University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2003 Instructor: Dave Patterson 2003-11-19 v1.9

CS 152 Exam #2 Solutions

Personal Information

First and Last Name	Peter Perfect
Your Login	cs152-____ ____
Lab/Discussion Section Time & Location ***you attend***	
"All the work is my own. I have no prior knowledge of the exam contents nor will I share the contents with others in CS152 who have not taken it yet."	***(Please sign)***

Instructions

- Partial credit may be given for incomplete answers, so please write down as much of the solution as you can.
- Please write legibly! If we can't read it from 3 feet away, we won't grade it!
- Put your name and login on each page.
- This exam will count for 16% of your grade.

Grading Results

Question	*Max. Points*	*Points Earned*
1	**40**	
2	**30**	
3	**30**	
Total	**100**	

Name: _____ **Login:** _____

Question 1: Potpourri (Jack and Dave's Question)

Part A: [4 points]

TLBs entries have valid bits and dirty bits. Data caches have them also. Which of the following are true? Circle the correct answer(s).

A. The valid bit means the same in both: if valid = 0, it must miss in both TLBs and Caches.
B. The valid bit has different meanings. For caches, it means this entry is valid if the address requested matches the tag. For TLBs, it determines whether there is a page fault (valid=0) or not (valid=1).
C. The dirty bit means the same in both: the data in this block in the TLB or Cache has been changed.
D. The dirty bit has different meanings. For caches, it means the data block has been changed. For TLBs, it means that the page corresponding to this TLB entry has been changed.

Explain (briefly):

A: True. If valid is 9, that means that there is no block or mapping there or the block or mapping has become bad somehow. Therefore, the TLB and cache will miss.
B. False. No, an invalid bit means that the entry in the TLB is garbage.
C. False. A TLB is not in control of VA->PA mappings, therefore it could never have anything to "writeback" the dirty data to.
D. True. Yes, since a TLB is a cache of VA->PA mappings, if it is "dirty" it means that that page is dirty.

Grading:

2 points for each correct marking (A through d). No partial credit.

Name: _____ **Login:**_____

Part B: [4 Points]

Buses and networks share some common characteristic yet retain some differences. Which of the following are true? Circle the correct answer(s).

A. Multimaster buses need to resolve arbitration before using the bus, while networks don't.
B. Both buses and networks transfer multiple words to increase communication bandwidth.
C. Networks are often connected in a hierarchy while buses are not connected in such a way.
D. Buses usually connect computers together while networks usually connect I/O peripherals to processors.

Explain (briefly):

A: False. Both need arbitration of some sort. As an example, Ethernet collision detection ("transmit and if there is a collision, transmit again") counts as arbitration.

B. Depends. This is certainly true for most networks and buses, but not true for all. We gave credit for any answer with a decent explanation.

C. False. Networks obviously are, but consider the FSB/PCI/USB hierarchy in pcs.

D. False. Just the opposite.

Grading:

We decided that parts A and B were not worded properly, so we ignored them. Parts C and D were worth +2 each.

Name: _____ **Login:**_____

Question 1: Potpourri (Jack and Dave's Question) Continued …

Part C: [8 Points]

What would be the bottleneck if we tried to turn an ordinary single issue non-speculative tomasulo machine into a dual issue machine by changing only the issue (and Icache) unit to issue 2 instructions at once?

Well, all sorts of things. Primarily, we would have a problem with the **CDB**, since we would be issuing 2 instructions per cycle, but the most that we could "commit" to be broadcast by the CDB during any given cycle would be 1. We could also have problems with not having enough **FUs** or **Stations.**

Grading:

We accepted pretty much any answer that made any sense at all. Just about every other component could possibly be the bottleneck.

Name: _____ **Login:**_____

Part D: [8 points]

Briefly explain how the non-speculative Tomasulo algorithm resolves the following classes of hazards:

RAW: By keeping pointers to the most recent writes in Register Status fields, and since ops are issued in program order, reads will always get the correct value because they will not read operands until the providing FUs broadcast them.

RAR: This is not a hazard. You TAs are soooo sneaky.

WAW: The register status fields, which ensure that all subsequent reads will get the last write to their source operands if the result is still in flight.

WAR: Again, register renaming will ensure that, for this antidependence, if the write executes before the read, the write will go to a renamed register.

Grading:

+2 for each correct. We accepted just about any answer that demonstrated understanding of the tomasulo architecture. We gave +0 on the RAR hazard for anyone who did tried to use some complicated mechanism to explain it away.

