The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Final Exam Solution
Fall 2002

1. (3 points) The three C's of caches are ____capacity___, ___conflict____ and ___compulsory___.

2. (1 point) One addressing mode supported by the MIPS processor is __immediate___.

3. (5 points) Given the bit pattern:

`0000 0000 0000 1000 0101 0011 1100 0000`$_2$

what does it represent, assuming that it is a MIPS instruction? Rearranging a bit,

```
000000 00000 01000 01010 01111 000000 or
000000 00000 01000 0101001111000000
```

are the two possibilities. Since the opcode is `000000,` the instruction is R-type. The remaining fields are `rs = 00000, rt = 01000, rd = 01010, shamt = 01111, and funct = 000000.` The instruction which has 0 values for opcode, rs, and funct is sll. The instruction is

`sll $10, $8, 15` or `sll $t2, $t0, 15`

4. (5 points) Given the pipelined processor of Chapter 6 with forwarding, determine which instruction is being executed in each stage of the pipeline in cycle 11 of the following instruction sequence if it begins executing in cycle 1.

```
1.    addi  $23, $29, 12
2.    sw    $2, 0($29)
3.    sw    $15, 4($29)
4.    sw    $16, 8($29)
5.    muli  $8, $5, 4
6.    add   $7, $4, $2
7.    lw    $15, 0($2)
8.    lw    $16, 4($2)
9.    sw    $17, 0($2)
10.   sw    $18, 4($2)
11.   lw    $2, 0($29)
12.   lw    $15, 4($29)
13.   lw    $16, 8($29)
14.   addi  $29, $29, 12
```

IF ____11_____ ID ____10_____ EX _____9_____ MEM ____8_____ WB ___7_____

5. (15 points) A program repeatedly performs a three-step process: It reads in a 8-KB block of data from disk, does some processing on that data, and then writes out the result as another 8-KB block elsewhere on the disk. Each block is contiguous and randomly located on a single track on the disk. The disk drive rotates at 9600 RPM, has an average seek time of 8 ms, and has a transfer rate of 32 MB/sec. The controller overhead is 2 ms. No other program is using the disk or processor, and there is no overlapping of disk operation with processing. The processing step takes 20 million clock cycles, and the clock rate is 500 MHz. What is the overall speed of the system in blocks processed per second?

Disk Access = seek + rotational latency + controller + transfer
= 8 ms + 0.5rotations * 1 minute/ 9600 rotations * 60 s/min + 2 ms + 8 KB/32MB/s
= 8 ms + 3.1 ms + 2 ms + 0.25 ms = 13.35 ms

Since each block processed involves two accesses (read and write), the disk component of the time is 27.7 ms per block processed. The (non-overlapped) computation takes 20 million clock cycles at 500 MHz, or another 40 ms. Thus, the total time to process one block is 67.7 ms, and the number of blocks processed per second is simply 1/0.0677 = 14.8.

6. (1 point) ____Communication____ is the hardest problem.

7. (10 points) Consider a virtual memory system with the following properties:

- 40-bit virtual byte address
- 16-KB pages
- 36-bit physical byte address

What is the total size of the page table for each process on this machine, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and that all the virtual pages are in use? (Assume that disk addresses are not stored in the page table.)

# of virtual pages = $\frac{2^{40}}{2^{14}} = 2^{26}$

#bits in physical page address = 36-14 = 22

#total bits = $2^{26} \cdot (22+4) = 2^{26} \cdot 26$

8. (10 points) Suppose we are considering a change to an instruction set. The base machine initially has only loads and stores to memory, and all operations work on the registers. Such machines are called load-store machines (like the MIPS). Measurements of the load-store machine showing the instruction mix and clock cycle counts per instruction are given in the table below:

Instruction Type	Frequency	Clock cycle count
ALU ops	43 %	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

Let's assume that 25% of the arithmetic logic unit (ALU) operations directly use a loaded operand that is not used again. We propose adding ALU instructions that have one source operand in memory. These new register-memory instructions have a clock cycle count of 2. Suppose that the extended instruction set increases the clock cycle count for branches by 1, but it does not affect the clock cycle time. Would this change improve CPU performance?

