The University of Alabama in Huntsville
ECE Department
CPE 431 01
Test 2 Solution
Fall 2018

1. (3 points) The three C's of cache misses are _**capacity**_, _**conflict**_, and _**compulsory**_.
2. (1 point) _**Virtual memory**_ is a technique that uses main memory as a "cache" for secondary storage.
3. (1 point) A _**context switch**_ is a changing of the internal state of the processor to allow a different process to use the processor that includes saving the state needed to return to the currently executing process.

4. (10 points) Calculate the total number of bits required for a cache with the parameters listed below, assuming a 32-bit address and byte addressability. Include valid and dirty bits.

Cache Data Size: 32 KiB
Cache Block Size: 2 64-bit words
Cache Set Associativity: 4-way
Cache Access Time: 1 cycle

32 KiB x 1 word/8 bytes x 1 block/2 words x 1 set/4 blocks = 512 sets
Byte offset = 3 bits (8 bytes per word)
Block offset = 1 bit (2 words per block)
Index = 9 bits (512 sets)
Tag = 32 – (9 + 1 + 3) = 32 – 13 = 19 bits
Each block has data plus tag plus dirty and valid bits

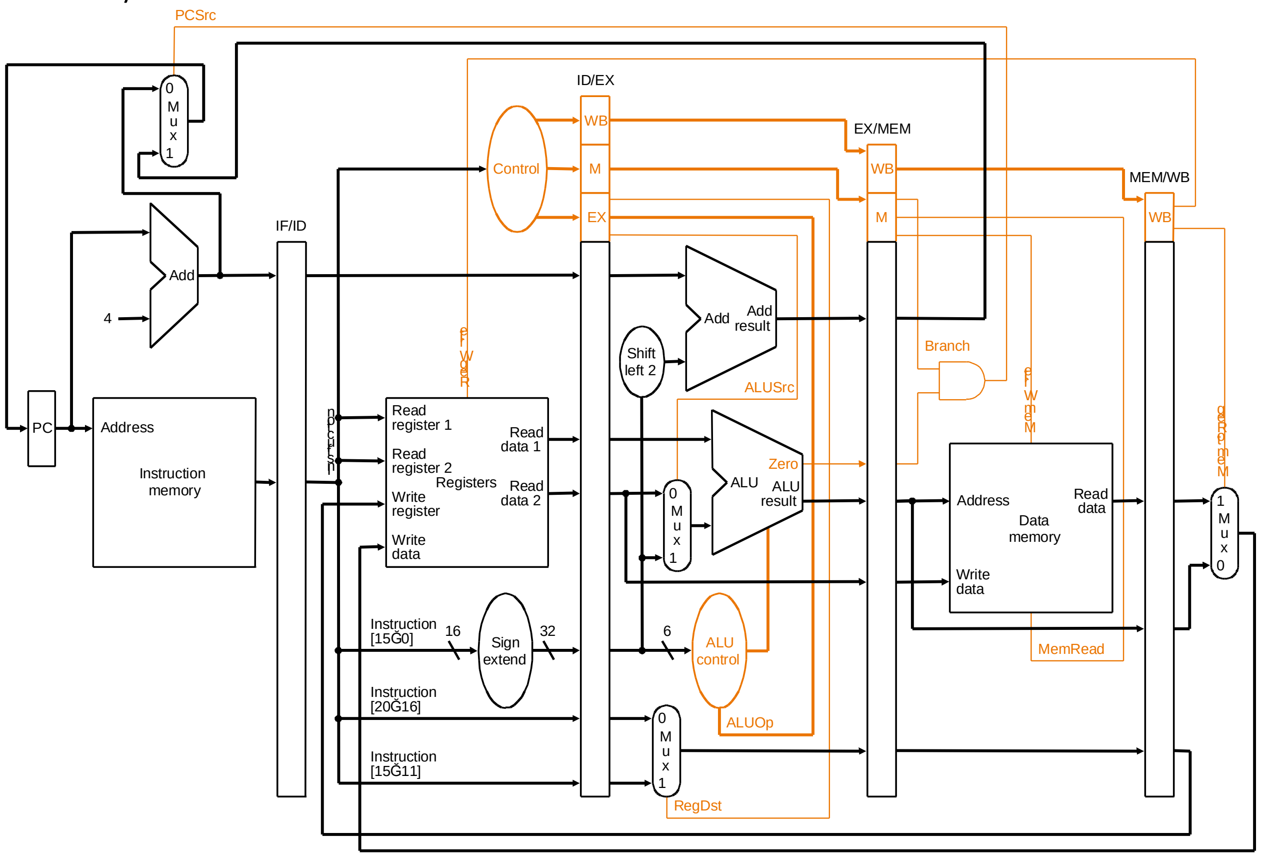
Total Bits = 512 sets x 4 blocks/set x (128 + 19 + 2) bits/block = 2048 x 149 bits = 305,152 bits

