

Computer System Architecture
6.823 Quiz #2
April 4th, 2014
Professors Daniel Sanchez and Joel Emer

This is a closed book, closed notes exam.

80 Minutes
15 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Show your work to receive full credit.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

Part A	_____	30 Points
Part B	_____	25 Points
Part C	_____	30 Points
Part D	_____	15 Points

TOTAL _____ **100 Points**

Part A: Virtual Memory (30 pts)

Ben Bitdiddle decides to build a cache with the following properties:

- 16 blocks with 64-bytes per block.
- 2-way set associative organization.
- Virtually indexed, physically tagged.
- LRU replacement.

His memory system looks like the following:

- 16-bit virtual and physical addresses.
- 1024-byte pages.
- A single-level page table stored in physical memory. The page table base register (equal to 0×1000) holds the address of the start of the page table.
- Page table entries are 16 bits, with the highest order bits indicating the physical page number, and the rest as status bits.
- 8-entry fully associative TLB.

While running his architecture, he is curious about the performance of caches, so he asks for some help running through some operations. Fill out the final state of the cache and the TLB after the following virtual memory address accesses:

```
0x05DB: 0000 0101 1101 1011
0x0B49: 0000 1011 0100 1001
0x17FB: 0001 0111 1111 1011
0x1C5E: 0001 1100 0101 1110
0x35E3: 0011 0101 1110 0011
```

The contents of physical memory follow after the cache diagrams on the next page.

Name _____

Cache:

Index	Way 0		Way 1	
	Valid?	Tag	Valid?	Tag
0				
1				
2				
3				
4				
5				
6				
7				

TLB:

VPN	PPN

Name _____

Memory contents:

Page Table Base Register

0x1000

Physical Memory

Address		Value
0x11FE	:	0x7777
0x11FC	:	0xBABA
0x11FA	:	0xAB00
0x11F8	:	0x1BD1
0x11F6	:	0x9001
0x11F4	:	0xAAAA
0x11F2	:	0xB789
0x11F0	:	0xDEF0

...

0x1022	:	0xBADE
0x1020	:	0xBEAD
0x101E	:	0xDEAF
0x101C	:	0xFACE
0x101A	:	0x9ABC
0x1018	:	0x5678
0x1016	:	0x1234
0x1014	:	0xFEED
0x1012	:	0xD666
0x1010	:	0xDEAD
0x100E	:	0xADD0
0x100C	:	0x00F0
0x100A	:	0x7734
0x1008	:	0x3704
0x1006	:	0x1337
0x1004	:	0xDAB0
0x1002	:	0xBABE
0x1000	:	0xACE0

Part B: Complex Pipelining (25 points)

A	PC Generation/Mux
P	Instruction Fetch Stage 1
F	Instruction Fetch Stage 2
B	Branch Address Calc/Begin Decode
I	Complete Decode
J	Steer Instructions to Functional units
R	Register File Read
E	Integer Execute
⋮	Remainder of execute pipeline (+ another 6 stages)

You are designing a processor with the complex pipeline illustrated above. For this problem assume there are no unconditional jumps or jump register—*only* conditional branches.

Suppose the following:

- Each stage takes a single cycle.
- Branch addresses are known after stage Branch Address Calc/Begin Decode.
- Branch conditions (taken/not taken) are known after Register File Read.
- Initially, the processor *always* speculates that the next instruction is at PC+4, without any specialized branch prediction hardware.
- Branches always go through the pipeline without any stalls or queuing delays.

Question 1 (5 points):

How much work is lost (in cycles) on a branch misprediction in this pipeline?

Question 2 (5 points):

If one quarter of instructions are branches, and half of these are taken, then how much should we expect branches to increase the processor's CPI (cycles per instruction)?

Question 3 (5 points):

You are unsatisfied with this performance and want to reduce the work lost on branches. Given your hardware budget, you can add only one of the following:

- A branch predictor to your pipeline that resolves after Instruction Fetch Stage 1.
- Or a branch target buffer (BTB) that resolves after Instruction Fetch Stage 2.

If each make the same predictions, which do you prefer? In one or two sentences, why?

Question 4 (10 points):

You decide to add the BTB (not the branch predictor). Your BTB is a fully tagged structure, so if it predicts an address other than PC+4 then it always predicts the branch address of a conditional branch (but not the condition!) correctly. **For partial credit, show your work.**

If the BTB correctly predicts a next PC other than PC+4, what is the effect on the pipeline?

