The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Test 2 Solution
Fall 2008

1. (1 point) _True_(True or False) The number of sets in a fully associative cache is one.
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) _Loop unrolling_ is a technique to get more performance from loops that access arrays, in which multiple copies of the loop body are made and instructions from different iterations scheduled together.
4. (1 point) _Hot-swapping_ is replacing a hardware component while the system is running.
5. (1 point) _False_(True or False) The number of sets in a two-way set associative cache is two.
6. (15 points) Consider the MIPS five stage pipeline. For the base case, an instruction fetch takes 100 ps, a register read 60 ps, an ALU operation 150 ps, a data access 100 ps, and a register write 60 ps. If the register reads and write times can be shortened by 30 %, will the speedup obtained from pipelining be affected? If yes, by how much? Otherwise, why? What if the register reads and writes now take 30 % more time?

   a. If the time for register file reads/writes is reduced to 42 ps, the pipeline can not be changed because the clock cycle still needs to be 150 ps to accommodate the ALU. The register file time in this scenario is 42 ps (write) + 42 ps (read) = 84 ps, still much less than 150 ps.
   b. If the time for register file reads/writes is increased to 78 ps, the pipeline must be changed because the time needed for the register file is greater than 150. The register file time in this scenario is 78 ps (write) + 78 ps (read) = 156 ps. The clock cycle must be adjusted to 156 ps.

   Original

$$\frac{P_{pipe}}{P_{nonpipe}} = \frac{Timebetweeninstructions_{nonpipe}}{Timebetweeninstructions_{pipe}} = \frac{470\,ps}{150\,ps} = 3.13$$

   Longer register file read/write

$$\frac{P_{pipe}}{P_{nonpipe}} = \frac{Timebetweeninstructions_{nonpipe}}{Timebetweeninstructions_{pipe}} = \frac{506\,ps}{156\,ps} = 3.24$$

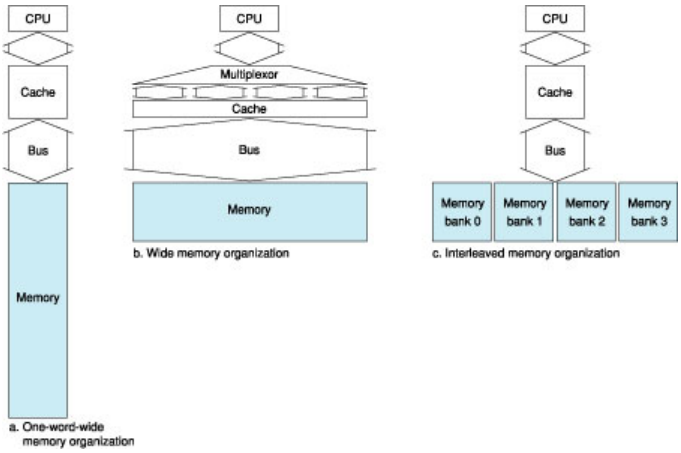
7. (10 points) The speed of light is approximately 3 x 10^8 meters per second, and electrical signals travel at about 50% of this speed in a conductor. When the term high speed is applied to a network, it is the bandwidth that is higher, not necessarily the velocity of the electrical signals. Calculate the "flight time", or latency, for the electrical signals. Consider two computers that are 20 meters apart and two computers that are 2000 kilometers apart.

$$flight\_time = \frac{distance}{effective\_velocity}$$

   For 20 m, $flight\_time = \frac{20m}{effective\_velocity} = \frac{20m}{0.5 * 3 \times 10^8 \frac{m}{s}} = 133ns$

For 2000 km, $flight\_time = \frac{2000km}{effective\_velocity} = \frac{20km}{0.5 * 3 \times 10^8 \frac{m}{s}} = 13.3ms$

8. (10 points) Consider a memory hierarchy using one of the three organizations for main memory shown. Assume that the cache block size is 32 words, that the width of organization (b) of the figure is four words, and that the number of banks in organization (c) is eight. If the main memory latency for a new access is 10 memory bus cycles and the transfer time is 1 memory bus cycle, what are the miss penalties for these organizations?



