CS152
COMPUTER ARCHITECTURE AND ENGINEERING

EXAMINATION #2

NAME:_____

DISCUSSION SECTION TIME:_____

PROBLEM NUMBER	SCORE
# 1	
# 2	
# 3	
# 4	
TOTAL SCORE	

NOTE: Please show your work CLEARLY for all problems. I hope you enjoy the test!

PROBLEM 1: PIPELINING/HAZARDS

For all five parts of this question, assume that we are using the five-stage pipelined MIPS machine described in the CS152 textbook.

a. (6 points) The following is some code from Mr. Oza's Nut Factory. Assume that the pipelined datapath has NO FORWARDING. Find the register hazards in the following code. Enter your answers in the table on the next page. Also, for each hazard that you find, classify the hazard (under "Hazard Type" in the table below) into one of the following three types:

> (1). The write register of the instruction in the EXECUTION stage is the same as the read register of the instruction in the INSTRUCTION DECODE stage.
>
> (2). The write register of the instruction in the MEMORY stage is the same as the read register of the instruction in the INSTRUCTION DECODE stage.
>
> (3). The write register of the instruction in the WRITE-BACK stage is the same as the read register of the instruction in the INSTRUCTION DECODE stage

When designating the two instructions between which there is a hazard (under Instruction#1 and Instruction #2 below), use the number to the left of the instruction. When designating the type of hazard, use the number corresponding to one of the three hazards listed above.

1. add $3,$1,$2
2. lw $1,0($4)
3. and $5,$3,$4
4. and $6,$1,$2
5. or $1,$3,$6
6. sw $1,4($4)
7. lw $2,4($4)
8. sub $3,$5,$6

Instruction #1	Instruction #2	Register(s)	Hazard Type
1	3	$3	2
2	4	$1	2
4	5	$6	1
5	6	$1	1

b. (5 points) Ben "The Hazard Buster" Bitdiddle (remember him?) says that instruction 7 might not load the value stored by instruction 6. Is this true? Why or why not?

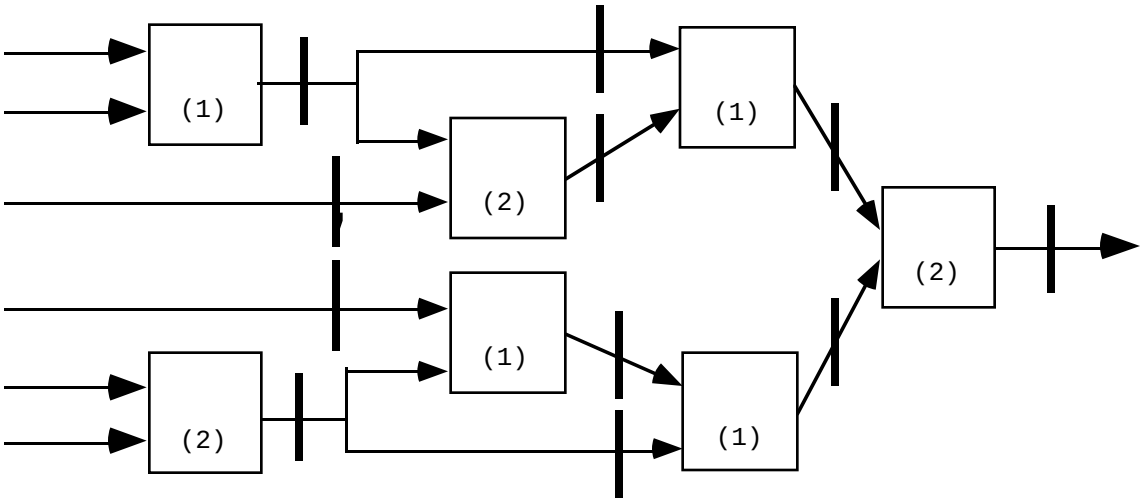
Solution: This is NOT true. The reason is that memory reads AND writes take place in only one stage of the pipeline (the MEM stage). Since the MEM stage of an earlier instruction ALWAYS comes before the MEM stage of a later instruction, memory reads/writes of earlier instructions are always completed before those of subsequent instructions.

c. (6 points) Rewrite the code above WITHOUT CHANGING ITS EFFECT so that it has the fewest number of nop instructions. Again, assume there is NO FORWARDING.

