CS232 Midterm Exam 3
April 23, 2004

Name: **Bill**

Section: **Kill**

- This exam has 7 pages including the pipelined datapath diagram on the last page, which you are free to tear off.
- You have 50 minutes, so budget your time carefully!
- No written references or calculators are allowed.
- To make sure you receive credit, please write clearly and show your work.
- We will not answer questions regarding course material.

Question	Maximum	Your Score
1	10	**10**
2	45	**45**
3	45	**45**
Total	100	**7**

Question 1: Conceptual Questions (10 points)

Limit your answers to two sentences.

Part (a)

The statement “Pipelining improves both instruction latency and throughput” is inaccurate or imprecise. Explain why this statement isn’t entirely true (be specific). (5 points)

Pipelining improves throughput, because it allows running multiple instructions at once. But it does not improve latency – all instructions are forced to take 5 cycles to execute.

Part (b)

Why is performance of a write-back cache generally better than a write-through cache? What property of programs does write back exploit? (5 points)

Not every write is sent to memory, which conserves memory bandwidth. Write back exploits temporal locality of data writes.

Question 2: Pipelining (45 points)

Part (a)
Consider executing the following *dependence-free* MIPS code on the pipelined datapath attached at the end of the exam:

```
add  $1, $2, $3
sub  $4, $5, $6
sub  $7, $8, $9
add $10,$11,$12
add $13,$14,$15
```

During the **fifth** cycle of execution, which registers (in the register file) are being read and which register will be written? (5 points)

Writing: $1
Reading: $11 and $12.

Part (b)
If the pipeline was elongated to eight stages:

IF	DE	RR	EX1	EX2	MEM1	MEM2	WB

where decode is split into two stages Decode (DE) and Register Read (RR) and both the Execute and Memory Access stages each take two stages (*i.e.*, all ALU operations take two cycles).

How many inputs would the forwarding muxes in the EX1 stage have? Explain. (5 points)
4 inputs
The control inputs to the forwarding units should not be confused with this question.
The following were the inputs to the MUX
from RR (the regular input which feeds data to the EX stage)
from MEM1 (forwarded)
from MEM2 (forwarded)
from WB (forwarded)
(the latter is for dependent instructions with 2 independent instructions in the middle)
There is no input to the MUX from the middle of the two EX stages since the result is only available when the entire EX stage is complete.

Assuming that forwarding is present, how many cycles would the machine need to stall when a loaded value is used in the cycle after the load? Explain. (5 points)

3 cycles.
Loaded value is available at the beginning of WB stage.

IF	DE	RR	EX1	EX2	MEM1	MEM2	WB	
	IF	DE	RR	SS	SS	SS	EX1	EX2

Question 2, continued

Part (c)

The inner loop for the C code shown can be written as:

```
sum = 0;
for (int i = 0 ; i < max ; ++ i) {
    sum += A[i] * B[i];
}
```

```
LOOP: add  $3,  $3,  4
      lw   $5,  -4($3)
      add  $4,  $4,  4
      lw   $6,  -4($4)
      mul  $7,  $5,  $6
      add  $8,  $8,  $7
      bne  $3,  $9,  LOOP
```

Note that $5 in mul and $3 in bne are not dependencies when forwarding is available.

If the loop executes many times, we are mostly concerned with the performance of the "steady state" execution where the branch is taken *(i.e., make the common case fast).* Assume the pipeline shown on the last page, but ***without*** any forwarding (you can assume that a register can be read in the same cycle it is written) but ***with*** branch prediction (predict not-taken).

This question has three parts: First, mark the dependencies. Second, using the grids below, indicate the pipeline stage each instruction is in for each cycle (stalls can be marked with an **S** or --). Third, compute the steady state CPI of the loop by computing the time between the start of one iteration and the next. State any assumptions you make. (30 points)

Inst	iter	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
Add	N	F	D	E	M	W																	
Lw	N		**F**	**D**	**S**	**S**	**E**	**M**	**W**														
Add	N			**F**	**S**	**S**	**D**	**E**	**M**	**W**													
Lw	n						**F**	**D**	**S**	**S**	**E**	**M**	**W**										
Mul	n							**F**	**S**	**S**	**D**	**S**	**S**	**E**	**M**	**W**							
Add	n										**F**	**S**	**S**	**D**	**S**	**S**	**E**	**M**	**W**				
Bne	N													**F**	**S**	**S**	**D**	**E**	**M**	**W**			
Add	N+1																**S**	**F**	**D**	**E**	**M**	**W**	

Steady state CPI = ___**16/7**_________

(you can leave your answer as a fraction)

