CS232 Final Exam
May 10, 2002

Name: _____

- This exam has 15 action-filled pages, including this cover.
- There are six questions, worth a total of 150 points.
- Pages 14-15 contain a summary of the MIPS instruction set and a diagram of a pipelined datapath. You may tear these sheets out.
- No other written references or calculators are allowed.
- Write clearly and show your work. State any assumptions you make.
- You have 3 hours. Budget your time!

Question	Maximum	Your Score
1	25	
2	25	
3	25	
4	25	
5	25	
6	25	
Total	150	

Question 1: MIPS programming

We'll start off by writing some MIPS functions to solve a common technical interview question: how can you reverse the order of the words in a string? For example, reversing the words in the string *I Love Lucy* would result in *Lucy Love I*.

A string is just an array of bytes, with each byte representing one character. The last byte of the array is always 0, to indicate the end of the string. Also, a space is represented by the value 32. For instance, the string above might be stored in memory from addresses 500–511 as follows.

Address	500	501	502	503	504	505	506	507	508	509	510	511
Byte	73	32	76	111	118	101	32	76	117	99	121	0
Character	I		L	o	v	e		L	u	c	y	\0

Part (a)

First write a function *rev*, which simply reverses bytes in an array. The arguments passed in $a0 and $a1 are the starting address and the number of bytes to reverse. For example, invoking *rev* for the string above with $a0 and $a1 set to 502 and 4 should change the string to *I evoL Lucy*. There is no return value. Make sure to follow all MIPS calling and register save conventions. (10 points)

Question 1 continued

Part (b)

Using the *rev* function, you should now be able to write *revwords* to reverse the order of the words within a string. The only argument is the starting address of the string, passed in $a0. Again, there is no return value. You may write additional "helper" functions if you like, but be sure you follow all MIPS calling and register save conventions. (15 points)

You can implement *revwords* according to the following basic algorithm.

1. Call *rev* to completely reverse the original string. For example, *I Love Lucy* becomes *ycuL evoL I*. Remember that strings are terminated by a 0 byte.
2. Then use *rev* to reverse each word in the new string—*ycuL evoL I* becomes *Lucy Love I*. You can assume that "words" are separated by a space character, with the byte value 32.

Question 2: Forwarding

Consider the pipelined datapath shown on page 15.

Part (a)
Describe the difficulty in executing the following instructions in this datapath. How many stall cycles must be inserted to correctly execute this code? (5 points)