Solution:

```
1.    add $3,$1,$2
2.    lw  $1,0($4)
      nop
      nop
3.    and $5,$3,$4
4.    and $6,$1,$2
      nop
      nop
      nop
5.    or  $1,$3,$6
8.    sub $3,$5,$6    /* Note that instruction 8 has been moved up
      nop
      nop
6.    sw  $1,4($4)
7.    lw  $2,4($4)
```

d. (4 points) Pipeline the following circuit for maximum throughput by adding pipeline registers (by drawing vertical lines on one or more wires) at appropriate places. Use as few pipeline registers as possible. On each component, the number in parentheses is that component's latency. THERE ARE NO WIRE DELAYS OR OTHER DELAYS IN THE CIRCUIT. None of the components are clocked; therefore, you have to make sure that, for each component, both inputs arrive at the same time.



e. (4 points) What is the lowest possible cycle time for the pipelined version of the circuit above? How can you figure it out without actually pipelining the circuit?

Solution: The lowest possible cycle time for the pipelined version of this circuit is 2. This is true because the component with the longest latency has latency 2. During a cycle, all components must have a chance to finish their computation, so the cycle time needs to be at least 2 to allow the slowest components to finish.

PROBLEM 2: BUSES

a. (6 points) Name three traditionally classified buses and their features:

1. Processor-memory buses: short, genrally high speed, and matched to the memory system so as to maximize memory-processor bandwidth.
2. I/O buses: lengthy, can have many types of devices connected to them, and often have a wide range in the data bandwidth of the devices connected to them.
3. Backplane buses: designed to allowprocessor, memory, and I/O devices to coexist on a single-bus.

b. (16 points) We want to compare the maximum bandwidth for a synchronous and an asynchronous bus.

- The synchronous bus has a clock cycle time of 30ns, and each bus transmission takes 2 clock cycles.
- The asynchronous bus requires 35ns per handshake.
- The data portion of both buses is 32 bits wide.
- Addresses are 32 bits(word) long but the returning data are 64 bits(double word) long.

Find the bandwidth for each bus when performing one-double-word READS from a 100ns memory (64 bits data).

Calculations for Synchronous bus (4 points):

1. Send the address to memory: 30ns x 2 cycles = 60ns
2. Read the memory: 100ns
3. Send the first half of the data to the device: 30ns x 2 cycles = 60ns
4. Send the second half of the data to the device: 30ns x 2 cycles = 60ns

Thus, the total time is 280ns. This yields a maximum bus bandwidth of
8 bytes every 280ns, or

$$\frac{8 \text{ bytes}}{280\text{ns}} = 28.57 \frac{\text{MB}}{\text{second}}$$

Synchronous Bus bandwidth (4 points): 36.4 MB/second (Megabytes/Second)

Calculations for Asynchronous bus (4 points):

1. Mem sees the Readreq line: 35ns
2. I/O devices sees the Ack: 35ns
3. Mem sees the Readreq low: 35ns
4* When data is ready: Max ((100ns-70ns),(35ns*3-70ns)) = 35ns
5. I/O sees DataRdy: 35ns
6. Mem sees Ack: 35ns
7. I/O sees DataRdy low: 35ns
8* Mem sees Ack Low: 35ns
9. I/O sees DataRdy: 35ns
10. Mem sees Ack: 35ns
11. Finally I/O seeing DataRdy go low, drops the Ack line: 35ns

Step 1: 35ns
Steps 2,3,4: maximum (3x35ns, 100ns)=105ns
Steps 5,6,7,8,9,10,11: 7x35ns=245ns