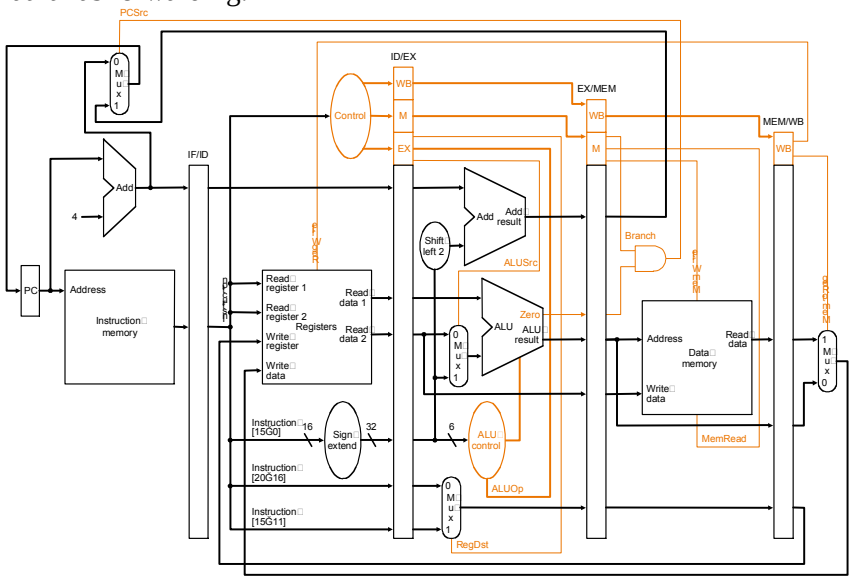
send address 1 cycle, memory access 10 cycles, bus transfer 1 cycle

a. total = send address + 32(memory access + bus transfer)
= 1 + 32(10+1) = 1 + 352
= 353 cycles

b. total = send address + 32/4(memory access + bus transfer)
= 1 + 8 (10+1) = 1 + 88
= 89 cycles

c. total = send address + 32/8(memory access) + 32(bus transfer)
= 1 + 4(10) + 32(1) = 1 + 40 + 32
= 73 cycles

