CS232 Midterm Exam 3
December 1, 2004

Name: _____

Section: _____

- 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	20	
2	40	
3	40	
Total	100	

Question 1: Conceptual Questions (20 points)

.

Part (a)

An ideal pipelined machine has a sustained CPI of 1. List causes (discussed in class) for a pipelined machine having a CPI above 1. We're looking for 3 distinct causes, but you may write up to 5 answers. (10 points)

1. Data hazards such as LW $t0, 0($t1) followed by ADD $t3, $t0, $t0

2. Cache misses

3. Miss predicted or flushed branches

4. Pipeline fill time

5. Structural and control hazards

Part (b)

What does LRU stand for? What is it? What property of programs does it exploit? *Be specific, but limit your answers to three sentences.* (10 points)

LRU stands for Least Recently Used. It is a scheduling and replacement policy used throughout computer science that exploits the temporal locality of programs and data. In LRU, data items (eg cache blocks) which haven't been referenced in the longest time are replaced first.

Question 2: Cache Organization and Performance (40 points)

Part (a)

Compute how many bits of storage are required to implement the following cache for a machine with 32-bit addresses (include all data, tag, and control state): a write-back, write-no-allocate, 4-way set associative cache with 1024 32-byte blocks. Five bits are required *for each set* to store full LRU ($4! = 4x3x2x1 = 24 < 2^5$). You can leave your answer as an expression. Show work for partial credit. (10 points)

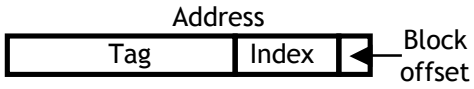
4! (2^5) LRU => 5bits LRU/set
32 (2^5) byte block => 5bit block offset
1024 (2^10) blocks 1 set/4 blocks = 2^8 sets =>8bit index
32bit address – 5bit block -8bit index => 19bit tag

Bits of Storage = LRU/set*(number of sets) + (tag bits)/block*(number of blocks) +
(1 valid + 1 dirty bit)/block * (number of blocks) + bits/block*(number of blocks)

Bits of Storage = 5*2^8 + 19*2^10 + 2*2^10 + 32*8*2^10

Part (b)

Given a direct-mapped cache with 4 blocks of 4 bytes, which of the following byte accesses hit? (10 points)



Address (binary)	Hit/Miss
110001	Miss
100111	Miss
001111	Miss
001100	Hit
010001	Miss
110010	Miss
100101	Hit
001110	Hit
100001	Miss
110101	Miss

Question 2, continued

Part (c)
Compute the average memory access time for a 4-way set-associative, unified L2 cache with 64B blocks and a 95% hit rate. The access time for this 1MB cache is 8 cycles, and it take 80 cycles to move a 64B block from memory to the cache. You can leave your answer as an expression. (10 points)

AMAT = (hit time) + (miss rate * miss penalty)
AMAT = 8 + (.05 * 80)

Part (d)
Consider a cache that holds two blocks of 1-byte each. Write 2 sequences of accesses using the following notation: A_1, A_2, A_3, etc. are distinct addresses that map to the first block, and B_1, B_2, B_3, etc. are distinct addresses that map to the second block of a direct mapped cache. For example: A_2, B_1, B_1, A_9, B_7, A_6.

Sequence 1:
Write a sequence of at most 5 accesses that has a better hit rate on a direct mapped cache than on a fully-associative cache (using LRU). (5 points)

A_1, B_1, B_2, A_1

Sequence 2:
Write a sequence of at most 5 accesses that has a better hit rate on a fully-associative cache (using LRU) than on a direct mapped cache. (5 points)

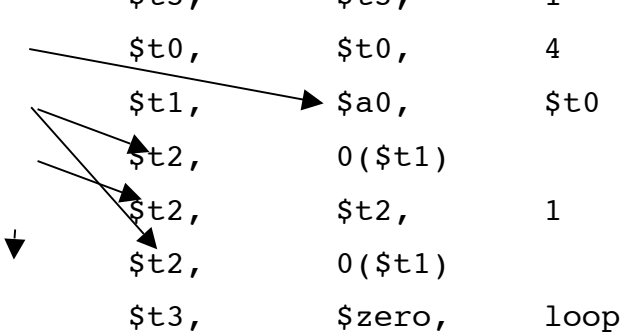
A_1, A_2, A_1, A_2

Question 3: Pipelining (40 points)

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

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

```
loop: sub   $t3,   $t3,   1
      add   $t0,   $t0,   4
      add   $t1,   $a0,   $t0
      lw    $t2,   0($t1)
      add   $t2,   $t2,   1
      sw    $t2,   0($t1)
      bgt   $t3,   $zero, loop
```



Part (a)

Label all dependences within one iteration of this code (not just the ones that will require forwarding). *One iteration is defined as the instructions between the* `sub` *and* `bgt`*,inclusive.* (5 points)

Part (b)

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 **with forwarding** (you can assume that a register can be read in the same cycle it is written) but **without branch prediction** shown on the last page. Using the grids below, indicate the pipeline stage each instruction is in for each cycle (stalls can be marked with an **S** or --). (15 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
sub	N	F	D	E	M	W																	
add	N		F	D	E	M	W																
add	N			F	D	E	M	W															
lw	N				F	D	E	M	W														
add	N					F	D	S	E	M	W												
sw	N						F	S	D	E	M	W											
bgt	N							S	F	D	E	M	W										
sub	N+1									F	S	D	E	M	W								