Thus, total time to perform the transfer is 385ns, and the maximum bandwidth is

$$\frac{8 \text{ bytes}}{385\text{ns}} = 20.78 \frac{\text{MB}}{\text{second}}$$

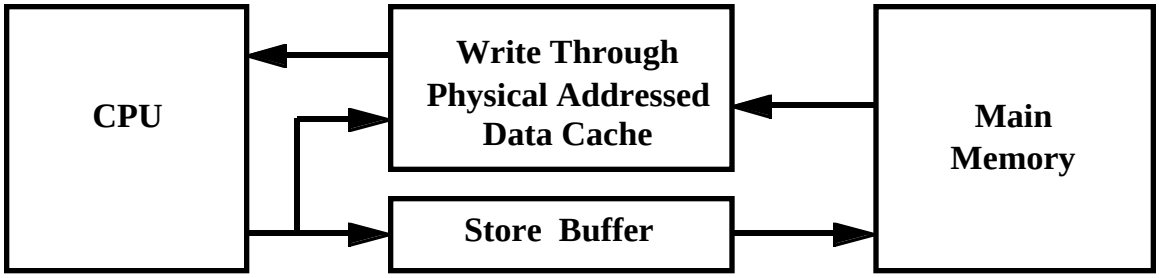
Asynchronous Bus bandwidth: 20.78 MB/second

c. (3 points) Which of two buses scales better with technology changes when used as I/O bus? Why?

Asynchronous buses are used as I/O buses due to its capacity in terms of physical distance and the number of devices that can be connected to the bus. As technology changes, asynchronous buses are more flexible to be able to support a wider variety of device response speed.

PROBLEM 3: MEMORY SYSTEM DESIGN

In this problem, you are going to design a memory system for a computer. One thing we want you to learn from this class is how to solve a BIG problem by solving a series of little problems. So we have divided this problem into multiple sub-problems.



First here are some numbers and equations you may need to solve this problem:

Virtual Memory Page Size = 4K Byte
DRAM Chips Available:
Size: 4 Megabit = 256K words x 8-bit
Read Access Time = Write Access Time = 50ns
Read Cycle Time = Write Cycle Time = 100ns

Data cache Options	Hit Time	Miss Rate	Miss Penalty
J-byte Direct-Mapped	1 cycle	8%	4 cycles
L-byte 2-way Set Associative	1 cycle	4%	6 cycles
M-byte 4-way Set Associative	1 cycle	2%	8 cycles

Note: J, L, M are values you need to pick in Part (a) of this problem

Queuing Theory Review:
Utilization = Request Rate / Service Rate
Average Q Length = Utilization / (1 - Utilization)
Probability of Overflow of a N-entry Q = $(\text{Utilization})^N$

Part (a) Physically Addressed Data Cache (12 points)

Part (a): The first thing you need to pick is the configuration of the write-through, physically addressed, data cache. Your options are:

- J-byte Direct Mapped
- L-byte 2-Way Set Associative
- M-byte 4-Way Set Associative

(a.1) (2 points): What are J, L, and M in order to index these caches with the virtual address?

Answer: J = 4 KB L = 2 x 4 KB = 8KB M = 4 x 4 KB = 16 KB

a.2 (6 points): Assume Load instructions have a CPI of 1.2 if they hit in the data cache and if they miss, they have a CPI of (1.2 + Miss Penalty). All other instructions have a CPI of 1.2. Furthermore assume 20% of the instructions are loads. What is the machine's CPI for each of the 3 cache options (keep 3 decimal points for your answer, example: 1.xxx)? Which cache should you use if CPI is the only criterion?

CPI (J-byte Direct Mapped) = 0.8 x 1.2 + 0.2 x (1.2 x 92% + (1.2+4) x 8%) = 1.264 (1.5 pt)

CPI (K-byte 2-Way) = 0.8 x 1.2 + 0.2 x (1.2 x 96% + (1.2+6) x 4%) = 1.248 (1.5 pt)

CPI (M-byte 4-Way) = 0.8 x 1.2 + 0.2 x (1.2 x 98% + (1.2+8) x 2%) = 1.232 (1.5 pt)

Since the 4-way set associative cache has the lowest CPI, we should use that if the CPI is the ONLY criterion (1.5 pt).

a.3 (4 points) Assume if the direct-mapped cache is used, the machine cycle time is 20ns. If the 2-way cache is used, cycle time is 22ns. 4-way cache is use, cycle time is 24ns. What is the average time per instruction for each option and what cache should you use?

