**COMP 230: Computer Architecture and Organization** **November 13, 2017**

# Exam 2

Instructions:

- This exam is open book and notes. However, you are not allowed to use laptops, cell phones, or other electronics.
- If you do not show your work, do not expect partial credit for incorrect answers.
- If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution.
- In all cases, clearly state any assumptions you make in your answers.

| **Name** | |
|---|---|

| **Question** | **Points Possible** | **Grade** |
|---|---|---|
| **1** | 20 | |
| **2** | 20 | |
| **3** | 20 | |
| **4** | 20 | |
| **5** | 20 | |
| **Total** | 100 | |

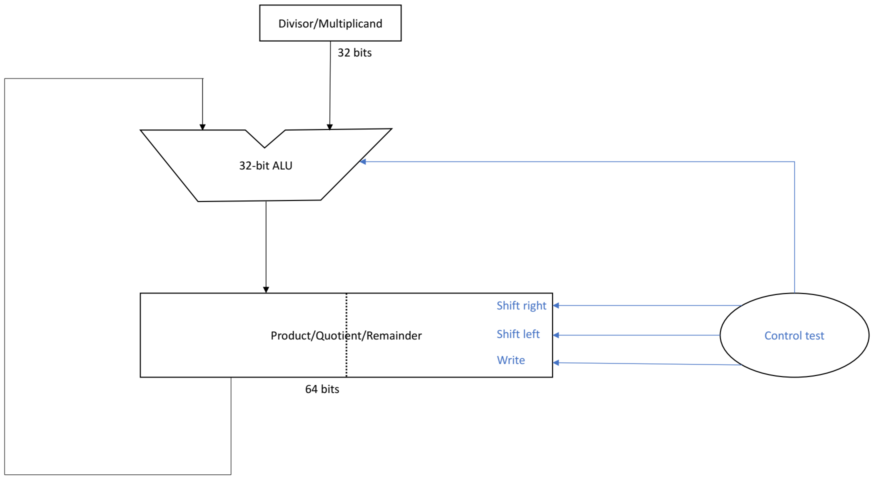


Figure 1: Hardware components for multiplication and division.

1. (a) Complete the following table to multiply two 4-bit unsigned numbers: $7 * 9$. Use the hardware shown in Figure 1.

| Iteration | Step | 8 Bits |
|---|---|---|
| 0 | initial | 0000 1001 |
| 1 | add? | 0111 1001 |
| | shift | 0011 1100 |
| 2 | add? | - - |
| | shift | 0001 1110 |
| 3 | add? | - - |
| | shift | 0000 1111 |
| 4 | add? | 0111 1111 |
| | shift | 0011 1111 |

(b) Consider the following code, where `$t0` contains 7 and `$t1` contains 9.

```
mul $t0, $t1
mflo $t2
```

If our MIPS machine operates on 4-bit words (so that we only have 4-bits to represent our integers), what does number does `$t2` contain?

Overflowed the register: 15.

2. (a) What does the bit pattern `0x7FFC1A55` represent, assuming a single-precision floating-point format?

NaN

(b) What is the purpose of the sticky bit during floating point arithmetic operations?

The sticky bit helps us round more correctly when we have numbers that are in between two numbers we can actually represent in our floating-point format. If we are exactly in between two numbers, we normally use the round-to-even scheme. The sticky bit tells us if we have a tiny bit more than that, so that we will round up.

3. Consider an 8-bit floating point format, as in the homework: 1 sign bit, two exponent bits with a bias of 1, and 5 fraction bits. How many *real* numbers are represented in this format, NOT including infinities or NaN objects?

$$\begin{aligned} 1 \text{ sign bit} &\Longrightarrow 2 \text{ possibilities} \\ 2 \text{ exponent bits} &\Longrightarrow 4 \text{ possibilities, but 1 means infinity or NaN} \\ 5 \text{ fraction bits} &\Longrightarrow 32 \text{ possibilities} \end{aligned}$$

Each of these are independent of each other, so we have $2 \times 3 \times 32 = 192$. Except we have actually represented 0 twice: $+0$ and $-0$, so really there are 191 numbers.

