The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Test 2 Solution
Fall 2012

1. (1 point) **_Loop unrolling_** is a technique in which multiple copies of a loop body are made and instructions from different iterations are scheduled together.
2. (1 point) **_Superscalar_** is an advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle.
3. (1 point) The **_miss penalty_** is the time required to fetch a block into a level of the memory hierarchy from the lower level.
4. (1 point) Getting a missing data item early from the internal resources of the pipeline is called **_forwardin**g_.
5. (1 point) **_T_**(True or False) The number of sets in a fully associative cache is one.

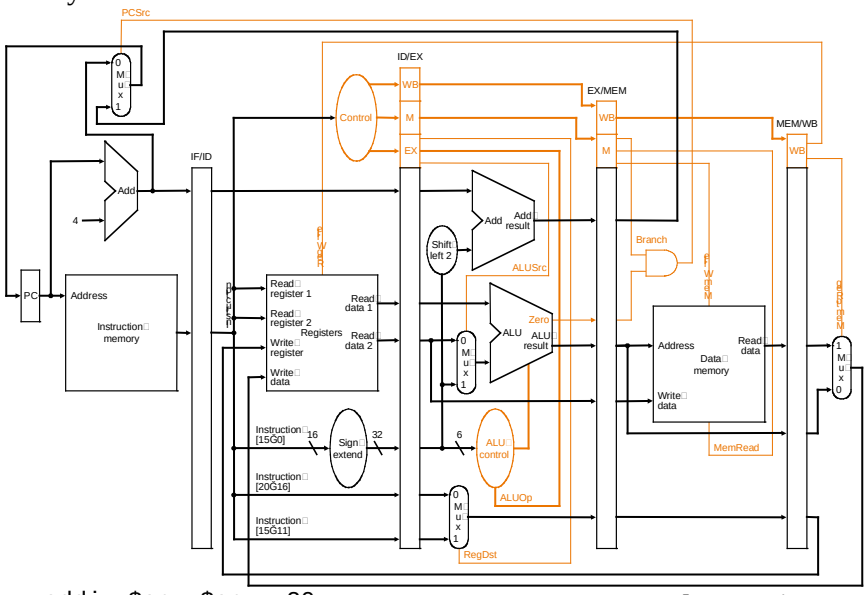
6. (10 points) It is possible to have an even greater cache hierarchy than two levels. Consider a processor with the following parameters.

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
2.0	3 GHz	75 ns	5 %	15 cycles	30 %	50 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.

CPI = CPI_{base} + CPImiss_penalty
= CPI_{base} + Memory accesses per instruction(CPI_{L1_Hit} + $L1_{miss\_rate}$($L2_{hit\_time}$ + $L2_{miss\_rate}$($L3_{hit\_time}$ + $L3_{miss\_rate}$*Main Memory access time))
= 2.0 + 1.36(0 + 0.05(15 cycles + 0.3(50 cycles + 0.35*75 ns*3*10^9cycles/s))
= 2.0 + 1.36(0.0 + 0.05(15 + 0.3(50 + 78.75))
= 2.0 + 1.36(0.05(15 + 38.625))
= 2.0 + 1.36(2.68125)
= 2.0 + 3.6465
= 5.65

7. (10 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 `lw $t4` instruction six instructions after the `for2tst` label begins executing in cycle 1 and the `beq $t0, $zero, exit2` is taken, what instructions are found in each of the five stages of the pipeline in the 12th cycle? Show the instructions being executed in each stage of the pipeline during each cycle.

Cycle	IF	ID	EX	MEM	WB
1	`lw $s4`				
2	`slt $t0`	`lw $s4`			
3	`beq $t0`	`slt $t0`	`lw $s4`		
4	`beq $t0`	`slt $t0`	`bubble`	`lw $s4`	
5	`add $a0`	`beq $t0`	`slt $t0`	`bubble`	`lw $s4`
6	`add $a0`	`beq $t0`	`bubble`	`slt $t0`	`bubble`
7	`add $a1`	`add $a0`	`beq $t0`	`bubble`	`slt $t0`
8	`jal swap`	`add $a0`	`add $a0`	`beq $t0`	`bubble`
9	`addi $s0`	`bubble`	`bubble`	`bubble`	`beq $t0`
10	`j for1tst`	`addi $s0`	`bubble`	`bubble`	`bubble`
11	`lw $s0`	`j for1tst`	`addi $s0`	`bubble`	`bubble`
12	`slt $t0`	`bubble`	`j for1tst`	`addi $s0`	`bubble`

8. (10 points) Consider the case of a seven-deep pipeline where the branch is resolved at the end of the fourth stage for unconditional branches and at the end of the sixth cycle for conditional branches. The program run on this pipeline has the following branch frequencies (as percentages of all instructions) are as follows:

Conditional branches	30%
Jumps and calls	15%
Conditional branches	70% are taken

Assuming that the CPI of the program, neglecting branch hazards, is 1.0, how much slower is the real number, when branch hazards are considered?

