The University of Alabama in Huntsville
ECE Department
CPE 431 01
Test 2 Solution
Fall 2017

1. (1 point) The only dependence/hazard where full forwarding cannot save the day is between a _**lw**_ and a _**use of that lw data**_.

2. (1 point) The minimum unit of information that may be either present or not present in physical memory is a _**page**_.

3. (1 point) _**Superscalar**_ is an advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle.

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	3.5 GHz	90 ns	5 %	20 cycles	40 %	50 cycles	30 %

Calculate the CPI for the processor given that the Memory accesses/instruction = 1.36 and that hits in the L1 cache incur a 1 cycle stall.

$CPI_{total} = CPI_{base} + 1.36(L1_{hit\_time} + L1_{miss}(L2_{hit\_time} + L2_{miss}(L3_{hit\_time} + L3_{miss} * Main\_memory_{hit\_time}/cycle\ time)))$

$CPI_{total} = 2.0 + 1.36(1 + 0.05(20 + 0.4(50 + 0.3*90\ ns*3.5\ E9\ cycles/s)))$

$CPI_{total} = 2.0 + 1.36(1 + 0.05(20 + 0.4(50 + 94.5)))$

$CPI_{total} = 2.0 + 1.36(1 + 0.05(20 + 57.8))$

$CPI_{total} = 2.0 + 1.36(1 + 3.89) = 2.0 + 6.65 = 8.65$

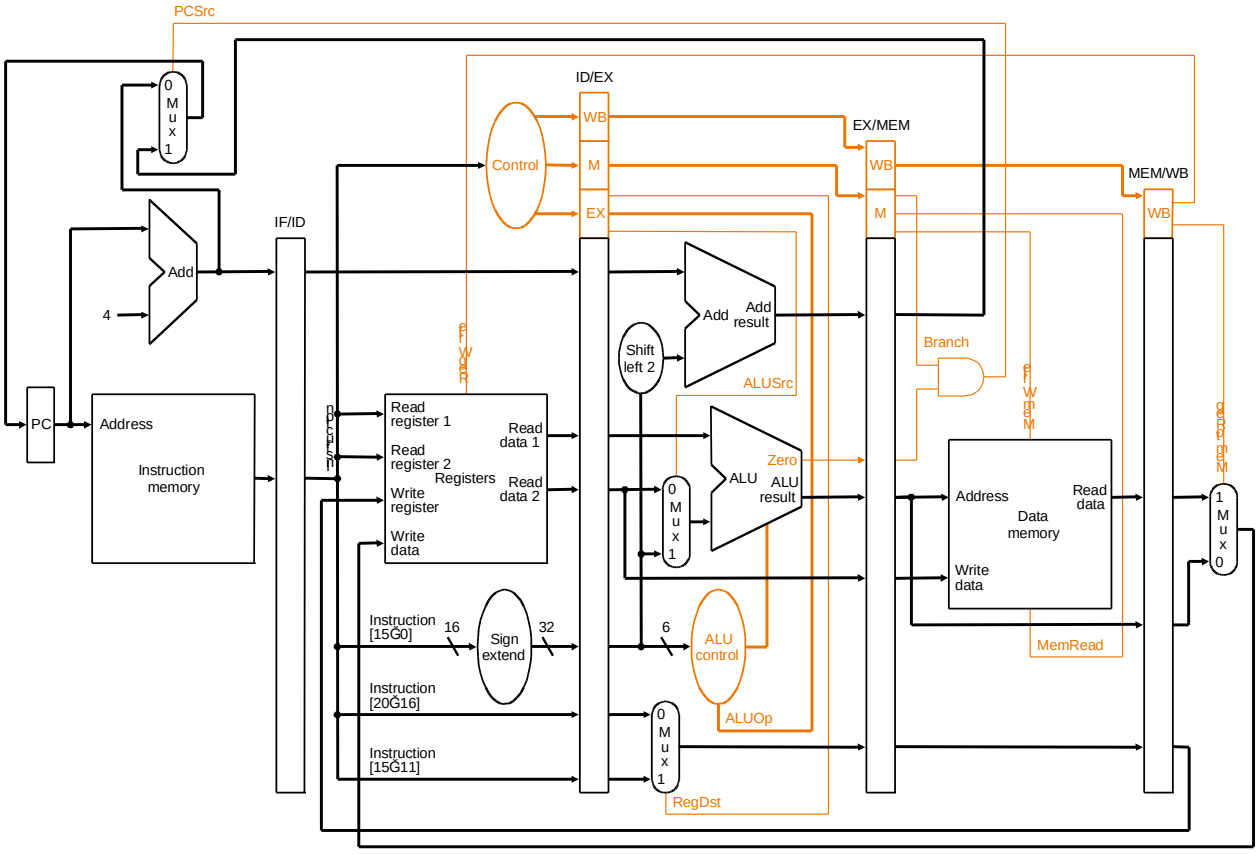
5. (15 points) Here is a series of address references given as byte addresses: 118, 483, 2069, 321, 368, 1505, 812, 2832, 373, 1411, 511, 122, 690, 4820, 1714, 1508, 1080. Assuming a two-way set associative-mapped cache with two-word blocks and a total size of 16 64-bit words that is initially empty 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.