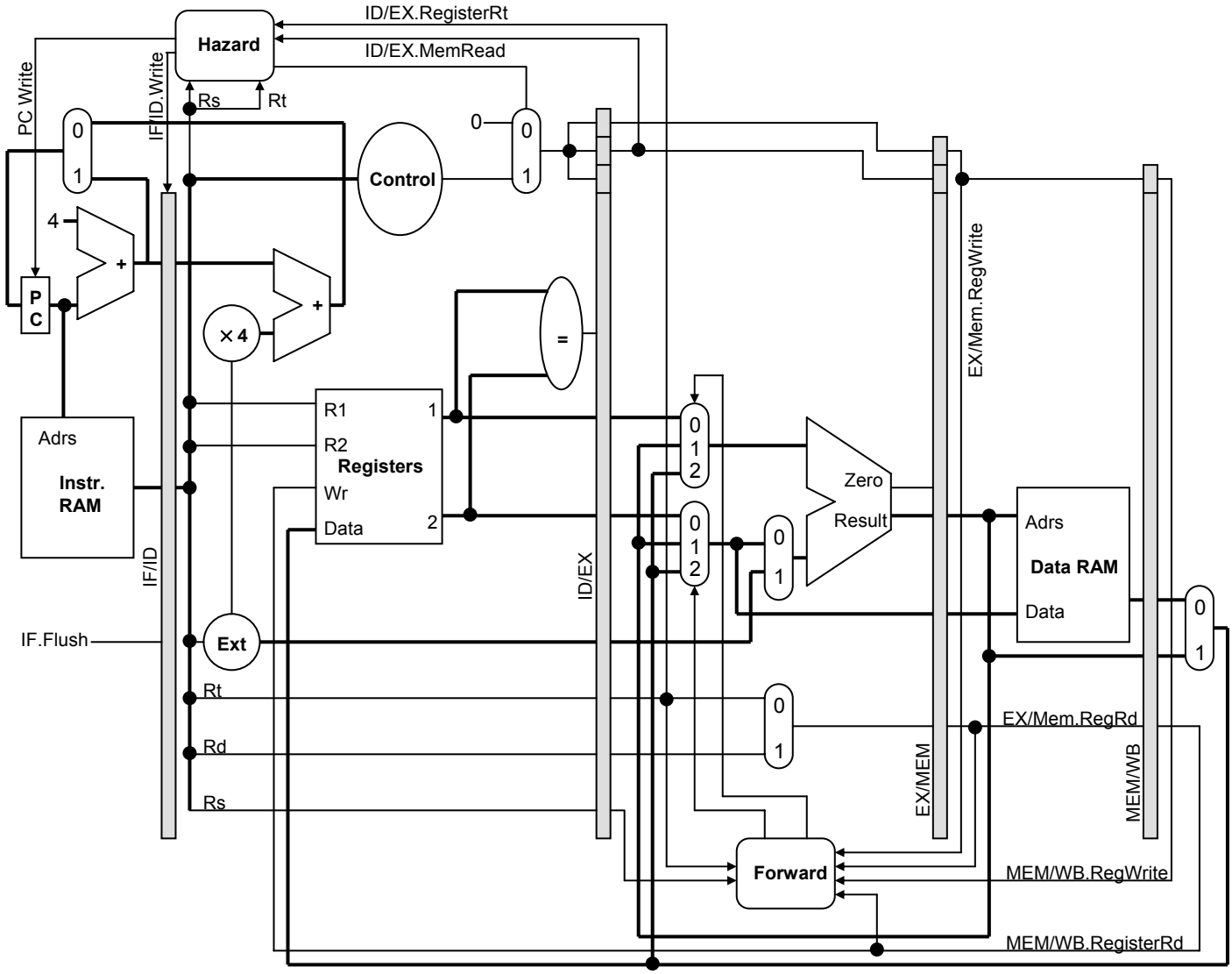
```
sub   $a3, $a3, $a1
beq   $a3, $0, SomewhereOverTheRainbow
```

Part (b)
Explain how the number of stalls could be minimized with forwarding. Show how muxes might be added to the pipelined datapath on the *next* page, and discuss how they can be controlled by a forwarding unit. Giving equations is the most concise approach, but in any case make sure your description is clear and precise.

You can assume the register file and ALU have the same latency, while muxes have negligible delays. You only need to consider dependencies between branches and R-type arithmetic instructions with the format shown below—you don't have to worry about loads or other I-type instructions. (20 points)

Fields	op	rs	rt	rd	shamt	func
Bits	31-26	25-21	20-16	15-11	10-6	5-0

Question 2 continued



Question 3: Pipelining performance

Here is a short MIPS assembly language loop. (This is a simpler version of a very common operation in scientific applications.) Assume that we run this code on the pipelined datapath shown on page 15.

```
Frodo:
    lw    $t0, 0($a1)
    mul   $t1, $t0, $a2
    lw    $t0, 0($a0)
    add   $t0, $t0, $t1
    sw    $t0, 0($a0)
    sub   $a3, $a3, 1
    addi  $a0, $a0, 4
    addi  $a1, $a1, 4
    bne   $a3, $0, Frodo
```

Part (a)
Find the number of clock cycles needed to execute this code, accounting for all possible stalls and flushes. Assume that $a3 is initially set to 100. (5 points)

Part (b)
Rewrite the code to eliminate as many stalls as possible. Show your modified assembly language program below. (10 points)

Question 3 continued

Part (c)
What is the performance improvement of your new function as compared to the original one? Again, assume that $a3 is initially set to 100. (5 points)

Part (d)
Suggest a way to rewrite this code to reduce the number of cycles lost to flushes. You do *not* have to show any actual code. (5 points)

Question 4: Cache performance

$$\text{AMAT} = \text{Hit time} + (\text{Miss rate} \times \text{Miss penalty})$$
$$\text{Memory stall cycles} = \text{Memory accesses} \times \text{miss rate} \times \text{miss penalty}$$

The Corleone 2000 processor has two levels of data caches, with the characteristics shown below. You can also assume that it takes 50 clock cycles to request and complete a 32-byte transfer between main memory and the L2 cache.

