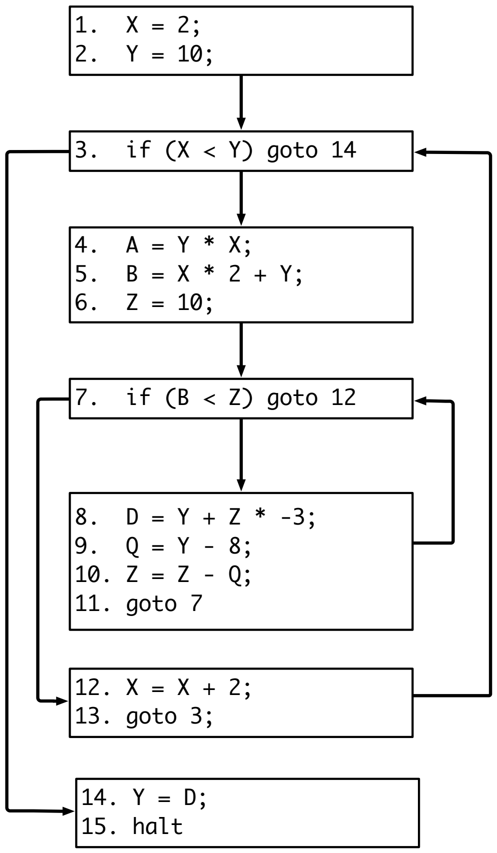
Loop transformations
For the following problems, consider the code below:

```
1.   X = 2;
2.   Y = 10;
3.   if (X < Y) goto 14
4.     A = Y * X;
5.     B = X * 2 + Y;
6.     Z = 10;
7.     if (B < Z) goto 12 //This was a mistake in the original HW!
8.        D = Y + Z * -3;
9.        Q = Y - 8;
10.       Z = Z - Q;
11.       goto 7;
12.    X = X + 2;
13.    goto 3;
14.  Y = D;
15.  halt;
```

1. Draw the CFG for the code above. Identify the loops in the code.

   Answer:

   Loop headers: BBs starting with instruction 3 and instruction 7. The BBs in the loop for BB3 are: 3, 4, 7, 8, 12. The BBs in the loop for BB7 are: 7, 8.



```
1.  X = 2;
2.  Y = 10;

3.  if (X < Y) goto 14

4.  A = Y * X;
5.  B = X * 2 + Y;
6.  Z = 10;

7.  if (B < Z) goto 12

8.  D = Y + Z * -3;
9.  Q = Y - 8;
10. Z = Z - Q;
11. goto 7

12. X = X + 2;
13. goto 3;

14. Y = D;
15. halt
```

2. Which statements are loop invariant? Can they be moved outside their enclosing loop? Show the code that results after hoisting any loop invariant code outside the loop.

Answer:

Loop invariant statements: 6, 9. 6 cannot be moved, because Z is defined twice inside the loop (note that if we try to move it outside the loop, Z will not get reinitialized before the inner loop). 9 can be moved, though. Note that first it gets moved outside the inner loop, but then is *still* loop invariant (because Y is only defined outside the loop), so can be moved outside the outer loop, as well:

```
1.    X = 2;
2.    Y = 10;
9'.   Q = Y - 8;
3.    if (X < Y) goto 14
4.      A = Y * X;
```

```
5.     B = X * 2 + Y;
6.     Z = 10;
7.     if (B < Z) goto 12
8.        D = Y + Z * -3;
10.       Z = Z - Q;
11.       goto 7;
12.    X = X + 2;
13.    goto 3;
14.  Y = D;
15.  halt;
```

3. Identify the induction variables in this code. Show the code that results after performing any possible strength reduction.

   We'll break this up by loop. In the inner loop, Z is an induction variable (because Q is loop invariant), and D is a mutual induction variable. After performing strength reduction on the inner loop, we get:

```
1.   X = 2;
2.   Y = 10;
9'.  Q = Y - 8;
3.   if (X < Y) goto 14
4.     A = Y * X;
5.     B = X * 2 + Y;
6.     Z = 10;
8'.    D' = Y + Z * -3;
7.     if (B < Z) goto 12
8.        D = D'
10.       Z = Z - Q;
10'.      D' = D' + -3 * -Q;
11.       goto 7;
12.    X = X + 2;
13.    goto 3;
14.  Y = D;
15.  halt;
```

   We then turn to the outer loop. X is an induction variable, and both A and B are mutual induction variables. After strength reduction, we get:

```
1.   X = 2;
```

```
2.   Y = 10;
9'.  Q = Y - 8;
4'.  A' = Y * X;
5'.  B' = X * 2 + Y;
3.   if (X < Y) goto 14
4.     A = A'
5.     B = B'
6.     Z = 10;
6'.    D' = Y + Z * -3;
7.     if (B < Z) goto 12
8.        D = D'
10.       Z = Z - Q;
10'.      D' = D' + -3 * -Q;
11.       goto 7;
12.    X = X + 2;
12'.   A' = A' + 2 * Y;
12''.  B' = B' + 4;
13.    goto 3;
14.  Y = D;
15.  halt;
```

4. Show the code after performing any possible linear test replacement.

   Answer: At this point, X is only used to drive the outer loop, so can be removed with linear test replacement. We have our choice of using A or B as the replacement. Let's use A.

```
1.   X = 2;
2.   Y = 10;
9'.  Q = Y - 8;
4'.  A' = Y * X;
5'.  B' = X * 2 + Y;
3.   if (A' < Y * Y) goto 14
4.     A = A'
5.     B = B'
6.     Z = 10;
6'.    D' = Y + Z * -3;
7.     if (B < Z) goto 12
8.        D = D'
10.       Z = Z - Q;
10'.      D' = D' + -3 * -Q;
```

```
11.       goto 7;
12.    // X = X + 2;
12'.   A' = A' + 2 * Y;
12''.  B' = B' + 4;
13.    goto 3;
14.  Y = D;
15.  halt;
```

Note that it looks like Z is only used to drive the inner loop. But because Z is used in line 6', we can't actually remove its use (without more sophisticated compiler analysis).