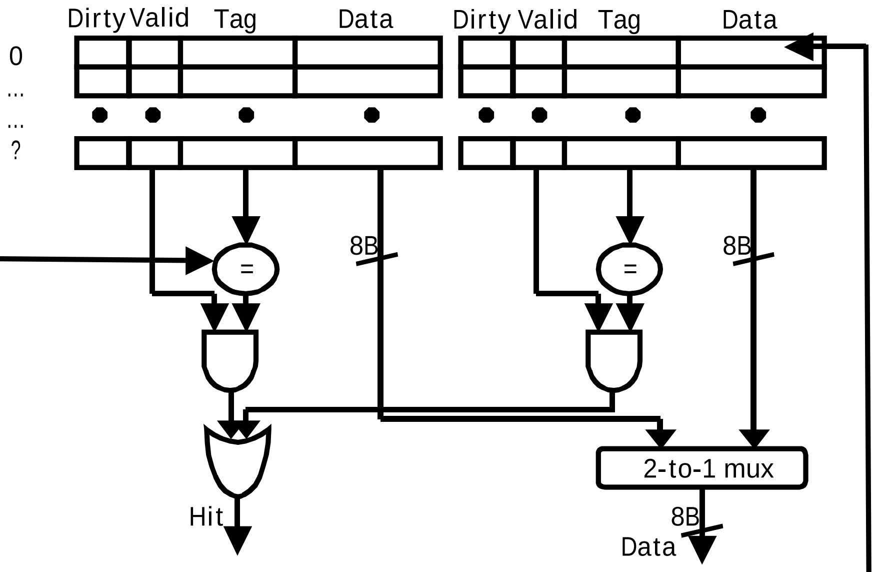
It's a flush

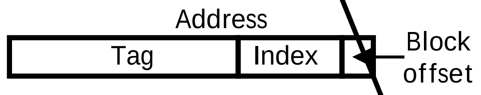
Question 3: Cache Organization and Performance (45 points)

Part (a)

The organization of a cache for a **byte-addressable** memory system is shown below. The cache holds 256 bytes of data in 8 byte blocks (recall that 8B = 8 bytes, where 8b = 8 bits). Implementing the cache requires 320 bytes of storage.

Overhead: 320-256 = 64 bytes or 1 byte for 4 bytes of data. Each block has 8 bytes of data so 2 bytes of overhead. 2 bytes = 16 bits – 2 bits (valid & dirty) = 14 bits tag





Answer the following questions: (write "unknown" if the answer cannot be discerned from the information provided) (20 points)

associativity	**2-way set associative – 2 blocks per line**
number of sets	**16 256 bytes / 8 bytes per block / 2 sets**
write-back or write-through	**write-back – has dirty bit**
write-allocate or write-no-allocate	**unknown**
# of block offset bits	**3 8-byte block, 8 = 2^3**
# of index bits	**4 16 sets, 16 = 2^4**
# of tag bits	**14**
# of address bits total	**21 = 3 + 4 + 14**

Part (b)

Given a processor that would have a CPI of 1.5 with a perfect data cache, what would be its CPI, if loads and stores are 20% of the dynamic instructions, its cache provides a hit rate of 95%, and the miss penalty is 200 cycles. You can leave your answer as an expression. (5 points)

CPI = Base CPI + (%memory accesses * miss rate * miss penalty) =
1.5 + (.2 * .05 * 200) = 1.5 + 2 = 3.5

Question 3, continued

Part (c)
For this question, you are given a 16-byte cache (initially empty) and the sequence of memory accesses (byte loads) shown in the table. For each of the following cache configurations explain whether it can produce this sequence. If so, what is the result of the last access (shown as ???) for each of the matching sequences? (20 points)

addr	hit/miss
0	(miss)
12	(miss)
1	(hit)
17	(miss)
15	(hit)
3	(miss)
7	(???)

1. 4-byte blocks, 2-way set-associative (*e.g.*, 2 sets of 2 blocks each)

0	16
12	

Can't produce this sequence; 3 should be a Hit

1. 4-byte blocks, direct-mapped (4 blocks)

0, 16
7
12

Yes, 7 is a Miss

2. 8-byte blocks, direct-mapped (2 blocks)

0-7, 16-23, 0-7
8-15

Yes, 7 is a Hit

3. 4-byte blocks, fully associative (4 blocks)