If the BTB predicts the next PC incorrectly, what is the effect on the pipeline?

Assume the BTB predicts PC+4 90% of the time. When the BTB predicts PC+4 it is accurate 90% of the time. Otherwise it is accurate 80% of the time. How much should we expect branches to increase the CPI of the BTB design? (*Don't bother trying to compute exact decimal values.*)

Part C: Out-of-order Execution (30 points)

In this problem, we are going to update the state of the processor when different events happen. You are given an out-of-order processor in some initial state, as described by the registers (renaming table, physical registers, and free list), one-bit branch predictor, and re-order buffer. Your job is to show the changes that occur when some event occurs, starting from the same initial state except where noted. For partial credit, briefly describe what changes occur.

Question 1 (10 points):

Show the state of the processor if the first load completes (but does not commit).

```

00: LD R1, 0(R2)
04: ADD R3, R1, R4
08: ADD R2, R1, R2
0c: BGEZ R4, A
10: LD R3, 0(R2)
A: 14: SUB R1, R3, R2
   18: ADD R4, R3, R1
    
```

} INSTRUCTIONS

BRANCH PREDICTOR	
00	1
01	0
10	1
11	0

RENAMING TABLE	
R1	P4
R2	P6
R3	P5
R4	P3

PHYS. REG. FILE		
P0	(R1)	p
P1	(R2)	p
P2	(R3)	p
P3	(R4)	p
P4		
P5		
P6		
P7		

FREE LIST
P7

RE-ORDER BUFFER (ROB)										
	Use?	Ex	Op	P1	PR1	P2	PR2	Rd	LPRd	PRd
Next to commit →	X		LD	p	P1			R1	P0	P4
	X		ADD		P4	p	P3	R3	P2	P5
	X		ADD		P4	p	P1	R2	P1	P6
	X		BGEZ	p	P3					
Next available →										

Question 2 (10 points):

Show the state of the processor after the next instruction is issued.

```

00: LD R1, 0(R2)
04: ADD R3, R1, R4
08: ADD R2, R1, R2
0c: BGEZ R4, A
10: LD R3, 0(R2)
A: 14: SUB R1, R3, R2
   18: ADD R4, R3, R1
    
```

} INSTRUCTIONS

BRANCH PREDICTOR	
00	1
01	0
10	1
11	0

RENAMING TABLE	
R1	P4
R2	P6
R3	P5
R4	P3

PHYS. REG. FILE		
P0	(R1)	p
P1	(R2)	p
P2	(R3)	p
P3	(R4)	p
P4		
P5		
P6		
P7		

FREE LIST
P7

RE-ORDER BUFFER (ROB)										
	Use?	Ex	Op	P1	PR1	P2	PR2	Rd	LPRd	PRd
Next to commit →	X		LD	p	P1			R1	P0	P4
	X		ADD		P4	p	P3	R3	P2	P5
	X		ADD		P4	p	P1	R2	P1	P6
	X		BGEZ	p	P3					
Next available →										

Question 3 (5 points):

From the state at the end of Question 2, as the next action can the processor issue (not execute) another instruction?

In one or two sentences, what does this say about our design? How can we improve it?

Question 4 (5 points):

Show the state of the processor if the first LD triggers a page fault and after abort finishes.

```

00: LD R1, 0(R2)
04: ADD R3, R1, R4
08: ADD R2, R1, R2
0c: BGEZ R4, A
10: LD R3, 0(R2)
A: 14: SUB R1, R3, R2
   18: ADD R4, R3, R1
    
```