	L1	L2
Data size	32 KB	256 KB
Block size	8 bytes	32 bytes
Associativity	Direct-mapped	4-way
Hit time	1 cycle	19 cycles
Miss rate	5%	2%

Part (a)
How many bits of a 32-bit main memory address would be used as the tag, index and block offset fields, for each of the two caches above? Assume that 1 KB = 2^{10} bytes. Also, 32 = 2^5 and 256 = 2^8. (5 points)

	L1	L2
Tag size		
Index size		
Block offset		

Part (b)
Find the average memory access time for both the L2 and L1 caches. (5 points)

Question 4 continued

Part (c)

While you're still in the mood, let's look at the `Frodo` code again. Assume that $a0 and $a1 are initially 3000 and 4000, and the loop iterates 100 times. Both level 1 and level 2 data caches are empty at first. How many memory stall cycles would be required for the execution of this code, according to the formula and table given on the previous page? (5 points)

```
Frodo:
    lw    $t0, 0($a1)
    mul   $t1, $t0, $a2
    lw    $t0, 0($a0)
    add   $t0, $t0, $t1
    sw    $t0, 0($a0)
    sub   $a3, $a3, 1
    addi  $a0, $a0, 4
    addi  $a1, $a1, 4
    bne   $a3, $0, Frodo
```

Part (d)

Briefly discuss how each of the different write policies (write-back or write-through for hits, and allocate-on-write or write-around for misses) might affect the performance of the `Frodo` function. (10 points)

Question 5: Cache computations

Here is a short C program that we can use to determine some information about a processor's cache.

```
char a[LENGTH];          // array of bytes
int i, j, temp;

for (i = 0; i < 10000; i = i + 1)
   for (j = 0; j < LENGTH; j = j + STRIDE)
      temp = temp + a[j];
```

The inner loop reads data from an array of *bytes* in a pattern that depends on `STRIDE`; different values of `LENGTH` and `STRIDE` may cause the inner loop body to execute different numbers of times. The entire inner loop is repeated 10,000 times. You can assume that the variables `i`, `j`, `LENGTH`, `STRIDE`, `temp` and `a` are all stored in registers, and only the array contents are kept in main memory.

We run this program on a processor with several different values of `LENGTH` and `STRIDE`, and compute the average amount of time to execute "`temp = temp + a[j]`" once. The results are summarized in the table below, with `LENGTH` and `STRIDE` given in bytes and average times in nanoseconds.

LENGTH \ STRIDE	1	2	4	8	16	32	64	128
8	7 ns	7 ns	7 ns	7 ns	6 ns	7 ns	7 ns	7 ns
16	8 ns	7 ns	7 ns	6 ns	6 ns	7 ns	7 ns	7 ns
32	7 ns	8 ns	7 ns	7 ns	7 ns	7 ns	6 ns	7 ns
64	16 ns	21 ns	45 ns	45 ns	7 ns	8 ns	8 ns	9 ns
128	14 ns	25 ns	47 ns	45 ns	45 ns	7 ns	7 ns	7 ns
256	17 ns	25 ns	43 ns	45 ns	45 ns	46 ns	8 ns	7 ns

Use the data in this table to answer the questions on the next page. Explain your reasoning in each part.

Question 5 continued

Part (a)
What is the size of the cache, not including valid and tag bits? (5 points)

Part (b)
What is the block size? (5 points)

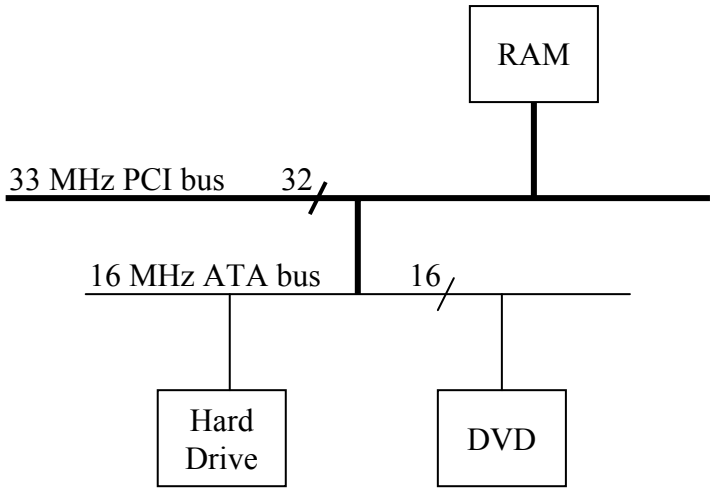
Part (c)
Is the cache direct mapped, fully associative, or *n*-way associative (and what is *n*)? (10 points)

Part (d)
What are the approximate hit and miss times? (5 points)

Question 6: Input/Output

Big Howie is upgrading his computer system!