4. Write MIPS assembly code to for a procedure `overflow` that takes in two integers as parameters and returns 1 if the product of the two would overflow a 32-bit register, and 0 otherwise.

```
overflow:  mul   $a0,  $a1
           mfhi  $v0
           beq   $v0,  $zero,  exit
           addi  $v0,  $zero,  1
exit:      jr    $ra
```

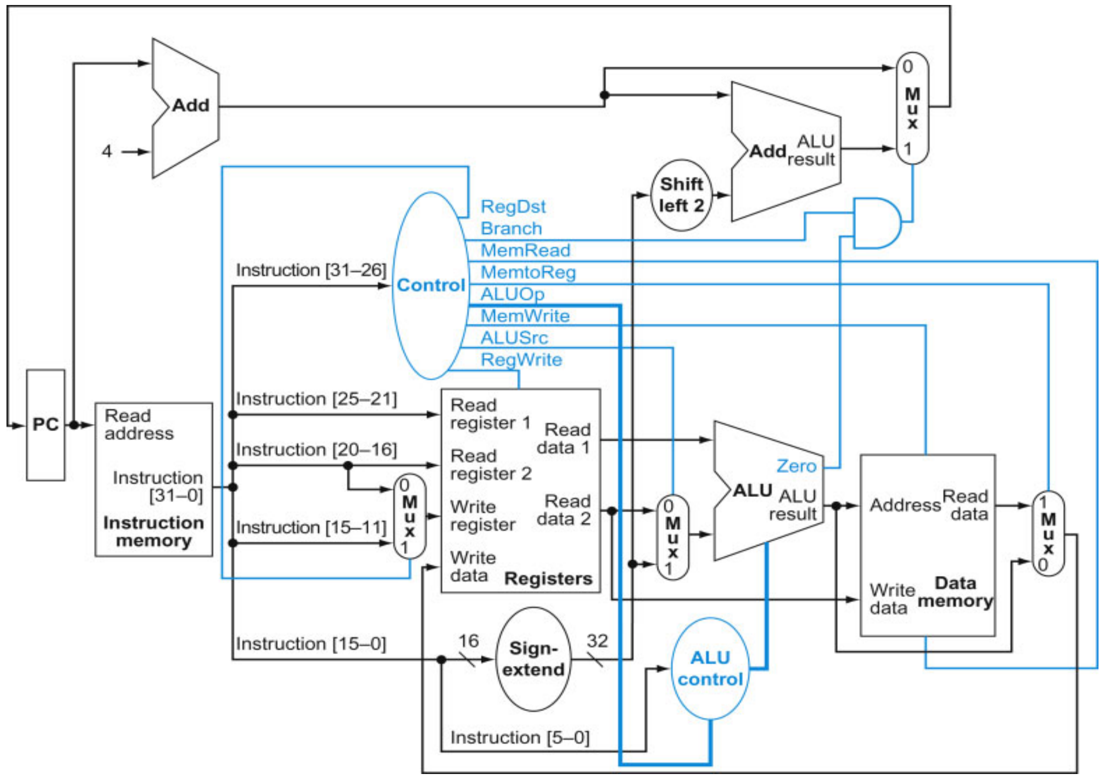


Figure 2: A simplified datapath for the MIPS architecture.

5. Consider the execution of a `lw $s3, -8($t0)` instruction using the datapath in figure 2.

   (a) Circle the inputs that would be chosen to pass through each of the multiplexors.

   From top-to-bottom, left-to-right: 0, 0, 1, 1

   (b) Which operation will the ALU perform? `add`

   (c) What 32 bits, represented in hexadecimal, would be sent to the bottom input of the top ALU?

   The last 16 bits of the instruction will represent $-8$, which is 1111 1111 1111 1000. We sign extend to 32 bits and shift this left two places, yielding 1111 1111 1111 1111 1111 1111 1110 0000. Thus the answer is `0xFFFFFFE0`.