The University of Alabama in Huntsville
ECE Department
CPE 431 01
Test 2 Solution
Fall 2014

1. (3 points) The three Cs of caches are _**capacity**_, _**conflict**_ and _**compulsory**_ .

2. (1 point) A _**tag**_ is a field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word.

3. (1 point) _**Write back**_ is a scheme that handles writes by updating values only the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

4. (10 points) It is possible to have an even greater cache hierarchy than two levels. Consider a processor with the following parameters.

Base CPI, no memory stalls	Processor speed	Main memory access time	First-level cache miss rate	Second-level cache, direct-mapped speed	Local miss rate with second-level cache, direct-mapped	Third-level cache, eight-way set associative speed	Local miss rate with third-level cache, eight-way set associative
2.5	3 GHz	90 ns	6 %	25 cycles	30 %	60 cycles	35 %

Calculate the CPI for the processor given that the Memory accesses/instruction = 1.36 and that hits in the L1 cache incur no miss stalls.

$$\begin{aligned}
\mathbf{CPI} &= \mathbf{CPI_{base} + CPImiss\_penalty} \\
&= \mathbf{CPI_{base} + \text{Memory accesses per instruction}(CPI_{L1\_Hit} + L1_{miss\_rate}(L2_{hit\_time} +} \\
&\qquad \mathbf{L2_{miss\_rate}(L3_{hit\_time} + L3_{miss\_rate} * \text{Main Memory access time}))} \\
&= \mathbf{2.5 + 1.36(0 + 0.06(25\ \text{cycles} + 0.3(60\ \text{cycles} + 0.35*90\ \text{ns}*3*10^{9}\text{cycles/s}))} \\
&= \mathbf{2.5 + 1.36(0.0 + 0.06(25 + 0.3(60 + 94.5))} \\
&= \mathbf{2.5 + 1.36(0.06(25 + 46.35))} \\
&= \mathbf{2.5 + 1.36(4.281)} \\
&= \mathbf{2.5 + 5.822} \\
&= \mathbf{8.322}
\end{aligned}$$

5. (20 points) Here is a series of address references given as byte addresses: 118, 483, 2069, 321, 368, 1077, 1505, 812, 2832, 373, 1411, 511, 1463, 690, 4820, 1714, 1508. Assuming a two-way set associative-mapped cache with two-word blocks and a total size of 32 words that is initially empty and uses LRU, (a) label each reference in the list as a hit or a miss and (b) show the entire history of the cache, including tag and data.

$$32 words \times \frac{1 block}{2 words} \times \frac{1 set}{2 blocks} = 8$$

$sets$

	Index Block offset byte offset	
118	0000 0000 01 110 1 10	miss
483	0000 0001 11 100 0 11	miss
2069	0000 1000 00 010 1 01	miss
321	0000 0001 01 000 0 01	miss
368	0000 0001 01 110 0 00	miss
1077	0000 0100 00 110 1 01	miss
1505	0000 0101 11 100 0 01	miss
812	0000 0011 00 101 1 10	miss
2832	0000 1011 00 010 0 00	miss
373	0000 0001 01 110 1 01	hit
1411	0000 0101 10 000 0 11	miss
511	0000 0001 11 111 1 11	miss
1463	0000 0101 10 110 1 11	miss
690	0000 0010 10 110 0 10	miss
4820	0001 0010 11 010 1 00	miss
1714	0000 0110 10 110 0 10	miss
1508	0000 0101 11 100 1 00	hit

	Tag	Data	Tag	Data
0	5	M[320:327]	22	M[1408:1415]
1				
2	44	M[2832:2839]	32, 75	M[2064:2071], M[4816:4823]
3				
4	23	M[1504:1511]	7	M[480:487
5			12	M[808:815]
6	2, 16, 22, 26	M[112:119], M[1072:1079]. M[1456:1473], M[1712:1719]	5,10	M[368:375], M[688:695]
7	7	M[504:511]		

6. (15 points) a) Schedule the following code on a 2-issue pipeline in which one slot takes lw/sw instructions and the other slot takes R-type and branch instructions.

```
Loop:  lw    $s1, 0($t1)
       add   $s2, $s2, $s1
       sw    $s2, 0($t1)
      lw    $s1, -4($t1)
       add   $s2, $s2, $s1
       sw    $s2, -4($t1)
      lw    $s1, -8($t1)
       add   $s2, $s2 $s1
       sw    $s2, -8($t1)
       addi  $t1, $t1, -12
       bne   $t1, $s0, Loop
```

