CPSC 313 Sample Test 2
May 2012

[6] 1. Branch Prediction

[2] a. As stated in class, it does not matter how an Instruction Set Architecture (ISA) predicts branches as long as the compiler for the high-level language knows the algorithm used. Why?

Solution : Because the compiler can adjust the code it generates to take advantage of the algorithm used by the ISA, when it knows whether or not a branch is likely to be taken based on the structure of the high-level language program.

[2] b. Why is jump prediction for `ret` instructions harder to implement than jump prediction for conditional branches such as `jg` and `jne`?

Solution : For conditional branches, there are only two possible addresses from which the next instruction will come: the branch target, or the address following the current instruction. For `ret` instructions, the number of possible locations for the next instruction is much, much larger.

[2] c. In a pipelined CPU, is a random jump prediction strategy better or worse than no jump prediction at all? Justify your answer.

Solution : It is better: with no jump prediction, the CPU is forced to stall until it knows whether or not the jump is taken. With jump prediction, there is at least some probability that the CPU will guess correctly, and avoid stalling.

[6] 2. Consider the following sequence of instructions executed by the `PipeMinus` version of our Y86 CPU (this is the version that stalls whenever a hazard is detected):

```
start: irmovl $8,   %eax
       subl   %ebx, %ebx
       xorl   %ecx, %ecx
       addl   %eax, %eax
       mull   %eax, %ecx
       popl   %edi
       rrmovl %edi, %edi
       jg     start
       halt
```

The first row of the following table shows the instruction that is about to enter each stage of our pipeline at the beginning of the sixth clock cycle. Complete the table by indicating, for each of the following ten clock cycles, which instruction will enter each of the pipeline stages during that clock cycle. Recall that a pipeline bubble is represented by a `nop` instruction.

Solution :

F	D	E	M	W
popl	mull	addl	nop	xorl
popl	mull	nop	addl	nop
popl	mull	nop	nop	addl
popl	mull	nop	nop	nop
rrmovl	popl	mull	nop	nop
jg	rrmovl	popl	mull	nop
jg	rrmovl	nop	popl	mull
jg	rrmovl	nop	nop	popl
jg	rrmovl	nop	nop	nop
irmovl	jg	rrmovl	nop	nop
irmovl	nop	jg	rrmovl	nop

[5] 3. Now, repeat the same exercise for the `Pipe` version of our Y86 CPU, on the same piece of code:

```
start: irmovl $8,   %eax
       subl   %ebx, %ebx
       xorl   %ecx, %ecx
       addl   %eax, %eax
       mull   %eax, %ecx
       popl   %edi
       rrmovl %edi, %edi
       jg     start
       halt
```

Recall that this version of the CPU forwards values from the `write-back`, `memory` and `execute` stages to the `decode` stage, and predicts that conditional jumps are always taken. One again, the first row of the table shows the instruction that is about to enter each stage of our pipeline at the beginning of the sixth clock cycle. This time, you only need complete the table for the following six clock cycles.

Solution :

F	D	E	M	W
popl	mull	addl	xorl	subl
rrmovl	popl	mull	addl	xorl
jg	rrmovl	popl	mull	addl
jg	rrmovl	nop	popl	mull
irmovl	jg	rrmovl	nop	popl
subl	irmovl	jg	rrmovl	nop
halt	nop	nop	jg	rrmovl

[8] 4. The following table shows the state of each pipeline stage register when the `popl` instruction from the previous program is about to enter the Fetch stage:

F		D		E		M		W	
f.prPC:	**10E**	d.iCd:	**6**	e.iCd:	**6**	m.iCd:	**6**	w.iCd:	**6**
		d.iFn:	**4**	e.iFn:	**0**	m.bch:	**1**	w.valE:	**0**
		d.rA:	**0**	e.valC:	**0**	m.valE:	**0**	w.valM:	**0**
		d.rB:	**1**	e.valP:	**10C**	m.valA:	**0**	w.dstE:	**3**
		d.valC:	**0**	e.valA:	**8**	m.dstE:	**1**	w.dstM:	**F**
		d.valP:	**10E**	e.valB:	**8**	m.dstM:	**F**	w.valP	**108**
				e.dstE:	**0**	m.valP:	**10A**		
				e.dstM:	**F**				
				e.srcA:	**0**				
				e.srcB:	**0**				

Fill in the following table with the values that each stage register will contain one clock cycle later.

Solution :

F		D		E		M		W	
f.prPC:	**110**	d.iCd:	**B**	e.iCd:	**6**	m.iCd:	**6**	w.iCd:	**6**
		d.iFn:	**0**	e.iFn:	**4**	m.bch:	**1**	w.valE:	**0**
		d.rA:	**7**	e.valC:	**0**	m.valE:	**10**	w.valM:	**0**
		d.rB:	**F**	e.valP:	**10E**	m.valA:	**8**	w.dstE:	**1**
		d.valC:	**0**	e.valA:	**10**	m.dstE:	**0**	w.dstM:	**F**
		d.valP:	**110**	e.valB:	**0**	m.dstM:	**F**	w.valP:	**10A**
				e.dstE:	**1**	m.valP:	**10C**		
				e.dstM:	**F**				
				e.srcA:	**0**				
				e.srcB:	**1**				