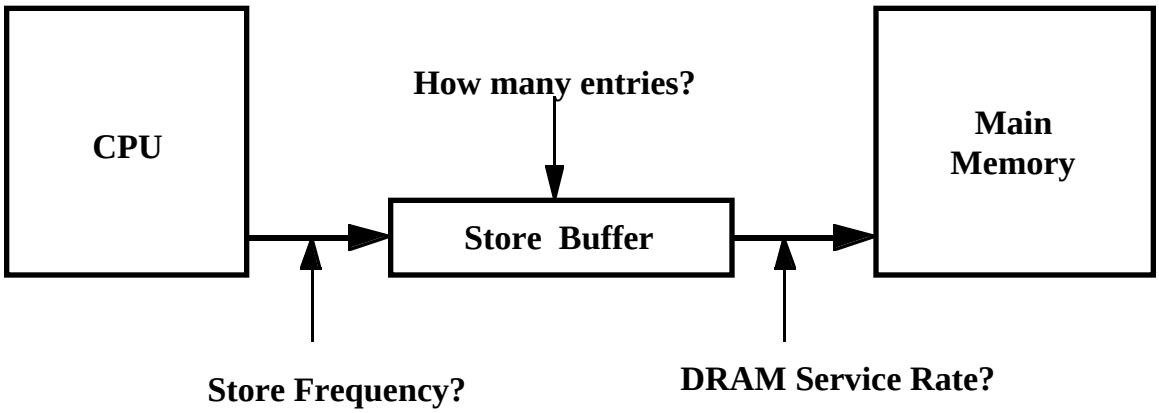
Average Time (Direct Mapped) = 1.262 x 20ns = 25.280ns/instruction (1pt)

Average Time (2-way) = 1.248 x 22ns = 27.456ns/instruction (1pt)

Average Time (4-way) = 1.232 x 24ns = 29.568ns/instruction (1pt)

We should pick the direct-mapped cache!!! (1 pt) Lesson: CPI alone can be misleading.

Part (b) Store Buffer Design (8 points)



(b) In this part, you need to pick the length of the store buffer. For this part of the problem we will assume (do NOT use the result of Part(a) for this part):

Machine's CPI = 1.3 Cycle Time = 20 ns

Percentage Instructions that are stores = 15 %

b.1 (1 point): What is the store frequency with respect to time?

Average Instruction Rate = 1 / (1.3 x 20ns) = 3.846 x 10**7 instr./sec
Store Frequency = 15% * (3.846 x 10**7) = 5.769 x 10**6 stores/sec

b.2 (1 point): What is the maximum rate at which DRAM can service store requests? Assume that the DRAM Write Cycle Time = 100ns.

DRAM Write Cycle Time = 100ns
Max. DRAM Service Rate = 1/100ns = 1.00 x 10**7 requests/sec

b.3 (2 points) What is the utilization and mean Q length based on queuing theory?

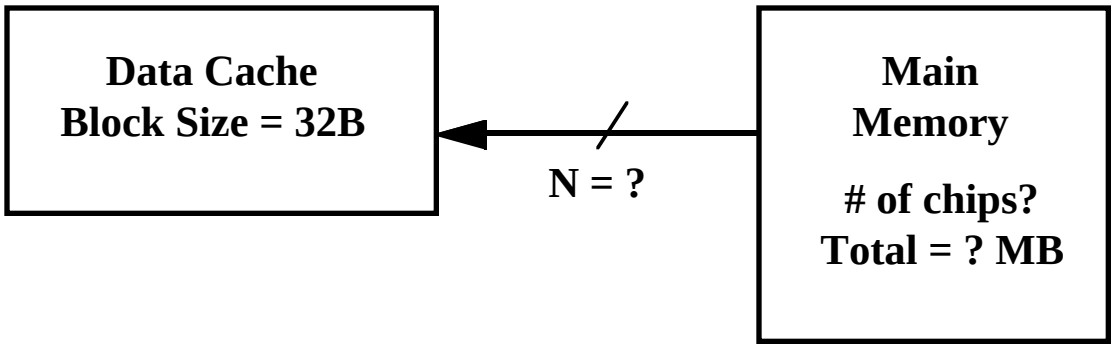
Utilization = 5.769*10**6/10 **7= 0.5769
Q Length= 0.5769 / (1 - 0.5769) = 1.363

b.4 (1 point) What is the probability of overflow if the queue has two entries?

