The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Test 2 Solution
Fall 2009

Show all work. You will not receive full credit for a problem if you do not show your work!

1. (1 point) _Swap space_ is the space on the disk reserved for the full virtual memory of a process.
2. (1 point) MIPS exceptions are collected in the _Cause_ register.
3. (1 point) The _tags_ contain the address information required to identify whether a word in the cache corresponds to the requested word.
4. (1 point) Getting a missing data item early from the internal resources of the pipeline is called _forwarding_.
5. (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_.
6. (15 points) Virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. Consider this stream of virtual byte addresses as seen on a system: 9452, 30964, 19136, 46502, 38110, 16653, 48480. Assume 4KB 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 initial state of the TLB and the page table, show the final state of the system. Also list for each reference it is a hit in the TLB, a hit in the page table, or a page fault.

TLB

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

Page table

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

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 4KB (4096) and taking the integer part of the result.

	VPN	TLB?	PT?	PF?	Replacing 3 TLB?	PT?	PF?
9452	2	M	M	Y	M	M	Y
30964	7	H			H		
19136	4	M	H	N	M	Y	N
46502	11	H			M	Y	N
38110	9	M	M	Y	M	N	Y
16653	4	H			H		
48480	11	H			M	Y	N

Note: When bringing in page 4 to the TLB, 11 and 3 are equivalent choices for LRU, could have evicted 11.

Valid	Tag	PPN
1	~~11~~,4	~~12~~, 9
1	7	4
1	~~3~~, 11	~~6~~, 12
~~0~~, 1	~~4~~, ~~2~~, 9	~~9~~, ~~13~~, 14

7. (15 points) Unroll the following loop once and schedule it for maximum performance on a MIPS pipeline. Assume that the loop always executes an even number of iterations. You can use registers **$10** through **$20** when changing the code to eliminate dependences. The MIPS pipeline has full forwarding, no delay slots, branch prediction of not taken and branches completing in the ID stage. If the original loop executes 20 times, what is the speedup for the unrolled, scheduled loop as compared to the original loop?

```
Loop: lw    $1, 40($6)
      add   $5, $5, $1
      sw    $1, 20($5)
      addi  $6, $6, 4
      addi  $5, $5, -4
      beq   $5, $0, Loop
```

Original

```
Loop: lw    $1, 40($6)
      add   $5, $5, $1
      sw    $1, 20($5)
      addi  $6, $6, 4
      addi  $5, $5, -4
      beq   $5, $0,
```

C	IF	ID	EX	MEM	WB
1	lw $1				
2	add $5	lw $1			
3	sw $1	add $5	lw $1		
4	sw $1	add $5	bubble	lw $1	
5	addi $6	sw $1	add $5	bubble	lw $1
6	addi $5	addi $6	sw $1	add $5	bubble
7	beq	addi $5	addi $6	sw $1	add $5
8	after 1	beq	addi $5	addi $6	sw $1
9	lw $1	bubble	beq	addi $5	addi $6

19 iterations * 8 cycles per iteration + 1 iteration * 7 cycles per iteration = 159 cycles

Unrolled, Scheduled

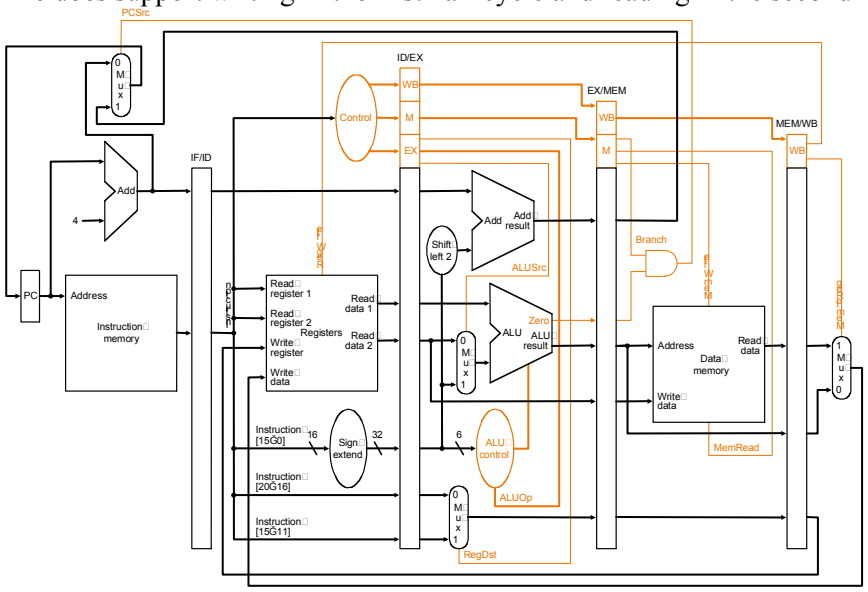
```
Loop: lw    $1, 40($6)
      lw    $10, 44($6)
      add   $5, $5, $1
      sw    $1, 20($5)
      add   $5, $5, $10
      sw    $1, 16($5)
      addi  $6, $6, 8
      addi  $5, $5, -8
      beq   $5, $0, Loop
```