Name: _____ **Login:**_____

Question 1: Potpourri (Jack and Dave's Question) Continued …

Part E: 5 points

Again, assume we have a dual-issue Tomasulo machine with exactly three reservation stations for arithmetic instructions. The reservation stations/functional units along with their execution latencies are as follows:

adds and subs -- *a* cycles
multiply -- *2a* cycles
divide -- *5a* cycles

(Loads, stores, and branches are handled separately.)

What is the minimum number of entries in each of the reservation stations and in the ROB necessary to guarantee that we won't stall on a structural hazard while trying to issue an arithmetic instruction?

Add/sub reservation station: ____∞____

Multiply reservation station: ____∞____

Divide reservation station: ____∞____

Number of entries in the Reorder Buffer: ____∞____

Solution: The key is "dual-issue" – if our program segment consists, for example, of a stream of adds, then our single add function unit will commit at most one add per cycle (even if it is pipelined). Therefore, since we issue two but can only commit one, we will need an infinite number of entries in both the add reservation station and the ROB to ensure no structural hazards. The same reasoning can be applied to mult and div instructions.

Grading:

*5/5: for putting ∞/∞/∞/∞.
* 2/5 for putting 2a/4a/10a/16a (for recognizing that dual-issue doubles our entry requirements) or a/2a/5a/8a.
* 0/5 for anything else

Name: _____ **Login:**_____

Part F:

The x86 instruction set only has 8 ISA-defined general purpose registers. Assume that each instruction only writes to one register, but may read from up to 3 registers (those darn CISC instructions!) If the maximum number of instructions that can be in flight at any given time is 32, how many physical registers must we have in order to implement explicit register renaming?

Number of physical registers: _____40_____

Explanation: Assume a program segment as follows (this isn't real x86):

Add $t0 $t0 $t1
Add $t0 $t0 $t2
Add $t0 $t0 $t3
{etc.}

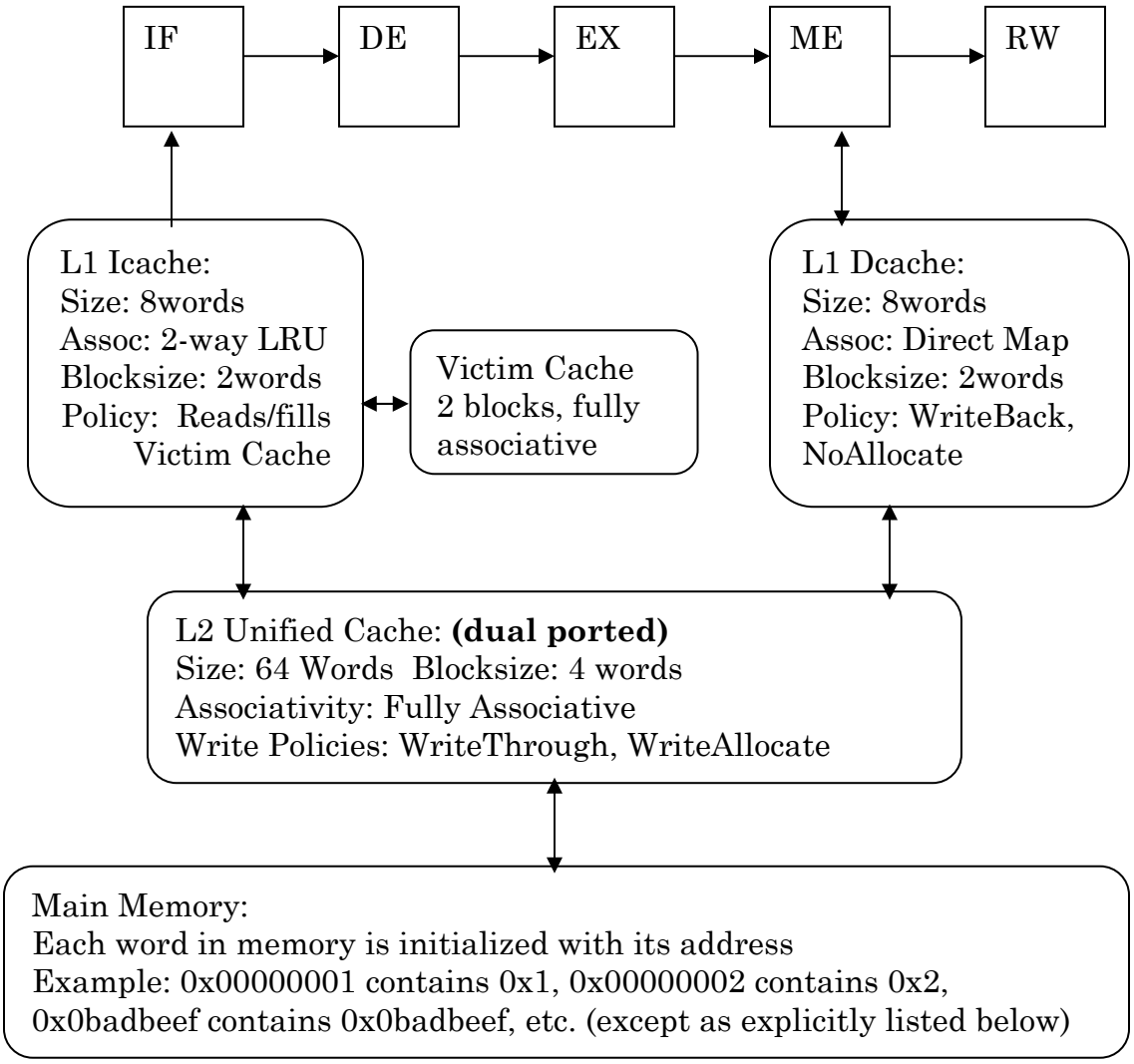
In this case, if 32 of these adds can be in flight at one time, we will need 32 destination regs to hold their results. Additionally, we will need 8 registers to hold the ISA-defined values, bringing our total to 40.

