The University of Alabama in Huntsville
ECE Department
CPE 431 01
Test 2 Solution
Fall 2016

1. (1 point) A _**structural hazard**_ means that the hardware cannot support the combination of instructions that we want to execute in the same clock cycle.

2. (10 points) The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-type	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the (a) always-taken predictor, (b) always-not-taken predictor and (c) the 2-bit predictor? Assume that branch outcomes are determined in the MEM stage, that there are no data hazards, and that no delay slots are used.

Mispredict penalty – 3 cycles

IF	**ID**	**EX**	**MEM**	**WB**
			beq	
target	**bubble**	**bubble**	**bubble**	**beq**

$CPI_{mispredict}$ = % beq * % mispredict * mispredict penalty

(a) $CPI_{mispredict}$ = 0.25 * (1-0.45) * 3 = 0.4125
(b) $CPI_{mispredict}$ = 0.25 * (1-0.55) * 3 = 0.3375
(c) $CPI_{mispredict}$ = 0.25 * (1-0.85) * 3 = 0.1125

3. (20 points) Here is a series of address references given as byte addresses: 118, 483, 2069, 321, 368, 1505, 812, 2832, 373, 1411, 511, 122, 690, 4820, 1714, 1508, 1080. Assuming a two-way set associative-mapped cache with two-word blocks and a total size of 16 words (32 bits) 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.

$$16\ words \times \frac{1\ block}{2\ words} \times \frac{1\ set}{2\ blocks} = 4\ sets$$

Byte offset = 2 bits, Block offset = 1 bit, Index = 2 bits

Byte Address (Decimal)	Byte Addres (Hexadecimal)		
118	76	0000 0000 011 10 1 10	Miss
483	1E3	0000 0001 111 00 0 11	Miss
2069	815	0000 1000 000 10 1 01	Miss
321	141	0000 0001 010 00 0 01	Miss
368	170	0000 0001 011 10 0 00	Miss
1505	5E1	0000 0101 111 00 0 01	Miss
812	32C	0000 0011 001 01 1 00	Miss
2832	B10	0000 1011 000 10 0 10	Miss
373	175	0000 0001 011 10 1 01	Hit
1411	583	0000 0101 100 00 0 11	Miss
511	1FF	0000 0001 111 11 1 11	Miss
122	7A	0000 0000 011 11 0 10	Miss
690	2B2	0000 0010 101 10 0 10	Miss
4820	12D4	0001 0010 110 10 1 00	Miss
1714	6B2	0000 0110 101 10 0 10	Miss
1508	5E4	0000 0101 111 00 1 00	Hit
1080	438	0000 0100 001 11 0 00	Miss

	Entry A		Entry B	
Set	Tag	Data	Tag	Data
0	~~15~~, 47	~~M[480..487],~~ M[1504..1511]	~~10~~, 44	~~M[320..327],~~ M[1408..1415]
1			25	M[808..815]
2	~~64~~, ~~88~~, ~~21~~, 53	~~M[2064..2071],~~ ~~M[2832..2839],~~ ~~M[688..695],~~ M[1712..1719]	~~3~~, ~~11~~, ~~150~~, 64	~~M[112..119],~~ ~~M[368..375],~~ M[4816..4823]
3	~~15~~, 33	~~M[504..511],~~ M[1080..1087]	3	M[120..127]

4. (1 point) A _page fault_ occurs when an accessed page is not present in main memory.

5. (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. You have forwarding from the EX stage only and the branch completes in the ID stage.

```
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)
      addi  $s0, $s0, 12
      bne   $s0, $t3, Loop
```

Cycle	Issue Slot R type/Branch	Issue Slot lw/sw
1		lw $s2, 0($s0)
2		lw $t0, 4($s0)
3	addi $s0, $s0, 12	lw $t1, 8($s0)
4	sub $s4, $s2, $s3	
5	sub $s4, $t0, $s3	sw $s4, 0*($s0)
6	sub $s4, $t1, $s3	sw $s4, -8($s0)
7	bne $s0, $t3,Loop	sw $s4, -4($s0)
8		* no forwarding from MEM, s0 value is old one

6. (1 point) The _tag_ contains the address information required to identify whether a word in the cache corresponds to the requested word.

7. (15 points) In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 32-bit integer.

```
for (I = 0; I < 8; I++)
  for (J = 0; J < 8000; J++)
    for (K = 0; K < 2000; K++)
      A[I][J][K] = B[J][0][I] + A[I][J][K];
```

(a) References to which variables exhibit temporal locality?
(b) References to which variables exhibit spatial locality?