5. (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, 2070, 1080. Assuming a two-way set associative-mapped cache with 4-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/4 words x 1 set/2 blocks = 2 sets**

Byte Address (Decimal)	Byte Address (Hexadecimal)	Byte Address (Binary)	Offsets	Hit/Miss
118	76	`0000 0000 01 1 10 110`	22	Miss
483	1E3	`0000 0001 11 1 00 011`	3	Miss
2069	815	`0000 1000 00 0 10 101`	21	Miss
321	141	`0000 0001 01 0 00 001`	1	Miss
368	170	`0000 0001 01 1 10 000`	16	Miss
1505	5E1	`0000 0101 11 1 00 001`	1	Miss
812	32C	`0000 0011 00 1 01 100`	12	Miss
2832	B10	`0000 1011 00 0 10 000`	16	Miss
373	175	`0000 0001 01 1 10 101`	21	Miss
1411	583	`0000 0101 10 0 00 011`	3	Miss
511	1FF	`0000 0001 11 1 11 111`	31	Miss
122	7A	`0000 0000 01 1 11 010`	28	Miss
690	2B2	`0000 0010 10 1 10 010`	18	Miss
4820	12D4	`0001 0010 11 0 10 100`	20	Miss
1714	6B2	`0000 0110 10 1 10 010`	18	Miss
1508	5E4	`0000 0101 11 1 00 100`	4	Miss
2070	816	`0000 1000 00 0 10 110`	22	Miss
1080	438	`0000 0100 00 1 11 000`	24	Miss

Set	Tag	Data	Tag	Data
0	~~5~~, ~~22~~, ~~32~~	~~M[320..351],~~ ~~M[1408..1439],~~ M[2048..2069]	~~32~~, 44, 75	~~M[2048..2079],~~ ~~M[2816..2847],~~ M[4800..4831]
1	~~1~~, ~~5~~, ~~12~~, ~~7~~, ~~10~~, 23	~~M[96..127],~~ ~~M[352..383],~~ ~~M[800..831],~~ ~~M[480..511],~~ ~~M[672..703],~~ ~~M[1504..1535]~~	~~7~~, ~~23~~, ~~5~~, ~~1~~, ~~26~~, 16	~~M[480..511],~~ ~~M[1504..1535],~~ ~~M[352..383]~~, ~~M[96..127],~~ ~~M[1696..1727],~~ M[1056..1087]

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 full forwarding. The register file supports 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
228           jal   swap
232           addi  $s1, $s1, -1
236           j     for2tst
240 exit2:    addi  $s0, $s0, 1
244           j     for1tst
248 exit1:    lw    $s0, 0($sp)
252           lw    $s1, 4($sp)
256           lw    $s2, 8($sp)
260           lw    $s3, 12($sp)
264           lw    $ra, 16($sp)
268           addi  $sp, $sp, 20
272           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 ReadData1 field of the ID/EX 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	**beq $t0**	**slt $t0**	**lw $t4**	**lw $t3**	**add $t2**
6	**beq $t0**	**slt $t0**	**bubble**	**lw $t4**	**lw $t3**
7	**add $a0**	**beq $t0**	**slt $t0**	**bubble**	**lw $t4**
8	**add $a1**	**add $a0**	**beq $t0**	**slt $t0**	**bubble**
9	**jal swap**	**add $a1**	**add $a0**	**beq $t0**	**slt $t0**
10	**addi $s0**	**bubble**	**bubble**	**bubble**	**beq $t0**
11	**j for1test**	**addi $s0**	**bubble**	**bubble**	**bubble**

Instruction of interest is the bubbled jal swap. Assuming swap is close by, the rs field is 0, so the value is 20.

ID/EX.ReadData1 = **20**

7. (20 points) Consider the execution of the following loop in a statically scheduled superscalar processor that has full forwarding. Additionally, any branches are resolved in the EX stage.

```
Loop:  lw    $1, 40($6)
       add   $1, $7, $1
       sw    $1, 20($5)
       addi  $6, $6, 4
       addi  $5, $5, -4
       bne   $5, $0, Loop
```

a. (15 points) Unroll this loop so that four iterations of it are done at once and schedule it for maximum performance on a 2-issue static superscalar processor that has one slot for R-type/branch instructions and one slot for lw/sw instructions. Assume that the loop always executes a number of iterations that is a multiple of 4. You can use registers **$10** through **$20** when changing the code to eliminate dependences.
b. (5 points) Calculate the number of cycles for the original and for the unrolled, scheduled code and the speedup of unrolled, scheduled compared to the original.