Grading:
+5: 40 or 39
+2: 32
+0: Otherwise

Name: _____ **Login:**_____

Question 2: Cache This! (Kurt's Question)

Kurt's rinky-dink computer has the following organization:



His computer has **32-bit words and addresses** and **no virtual memory** system.

The worst-case latency of the entire memory system is **1 cycle**. (I.e., the cycle time is long enough such that L1Dcache, L1Icache, and L2 can all miss on the same request, fill their blocks, and L1I and L1D can return the requested data all in the same cycle. This assumption is totally unrealistic and defeats the purpose of having a cache, but it will make your calculations easier.)

Assume further that the L2 cache is **dual-ported** but services **Icache requests before Dcache requests**.

Name: _____ **Login:**_____

Question 2: Cache This! (Kurt's Question) Continued …

Part A:

Please show the structure of each of the 4 caches in a table format. We've done the L1 Dcache for you; your answers should contain the same types of information as ours. Be sure to include the size of all fields in the cache.

L1 Icache:

Index	Tag	Word0	Word1
0			
1			

Tag: 28
Index: 1
Block: 1

+2 correct index size
+2 correct tag size

L1 Icache Victim Cache

Tag	Word0	Word1

Tag: 29
Index: 0
Block: 1

+2 correct tag size
+2 correct index size

L1 Dcache:

Index #	Tag	Word0	Word1
00			
01			
10			
11			

Tag will be 27 bits for each block.
Index will be 2 bits from address.
Block offset will be 1 bit.
(Byte offset is 2 bits.)

L2 Unified

Tag	W0	W1	W2	W3

{16 total lines}

Tag: 28 Index: 0 Block: 2

+2 correct tag size
+2 correct block size

Name: _____ **Login:**_____

Question 2: Cache This! (Kurt's Question) Continued …

Part B:

Assume that Kurt just turned on his computer (with memory initialized as described above except for the addresses below) and then started executing these instructions. For each instruction, consider each of the 4 caches. For each cache, indicate whether the instruction hits in that cache ("H"), misses in that cache ("M"), or is never checked ("X".) For cache hits and misses, also include the cycle number on which the access occurred. **Some boxes may have more than one entry.** We have filled in a couple entries for you.

Don't forget that this is a pipelined processor!

We've provided a table on the next page which you may find helpful.
However, we will only grade your answers on this page.

Address	Instruction	L1 Icache	L1 Victim	L1 Dcache	L2
0x00000000	Lw $1 0xBAD0($0)	**M-1**	**M-1**	**M-4**	**M-1** M-4
0x00000004	Lw $2 0xBAD4($0)	H-2	X	H-5	X
0x00000008	Lw $3 0xBA04($0)	M-3	M-7	M-6	H-3 M-6
0x0000000C	Sw $1 0($0)	H-4	X	M-7	H-7
0x00000010	Sw $2 0x0C($0)	M-5	M-7	M-8	M-5 H-8
0x00000014	Sw $3 0x80($0)	H-6	X	M-9	M-9
0x00000018	Sw $3 0x0C($0)	M-7	M-7	M-10	H-10

Grading:
+2: Alternating hits/misses on L1I.
+2: L1I Hit -> X for L1Vic, L1I Miss -> Check L1Vic
+2: L1D hits on access to 0xBAD4 (inst 2)
+2: L1D misses twice on 0x000C (because of no-write-allocate)
+3: L2 not checked when both L1I and L1D hit (instt 2)
+3: L2 hits both times on 0x0C (inst 5 and 7)
+4: L2 accessed twice per inst when both L1I and L1D miss.

