The University of Alabama in Huntsville
ECE Department
CPE 531 01
Test 2 Solution
Fall 2015

1. (1 point) A _**sector**_ is the smallest amount of information that is read or written on a disk.
2. (1 point) _**M iss penalty**_ is the time required to fetch a block into a level of the memory hierarchy from the lower level.
3. (1 point) A _**fully associative**_ cache is a cache structure in which a block can be placed in any location in the cache.-
4. (1 point) A _**write buffer**_ is a queue that holds data while the data is waiting to be written to memory.
5. (1 point) A _**direct mapped**_ cache is a cache structure in which each memory location is mapped to exactly one location in the cache.

6. (10 points Consider the code snippet shown below. Suppose that it is executed on a system with a 2-way set associative 16 KB data cache with 8 word blocks, 32-bit words, and an LRU replacement policy. Assume that int is word sized. Also assume that the word address of a is 0x0, that `i` and `x` are kept in registers, and that the cache is initially empty. How many data cache misses are there? How many hits are there? Assume that there is no byte offset.

```
int i;
int a[1024*1024];
int x = 0;
for (i = 0; i < 1024; i++)
  x += a[i] + a[1024*i];
```

$$16\ KB \times \frac{1\ word}{4\ bytes} \times \frac{1\ block}{8\ words} \times \frac{1\ set}{2\ blocks} = 256\ sets$$

Address	Set	Hit/Miss
0	0	Miss
0	0	Hit
1	0	Hit
1024	128	Miss
2	0	Hit
2048	0	Miss
3	0	Hit
3072	128	Miss
4	0	Hit

4096	0	Miss
5	0	Hit
5120	128	Miss
6		Hit
6144	0	Miss
7		Hit
7168	128	Miss
8	1	Miss
8192	0	Miss
9	1	Hit
9216	128	Miss

`a[i]` hits 7 out of 8 accesses, `a[1024*i]` hits only once. Total hits 7*1024/8 + 1 = 897, Total misses = 2048 – 897 = 1151

8. (15 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, 1080. Assuming a two-way set associative-mapped cache with four-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.

$$32\ words \times \frac{1\ block}{4\ words} \times \frac{1\ set}{2\ blocks} = 4\ sets$$

Byte Address (Decimal)	Byte Addres (Hexadecimal)	Binary	
118	76	0000 0000 01 11 01 10	Miss
483	1E3	0000 0001 11 10 00 11	Miss
2069	815	0000 1000 00 01 01 01	Miss
321	141	0000 0001 01 00 00 01	Miss
368	170	0000 0001 01 11 00 00	Miss
1505	5E1	0000 0101 11 10 00 01	Miss
812	32C	0000 0011 00 10 11 00	Miss
2832	B10	0000 1011 00 01 00 00	Miss
373	175	0000 0001 01 11 01 01	Hit
1411	583	0000 0101 10 00 00 11	Miss
511	1FF	0000 0001 11 11 11 11	Miss
1463	5B7	0000 0101 10 11 01 11	Miss
690	2B2	0000 0010 10 11 00 10	Miss
4820	12D4	0001 0010 11 01 01 00	Miss
1714	6B2	0000 0110 10 11 00 10	Miss
1508	5E4	0000 0101 11 10 01 00	Hit
1080	438	0000 0100 00 11 10 00	Miss

	Entry A		Entry B	
Set	Tag	Data	Tag	Data
0	22	M[1408..1423]	5	M[320..335]
1	44	M[2832..2847]	~~32~~, 75	~~M[2064..2079],~~ M[4816..4831]
2	~~7~~, 12	~~M[480..495],~~ M[800..815]	23	M[1504..1519]
3	~~5~~, ~~22~~, 26	~~M[368..383]~~, ~~M[1456..1471]~~, M[1712..1727]	~~1~~, ~~7~~, ~~10~~, 16	~~M[112..127]~~, ~~M[496..511]~~, ~~M[688..703]~~, M[1072.. 1087]

8. (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 MEM 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	`addi $s0, $s0, 12`	`lw $s2, 0($s0)`
2		`lw $t4, 4($s0)`
3	`sub $s4, $s2, $s3`	`lw $t5, -4($s0)`
4	`sub $t4, $t4, $s3`	
5	`sub $t5, $t5, $s3`	`sw $s2, -12($s0)`
6		`sw $t4, -8($s0)`
7	`bne $s0, $t3, Loop`	`sw $t5, -4($s0)`

9. (15 points) Multilevel caching is an important technique to overcome the limited amount of space that a first level cache can provide while still maintaining its speed. Consider a processor with the following parameters.