Question 3, continued

Part (c)
Label each instruction with the forwarding paths it requires (1, 2, 3, 4: as labeled on the final page of this exam)? (5 points)

```
loop:   sub     $t3,    $t3,    1       _____
        add     $t0,    $t0,    4       _____
        add     $t1,    $a0,    $t0     ___3___
        lw      $t2,    0($t1)          ___1___
        add     $t2,    $t2,    1       ___2___
        sw      $t2,    0($t1)          ___3___
        bgt     $t3,    $zero,  loop    _____
```

Part (d)
Re-write the code on the previous page (same as the code above) for optimal performance (10 points).
In addition, try to minimize the number of required forwarding paths and explain which forwarding paths are still required. (5 points)

Here is one solution. The stall due to lw is gone, and forwarding path 2 is no longer needed

```
loop:   add     $t1,    $a0,    $t0     NO FORWARD
        lw      $t2,    4($t1)          PATH #1
        sub     $t3,    $t3,    1       NO FORWARD
        add     $t0,    $t0,    4       NO FORWARD
        add     $t2,    $t2,    1       NO FORWARD
        sw      $t2     4($t1)          PATH #3
        bgt     $t3,    $zero   loop    NO FORWARD
```

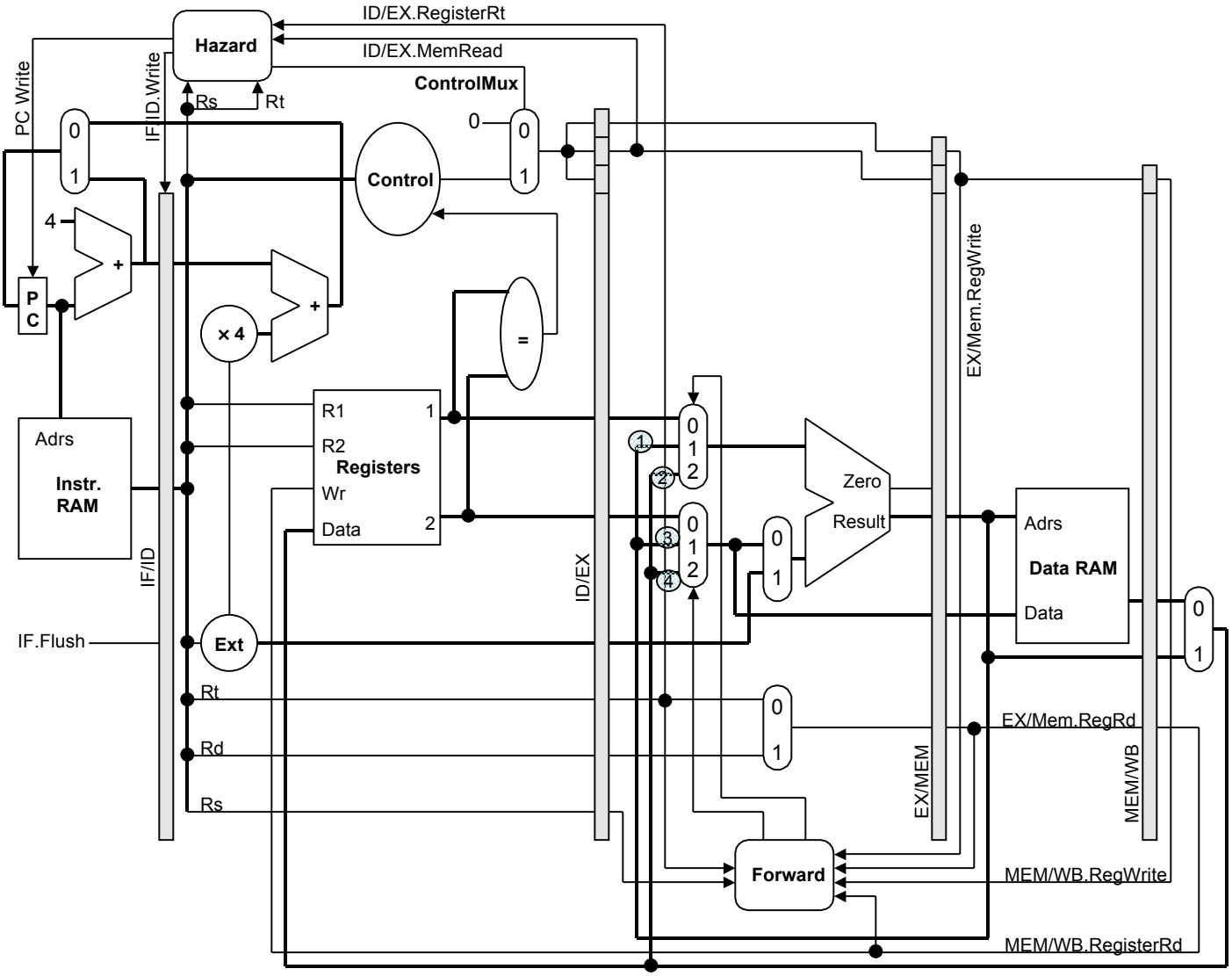
Extra Credit:
Explain (in words) how the code could be re-written so that it executes without stalls or unnecessary instructions on hardware with no bypassing. (+5 points)

You need to unroll the loop 3 times.

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.
- 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