(0.5769)**4 = 0.11 => 11% still too high
(0.5769)**6 = 0.038 => 3.8% => So 6 entries is good enough.

b.5 (3 points) What is the minimum number of entries the Store Buffer needs to have to lower the probability of buffer overflow to less than 4%?

Part (c) Main Memory Design (5 points)



(c) In this part, assume the data cache you designed in Part (a) has a Block Size = 32 B. On a cache miss, you want to fill up the cache in 2 DRAM read cycles.

c.1 (2 point) How wide does the datapath between the DRAM and your data cache have to be in order to achieve our goal of filling the cache in 2 DRAM read cycles?

The path has to be 32B/2 = 16 bytes or 128 bits wide.

c.2 (3 points) What is the minimum number of DRAM chips (256K x 8) you need for the main memory and what is the minimum memory size?

The DRAM chips are 256K x 8. In order to form the 128 bit wide path, we need 128/8 = 16 chips.

Total memory: (256 KB/ chip) x 16 chips = 4 MB

PROBLEM 4: MICROCODE

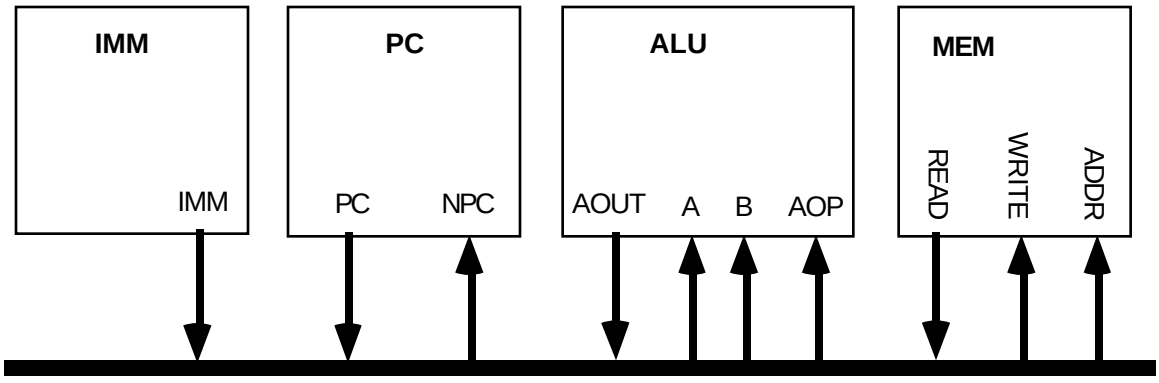
In this problem we will consider a very simple computer architecture design. This design has no general purpose registers, and all components are connected together via a common bus. For this question, you will have to write some simple programs. The instructions for this processor are very simple, being of the form:

Dest <- Source (i.e. "NPC <- AOUT")

Where Dest is one of {NPC, AOP, A, B, ADDR, WRITE} and Source is one of {PC, AOUT, READ, IMM}. If Source is IMM, there is some integer immediate data associated along with it. In this case, the instruction is written:

Dest <- Immediate (i.e. "A <- 5")

The physical architecture of the microprocessor looks like this:



In this architecture, there are no general purpose registers. Registers are emulated using a reserved section of main memory. Address 0 corresponds to register 0, address 1 to register 1, etc...

Program flow is controlled using the PC and NPC registers. The PC of the current instruction can be read from the PC register. If a value is written to the NPC register, the next instruction will be fetched from that address, effecting a jump. In a particular cycle, if no value is written to the NPC register, the PC advances and executes the next sequential instruction.

