Full Name:______________________________________

Andrew ID:______________________________________

Section :______________________________________

CS 15-347, Spring 1996

Exam 2

March 21, 1996

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew ID *on every page indicated*.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 50 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like.

Do not write below this line

1. _________	3. _________	5. _________	7. _________
2. _________	4. _________	6. _________	8. _________

Introduction to Problem 1

Consider the pipelined pMIPS architecture used in Lab 2. Remember that it has the following properties:

- The register file does a write before the two reads.
- Forwarding is used to handle hazards wherever possible. When hazards cannot be resolved through forwarding, the appropriate pipe registers are stalled.
- When a branch is taken, the pc pipe register is written during the MEM stage of the instruction, and the new PC is used on the next cycle. A branch has one delay slot; the instructions after the delay slot that have been fetched are canceled.
- Jumps are handled like branches.
- Loads have no delay slot. An instruction following a load can use the loaded result as an operand. This is handled by stalling the following instruction for one cycle and then forwarding the load result to it.

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 1. (9 points):

Consider this piece of machine code that searches a sequence of words for a terminating zero word:

```
$L1:    lw $2, 0($3)
        bne $2, $0, $L1
        addi $3, $3, 4
        nop
        nop
```

In each of the following questions, assume that the branch is taken.
Which of the following pairs of instructions require the processor to stall? Circle the pairs of instructions that would have at least one stall cycle between them, and cross out those that would not. Note that pipeline bubbles created by cancelled instructions don't count as stalls.

A. From `lw` to the `bne`

B. From `bne` to the `addi`

C. From `addi` to the `lw` of the next iteration

Which of the following pairs of instructions are handled by forwarding? Circle the pairs of instructions that represent a hazard that is handled (possibly partially) by forwarding, and cross out the ones that don't. (Remember to assume that the branch is taken).

A. From `lw` to the `bne`

B. From `lw` to the `addi`

C. From `addi` to the `lw` of the next iteration

D. From `addi` to the `addi` of the next iteration

Answer each of the following questions.

A. What is the interval between successive iterations of this loop? i.e., how many cycles are there between the beginning of one `lw` instruction and the beginning of the `lw` instruction in the next iteration?

B. What is the latency of one iteration of this loop? I.e., how many cycles does it take from the beginning of the `lw` instruction to the end of the `addi` instruction?

Introduction to Problems 2 and 3

In Lab 2, we assumed that the register file was capable of performing a write at the beginning of a clock period and then passing through this new value to its read ports shortly thereafter. That is, the register file supplied the correct result if the instruction in the ID stage read the same register written by the instruction in the WB stage. We might refer to this as a "write-before-read" register file.

Now, suppose that the register file does not have this capability – that is, that the register file supplies the old, out-of-date value if the instruction in the ID stage reads the same register written by the instruction in the WB stage. We might refer to this as a "read-before-write" register file.

Suppose we start off with a datapath and hazard detection unit that correctly handled data hazards in the case where the register file was write-before-read. Specifically, suppose we start off with the datapath shown in Figure 1, and that, with a write-before-read register file, this datapath is used to implement a processor with the same properties as the pMIPS architecture outlined in Problem 1 of this exam (and in Lab 2).

Unfortunately, the datapath in Figure 1 does *not* correctly implement the pMIPS architecture in the case where the register file is read-before-write.

Some important points on the datapath have been labelled with the letters 'A' through 'I'.