Name: _____ **Login:**_____

This page will not be graded!! **This page will not be graded!!**

		Instr	00	04	08	0C	10	14	18
Cycle	Cache		Lw $1	Lw $2	Lw $3	Sw $1	Sw $2	Sw $3	Sw $4
1	L1-I		m						
	L1-Vic		m						
	L1-D								
	L2		m						
2	L1-I			h					
	L1-Vic			x					
	L1-D								
	L2			x					
3	L1-I				m				
	L1-Vic				m				
	L1-D								
	L2				h				
4	L1-I					h			
	L1-Vic								
	L1-D		m			x			
	L2		m			x			
5	L1-I						m		
	L1-Vic						m		
	L1-D			h					
	L2			x			m		
6	L1-I							h	
	L1-Vic								
	L1-D				m			x	
	L2				m			x	
7	L1-I								m
	L1-Vic								m
	L1-D					m			
	L2					h			h
8	L1-I								
	L1-Vic								
	L1-D						m		
	L2						h		
9	L1-I								
	L1-Vic								
	L1-D							m	
	L2							m	
10	L1-I								
	L1-Vic								
	L1-D								m
	L2								h
11	L1-I								
	L1-Vic								
	L1-D								
	L2								

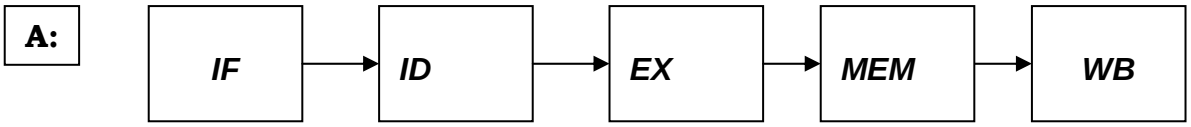
Name: _____ **Login:**_____

Name: _____ **Login:**_____

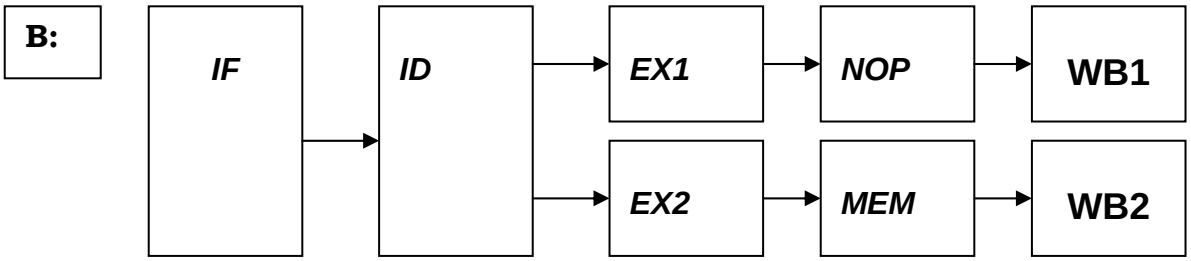
Question 3: Superscalar (John's Question)

You are an engineer working at Advanced Intelligent Devices. Your workers have proposed 3 different MIPS 2000 processor designs to you.

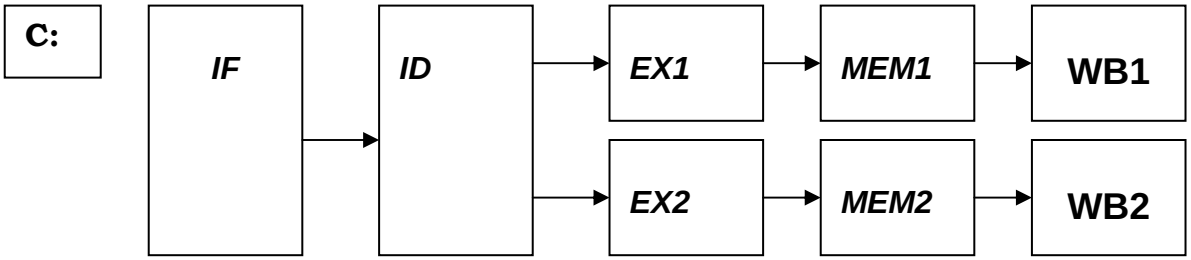
Processor A is a 5-stage pipeline identical to the one described in your Fall 2003 CS152 class. It has a full five stages and writes to the register file can be read during the same cycle:



