CPSC 313 Sample Test 2
May 2012

Name: ____________________ Student ID: ____________________
Signature: ____________________

– You have 45 minutes to write the 4 questions on this examination. A total of 25 marks are available.

– No notes, books or electronic devices are allowed.

– **Justify all of your answers.**

– Keep your answers short. If you run out of space for a question, you have written too much.

– The number in square brackets to the left of the question number indicates the number of marks allocated for that question. Use these to help you determine how much time you should spend on each question.

– Use the back of the pages for your rough work.

– **Good luck!**

Question	Marks
1	
2	
3	
4	
Total	

[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?

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

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

[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.

F	**D**	**E**	**M**	**W**
popl	mull	addl	nop	xorl

[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.

F	**D**	**E**	**M**	**W**
popl	mull	addl	xorl	subl

[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.

F	D	E	M	W
f.prPC:	d.iCd:	e.iCd:	m.iCd:	w.iCd:
	d.iFn:	e.iFn:	m.bch:	w.valE:
	d.rA:	e.valC:	m.valE:	w.valM:
	d.rB:	e.valP:	m.valA:	w.dstE:
	d.valC:	e.valA:	m.dstE:	w.dstM:
	d.valP:	e.valB:	m.dstM:	w.valP:
		e.dstE:	m.valP:	
		e.dstM:		
		e.srcA:		
		e.srcB:		

Register Names

0	%eax	%esp	4
1	%ecx	%ebp	5
2	%edx	%esi	6
3	%ebx	%edi	7

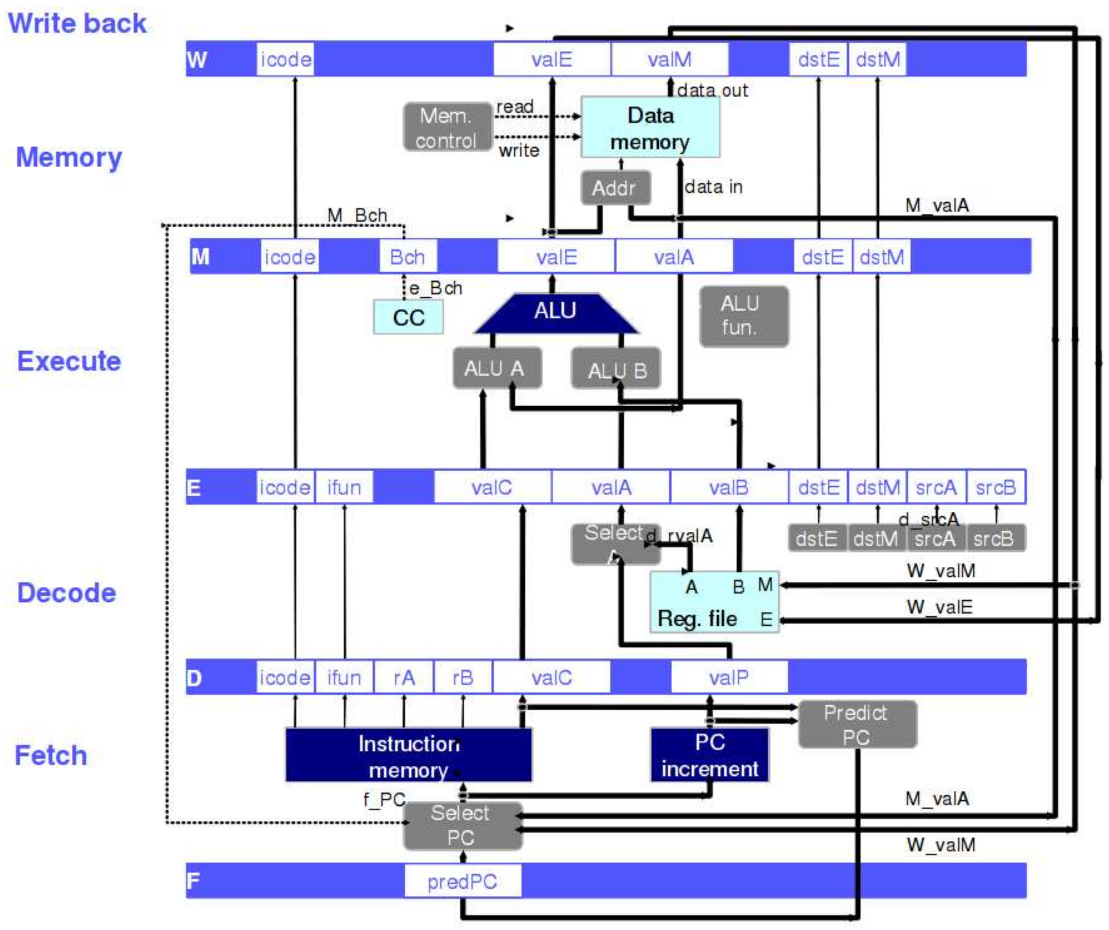
Function numbers

0	addl	andl	2
1	subl	xorl	3

Instructions Encoding

Byte	0		1		2	3	4	5
halt	0	0						
nop	1	0						
rrmovl **rA**, **rB**	2	0	**rA**	**rB**				
cmovXX **rA**, **rB**	2	fn	**rA**	**rB**				
irmovl **V**, **rB**	3	0	F	**rB**	**V**			
rmmovl **rA**, **D**(**rB**)	4	0	**rA**	**rB**	**D**			
mrmovl **D**(**rB**), **rA**	5	0	**rA**	**rB**	**D**			
OPl **rA**, **rB**	6	fn	**rA**	**rB**				
jXX **Dest**	7	fn	**Dest**					
call **Dest**	8	0	**Dest**					
ret	9	0						
pushl **rA**	A	0	**rA**	F				
popl **rA**	B	0	**rA**	F				

CPU hardware structure



Write back
W
icode
valE
valM
dstE
dstM
data out
read
Mem. control
write
Data memory
Memory
Addr
data in
M_valA
M_Bch
M
icode
Bch
valE
valA
dstE
dstM
e_Bch
CC
ALU
ALU fun.
Execute
ALU A
ALU B
E
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
Select A
d_rvalA
dstE
dstM
srcA
srcB
W_valM
A
B
M
Decode
Reg. file
E
W_valE
D
icode
ifun
rA
rB
valC
valP
Predict PC
Fetch
Instruction memory
PC increment
f_PC
M_valA
Select PC
W_valM
F
predPC