Base CPI, no memory stalls	Processor speed	Main memory access time	First-level cache miss rate per instruction	Second-level cache, direct-mapped speed	Global miss rate with second-level cache, direct-mapped	Second-level cache, eight-way s et associative speed	Global miss rate with second-level cache, eight-way set associative
2.0	**2.5 GHz**	**125 ns**	**4.8 %**	**15 cycles**	**2.2 %**	**25 cycles**	**1.4 %**

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 for a) only a first-level cache, b) a second-level dixect-mapped cache, and c) a second-level eight-way set-associative cache

a) $CPI = CPI_{base}$ + Memory accesses/per instruction (First level miss rate per instruction x Main memory access (in cycles))
CPI = 2.0 + 1.36(0.048 x 125 ns x 2.5 E9 cycles/s) = 2.0 + 20.4 = 22.4

b) $CPI = CPI_{base}$ + Memory accesses/per instruction(First level miss rate per instruction x Second level cache access time + Second level miss rate per instruction x Main memory access (in cycles))
CPI = 2.0 + 1.36(0.048 x 15 + 0.022 x 125 ns x 2.5 E9 cycles/s) = 2.0 + 1.36(0.72 + 6.875) = 2.0 + 1.36 x 7.595= 2.0 + 10.3292 = 12.3292

c) $CPI = CPI_{base}$ + Memory accesses/per instruction(First level miss rate per instruction x Second level cache access time + Second level miss rate per instruction x Main memory access (in cycles))
CPI = 2.0 + 1.36(0.048 x 25 + 0.014 x 125 ns x 2.5 E9 cycles/s) = 2.0 + 1.36(1.2 + 4.375) = 2.0 + 1.36 x 5.575= 2.0 + 7.582 = 9.582

10 (15 points) Virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this table must be updated as addresses are accessed. The following table is a stream of virtual addresses as seen on a system. Assume 8 KB pages, 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

	VPN	**TLB?**	**PT?**	**PF?**
12948	**1**	**M**	**M**	**Y**
49419	**6**	**M**	**M**	**Y**
46814	**5**	**M**	**H**	**N**
13975	**1**	**H**	**H**	**N**
40004	**4**	**M**	**H**	**N**
12707	**1**	**H**	**H**	**N**
52236	**6**	**H**	**H**	**N**

11. (15 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. If the original loop executes 50 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.

```
Loop:  lw    $s0, 0($t1)
       stall
       mul   $s0, $s0, $s2
       lw    $s4, 0($t2)
       stall
       add   $s0, $s0, $s4
       sw    $s0, 0($t2)
       addi  $t1, $t1, 4
       addi  $t2, $t2, 4
       bne   $t1, $zero, Loop
```

Cycles$_{original}$ 50 x (8 instructions + 2 data stalls + 1 control stall) -1 = 549

```
Loop   lw    $s0, 0($t1)
       lw    $s4, 4($t2)
       mul   $s0, $s0, $s2
       add   $s0, $s0, $s4
       sw    $s0, 0($t2)
       lw    $s0, 4($t1)
       mul   $s0, $s0, $s2
       lw    $s4, 4($t2)
       add   $s0, $s0, $s4
       addi  $t1, $t1, 8
       sw    $s0, 4($t2)
       addi  $t2, $t2, 8
       bne   $t1, $zero, Loop
```

Cycles$_{unrolled}$ 25 x (13 instructions + 1 control stall) -1 = 349

12. (10 points) Use a single error correcting/double error detecting code to encode the data word 100_1000_1001

```
PPDP DDDD DDDD DDD
1234 5678 9111 111
           012 345
__1_ 001_ 0001 001
```

$$P1 = D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 \oplus D13 \oplus D15$$
$$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$$
$$P2 = D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 \oplus D13 \oplus D15$$
$$= 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$$
$$P4 = D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 \oplus D13 \oplus D15$$
$$= 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$$
$$P8 = D3 \oplus D5 \oplus D7 \oplus D9 \oplus D11 \oplus D13 \oplus D15$$
$$= 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0$$

$$P = P1 \oplus P2 \oplus D3 \oplus P4 \oplus D5 \oplus D6 \oplus D7 \oplus P8 \oplus D9 \oplus D10 \oplus D11 \oplus D12 \oplus D13 \oplus D14 \oplus D15 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$$