Variable	I	J	K	A[I][J][K]	B[J][0][I]
Spatial	Unknown	Unknown	Unknown	Yes, Access pattern is A[0][0][0] A[0][0][1] A[0][0][2] … A[0][0][1999] A[0][1][0], distance between items in memory is one element	No, Access pattern is B[0][0][0] B[1][0][0] B[2][0][0] … B[0][0][1] B[1][0][1] distance between items in memory is at least 16 E6 elements

Temporal	**Yes, used 128 E6 times to access array elements plus loop control plus update**	**Yes, used 128 E6 times to access array elements plus loop control plus update**	**Yes, used 128 E6 times to access array elements plus loop control plus update**	**Yes, each item is referenced twice, once to read, once to write**	**Yes, the same element is accessed 2000 times in a row**

8. (1 point) _**Spatial locality**_ states that if an item is referenced, items whose addresses are close by will tend to be referenced soon.

9. (1 point) In the case where memory and cache have different values for the same memory location, the cache and memory are said to be _**inconsistent**_.

10. (15 points) Virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. The following table is a stream of virtual addresses as seen on a system. Assume 4 KiB pages, byte addressing, a four-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number. Given the address stream, and the shown initial state of the TLB and page table, show the final state of the system. Also list for each reference if it is a hit in the page table, or a page fault.

12948, 49419, 46814. 13975, 40004, 12707, 52236

TLB

Valid	Tag	Physical Page Number
1	11	12
1	7	4
1	3	6
0	4	9

Page table

VPN	Valid	Physical page or in disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
10	1	3
11	1	12
12	0	Disk

These are byte addresses. The page offset is $\log_2$ 4K = 12. The virtual page number can be obtained by dividing the byte address by 4 KiB (4096) and taking the integer part of the result.

	VPN	TLB?	PT?	PF?
12948	3	H		
49419	12	M	M	Y
46814	11	H		
13975	3	H		
40004	9	M	M	Y
12707	3	H		
52236	12	H		

TLB

Valid	Tag	Physical Page Number
1	11	12
1	~~7~~, 9	4, 14
1	3	6
~~0~~, 1	4, 12	9

Page table

	Valid	Physical page or in disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	~~0~~, 1	~~Disk~~, 14
10	1	3
11	1	12
12	~~0~~, 1	~~Disk~~, 13

11. (20 points) Using the code below, unroll the loop so that two iterations are executed. Arrange the unrolled code to maximize performance. You may assume that the loop executes a multiple of two times. Calculate the number of cycles for the original and for the unrolled, rearranged code. Assume full forwarding and one cycle delay for taken branches.

```
        addi  $t2, $zero, $zero
Loop:   sll   $t3, $t2, 2
        add   $t0, $t3, $s6
        lw    $t0, 0($t0)
        stall
        lw    $t0, 0($t0)
        stall
        mul   $t1, $t0, $s2
        add   $t0, $t3, $s7
        lw    $t0, 0($t0)
        stall
        lw    $t0, 0($t0)
        stall
        mul   $t0, $t0, $s3
        add   $t1, $t0, $t1
        add   $t0, $t3, $s8
        sw    $t1, 0($t0)
        addi  $t2, t2, 1
        slt   $t4, $t2, 48
        bne   $t4, $zero, Loop
        stall
```

$Cycles_{original}$ = addi + Number of iterations*(Number of Instructions + Data Stalls + Control Stall) – Control Stall last iteration

$Cycles_{original}$ = 1 + 48(15 + 4 + 1) – 1 = 48*20 = 960

```
        addi  $t2, $zero, $zero
Loop:   sll   $t3, $t2, 2
        add   $t0, $t3, $s6
        add   $t5, $t3, $s7
        add   $t8, $t3, $s8
        lw    $t6, 0($t0)
        lw    $t7, 0($t5)
        lw    $t6, 0($t6)
        lw    $t7, 0($t7)
        mul   $t6, $t6, $s2
        mul   $t7, $t7, $s3
        add   $t1, $t6, $t7
        sw    $t1, 0($t8)
        lw    $t6, 4($t0)
        lw    $t7, 4($t5)
        lw    $t6, 0($t6)
        lw    $t7, 0($t7)
        mul   $t6, $t6, $s2
        mul   $t7, $t7, $s3
        add   $t1, $t6, $t7
        sw    $t1, 4($t8)
        addi  $t2, t2, 2
        slt   $t4, $t2, 48
        bne   $t4, $zero, Loop
        stall
```

$Cycles_{unrolled,rearranged}$ = addi + Number of iterations(Number of instructions + control stall) – control stall last iteration

$Cycles_{unrolled,rearranged}$ =1 + 24(23 + 1) – 1 = 24*24 = 576