C	IF	ID	EX	MEM	WB
1	lw $1				
2	lw $10	lw $1			
3	add $5	lw $10	lw $1		
4	sw $1	add $5	lw $10	lw $1	
5	add $5	sw $1	add $5	lw $10	lw $1
6	sw $1	add $5	sw $1	add $5	lw $10
7	addi $6	sw $1	add $5	sw $1	add $5
8	addi $5	addi $6	sw $1	add $5	sw $1
9	beq	addi $5	addi $6	sw $1	add $5
10	after 1	beq	addi $5	addi $6	sw $1
11	lw $1	bubble	beq	addi $5	addi $6

9 iterations * 10 cycles per iteration + 1 iteration * 9 cycles per iteration = 99 cycles

$$Speedup = \frac{P_{unrolled,scheduled}}{P_{original}} = \frac{ET_{original}}{ET_{unrolled,scheduled}} = \frac{159 cycles}{99 cycles} = 1.6$$

8. (15 points) Consider executing the following code on a pipelined datapath like the one shown except that it supports j instructions that cause no control hazards and has full forwarding. The register file does support writing in the first half cycle and reading in the second half cycle.




```
128  sort:   addi$sp, $sp, -20
132  sw      $ra, 16($sp)
136  sw      $s3, 12($sp)
140  sw      $s2, 8($sp)
144  sw      $s1, 4($sp)
148  sw      $s0, 0($sp)
152  add     $s2, $a0, $zero
156  add     $s3, $a1, $zero
160  add     $s0, $zero, $zero
164  for1tst:slt $t0, $s0, $s3
168  ***beq  $t0, $zero, exit1********
172  addi    $s1, $s0, -1
176  for2tst:slt $t0, $s1, $zero
180  bne     $t0, $zero, exit2
184  add     $t1, $s1, $s1
188  add     $t1, $t1, $t1
192  add     $t2, $s2, $t1
196    lw       $t3, 0($t2)
200    lw     $t4, 4($t2)
204    slt    $t0, $t4, $t3
208    beq    $t0, $zero, exit2
212    add    $a0, $s2, $zero
216    add    $a1, $s1, $zero
220    jal    swap
224    addi   $s1, $s1, -1
228    j      for2tst
232    exit2: addi   $s0, $s0, 1
236    j      for1tst
240    exit1: lw     $s0, 0($sp)
244    lw     $s1, 4($sp)
248    lw     $s2, 8($sp)
252    lw     $s3, 12($sp)
256    lw     $ra, 16($sp)
260    addi   $sp, $sp, 20
264    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 are the values stored in the following fields of the IF/ID pipeline register in the 12th cycle? Show the instructions being executed in each stage of the pipeline during each 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 `lw $t4` 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.

Fill in all of the fields, even if the current instruction in that stage is not using them.

IF/ID.PC+4 = 172
IF/ID.Instruction = 0001 0001 0000 0000 0000 0000 0001 0001 = 0x11000011
Opcode = 000100, rs = 01000, rt = 00000, offset – (240 – 172)/4 = 68/4 = 17

Cycle	IF	ID	EX	MEM	WB
1	lw $t4				
2	slt $t0	lw $t4			
3	beq $t0	slt $t0	lw $t4		
4	beq $t0	slt $t0	bubble	lw $t4	
5	add $a0	beq $t0	slt $t0	bubble	lw $t4
6	add $a1	add $a0	beq $t0	slt $t0	bubble
7	jal	add $a1	add $a0	beq $t0	slt $t0
8	addi $s0	bubble	bubble	bubble	beq $t0
9	j for1tst	addi $s0	bubble	bubble	bubble
10	slt $t0	j for1tst	addi $s0	bubble	bubble
11	beq $t0	slt $t0	j for1tst	addi $s0	bubble
12	addi $s1	beq $t0	slt $t0	j for1tst	addi $s0

The instruction of interest is the beq $t0, $zero, exit1 found at address 168.

9. (15 points) Here is a series of address references given as byte addresses: 0, 4, 16, 132, 232, 160, 1024, 30, 140, 3100, 180, 2180. Assuming a four-way set associative cache with one-word blocks and a total size of 8 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. The addresses shown in the data fields are bytes.

$8words \times \frac{1block}{1word} \times \frac{1set}{4blocks} = 2sets$, index = 1 bit, block offset = 0 bits, byte offset = 2 bits

	Tag	Index	Byte Offset	
0	0000 0000 0	0	00	miss
4	0000 0000 0	1	00	miss
16	0000 0001 0	0	00	miss
132	0000 1000 0	1	00	miss
