The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Test 1 Solution
Fall 2004

1. (10 points) Use the following code fragment:

```
      add   $t0, $zero, $zero
loop: lw    $t1, 0($s0)
      add   $t0, $t0, $t1
      addi  $t1, $t1, 4
      sw    $t1, 0($s0)
      addi  $s0, $s0, -4
      bne   $2, $3, loop
```

Assume that the initial value of $s0 is $a0 + 256 and that this code fragment is run on a machine with a 2-GHz clock that requires the following number of cycles for each instru ction:

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

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

ET = (# of initialization cycles +#of iterations * #of cycles/iteration) * CC

ET = (2+64*(3+2+1+3+1+2))*0.5 ns = (2+64*12)*0.5 ns = (2+768)* 0.5 ns = 770*0.5 ns = 385 ns

2. (1 point) ___Instructions____ are individual commands to a computer.

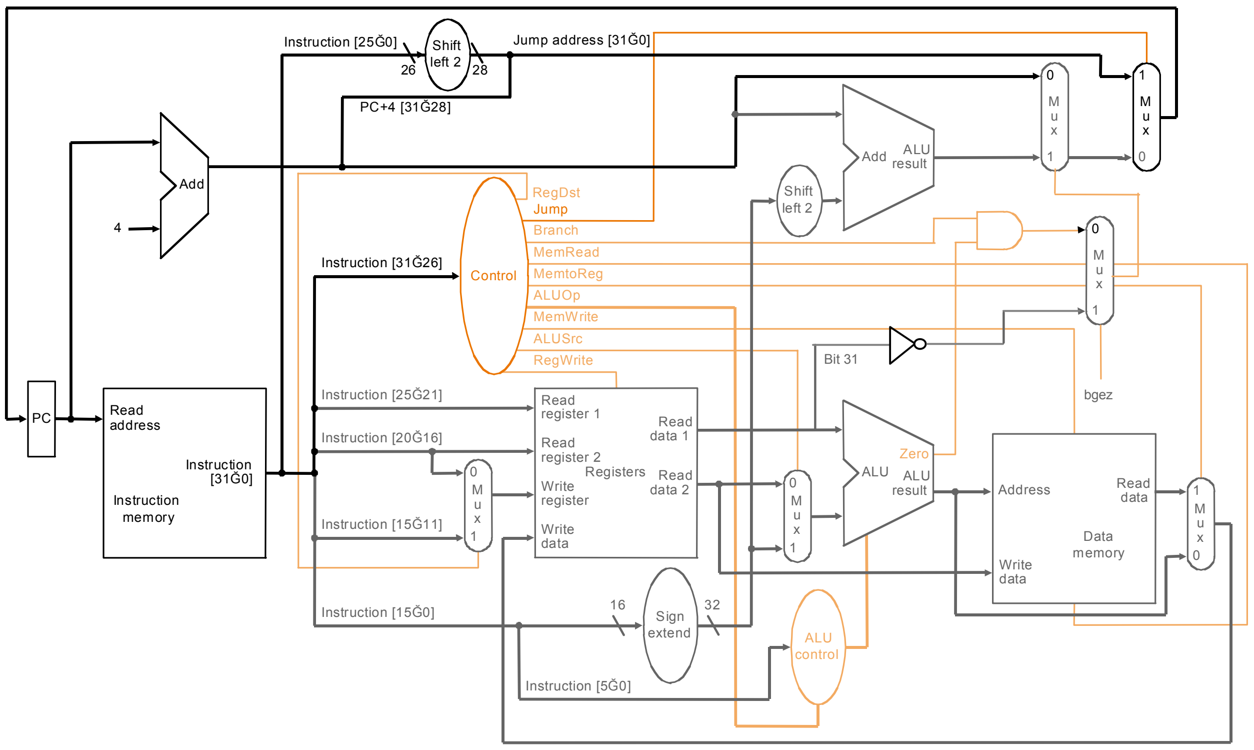
3. (1 point) The component of the processor that performs arithmetic operations is the _ALU_.

4. (1 point) The percentage of good die from the total number of die on the wafer is the __yield__.

5. (15 points) Add the instruction `bgez` (branch on greater than or equal to zero) to the single-cycle datapath shown in the figure below. The `begz` instruction is defined below. Add any necessary datapaths and control signals and show the necessary additions to the table of control signals given.

```
begz rs, label        if (rs >= 0)
                          PC ← PC + 4 + 4*offset
                      else
                          PC ← PC + 4
```

1	rs	1	offset



Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	bgez
R-format	1	0	0	1	0	0	0	1	0	0
lw	0	1	1	1	1	0	0	0	0	0
sw	d	1	d	0	0	1	0	0	0	0
beq	d	0	d	0	0	0	1	0	1	0
bgez	d	d	d	0	0	0	d	d	d	1

d-don't care

6. (1 point) _Overflow_ occurs when a number can't be represented in a computer.