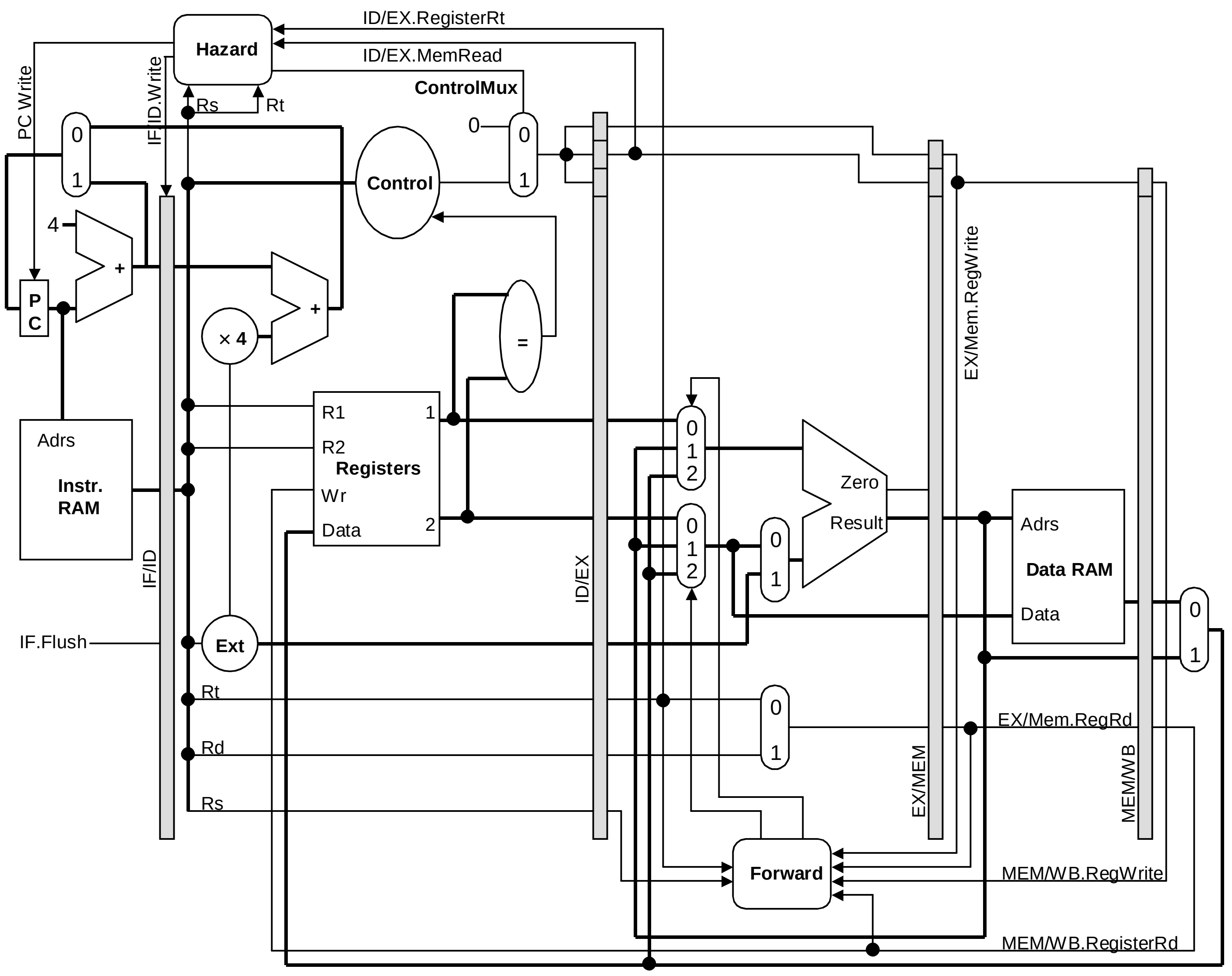
Can't produce this sequence; 3 should be a Hit

0-3	12-15	16-19	

Pipelined Datapath

Here is the final datapath that we discussed in class.

- Forwarding is performed from the EX/MEM and MEM/WB latches to the ALU inputs. The control equation for the Rs register input to the ALU is shown at the bottom of the page. The Rt input is similar.
- Branches are assumed to be not taken. Branch resolution takes place in the Decode stage.
- A hazard unit inserts stalls for lw instructions



```
if ((EX/MEM.RegWrite == 1) and
      (EX/MEM.RegisterRd == ID/EX.RegisterRs))
            then ForwardA = 1
else if ((MEM/WB.RegWrite == 1) and
         (MEM/WB.RegisterRd == ID/EX.RegisterRs))
            then ForwardA = 2
```

CPI = Cycles Per Instruction
AMAT = hit_time + (miss_rate * miss_penalty)
Memory Stall Cycles =
    # loads * miss_rate * miss_penalty