Processor B is a limited superscalar processor that can handle branches and jumps only in the first pipeline and memory operations only in the second pipeline. Integer operations can be executed in both pipelines as long as they are not dependent. Instructions are only issued if they are not dependent. The first pipeline always executes earlier instructions and the second pipeline always executes later instructions:



Processor C is a full superscalar processor that has no restrictions on placement of branches, jumps, or memory operations. The only restriction is that like the limited superscalar pipeline, earlier instructions are always in the first pipeline:



Name: _____ **Login:**_____

Question 3: Superscalar (John's Question) Continued ...

Each of these three processors detects hazards and issues stalls in the decode stage. In both of the superscalar processors the next four instructions will always be in the fetch and decode stages. The superscalar decode stages will issue 0, 1, or 2 instructions to the execute stages depending on inter-instruction dependencies and structural hazards. If a dependency prevents an instruction from being issued then it will not be issued to the EX unit on that cycle and a nop will take its place.

All three processors have a branch prediction unit that uses a branch target buffer and a 2-bit branch history table for prediction. The first time that a branch is encountered it is assumed that it will be taken.

The block size of both the instruction and data caches is 64 bytes. The data cache is also two-way set associative.

Part A:

We'd like you to consider the state of each of these 3 processors while they are executing the following code segment:

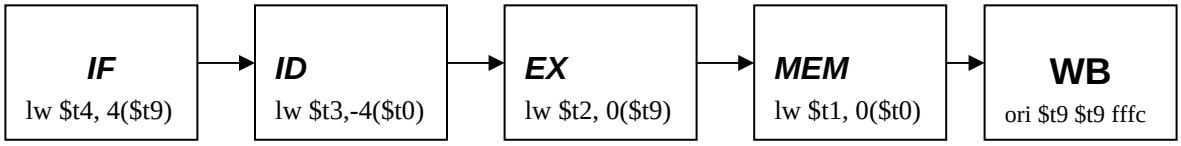
```
0x40000000          lui   $t0, 0x3fff
0x40000004          ori   $t0, $t0, 0xfffc
0x40000008          lui   $t9, 0x0000
0x4000000c          ori   $t9, $t9, 0xfffc
0x40000010   loop:  lw    $t1, 0($t0)
0x40000014          lw    $t2, 0($t9)
0x40000018          lw    $t3, -4($t0)
0x4000001c          lw    $t4, -4($t9)
0x40000020          addu  $t5, $t1, $t2
0x40000024          addu  $t6, $t3, $t4
0x40000028          sw    $t5, 0($t0)
0x4000002c          sw    $t6, -4($t0)
0x40000030          addiu $t9, $t9, -8
0x40000034          bne   $t9, $0, loop
0x40000038          addiu $t0, $t0, -8
```

Part A continues on the next page....

Name: _____ **Login:**_____

Question 3: Superscalar (John's Question) Continued ... [15 points]

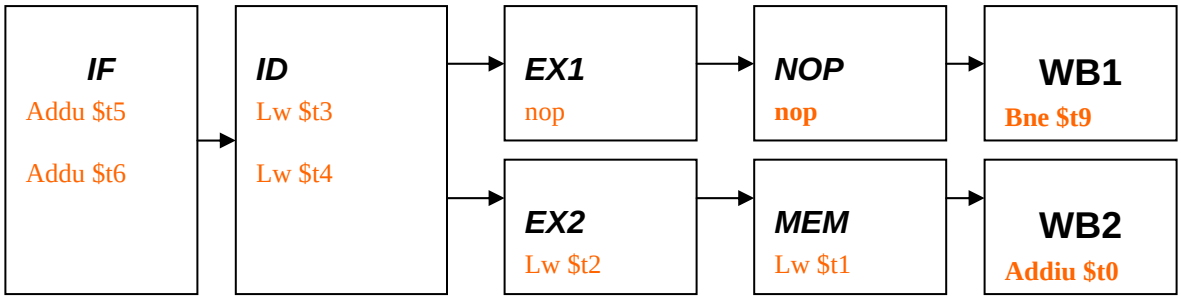
In the 5-stage pipeline, when instruction 0x400000**0c** is completing WB for the first time, the pipeline looks like this.