a. Unrolled, scheduled

Cycle	R-type/branch	lw/sw
1	`addi $5, $5, -16`	`lw $1, 40($6)`
2		`lw $10, 440($6)`
3	`add $1, $7, $1`	`lw $11, 48($6)`
4	`add $10, $7, $10`	`lw $12, 52($6)`
5	`add $11, $7, $11`	`sw $1, 36($5)`
6	`add $12, $7, $12`	`sw $10, 32($5`
7	`addi $6, $6, 16`	`sw $11, 28($5`
8	`bne $5, $0, Loop`	`sw $12, 24($5`

b. Unrolled, scheduled ET = 8 cycles (issue slots) + 2 cycles control stall
Original ET = 4 * (6 cycles (issue slots) + 1 data hazard stall (lw-add) + 2 cycles control stall)
Speedup = Original ET/Unrolled, scheduled ET = 36/10 = 3.6

8. (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 32 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~~, 1	~~12~~, 13
1	7	4
1	3	6
~~0~~, 1	4, ~~5~~, 4	~~9~~, ~~11~~, 9

Page table

VPN	Valid	Physical page or in disk
0	1	5
1	0	~~Disk~~, ~~13~~
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

Address	VPN	TLB Hit/Miss	Page Table Hit/Miss	Page Fault Y/N
184,760	**5**	**M**	**H**	**N**
110,824	**3**	**H**	**-**	**-**
236,100	**7**	**H**	**-**	**-**
119,200	**3**	**H**	**-**	**-**
56,312	**1**	**M**	**M**	**Y**
151,692	**4**	**M**	**H**	**N**
126,856	**3**	**H**	**-**	**-**
40,000	**1**	**H**	**-**	**-**

9. (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:

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:

```
add     $1, $1, 0($15)
add     $1, $1, 4($15)
load    $2 , 0($1)
add     $2, $2, 4($1)
store   $2, 8($1)
```

identify the data dependencies and draw a figure (using the multiple clock style) showing them as lines. You have space on the next page for this figure if you choose to use it.

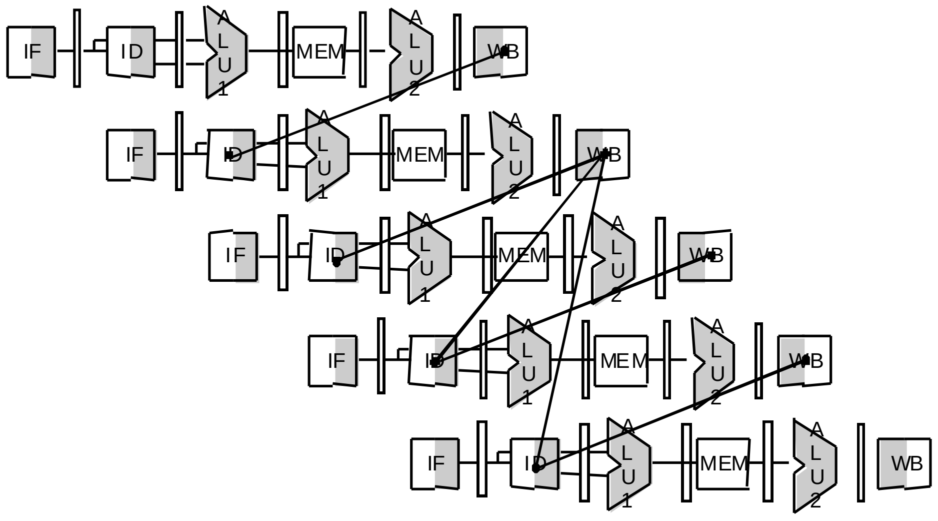
```
add $1,  $1, 0($15)

add $1, $1, 4($15)

add $2, $2, 0($1)

add $2, $2, 4($1)

store $2,  8($1)
```



10. (5 points) The following code is written in C, where elements within the same row are stored contiguously. Does `B(j, 0)` exhibit spatial locality? temporal locality? Explain your answers. A, B, and C are all arrays of integers 8000 by 8000.

```
for (i = 0; i< 8000; i++)
  for (j = 0; j < 8000; j++)
      A(i,j ) = B(j, 0) + A(j, i) + C(0, i);
```

`B(j, 0)` does not exhibit spatial locality, the access is `B(0, 0), B(1, 0),` … . Since they don't come from the same row, they are stored 8000 locations apart.
`B(j, 0)` exhibits temporal locality if once every 8000 times is soon, you could also argue that it doesn't because it changes every time in the inner loop