Solutions for Homework 6

CIS 501, Spring 1996
April 1, 1996

1. (4.3) For the following code fragment, list the control dependencies. For each control dependency, tell whether the statement can be scheduled before the if statement. Assume that all data references are shown, that all values are defined before use, and that only `b` and `c` are used again after this segment. You may ignore any possible exceptions.

```
if ( a > c ) {
  d = d + 5;                    /*  S1  */
  a = b + d + e;                /*  S2  */
} else {
  e = e + 2;                    /*  S3  */
  f = f + 2;                    /*  S4  */
  c = c + f;                    /*  S5  */
}
b = a + f;                      /*  S6  */
```

   Statements S1, S2, S3, S4, and S5 are control dependent on the conditional `if` $a > c$. S1 can be scheduled before the `if` statement, because d is not used in the code following the `if` statement and d is not used in the `else` portion of the code.

   None of the other statements can be scheduled before the `if` statement, because they conditionally change variables that are used within other portions of the `if` statement or after the `if` statement. S2 changes the value of a, which is used after the conditional in S6. S3 changes the value of e within the `else` portion of the code; however, e is used in the `then` portion of the code, where it must use the original value of e. S4 changes f, which is used in S6. S5 changes c which is used after the displayed segment, as specified in the problem.

   Note: S6 is not control dependent on $a > c$; however, S6 is data dependent on variables computed within the `if` statement, and thus, cannot be scheduled before the `if` statement.

2. Consider the following code fragment:

```
ADD     R3,R2,R4
LW      R4,8(R5)
MULT    R3,R3,R2
ADD     R5,R4,R3
LW      R3,8(R4)
SUB     R6,R6,R2
LW      R7,8(R6)
ADD     R7,R7,R6
```

   Assume that:

   - Reading register operands requires 1 cycle.
   - Once operands are read, execution takes:

     ALU 1 cycle
     LSU 2 cycles (data available at end of Mem phase)
     MULT 6 cycles
   - For the static pipeline, at most one new instruction can be initiated per cycle.
   - For the dynamic pipelines, at most 2 instructions can be fetched per cycle. Moreover, at most 3 instructions can be issued in a cycle.

- The DLX machine has 1 ALU (integer math unit), 1 MULT/DIV (integer multiply/divide unit) and 1 LSU (load/store unit).
- Each functional unit has 4 queue slots, and is fully pipelined (only relevant for the tomasulo algorithm).
- If a result is ready to write back, you can do so—there are always enough writeback busses.
- Once execution has completed, writing the results back takes one cycle, but the writeback happens at the start of the cycle.
- Bypassing is possible only in the static pipeline. On a dynamic machine, the value isn't available until the writeback phase. Bypassing is possible, but the mechanisms we have presented are NOT enough to manage it. So **do not** try to do bypassing in the dynamic pipelines.
- The first cycle is cycle 0.

For each of the following problems, draw a timing diagram showing the cycle(s) in which each instruction is fetched, issued and decoded, the operands become available, and the results are written back.

(a) For a simple, nondynamic pipeline.

(b) For a pipeline managed by scoreboarding.

(a) Simple, nondynamic pipeline

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ADD R3,R2,R4	F	R	X	M	W													
LW R4,8(R5)		F	R	X	M	W												
MULT R3,R3,R2			F	R	X1	X2	X3	X4	X5	X6	M	W						
ADD R5,R4,R3				F	R	-	-	-	-	-	X	M	W					
LW R3,8(R4)					F	-	-	-	-	-	R	X	M	W				
SUB R6,R6,R2											F	R	X	M	W			
LW R7,8(R6)												F	R	X	M	W		
ADD R7,R7,R6													F	R	-	X	M	W

(b) Pipeline managed by scoreboarding

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
ADD R3,R2,R4	F*	R	X	W																					
LW R4,8(R5)	F*	R	X	M	W																				
MULT R3,R3,R2		F	-	*	R	X1	X2	X3	X4	X5	X6	W													
ADD R5,R4,R3		F	-	*	-	-	-	-	-	-	-	-	R	X	W										
LW R3,8(R4)			F	-	-	-	-	-	-	-	-	*	R	X	M	W									
SUB R6,R6,R2			F	-	-	-	-	-	-	-	-	-	-	-	*	R	X	W							
LW R7,8(R6)				F	-	-	-	-	-	-	-	-	-	-	-	*	-	-	R	X	M	W			
ADD R7,R7,R6				F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	*	R	X	W

3. (4.10) Suppose we have a deeply pipelined processor, for which we implement a branch-target buffer for the conditional branches only. Assume that the misprediction penalty is always 4 cycles and the buffer miss penalty is always 3 cycles. Assume a 90% hit rate and a 90% accuracy, and 15% branch frequency. How much faster is the processor with the branch-target buffer versus a processor that has a fixed 2-cycle branch penalty? Assume a base CPI without branch stalls of 1.

4. For the processor with the branch-target-buffer,

$$\begin{aligned}\text{effective CPI} &= CPI_{base} + stalls_{BTB}\\ &= 1 + (\%\text{ branches})[(\%\text{ buffer hit rate} * \%\text{ incorrect prediction} * \text{misprediction penalty})\\ &\quad + (\%\text{ miss rate} * \text{miss penalty})]\\ &= 1 + (.15)(.90*.10*4 + .10*3)\\ &= 1.099\end{aligned}$$

For the processor with the 2-cycle branch penalty,

$$\begin{aligned}\text{effective CPI} &= CPI_{base} + stalls_{2bp} \\ &= 1 + (\text{\% branches * branch penalty}) \\ &= 1 + 2\text{*}.15 \\ &= 1.30\end{aligned}$$

$CPI_{2bp}/CPI_{BTB} = (1.3)/(1.099) = 1.18$

Thus, the processor with the branch-target-buffer is 18% faster.