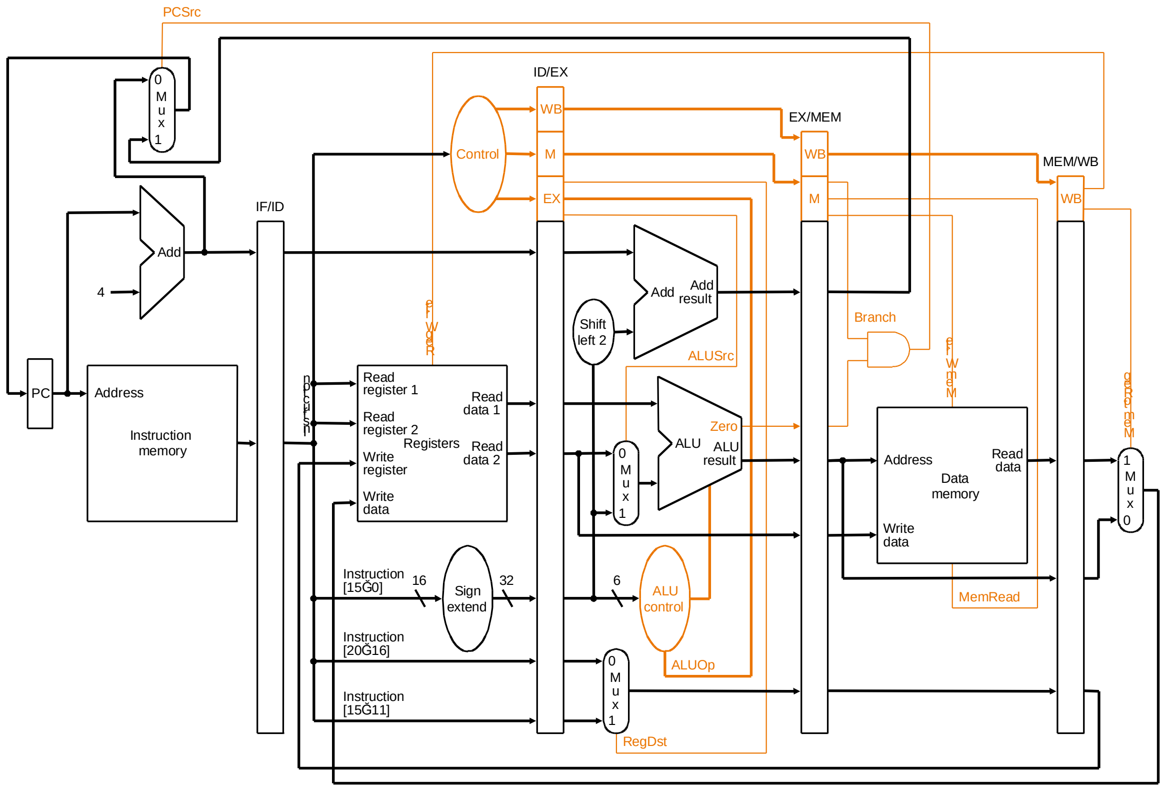
Cycle	Issue Slot R type/Branch	Issue Slot lw/sw
1	addi $t1, $t1, -12	lw $s1, 0($t1)
2		lw $t2, 8($t1)
3	add $s2, $s2, $s1	lw $t3, 4($t1)
4	add $s2, $s2, $t2	sw $s2, 12($t1)
5	add $s2, $s2, $t3	sw $s2, 8($t1)
6	bne $t1, $s0. Loop	sw $s2, 4($t1)

7. (15 points) The following code is written in MATLAB, where elements within the same column are stored contiguously. Consider each variable and answer whether it exhibits spatial locality and whether it exhibits temporal locality. A, B, and C are all arrays of integers 8000 by 8000.

```
for J=1:8000
   for I=1:8
      A(I,J) = B(J,1) + A(J, I) + C(1, I);
```

Variable	I	J	A(I ,J)	B(J, 1)	A(J, I)	C(1, I)
Spatial	Unknown	Unknown	Yes, elements are accessed by column	Yes, elements are accessed by column	No, elements are accessed by row	No, all elements are part of row 1
Temporal	Yes, used 64000 times to access array elements	Yes, used 64000 times to access array elements	No, a different element is accessed each time	Yes, each item is referenced 8 times in a row	No, a different element is accessed each time	Yes, used 8000 times, once each 8 times