- Howie's new DVD-ROM drive advertises a seek time of 85ms, a rotational speed of 7500 rpm (or 8 ms per rotation), and a transfer rate of 25,000,000 bytes per second.
- He also bought a new 40 GB hard drive that has 4 double-sided platters and 20,000 tracks on each surface. Each track has 500 sectors, and each sector contains 500 bytes. The drive's rotational speed is 9600 rpm, which works out to 6.25 ms per revolution.
- Finally, he has shiny new SDRAM memory that transfers data at a 133 MHz clock rate.



All of this precious hardware is connected as shown above. The main PCI system bus is 32-bits wide and runs at 33 MHz. The drives are connected to a slower, narrower "ATA" bus attached to the PCI bus.

Compute exact answers for the questions below, instead of giving unsimplified arithmetic expressions. Also be sure to specify the units for your solutions.

Part (a)
How much time would it take for the DVD-ROM drive to read a random 500,000-byte sector? (5 points)

Part (b)
Compute the hard drive's maximum data transfer rate per second, assuming that it can read and write data as fast as it spins. (10 points)

Question 6 continued

Part (c)
How long is needed to copy an entire 5 GB (5×10^9 bytes) DVD to the system's main memory? (5 points)

Part (d)
How long does it take to copy 5 GB (5×10^9 bytes) of data from RAM to the hard drive? (5 points)

MIPS Instructions

These are some of the most common MIPS instructions and pseudo-instructions, and should be all you need. However, you are free to use *any* valid MIPS instructions or pseudo-instruction in your programs.

Category	Example	Meaning
Arithmetic	add $t0, $t1, $t2	$t0 = $t1 + $t2
	sub $t0, $t1, $t2	$t0 = $t1 – $t2
	addi $t0, $t1, 100	$t0 = $t1 + 100
	mul $t0, $t1, $t2	$t0 = $t1 × $t2
	move $t0, $t1	$t0 = $t1
	li $t0, 100	$t0 = 100
Data Transfer	lw $t0, 100($t1)	$t0 = Mem[100 + $t1] (one word)
	sw $t0, 100($t1)	Mem[100 + $t1] = $t0 (one word)
	lb $t0, 100($t1)	$t0 = Mem[100 + $t1] (one byte)
	sb $t0, 100($t1)	Mem[100 + $t1] = $t0 (one byte)
Branch	beq $t0, $t1, Label	if ($t0 == $t1) go to Label
	bne $t0, $t1, Label	if ($t0 != $t1) go to Label
	bge $t0, $t1, Label	if ($t0 >= $t1) go to Label
	bgt $t0, $t1, Label	if ($t0 > $t1) go to Label
	ble $t0, $t1, Label	if ($t0 <= $t1) go to Label
	blt $t0, $t1, Label	if ($t0 < $t1) go to Label
Set	slt $t0, $t1, $t2	if ($t1 < $t2) then $t0 = 1; else $t0 = 0
	slti $t0, $t1, 100	if ($t1 < 100) then $t0 = 1; else $t0 = 0
Jump	j Label	go to Label
	jr $ra	go to address in $ra
	jal Label	$ra = PC + 4; go to Label

The second source operand of *sub*, *mul*, and all the branch instructions may be a constant.