$ET = IC * CPI * CC$ $\qquad$ $CC_{orig} = CC_{mod}$

$ET_{orig} = IC_{orig} * CPI_{orig} * CC_{orig}$, $\qquad$ $ET_{mod} = IC_{mod} * CPI_{mod} * CC_{mod}$

$CPI_{orig} = 0.43*1 + 0.57*2 = 0.43 + 1.14 = 1.57$

$CPI_{mod} = ((0.43/4)*2 + (0.43 – 0.43/4)*1 + (0.21 –0.25*0.43)*2 + 0.12*2 + 0.24*3)/0.89$
$= (0.22 + 0.32 + 0.2 + 0.24 + 0.72)/0.89 = 1.91$

$= 1.10 + 0.2 + 0.72 = 2.02$

$IC_{mod} = (1 - 0.25*0.43)*IC_{orig} = (1 - 0.11)* IC_{orig} = 0.89*IC_{orig}$

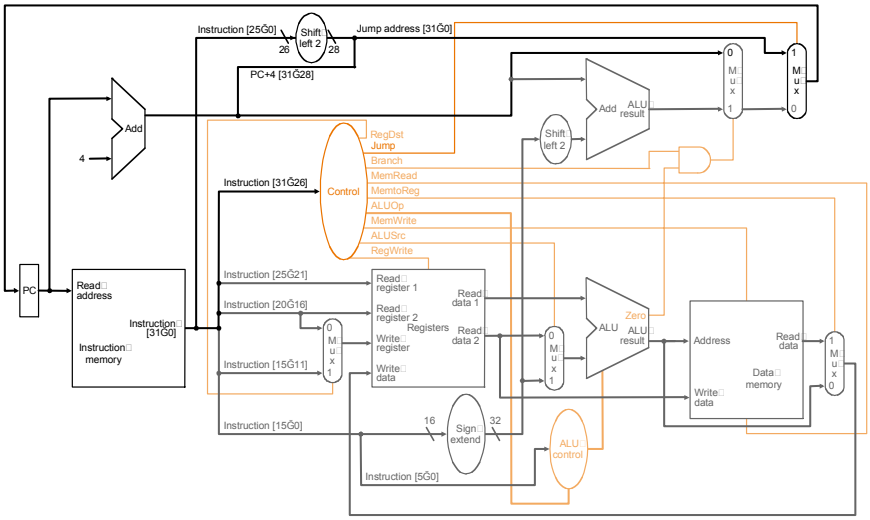
$$\frac{P_{\text{mod}}}{P_{orig}} = \frac{ET_{orig}}{ET_{\text{mod}}} = \frac{IC_{orig} * CPI_{orig} * CC_{orig}}{IC_{\text{mod}} * CPI_{\text{mod}} * CC_{\text{mod}}} = \frac{IC_{orig} * CPI_{orig}}{IC_{\text{mod}} * CPI_{\text{mod}}} = \frac{ICorig * 1.57}{0.89 * IC_{orig} * 1.91} = \frac{1.57}{1.7} = 0.92$$

No, this change would not increase CPU performance.