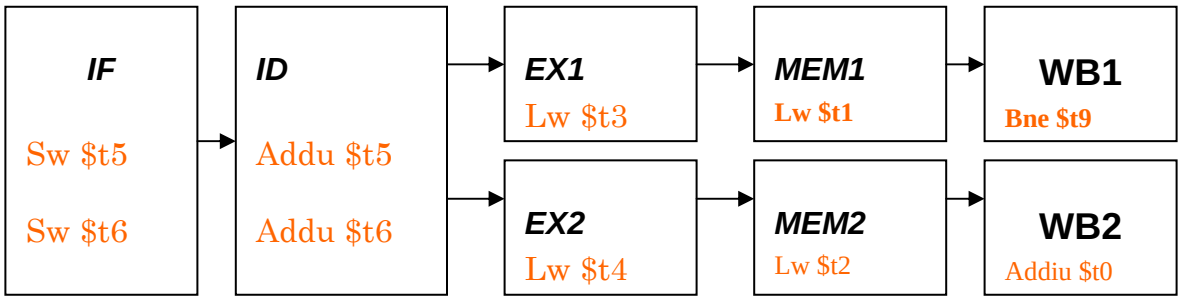
Now fill in the stages for each of the pipelines as instruction 0x400000**34** is completing WB for the **second** time. (If you like, you can use just the first two fields of the instruction.)

Processor A:

Processor B:



Processor C:



Grading: 1pt per stage per pipeline. If an assumption in a later stage messed up an earlier stage, we took off only ½ per stage.

Name: _____ **Login:**_____

Question 3: Superscalar (John's Question) Continued …

Part B: [5 points]

If you were asked to reorder the instructions to improve performance for either processor B or processor C, which would be easier and why?

The best answer is C, since C has fewer restrictions that need to be taken into account when optimizing. We accepted B if and only if you noticed that for B, you do not need to worry about basic blocks being aligned evenly or oddly.

Grading: All or nothing, depending on the value of your explanation.

Part C: [10 points]
Now reorder the instructions for the processor that you thought was easier. Indicate where the stalls will occur (if there are any left). You may not change the code size.
We've included some extra lines on the next page.

However, we will only grade this page – so put your final answer here!

Address	Label	Instruction
0x40000000		Lui $t0, 0x3fff #optimization for B
0x40000004		lui $t9, 0x0000
0x40000008		ori $t0, $t0, 0xfffc
0x4000000c		ori $t9, $t9, 0xfffc
0x40000010	Loop:	addiu $t9, $t9, -8
0x40000014		lw $t1, 0($t0)
0x40000018		addiu $t0, $t0, -8
0x4000001c		lw $t2, 8($t9)
0x40000020		lw $t3, 4($t0)
0x40000024		lw $t4, 4($t9)
0x40000028		addu $t5, $t1, $t2
0x4000002c		sw $t5, 0($t0)
0x40000030		addu $t6, $t3, $t4
0x40000034		bne $t9, $0, loop
0x40000038		sw $t6, -4($t0)
0x4000003C		
0x40000040		**(Processor C on next page)**
0x40000044		
0x40000048		

Name: _____ **Login:**_____

This page will not be graded!! **This page will not be graded!!**

Address	Label	Instruction
0x40000000		lui $t0, 0x3fff #optimization for C
0x40000004		lui $t9, 0x0000
0x40000008		ori $t0, $t0, 0xfffc
0x4000000c		ori $t9, $t9, 0xfffc
0x40000010	loop:	lw $t3, -4($t0)
0x40000014		lw $t4, -4($t9)
0x40000018		lw $t1, 0($t0)
0x4000001c		lw $t2, 0($t9)
0x40000020		addiu $t9, $t9, -8
0x40000024		addiu $t0, $t0, -8
0x40000028		addu $t6, $t3, $t4
0x4000002c		addu $t5, $t1, $t2
0x40000030		sw $t5, 8($t0)
0x40000034		bne $t9, $0, loop
0x40000038		sw $t6, 4($t0)
0x4000003C		
0x40000040		
0x40000044		
0x40000048		

Grading:

For whichever processor you chose, we compared the number of stalls in your code versus the number of stalls in our code (3 for B, 0 for C). For every additional stall in your code, we took off 4 points (3 if you recognized that it was a stall). If your optimization was broken, we gave 0 points.