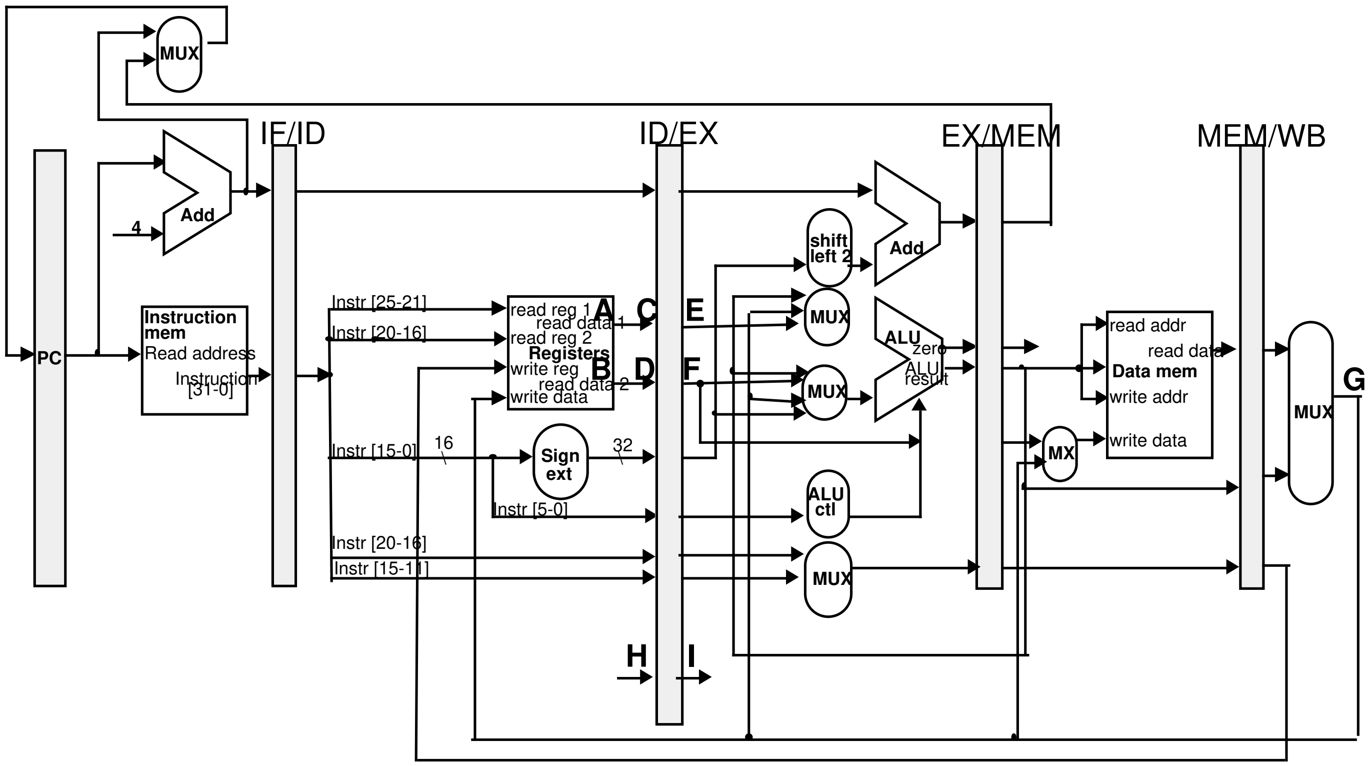

Figure 1: Datapath with forwarding for write-before-read register file

Full Name:______________________________________

Andrew ID:______________________________________

Problem 2. (8 points):

The instruction stream below creates data hazards that the previously correct datapath and forwarding scheme from Lab 2 no longer handle properly.

```
1               or      $3, $2, $1
2               and     $4, $3, $6
3               addu    $7, $3, $4
4               subu    $3, $2, $4
5               or      $8, $4, $3
6               la      $3, point1      #puts address of point1 in $3
7               jalr    $3
8               or      $4, $5, $3
9               and     $6, $31, $4
10              subu    $4, $5, $6
11              nop
12      point1: subu    $7, $8, $9
13              nor     $3, $3, $31
14              and     $2, $4, $5
15              ori     $7, $7, 42
```

Circle the pairs of instructions below that *would not* be handled correctly by the forwarding scheme from Lab 2, and cross out the pairs that would. [Remember to take into account the effects of stalling and instruction cancelling]