7. (1 point) To specify a register choice from a 128 register register file requires __7__ bits.

8. (15 points) When designing memory systems, it becomes useful to know the frequency of memory reads versus writes and also accesses for instructions versus data. Using the following average instruction-mix information, find
   a. (5 points) the percentage of all memory accesses for instructions
   b. (5 points) the percentage of memory accesses that are writes
   c. (5 points) the percentage of all data accesses that are reads.

Instruction	Percentage
`lw`	29
`sw`	15
`add`	18
`sub`	3
`lui`	7
`beq, bne`	6
`jump`	3

`and, or`	16
`mult`	3

a. $\frac{IC}{1.44 * IC} = 69.4\%$

b. $\frac{0.15 * IC}{1.44 * IC} = 10.4\%$

c. $\frac{0.29 * IC}{0.44 * IC} = 65.9\%$

9.. (5 points) What number does the two's complement binary number represent:

$\texttt{1101 1010 0011 1111}_2$?

$-2^{15} + 2^{14} + 2^{12} + 2^{11} + 2^{9} + 2^{5} + 2^{4} + 2^{3} + 2^{2} + 2^{1} + 2^{0} = -9665_{10}$

10. (7 points) Given the bit pattern:

$\texttt{0010 1100 1010 0100 0101 0011 1100 0000}_2$

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

opcode = 001011 (`sltiu`, so the instruction is I-type and the other fields are)
rs = 00101=$5 = $a1
rt = 00100 = $4 = $a0
offset = 0101 0011 1100 0000 = 21440

```
sltiu       $a0, $a1, 21440
```

11. (18 points) Pseudoinstructions are not part of the MIPS instruction set but often appear in MIPS programs. For each pseudoinstruction in the following table, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use $at for some of the sequences. In the table, big refers to a specific number that requires 32 bits to represent and small to a number that can fit in 16 bits.

Pseudoinstruction	What it accomplishes
`beq $t1, small, L`	`if ($t1 = small) go to L`
`clear $t0`	`$t0 = 0`
`lw $t5, big($t2)`	`$t5 = Memory[$t2 + big]`

```
beq   $t1, small, L             addi   $at, $zero, small
                                beq    $at, $t1, L

clear $t0                       add    $t0, $zero, $zero

lw    $t5, big($t2)             lui    $at, big_upper
                                ori    $at, $at, big_lower
                                add    $at, $at, $t2
                                lw     $t5, 0($at)
```

12. (10 points) Consider two different implementations, P1 and P2, of the same instruction set. There are five classes of instructions (A, B, C, D, E) in the instruction set. P1 has a clock rate of 4 GHz. P2 has a clock rate of 6 GHz. The average number of cycles for each instruction class for P1 and P2 is as follows:

Class	CPI on P1	CPI on P2
A	1	2
B	2	2
C	3	2
D	4	4
E	3	4

If the number of instructions executed in a certain program is divided equally among the classes of instructions except for class A, which occurs twice as often as each of the others, how much faster is P2 than P1?

$IC_{P1} = IC_{P2}$

$CPI = \sum w_i CPI_i$ over all i classes of instructions

$w_A = 2/6$, $w_B = 1/6$, $w_C = 1/6$, $w_D = 1/6$, $w_E = 1/6$ (The sum of all the weights must equal 1.)