8. (20 points) Consider executing the following code on a pipelined datapath like the one shown except that 1) it supports j instructions that complete in the ID stage, and 2) it has MEM/WB forwarding only. The register file does support writing in the first half cycle and reading in the second half cycle.



```
sort:     addi  $sp, $sp, -20
          sw    $ra, 16($sp)
          sw    $s3, 12($sp)
          sw    $s2, 8($sp)
          sw    $s1, 4($sp)
          sw    $s0, 0($sp)
          add   $s2, $a0, $zero
          add   $s3, $a1, $zero
⇒         add   $s0, $zero, $zero
for1tst:  slt   $t0, $s0, $s3
          beq   $t0, $zero, exit1
          addi  $s1, $s0, -1
for2tst:  slt   $t0, $s1, $zero
          bne   $t0, $zero, exit2
          add   $t1, $s1, $s1
          add   $t1, $t1, $t1
          add   $t2, $s2, $t1
          lw    $t3, 0($t2)
          lw    $t4, 4($t2)
          slt   $t0, $t4, $t3
          beq   $t0, $zero, exit2
          add   $a0, $s2, $zero
          add   $a1, $s1, $zero
          jal   swap
          addi  $s1, $s1, -1
          j     for2tst
exit2:    addi  $s0, $s0, 1
          j     for1tst
exit1:    lw    $s0, 0($sp)
          lw    $s1, 4($sp)
          lw    $s2, 8($sp)
          lw    $s3, 12($sp)
          lw    $ra, 16($sp)
          addi  $sp, $sp, 20
          jr    $ra
```

If the `add $s0` instruction one instructions before the `for1tst` label begins executing in cycle 1 and the `beq $t0, $zero, exit1` is taken, what instructions are found in each of the five stages of the pipeline in the 9th cycle? Show the instructions being executed in each stage of the pipeline during each cycle. What value is stored in the ALUResult of the EX/MEM pipeline register in the 9th cycle? Assume that before the instructions are executed, the state of the machine was as follows:

The PC has the value 200_{10}, the address of the add $s0 instruction

Every register has the initial value 20_{10} plus the register number.

Every memory word accessed as data has the initial value 10000_{10} plus the byte address of the word.

Cycle	IF	ID	EX	MEM	WB
1	**add $s0**				
2	**slt $t0**	**add $s0**			
3	**beq $t0**	**slt $t0**	**add $s0**		
4	**beq $t0**	**slt $t0**	**bubble**	**add $s0**	
5	**addi $s1**	**beq $t0**	**slt $t0**	**bubble**	**add $s0**
6	**addi $s1**	**beq $t0**	**bubble**	**slt $t0**	**bubble**
7	**slt $t0**	**addi $s1**	**beq $t0**	**bubble**	**slt $t0**
8	**bne $t0**	**slt $t0**	**addi $s1**	**beq $t0**	**bubble**
9	**lw $s0**	**bubble**	**bubble**	**bubble**	**beq $t0**

The instruction of interest is the one in the MEM stage during the 9th cycle, it is the addi $s1, $s0, -1 instruction that has been turned into a bubble by zeroing out the control lines.

EX/MEM.ALUResult = $s0 -1 = 0 -1 = -1

9. (15 points) Using the code below, unroll the loop four times. Arrange the unrolled code to maximize performance. You may assume that the loop executes a multiple of four times.

```
Loop: lw    $s2, 0($s0)
      sub   $s4, $s2, $s3
      sw    $s4, 0($s0)
      addi  $s0, $s0, 4
      bne   $s0, $s3, Loop
```

```
Loop: lw    $s2, 0($s0)
      sub   $s4, $s2, $s3
      sw    $s4, 0($s0)
      lw    $s2, 4($s0)
      sub   $s4, $s2, $s3
      sw    $s4, 4($s0)
      lw    $s2, 8($s0)
      sub   $s4, $s2, $s3
      sw    $s4, 8($s0)
      lw    $s0, 12($s0)
      sub   $s4, $s2, $s3
      sw    $s4, 12($s0)
      addi  $s0, $s0, 16
      bne   $s0, $s3, Loop
```

```
Loop: lw    $s2, 0($s0)
      addi  $s0, $s0, 16
      sub   $s4, $s2, $s3
      lw    $s2, -12($s0)
      sw    $s4, -16($s0)
      sub   $s4, $s2, $s3
      lw    $s2. -8($s0)
      sw    $s4, -12($s0)
      sub   $s4, $s2, $s3
      lw    $s2, -4($s0)
      sw    $s4, -8($s0)
      sub   $s4, $s2, $s3
      sw    $s4, -4($s0)
      bne   $s0, $s3, Loop
```