A. Line 1 and Line 4

B. Line 1 and Line 5

C. Line 2 and Line 4

D. Line 2 and Line 5

E. Line 7 and Line 9

F. Line 7 and Line 13

G. Line 8 and Line 14

H. Line 12 and Line 15

Full Name:______________________________________

Andrew ID:______________________________________

Problem 3. (5 points):

Continuing with the register file from Problem 2: Which of the following changes to the datapath in Figure 1 would fix the hazards that have been added due to the changed register file *without causing other hazards to be handled incorrectly*? (Assume that the appropriate changes are also made to the control logic for the MUXes.) Circle the changes that work, and cross out the ones that don't.

A. Add two MUXes between the register file and the ID/EX pipeline register. The first MUX takes inputs from point A (the first register file read port) and point G (the WB stage), and sends its output to point C (an ID/EX pipe register input); the second MUX takes inputs from point B (the second register file read port) and point (the WB stage) G, and sends its output to point D (another ID/EX pipe register input).

B. Move the two MUXes previously in front of the ALU so that they now lie between the register file and the ID/EX pipeline register. Change the two MUX inputs that used to come from points E and F to come from points A and B instead, and feed the MUX outputs into points C and D. Don't rewire any of the other MUX inputs. Take the inputs for the ALU directly from points E and F.

C. Send the output at point G to the extra ID/EX register input at point H, and add inputs to both MUXes that take their signals from the extra ID/EX register output at point I.

D. When a hazard occurs due to the change in the register file, stall the IF stage and allow the other stages to continue, inserting bubbles as necessary into the IF/ID pipeline register.

E. When a hazard occurs due to the change in the register file, stall the IF and ID stages and allow the other stages to continue, inserting bubbles as necessary into the ID/EX pipeline register.

Introduction to Problem 4

Problem 4 concerns the circuit shown in Figure 2. Each circle represents a block of combinational logic. Each block is referred to by a letter name, and is labeled with the delay of the block in nanoseconds. Each arrow represents a path between blocks.

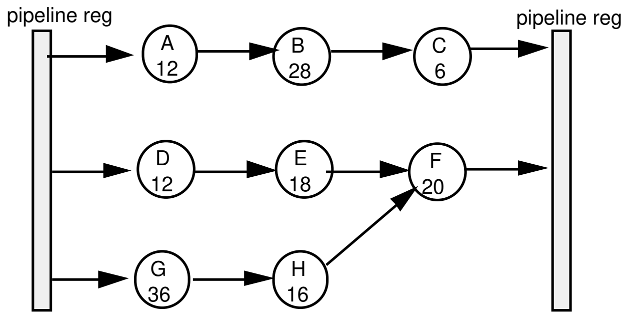

Figure 2: High-level depiction of circuit

Here are some assumptions and constraints for all questions:

- Assume that the propagation delay along an arrow is zero.
- Do not make any changes that increase latency without also increasing throughput (this makes the problem easier).

Full Name:_______________________________________
Andrew ID:_______________________________________

Problem 4. (8 points):

A. Suppose that you are not allowed to split any of the blocks into smaller blocks. Add additional pipeline registers to maximize the throughput of the circuit in Figure 2 and draw the new circuit. Label each combinational block with its letter name. Assume that each pipeline register adds 2 nanoseconds of delay.

B. What is the throughput of your circuit in MHz?

C. What is the latency of your circuit in nanonoseconds?

Now suppose that you can split any block into at most two smaller blocks. However the cost of the splitting is that the delay through each of the new "halves" is $3/4\,x$ (not $1/2\,x$), where x is the original delay. This delay of $3/4\,x$ does not include the delay through a pipeline register, which remains 2 ns.

A. Given this new freedom to split blocks, design a circuit that maximizes throughput and draw it below. If you split some block named Y, label the two new blocks Y' and Y''.