$$CPI_{P1} = \frac{2*1+2+3+4+3}{6} = \frac{14}{6} = 2.33$$

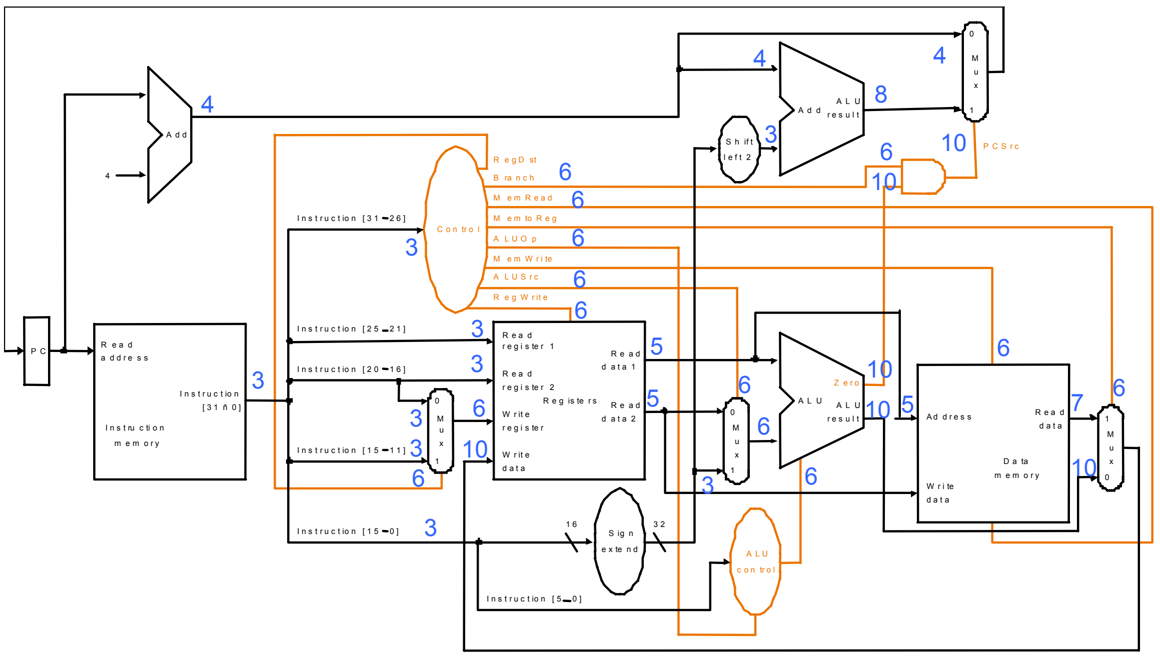
$$CPI_{P2} = \frac{2*2+2+2+4+4}{6} = \frac{16}{6} = 2.67$$

$$\frac{P_{P2}}{P_{P1}} = \frac{ET_{P1}}{ET_{P2}} = \frac{IC_{P1}*CPI_{P1}\ *CC_{P1}}{IC_{P2}*CPI_{P2}*CC_{P2}} = \frac{\frac{14}{6}*\frac{1}{4\times10^6\frac{cycles}{s}}}{\frac{16}{6}*\frac{1}{6\times10^6\frac{cycles}{s}}} = \frac{14*6*6}{16*6*4} = \frac{84}{64} = 1.3125$$

13. (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 want to make to the single-cycle datapath and control if this simplified architecture were to be used?



(b) (10 points) If the delay for the instruction memory is 3 ns, the data memory is 2 ns, the register file delay (read or write) is 2 ns, the control (not ALU control) is 3 ns, and the ALU delay is 4 ns, what are the clock cycle times 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?

`lw` is the longest for the unmodified datapath

Consider `lw`, `sw`, R-type, `beq`, `jump` for modified datapath

$CC_{\mathtt{lw}}$ = 3(fetch)+max(3, 2)(control, register file read)+2(data memory read)+2(register file write) = 3 + 3 + 2 + 2 = 10 ns

$CC_{\mathtt{sw}}$ = 3(fetch) + max(3, 2)(control, register file read) + 2(data memory write) = 3 + 3 + 2 = 8 ns

CC_{Rtype} = 3(fetch) + max(3, 2)(control, register file read) + 4(ALU) + 2(register file write) = 3 + 3 + 4 + 2 = 12 ns

$Cc_{\mathtt{beq}}$ = max((3(fetch) + max(3, 2)(control, register file read) + 4(ALU)), max(4, 3)(ADD, control) + 4 (ALU)) = max(3+3+4, 4+4) = max(10, 8) = 10 ns

$Cc_{\mathtt{jump}}$ = 3(fetch) = 3 ns

CC_{orig} = CClw= 3(fetch) + max(3,2)(control, register file read) + 4(ALU) + 2(data memory read) + 2(register file write) = 3 + 3 + 4 + 2 + 2 = 14 ns

CC_{mod} = max(CC_{lw}, CC_{sw}, CC_{Rtype}, Cc_{beq}, Cc_{jump}) = max(10, 8, 12, 10, 3) = 12 ns

For every load-store with a nonzero offset, an additional instruction must be executed. Thus, if the fraction of instructions which is a `lw` with a nonzero offset is x, the number of modified instructions is IC_{orig} (1 + x). Remembering that $CPI_{mod} = CPI_{orig} = 1$

$$\frac{P_{mod}}{P_{orig}} = \frac{ET_{orig}}{ET_{mod}} = \frac{IC_{orig} * CPI_{ori\,g} * CC_{orig}}{IC_{mod} * CPI_{mod} * CC_{mod}} = \frac{IC_{orig} * 1 * 14ns}{IC_{orig}(1+x) * 1 * 12ns} = 1$$

IC_{orig}*1*14 ns = IC_{orig}(1 + x)*1*12 ns

1.167 = 1 + x and x = 0.167

So, 16.7 % of the instructions could be `lw` with nonzero offsets and the total impact on performance would still be positive.