INSTRUCTIONS

BRANCH PREDICTOR	
00	1
01	0
10	1
11	0

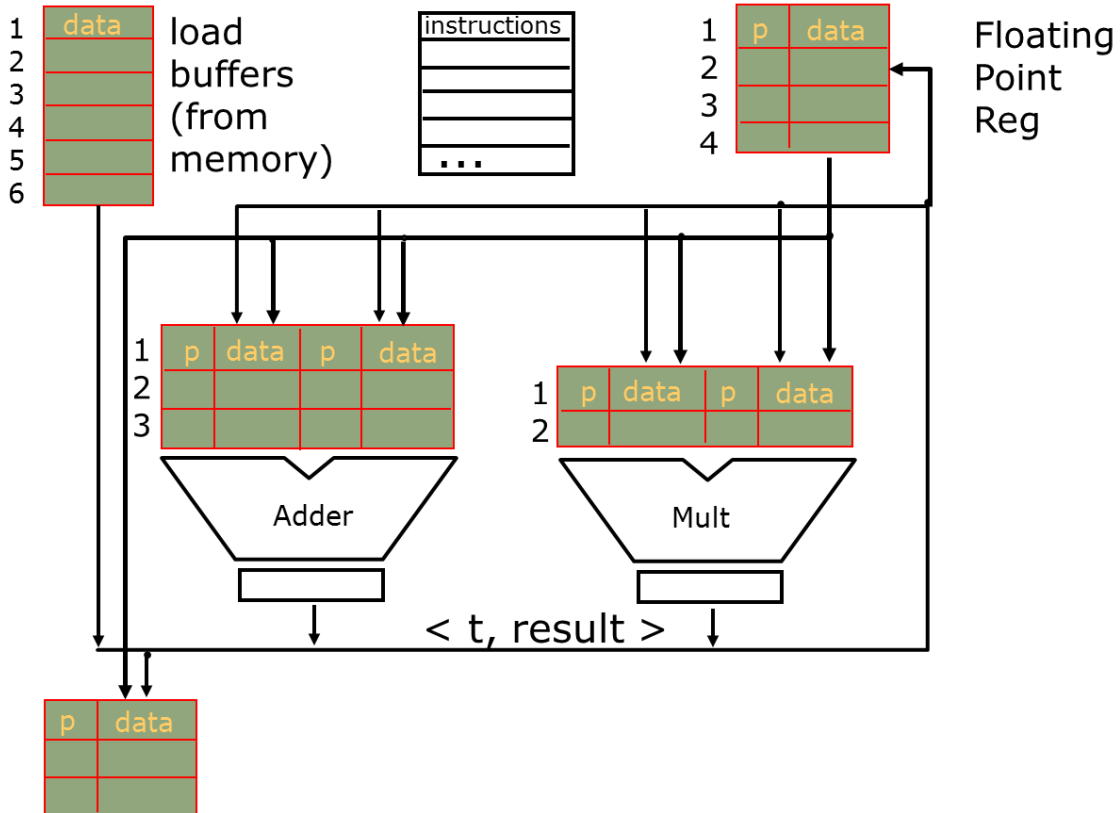
RENAMING TABLE	
R1	P4
R2	P6
R3	P5
R4	P3

PHYS. REG. FILE		
P0	(R1)	p
P1	(R2)	p
P2	(R3)	p
P3	(R4)	p
P4		
P5		
P6		
P7		

FREE LIST
P7

RE-ORDER BUFFER (ROB)										
	Use?	Ex	Op	P1	PR1	P2	PR2	Rd	LPRd	PRd
Next to commit →	X		LD	p	P1			R1	P0	P4
	X		ADD		P4	p	P3	R3	P2	P5
	X		ADD		P4	p	P1	R2	P1	P6
	X		BGEZ	p	P3					
Next available →										

Part D: Out-of-order Processor Design (15 points)



You are designing an out-of-order processor similar to the IBM 360/91 Tomasulo design shown above. This design distributes the re-order buffer around the processor, placing entries near their associated functional units. In such a design, the distributed ROB entries are called “reservation stations”. Entries are allocated when the instruction is decoded and freed when the instruction is dispatched to the functional unit.

Your design achieves an average throughput of 1.5 instructions per cycle. Two-thirds of instructions are adds, and one-third are multiplies. The latency of each instruction type *from allocation to completion* is 5 cycles for adds and 14 cycles for multiplies.

Type of operation	Fraction of instructions	Average latency
Add	2/3	5
Multiply	1/3	14

The adder and multiplier are each fully pipelined with full bypassing. *Once an instruction is dispatched to the FU*, the adder takes 2 cycles and the multiplier takes 5 cycles.

Throughput	Add latency	Multiply latency
1.5	2	5

Questions:

How many entries are in use, on average, in the reservation station at each functional unit (adder, multiplier) in the steady state? Assume there are infinite entries available if needed. What is the average latency of an instruction in this machine? *For partial credit, feel free to give any formulae you believe may be important to answer this question.*