16 words x 1 block/2 words x 1 set/2 blocks = 4 sets
Byte offset = 3 bits because 64-bit words have 8 bytes
Block offset = 1 bit because each block has 2 words
Index = 2 bits for 4 sets

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

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

6. (15 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 EX 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
200       add   $t2, $s2, $t1
204       lw    $t3, 0($t2)
208       lw    $t4, 4($t2)
212       slt   $t0, $t4, $t3
216       beq   $t0, $zero, exit2
220       add   $a0, $s2, $zero
224       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 $t2` instruction four 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 11th cycle? Show the instructions being executed in each stage of the pipeline during each cycle. What value is stored in the ALUResult of the IF/ID pipeline register in the 11th 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 $t2` 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 $t2**				
2	**lw $t3**	**add $t2**			
3	**lw $t4**	**lw $t3**	**add $t2**		
4	**slt $t0**	**lw $t4**	**lw $t3**	**add $t2**	
5	**slt $t0**	**lw $t4**	**bubble**	**lw $t3**	**add $t2**
6	**beq $t0**	**slt $t0**	**lw $t4**	**bubble**	**lw $t3**
7	**beq $t0**	**slt $t0**	**bubble**	**lw $t4**	**bubble**
8	**beq $t0**	**slt $t0**	**bubble**	**bubble**	**lw $t4**
9	**add $a0**	**beq $t0**	**slt $t0**	**bubble**	**bubble**
10	**add $a1**	**add $a0**	**beq $t0**	**slt $t0**	**bubble**
11	**jal**	**add $a1**	**add $a0**	**beq $t0**	**slt $t0**

IF/ID.PCInc = _**228**_ **The instruction of interest is the one in the ID stage, add $a1, It's value of PCInc is 224 + 4 = 228**

7. (1 point) _**False**_ (True/False) Knowing the number of words in a block is enough to know how many sets there are in a cache.

8. (1 point) _**False**_ (True/False) A 4-way set associative cache always has 4 sets.

9. (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
50%	20%	5%	20%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
65%	70%	90%

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 BEQ outcomes are determined in the EX stage, that JMP outcomes are determined in the ID stage, that there are no data hazards, and that no delay slots are used.

Mispredict penalty – 2 cycles

IF	ID	EX	MEM	WB
		beq		
target	bubble	bubble	beq	

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

(a) $CPI_{mispredict}$ = 0.2 * (1-0.65) * 2 = 0.14

(b) $CPI_{mispredict}$ = 0.2 * (1-0.7) * 2 = 0.12

(c) $CPI_{mispredict}$ = 0.2 * (1-0.9) * 2 = 0.04

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 16 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.

TLB

Valid	Tag	Physical Page Number
1	~~11~~, **8**	~~12~~, **14**
1	7	4
1	~~3~~, **2**	~~6~~, **16**
~~0~~, 1	4, ~~6~~, **9**	~~9~~, ~~13~~, **15**

Page table

VPN	Valid	Physical page or in disk
0	1	5
1	0	Disk
2	~~0~~, **1**	~~Disk~~, **16**
3	1	6
4	1	9
5	1	11
6	~~0~~, **1**	~~Disk~~, **13**
7	1	4
8	~~0~~, **1**	~~Disk~~, **14**
9	~~0~~, **1**	~~Disk~~, **15**
10	1	3
11	1	12
12	0	Disk

Address	Page Number	TLB Hit/Miss	Page Table Hit/Miss	Page Fault Y/N
184,760	**11**	**Hit**		
110,824	**6**	**Miss**	**Miss**	**Y**
56,312	**3**	**Hit**		
119,200	**7**	**Hit**		
136,100	**8**	**Miss**	**Miss**	**Y**
151,692	**9**	**Miss**	**Miss**	**Y**
126,856	**7**	**Hit**		
40,000	**2**	**Miss**	**Miss**	**Y**