B. What is the throughput of the new circuit in MHz?

C. What is the latency of the new circuit in nanoseconds?

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 5. (4 points):

For each of the following exceptions, circle the *earliest* (i.e., leftmost) pipeline stage in which the exception could be detected. If it is an external exception that is not related to any particular pipeline stage, circle n/a instead. There should be exactly one circled item in each row.

Arithmetic overflow	IF	ID	EX	MEM	WB	n/a
Hard reset:	IF	ID	EX	MEM	WB	n/a
Unaligned store:	IF	ID	EX	MEM	WB	n/a
Interrupt:	IF	ID	EX	MEM	WB	n/a

Introduction to Problem 6

Consider the following flawed instruction sequence:

```
and $1, $2, $3
ld  $4, 0($5)  -- unaligned address exception (MEM)
add $6, $7, $8
<illegal instr>-- illegal instruction exception (ID)
or  $9, $10, $11
```

where the `ld` instruction uses an unaligned address and the instruction following the `add` contains an illegal opcode. Suppose that the system has a single exception branch location (like MIPS), and that the first two instructions in the handler at that location are:

```
sw $1, 1000($0) -- start of interrupt handler
sw $2, 1004($0)
```

Full Name:____

Andrew ID:____

Problem 6. (4 points):

Suppose that both exceptions in the code sequence on the previous page are detected in clock cycle c_4 at runtime (as shown below). The unaligned address exception is detected in the MEM stage, and the illegal instruction exception is detected in the ID stage. Show the action of the processor during clock cycles c_5 and c_6 by filling in the 12 blank entries in the table below. If the instruction has been cancelled, indicate this with the word "nop", otherwise write the stage (WB, MEW, etc.) that the instruction is executing. Don't forget to write the names of the instructions in the first column.

	Clock cycle						
Instruction	c_0	c_1	c_2	c_3	c_4	c_5	c_6
and	IF	ID	EX	MEM	WB	—	—
ld	—	IF	ID	EX	MEM		—
add	—	—	IF	ID	EX		
<illegal instr>	—	—	—	IF	ID		
or	—	—	—	—	IF		
	—	—	—	—	—		
	—	—	—	—	—	—	

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 7. (6 points):

Consider a computer with a 4-bit word size, using a two's complement representation of (signed) integers. Suppose that the multiply instruction for this machine returns the low order 4 bits of the product, while the add instruction returns the low order 4 bits of the sum. Fill in the following table with the bit patterns and numeric values

Expression	Bit Pattern	Decimal Value
`-1`		
Most positive		
Most negative		
`5 * 5`		
`6 * 6`		
`6 + 5`		

Full Name:______________________________________

Andrew ID:______________________________________

Problem 8. (6 points):

Imagine the following, scaled down version of the IEEE Floating Point Format:

- There is no sign bit, only nonnegative numbers can be represented.
- There are only three exponent bits. The exponent bias is 3.
- There are only two significand bits.
- Otherwise, the rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NAN). Refer to the table at the end of Section 4.8.

The purpose of this problem is to translate bit patterns in this small format into reduced rational numbers. That is, you should express the final result as a number of the form n/d where n and d are expressed in base 10, and they are relatively prime (e.g., $3/8$ rather than $6/16$).
You should do this by filling in the table below, having the following columns:

Exponent Bits The 3-bit exponent pattern (given)

Significand Bits The 2-bit significand pattern (given)

Exponent Rational The number represented by the exponent bits.

Mantissa Binary The binary number (with binary point) represented by the significand bits plus any "hidden" 1.

Mantissa Rational The number represented by the mantissa.

Value Rational The number represented by this combination of mantissa and exponent.

Exponent Bits	Significand Bits	Exponent Rational	Mantissa Binary	Mantissa Rational	Value Rational
000	00				
000	01				
001	00				
011	10				
110	11				
111	00				