9. (20 points) Consider executing the following code on a pipelined datapath like the one shown except that it has forwarding.




```
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
200      add $t1, $s1, $s1
204      add $t1, $t1, $t1
208      add $t2, $s2, $t1
212      lw  $t3, 0($t2)
216      lw  $t4, 4($t2)
220      slt $t0, $t4, $t3
224      beq $t0, $zero, exit2
228      add $a0, $s2, $zero
⇒232     add $a1, $s1, $zero
236      jal swap
240      addi$s1, $s1, -1
244      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 $t1` instruction two 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 ID/EX pipeline register in the 12th 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 $t1` 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.

ID/EX.WB = 00_2
ID/EX.MEM = 000_2
ID/EX.EX = 0000_2
ID/EX.PCInc = 236_{10}
ID/EX.ReadData1 = $s1 = 37_{10}
ID/EX.ReadData2 = $zero = 0_{10}

ID/EX.SignExtend = 00101 00000 100000 = 00002820_{16}
ID/EX.WriteRt = 0_{10}
ID/EX.WriteRd = 5_{10}

Cycle	IF	ID	EX	MEM	WB
1	add $t1				
2	add $t1	add $t1			
3	add $t2	add $t1	add $t1		
4	lw $t3	add $t2	add $t1	add $t1	
5	lw $t4	lw $t3	add $t2	add $t1	add $t1
6	slt $t0	lw $t4	lw $t3	add $t2	add $t1
7	beq $t0	slt $t0	lw $t4	lw $t3	add $t2
8	beq $t0	slt $t0	bubble	lw $t4	lw $t3
9	add $a0	beq $t0	slt $t0	bubble	lw $t4
10	add $a1	add $a0	beq $t0	slt $t0	bubble
11	jal swap	add $a1	add $a0	beq $t0	slt $t0
12	addi $s0	bubble	bubble	bubble	beq $t0

The instruction of interest is the add $a1 instruction that has been turned into a bubble.

10. (15 points) Here is a series of address references given as word addresses: 1, 4, 8, 5, 20, 17, 19, 9, 56, 11, 4, 43, 5, 6. 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 final contents of the cache.

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

	Tag	Index	hit/miss
1	000000	1	miss
4	000010	0	miss
8	000100	0	miss
5	000010	1	miss
20	001010	0	miss
17	001000	1	miss
19	001001	1	miss
9	000100	1	miss
56	011100	0	miss
11	000101	1	miss
4	000010	0	hit
43	010101	1	miss
5	000010	1	miss
6	000011	0	miss

0	M[4]	~~M[8],~~ M[6]	M[20]	M[56]
1	~~M[1],~~ M[9]	~~M[5],~~ M[11]	~~M[17~~], M[43]	~~M[19],~~ M[5]

11. (15 points) You have been given asked to build a 1MB instruction cache for a processor with a 32-bit address and 32 bit words. You do not have a byte offset. You do need 2 bits of storage per block for valid, dirty and other status bits. How many 16K X 8 SRAM chips do you need if the cache is two-way set associative with two word blocks? Show the breakdown on the address into its cache access components and describe how the various SRAM chips are used.

Byte offset is 0 bits and Block offset is 1 bit. To calculate the number of sets and thus, the index, divide the total number of bytes in the cache by the number of bytes in a set.

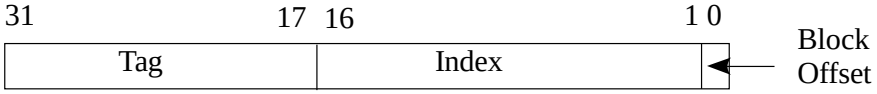