The ALU operation is controlled by the AOP register. AOP may be one of {ADD, SUB, SEQ}, which are all defined constants which may be used as the Immediate of an instruction. The two operands to the ALU are the A and B registers, and the result of the ALU operation can be read from AOUT. ADD is defined as A+B, SUB is defined as A - B, while SEQ is defined as 1 if (A == B), else 0 if (A != B).

The memory is a simple synchronous word-addressed one-cycle memory. For a read operation, the address is first placed in the ADDR register, then a data value can be read from the READ register. For a write, first place the address in the ADDR register, and then write the data to the WRITE register.

Immediate data values may be in the range -32 to 31 (6 bits signed 2's complement.) This system uses a shared instruction/data memory model. Large immediate values may be in-lined in the code and accessed using PC-relative addressing.

Show below are two examples, the first one very simple and the second one more complex.

example 1) Write out code equivalent to the MIPS instruction "addi r6, r5, #12"

AOP <- ADD	Get the ALU ready for addition.
ADDR <- 5	Get the memory ready to read 'register' 5.
A <- READ	Read the memory and store in register A.
B <- 12	Load the immediate constant 12.
ADDR <- 1	Get ready to write-back to 'register' 6.
WRITE<- AOUT	Write the ALU result to memory.

example 2) Write out code equivalent to the MIPS instruction "addi r6, r5, #-142"

AOP <- ADD	Get the ALU ready for addition.
A <- PC	Load the PC to calculate effective address.
B <- 11	13-2 = 11.(Data at line 13, PC from line 2)
ADDR <- AOUT	Save the address of the in-line immediate.
B <- READ	Load the in-line immediate data.
ADDR <- 5	Get ready to load 'register' 5.
A <- READ	Read the input data from memory.
ADDR <- 6	Get ready to save the result.
WRITE <- AOUT	Save the result of the ALU add.
A <- PC	Now we need to skip over line #13, because
B <- 4	it is immediate data and won't execute.
NPC <- AOUT	So use 14-10 = 4 added to the PC and jump!
-142	Data for the ALU add. Don't try to execute!

Below are four incomplete code fragments. Fill in all the missing blanks, INCLUDING COMMENTS.

a) (5 points) Write out code equivalent to the MIPS instruction "sub r10, r11, r12"

AOP <-	Get the ALU ready to do a subtract.
ADDR <- 11	Get ready to read memory location 11.
A <- READ	Read the memory into register A.
ADDR <-	
B <- READ	Read the memory into register B.
<-	
WRITE <- AOUT	Write the result to memory.

b) (5 points) Write out code equivalent to the MIPS instruction "jalr r10"

AOP <- ADD	Ready the ALU for adding a PC offset.
A <- PC	Return address is relative to the PC.
B <-	
ADDR <- 31	Get ready to write to 'register' 31.
<-	
<-	
NPC <- READ	Jump to the new address.

c) (5 points) Write out code equivalent to the MIPS instruction "lw r4, 16(r3)"

AOP <- ADD	Ready the ALU for calculating the address.
ADDR <- 3	Get ready to load address 3.
A <- READ	Read in the register.
B <- 16	Our immediate offset.
<-	Save the calculated address.
B <- 0	Set up the ALU to be a temporary register.
<-	Load the data from memory.
ADDR <- 4	Get ready to save at memory location 4.
<-	Save our result into main memory.

d) (10 points) Write out code equivalent to the MIPS instruction "bne r4, r5, -4"

0. <<< If r4 != r5, this branch will infinite loop >>>

AOP <- SEQ	Ready to ALU for the comparison
ADDR <- 4	Ready to read memory 4
A <- READ	Read location 4
ADDR <- 5	Ready to read memory 5
B <- READ	Read location 5
A <-	
B <-	
AOP <- ADD	No more comparison tests
A <- AOUT	
B <- PC	Compute offsets from here.
<-	
<-	
NPC <- AOUT	And jump!
- 9	!=, jump to line 1, 1-10 = -9.
6	==, jump to line 16, 16-10 = 6.
	<<< If r4 = r5, continue execution here >>>

THE END!!