Register Conventions

The *caller* is responsible for saving any of the following registers that it needs, before invoking a function:

$t0-$t9 $a0-$a3 $v0-$v1

The *callee* is responsible for saving and restoring any of the following registers that it uses:

$s0-$s7 $ra

Performance

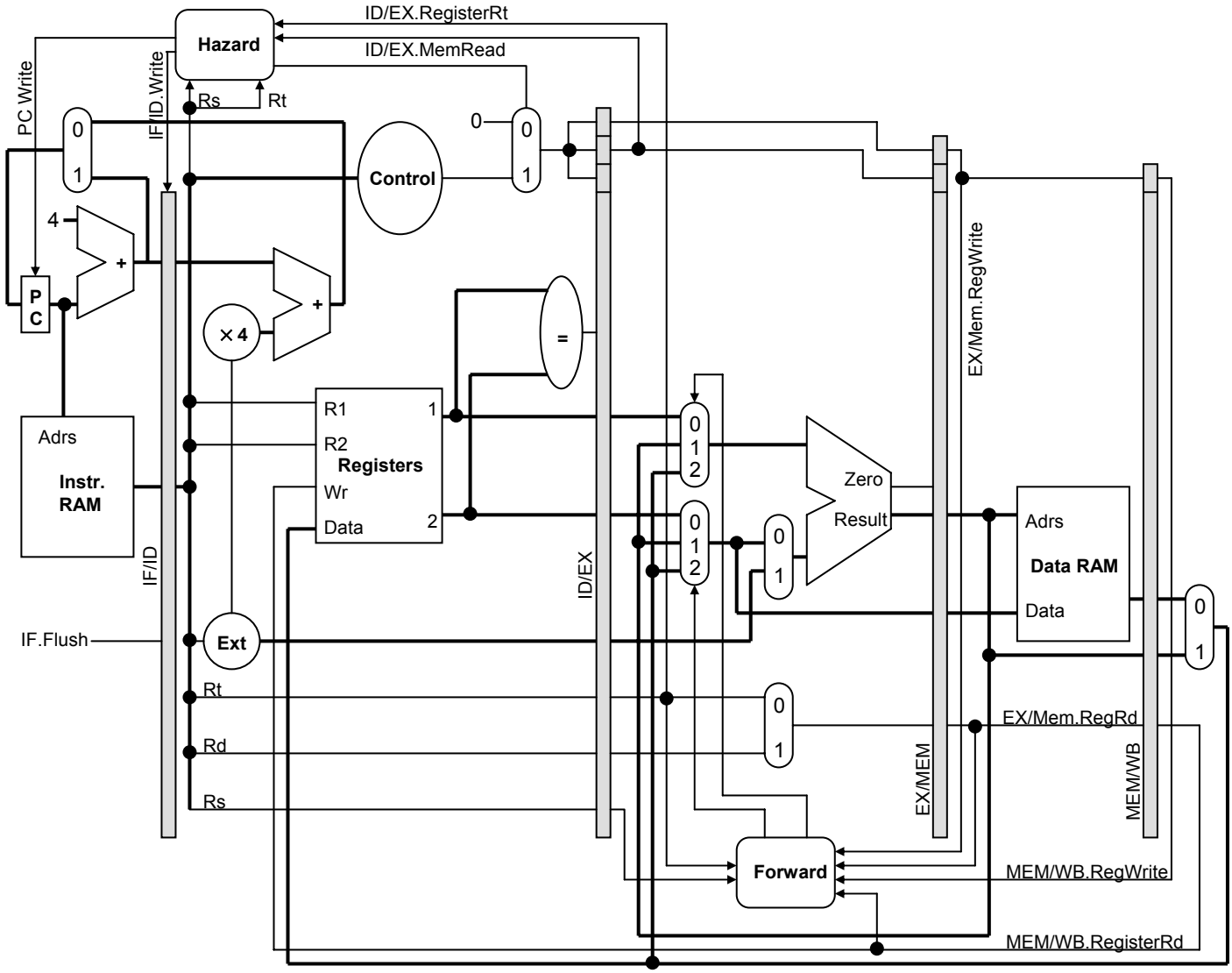
Formula for computing the CPU time of a program P running on a machine X:

$$\text{CPU time}_{X,P} = \text{Number of instructions executed}_P \times \text{CPI}_{X,P} \times \text{Clock cycle time}_X$$

CPI is the average number of clock cycles per instruction, or:

$$\text{CPI} = \text{Number of cycles needed} / \text{Number of instructions executed}$$

Pipelined Datapath

Here is the final datapath that we discussed in class.

- Forwarding for arithmetic instructions is done from the EX/MEM and MEM/WB stages to the ALU.
- A hazard detection unit can insert stalls for lw instructions.
- Branches are assumed to be not taken.
- To minimize the number of flushes on a misprediction, branch determination is done in the ID stage.