First, determine the penalties involved for the conditional and unconditional branches

Conditional Branches

after 5	**after 4**	**after 3**	**after 2**	**after 1**	**branch**	
target	**after 5**	**after 4**	**after 3**	**after 2**	**after 1**	**branch**

Unconditional Branches

after 3	**after 2**	**after**	**branch**			
target	**after 3**	**after 2**	**after 1**	**branch**		

So, the penalty for conditional branches is five clock cycles and the penalty for unconditional branches is three clock cycles.

$$CPI_{branch\ hazards} = CPI_{perfect} + \text{Conditional branches taken} * \text{penalty} + \text{Unconditional branches taken} * \text{penalty}$$
$$= 1.0 + 0.3*0.7*5 + 0.15*3 = 1 + 1.05 + 0.45 = 2.5$$

Slowdown = $CPI_{branch\ hazards}/CPI_{perfect}$ = 2.5/1
CPIbranch hazards is a slowdown of 2.5

9. (25 points) a) (5 points) Consider the following loop executing on a MIPS pipeline with full forwarding. Calculate the number of cycles it takes to execute this loop, neglecting pipeline fill cycles. b) (15 points) Unroll the loop so that 3 iterations of the loop are executed at once and schedule the unrolled code on a 2-issue pipeline in which any instruction can be issued in any slot. c) (5 points) Calculate the speedup from the original loop to the unrolled loop scheduled on a 2-issue pipeline.

```
      addi  $t1, $s0, 360
Loop: lw    $s1, 0($t1)
      add   $s2, $s2, $s1
      sw    $s2, 0($t1)
      addi  $t1, $t1, -4
      bne   $t1, $s0, Loop
```

89 iterations * (5 instructions + 1 cycle data hazard stall (lw $s1 – add $s2) + 1 cycle control hazard stall) + 1 iteration (5 instructions + 1 cycle data hazard stall) = 89 * 7 +6= 629 cycles

```
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, $s0, Loop
```

Cycle	Issue Slot 1	Issue Slot 2
1	lw $s1, 0($t1)	lw $t8, -4($t1)
2	lw $t9, -8($t1)	addi $t1, $t1, -12
3	add $s2, $s2, $s1	
4	sw $s2, 12($t1)	add $s2, $s2, $t8
5	sw $s2, 8($t1)	add $s2, Ss2, $t9
6	bne $t1, $s0, Loop	sw $s2, 4($t1)
7		

c. 29 iterations * (6 instructions + 1 cycle control hazard stall) + 1 iteration (6 instructions) = 29 * 7 + 6 = 209

Speedup = 629 cycles/209 cycles = 3.0

10. (10 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 I=1:8000
   for J=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	No, elements are accessed by row	Yes, elements are accessed by column	Yes, elements are accessed by column	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, used 8000 times, once each 8 times	No, a different element is accessed each time	Yes, each item is referenced 8 times in a row

11. (15 points) Here is a series of address references given as byte addresses: 0, 4, 16, 131, 232, 160, 1024, 30, 140, 3100, 179, 2180. Assuming a direct-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.

$$32words \times \frac{1block}{4words} \times \frac{1set}{1blocks} = 8sets$$

	Index	Block Offset	Byte Offset	
0	000	0000	00	m
4	000	0000	01	h
16	000	0001	00	m
131	000	0011	00	m
232	000	1110	00	m
160	000	1010	00	m
1024	000	0000	00	m
30	000	0001	10	h
140	000	1000	11	m
3100	000	1100	11	m
179	000	1011	11	m
2180	000	1000	01	m

0	MEM[0:15], MEM[128:143], MEM[1024:1039], MEM[128:143], MEM[2176:2191]
1	MEM[16:31], MEM[3088:3103]
2	MEM[160:175]
3	MEM[176:191]
4	
5	
6	MEM[224:239]
7	

12. (15 points) As described in Section 5.4, 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.

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

These are byte addresses. The page offset is $\log_2$ 8K = 13. The virtual page number can be obtained by dividing the byte address by 8KB (8192) and taking the integer part of the result.

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

TLB

Valid	Tag	Physical Page Number
1	~~11~~, 6	~~12~~, 14
1	~~7~~, 5	~~4~~, 11
1	~~3~~, 4	~~6~~, 9
~~0~~, 1	~~4~~, 1	~~9~~, 13

Page table

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