232	0000 1101 1	0	00	miss
160	0000 1010 0	0	00	miss
1024	0100 0000 0	0	00	miss
30	0000 0001 1	1	10	miss
140	0000 1000 1	1	00	miss
3100	1100 0001 1	1	00	miss
180	0000 1011 0	1	00	miss
2180	1000 1000 0	1	00	miss

Index	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0x00 0, 0x80 0	M[0:3], M[1024:1027]	0x0A 0	M[160:163]	0x01 0	M[16:19]	0x0D 1	M[232:235]
1	~~0x00 0~~, 0xC1 1	~~M[4:7],~~ M[31000:3103]	~~0x01 1~~, 0x88 0	~~M[28:31],~~ M[2180:2183]	0x08 1,	M[140:143],	~~0x08 0,~~ 0x0B 0	~~M[132:135],~~ M[180:183]

10. (10 points) The following code is written in C, where elements within the same row are stored contiguously. References to which variables exhibit spatial locality?

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

Variable	A[I][J]	B[J][0]	A[J][I]
First access	A[0][0]	B[0][0]	A[0][0]
Second access	A[1][0]	B[1][0]	A[0][1]
Spatial locality?	No, all elements of row 0 are stored before the first element of row 1 is stored, these two accesses are at least 8000 elements apart	Maybe, if matrix B has its second dimension = 1, this situation is unlikely. These accesses are as far apart as the second dimension of B.	Yes, the elements in row A[J] are accessed sequentially

11. (15 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	Fourth-level cache, fully associative speed	Local miss rate with fourth-level cache, fully associative
2.0	3 GHz	125 ns	5 %	15 cycles	60 %	50 cycles	43 %	100 cycles	62 %

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.

CPItotal = CPIbase + Memory accesses/instruction *(L1 Hit penalty cycles + Miss rate * Miss penalty cycles)

Miss rate * Miss penalty cycles = L1 miss rate *(L2 hit time cycles + L2 miss rate * (L3 hit time cycles+ L3 miss rate *(L4 hit time cycles + L4 miss rate * main memory penalty cycles)))

= 0.05 * (15 + 0.6 * (50 + 0.43 *(100 + 0.62 * 125*3)))
= 0.05 * (15 + 0.6 * (50 + 0.43 *(332.5))
= 0.05 * (15 + 0.6 * (192.98))
= 0.05 * (130.78)
= 6.54

CPItotal = 2.0 + 1.36 * (0 + 6.54) = 10.89

12. (10 points) Consider a pipeline for a register-memory architecture. The architecture has two instruction formats: a register-register format and a register-memory format. There is a single memory addressing mode (offset + base register). There is a set of ALU operations as follows:

ALUop← Rdest,Rsrc1,Rsrc2,offset
Rdest ← Rsrc1 ALUop Rsrc2
or Rdest ← Rsrc1 ALUop MEM[Rsrc2 + offset]
or
Rdest ← MEM[Rsrc2 + offset]
or
MEM[Rsrc2 + offset] ← Rsrc1

where the ALUop is one of the following:

Add, Subtract, And, Or(with or without offset)
Load(Rsrc1 omitted)
Store(Rdest omitted)

Rdest, Rsrc1 and Rsrc2 are registers.

Branches use a full compare of two registers and are PC-relative. Assume that this machine is pipelined so that a new instruction is started every clock cycle. The pipeline structure is

IF	RF	ALU1	MEM	ALU2	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	ALU1	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

The first ALU stage is used for effective address calculation for memory references and branches. The second ALU stage is used for operations and branch comparisons. RF is both a decode and register-fetch stage. Assume that when a register read and a register write of the same register occur in the same clock cycle, the write data is forwarded. For the following code fragment:

```
a  add   $1, $1, 0($15)
b  add   $1, $1, 4($15)
c  add   $2 ,$2, 0($1)
d  add   $2, $2, 4($1)
e  store $2, 8($1)
```

Dependencies
a & b, c & d
b & c, d & e
b & d
b & e

identify the data dependencies and draw a figure (using the multiple clock style) showing them as lines.