11. (15 points) Using the code below, unroll the loop so that three iterations are executed. Arrange the unrolled code to maximize performance. You may assume that the loop executes a multiple of three 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      # i = 0
Loop:   sll   $t3, $t2, 2            # $t3 = i*4
        add   $t0, $t3, $s6          # $t0 = &A[i]
        add   $t1, $t3, $s8          # $t1 = &C[i]
        lw    $t0, 0($t0)            # $t0 = A[i]
        add   $t0, $t0, $s3          # $t0 = A[i]+j
        lw    $t0, 0($t0)            # $t0 = A[A[i]+j]
        sw    $t0, 0($t1)            # C[i] = A[A[i]+j]
        addi  $t2, t2, 1             # i++
        slt   $t4, $t2, 90           # $t4 = 1 if i < 90
        bne   $t4, $zero, Loop       # go back if i < 90
```

Original with stalls:

```
        addi  $t2, $zero, $zero      # i = 0
Loop:   sll   $t3, $t2, 2            # $t3 = i*4
        add   $t0, $t3, $s6          # $t0 = &A[i]
        add   $t1, $t3, $s8          # $t1 = &C[i]
        lw    $t0, 0($t0)            # $t0 = A[i]
        stall
        add   $t0, $t0, $s3          # $t0 = A[i]+j
        lw    $t0, 0($t0)            # $t0 = A[A[i]+j]
        stall
        sw    $t0, 0($t1)            # C[i] = A[A[i]+j]
        addi  $t2, t2, 1             # i++
        slt   $t4, $t2, 90           # $t4 = 1 if i < 90
        bne   $t4, $zero, Loop       # go back if i < 90
        stall
```

Cycles = 1 + 89(10 + 2 data stalls + 1 control stall) + 10 + 2 data stalls = 1170

Unrolled:

```
        addi  $t2, $zero, $zero      # i = 0
Loop:   sll   $t3, $t2, 2            # $t3 = i*4
        add   $t5, $t3, $s6          # $t5 = &A[i]
        add   $t1, $t3, $s8          # $t1 = &C[i]
        lw    $t0, 0($t5)            # $t0 = A[i]
        add   $t0, $t0, $s3          # $t0 = A[i]+j
        lw    $t0, 0($t0)            # $t0 = A[A[i]+j]
        sw    $t0, 0($t1)            # C[i] = A[A[i]+j]
        lw    $t0, 4($t5)            # $t0 = A[i]
        add   $t0, $t0, $s3          # $t0 = A[i]+j
        lw    $t0, 4($t0)            # $t0 = A[A[i]+j]
        sw    $t0, 4($t1)            # C[i] = A[A[i]+j]
        lw    $t0, 8($t5)            # $t0 = A[i]
        add   $t0, $t0, $s3          # $t0 = A[i]+j
        lw    $t0, 8($t4)            # $t0 = A[A[i]+j]
        sw    $t0, 8($t1)            # C[i] = A[A[i]+j]
        addi  $t2, t2, 3             # i++
        slt   $t4, $t2, 90           # $t4 = 1 if i < 90
        bne   $t4, $zero, Loop       # go back if i < 90
```

```
Unrolled, Scheduled:
        addi   $t2, $zero, $zero       # i = 0
Loop:   sll    $t3, $t2, 2             # $t3 = i*4
        addi   $t2, t2, 3              # i+=3
        slt    $t4, $t2, 90            # $t4 = 1 if i < 90
        add    $t8, $t3, $s6           # $t8 = &A[i]
        add    $t1, $t3, $s8           # $t1 = &C[i]
        lw     $t7, 0($t8)             # $t7 = A[i]
        lw     $t6, 4($t8)             # $t6 = A[i+1]
        lw     $t5, 8($t8)             # $t5 = A[i+2]
        add    $t7, $t0, $s3           # $t7 = A[i]+j
        add    $t6, $t6, $s3           # $t6 = A[i+1]+j
        add    $t5, $t5, $s3           # $t5 = A[i+2]+j
        lw     $t7, 0($t0)             # $t7 = A[A[i]+j]
        lw     $t6, 4($t6)             # $t6 = A[A[i+1]+j]
        lw     $t5, 8($t5)             # $t5 = A[A[i+2]+j]
        sw     $t7, 0($t1)             # C[i] = A[A[i]+j]
        sw     $t6, 4($t1)             # C[i+1] = A[A[i]+j]
        sw     $t5, 8($t1)             # C[i+2] = A[A[i]+j]
        bne    $t4, $zero, Loop        # go back if i < 90
Cycles = 1 + 29(18 + 1 control stall) + 18 = 570
```

12. (15 points) In this exercise we look at memory locality properties of matrix computation. The following code is written in MATLAB, where elements within the same column are stored contiguously. Assume each word is a 32-bit integer.

```
for I = 1:8
  for J = 1:8000
    for K = 1:2000
      A[I][J][K] = B[J][1][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][1][I]
Spatial	Unknown	Unknown	Unknown	No, 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 at least 16 E6 elements	Yes, 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 one element
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