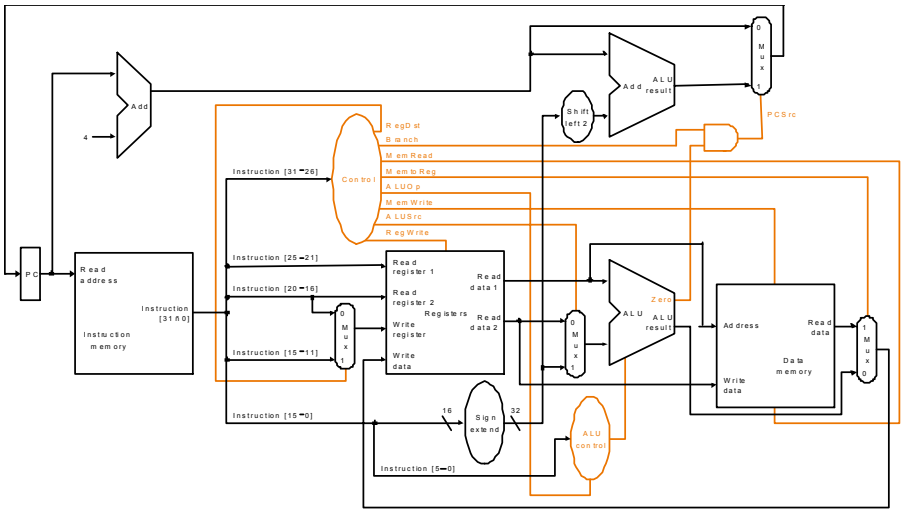
9. (15 points) Consider the following idea: Let's modify the instruction set architecture and remove the ability to specify an offset for memory access instructions. Specifically, all load-store instructions with nonzero offsets would become pseudoinstructions and would be implemented using two instructions. For example:

```
addi  $at, $t1, 104    # add the offset to a temporary
lw    $t0, $at         # new way of doing lw $t0, 104 ($t1)
```

(a) (5 points) What changes would you have to make to the single-cycle datapath and control if this simplified architecture were to be used?



The changes shown are that the output of the first register is fed directly to the data input of the data memory and the output of the ALU now only has the path around the data memory.



(b) (10 points) If the delay for the instruction and data memory is 2 ns, the register file delay (read or write) is 1 ns, and the ALU delay is 2 ns, what are the clock cycles required for the original datapath and for the modified datapath? What is the highest percentage of load-store instructions with offsets that could be tolerated without total performance being degraded?

The cycle time could be modified; the new instruction completion times are given below:

R-type	IM + REG + ALU + REG = 2 ns + 1 ns + 2 ns + 1 ns = 6 ns
Lw	IM + REG + DM + REG = 2 ns + 1 ns + 2 ns + 1 ns = 6 ns
Sw	IM + REG + DM = 2 ns + 1 ns + 2 ns = 5 ns
Branch	IM + REG + ALU = 1 ns + 2 ns + 2 ns

So, Ccnew = 6 ns, while Ccold = 8 ns (for lw, IM + REG + ALU + DM + REG = 1+2+2+2+1 = 8 ns).
Icnew = Icold + xICold, where x is the percentage of instructions which touch data memory (lw/sw).
CPInew = CPIold = 1.

$$1 = \frac{P_{new}}{P_{old}} = \frac{ETold}{ETnew} = \frac{IC_{old} * CPI_{old} * CC_{old}}{IC_{new} * CPI_{new} * CC_{new}} = \frac{IC_{old} * 1 * 8ns}{IC_{old}(1+x) * 1 * 6ns}$$, giving

(1 + x)*6ns = 8ns, 1 + x = 1.333
x = 0.333 or 33 %

10. (10 points) Use the following code fragment:

```
loop: lw    $1, 0($2)
      addi  $1, $1, 4
      sw    $1, 0($2)
      addi  $2, $2, 4
      bne   $2, $3, loop
```

Assume that the initial value of $2 is $3 – 396 and that this code fragment is run on a machine with a 500-MHz clock that requires the following number of cycles for each instruction:

Instruction	Cycles
addi	1
add	2
lw, sw	3
bne	4

In the worst case, how many seconds will it take to execute this code?

This code will execute (0 – (-392))/4 + 1 = 392/4 + 1 = 98 + 1 = 99 times.
Each iteration will take 3(lw) + 1(addi) + 3(sw) + 1(addi) + 4(bne) = 12 cycles

$$ExecutionTime = \#iterations * \frac{\#cycles}{iteration} * \frac{\#s}{cycle} = 99 * 12 * 2ns = 2.376\mu s$$

11. (10 points) What size messages would result in ATM outperforming Ethernet by a factor of ten, assuming the following latencies and bandwidths?