$$Bytes\_per\_set = 2blocks * \frac{2words}{1block} * \frac{4bytes}{1word} = 16bytes$$

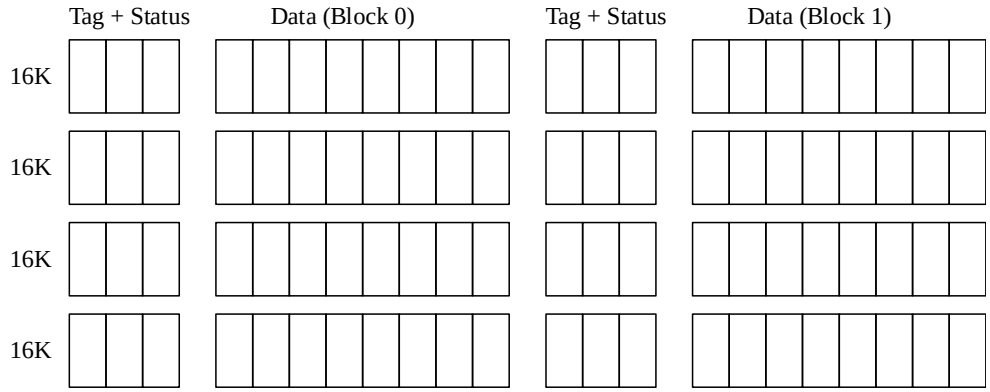
The number of sets is 2^{20}(1 Mbyte)/2^4(bytes per set) = 2^{16} sets.
Knowing the number of sets, we can calculate how many rows of chips are needed. The number of rows is 2^{16}(sets)/2^{14}(sets/chip) = 2^2 = 4.
To know how many columns of chips are needed, we need to calculate the tag.
Tag = address – index – block offset – byte offset = 32 – 16 – 1 – 0 = 15. However, we also need to store 2 status bits along with the tag, so 17 total bits are needed, requiring 3 chips for each tag/status group per block. ⌈17 bits / 8bits/chip⌉ = 3. The remaining component of the columns are the data chips. Each block takes ⌈64 bits / 8bits/chip⌉ = 8 chips. Each block has a tag/status group. That makes a total of 11 chips per block and 22 chips total.

So, we have 4 rows of 22 chips, 24 for tag/status, 64 for data.

The address is broken down as follows

Byte offset – 0 bits, Block offset – 1 bit, Index – 16 bits, Tag – 15 bits





12. (10 points) The MicroBlaze embedded soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx field programmable gate arrays (FPGAs). It has a 32-bit instruction word with three operands and two addressing modes. MicroBlaze instructions are either Type A or Type B. The instruction formats for each are given below:

Bits	0-5	6-10	11-15	16-20	20-31
Type A	opcode	Rd	Ra	Rb	00000000000
Type B	opcode	Rd	Ra	Immediate	
				16-31	

For arithmetic/logical instructions, we have:

Rd ← Ra op Rb, where op is +, -, AND, OR, etc. `add Rd, Ra, Rb`

For loads:

Rd ← MEM[Ra + Rb] `lw` Rd, Ra, Rb

or Rd ← MEM[Ra + Immediate] `lw Rd, Ra, Imm`

For stores:

MEM[Ra + Rb] ← Rd `sw Rd, Ra, Rb`

MEM[Ra + Immediate] ← Rd `sw Rd, Ra, Imm`

For beq:

PC ← Rb if Ra = 0 `beq Ra, Rb`

The MicroBlaze has a 3 stage pipeline as follows:

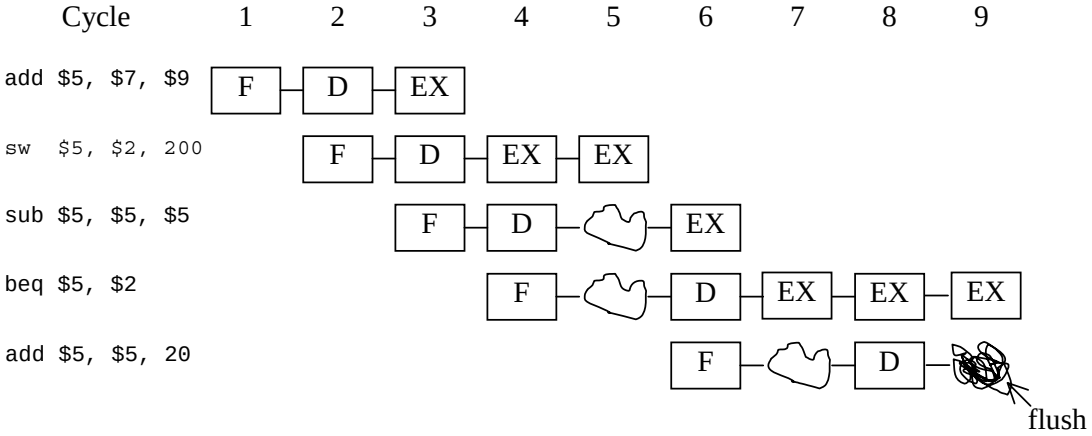
Fetch	Decode	Execute

The register file is read in the Decode state and the ALU and memory operations take place in the Execute stage. Even without data hazards, the pipeline has stalls. Loads and stores require an additional cycle to complete (a total of 4) and beq requires two extra cycles to complete (a total of 5) if taken. If the beq is not taken, no stalls are required.

For the following MicroBlaze code, how many cycles will it take to execute?

```
add   $5, $7, $9
sw    $5, $2, 200
sub   $5, $5, $5
beq   $5, $2
add   $5, $5, 20
```

One way to answer this is to use a multi-cycle diagram.



Another way to look at it is:

Cycle	Fetch	Decode	Execute
1	`add $5`		
2	`sw $5`	`add $5`	
3	`sub $5`	`sw $5`	`add $5`
4	`beq $5`	`sub $5`	`sw $5`
5	`beq $5`	`sub $5`	`sw $5`
6	`add $5`	`beq $5`	`sub $5`
7	after 1	`add $5`	`beq $5`
8	after 1	`add $5`	`beq $5`
9	after 1	`add $5`	`beq $5`
10	target	`bubble`	`bubble`

Either way, it takes 9 cycles.