Characteristic	Ethernet	ATM
Bandwidth from node to network	1.125 MB/sec	10 MB/sec
Interconnect latency	15 μs	50 μs
HW latency to/from network	6 μs	6 μs
SW overhead sending to network	200 μs	207 μs
SW overhead receiving from network	241 μs	360 μs

For ATM to be ten times as fast as Ethernet, the total message time must be one-tenth that of Ethernet. We can write

$15 + 6 + 200 + 241 + \text{Transmission time}_{\text{Ethernet}} = 10 \times (50 + 6 + 208 + 360 + \text{Transmission time}_{\text{ATM}})$

462 + X/1.125 = 10(623 +X/10)
11.25(462 +X/1.125) = 11.25(6230 + (10X/10)
5197.5 + 10X = 70087.5 + 11.25X
-1.25X = 64890
X = -5192 bytes, The number of bytes can't be negative so there is no message size for which this is possible.

12. (15 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:

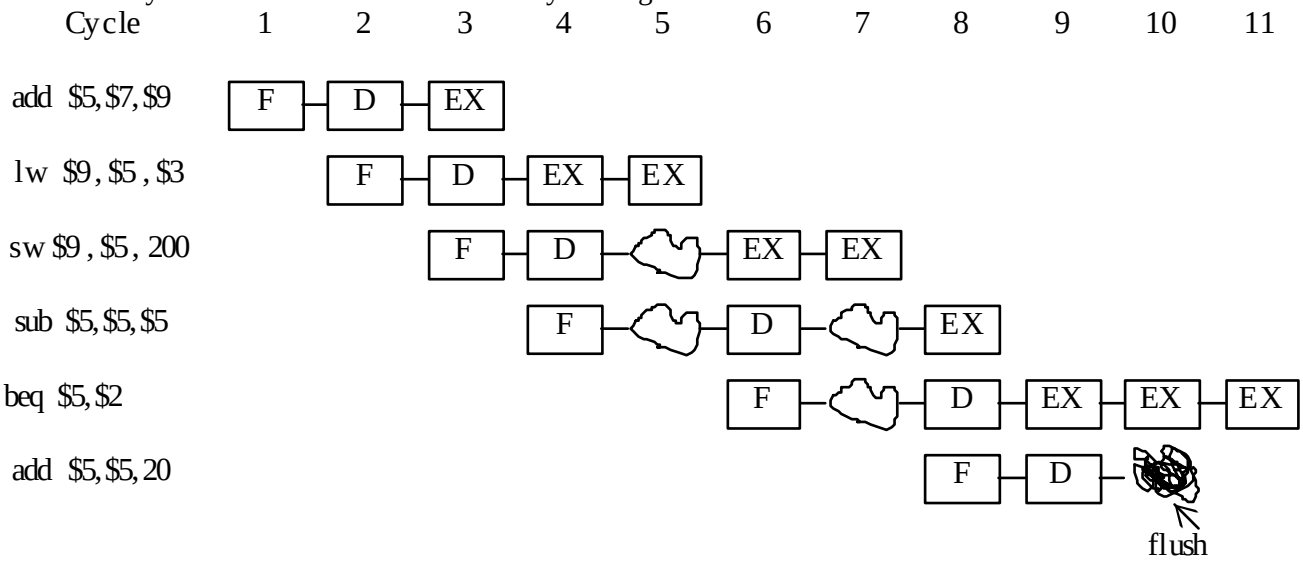
Fetch	Decode	Execute

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
lw    $9, $5, $3
sw    $9, $5, 200
sub   $5, $5, $5
beq   $5, $2
add   $5, $5, 20
```

The best 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	`lw $9`	`add $5`	
3	`sw $9`	`lw $9`	`add $5`
4	`sub $5`	`sw $9`	`lw $9`
5	`sub $5`	`sw $9`	`lw $9`
6	`beq $5`	`sub $5`	`sw $9`
7	`beq $5`	`sub $5`	`sw $9`
8	`add $5`	`beq $5`	`sub $5`
9	after 1	`add $5`	`beq $5`
10	`bubble`	`bubble`	`beq $5`
11	`bubble`	`bubble`	`beq $5`
12	target	`bubble`	`bubble`

Either way, it takes 11 cycles.