Facultad de Informática
Universidad Complutense de Madrid

Introduction to Computers II – Test exam
May 3rd, 2023

Analog-to-Digital Converters (ADCs) are widely used in electronic applications. Physical parameters (e.g. temperature, humidity, light, etc.) do not have discrete values, but follow continuous distributions. In order for a computer to process this information, it has to be digitalized first.
An example of this kind of process can be seen in food scales, which incorporate the HX711 ADC. The weight sensor is formed by a strain gauge, whose value is read by the ADC and transformed into grams.
The behavior is as follows. The measured data (weight) is a 24-bit unsigned integer. The sensor shares the memory with the processor in order to facilitate the reading process. To do so, we have to send a control sequence (a 0 followed by a 1) to the control address 0x1000, and then read a data bit (from most to least significant) from address 0x1004. After repeating this 24 times, we would get the whole data.
The following RISC-V code is provided, with a **read_sensor** function that implements this functionality:

```
.equ ADDR_CTRL, 0x1000
.equ ADDR_DATA, 0x1004

.text
read_sensor:
        // Prologue
        //...
        ////////////////////////

        li s0, ADDR_CTRL
        li s1, ADDR_DATA
        li s2, 24                   // number of iterations

        li t0, 0                    // iteration index
        li t1, 1                    // constant 1

        li a0, 0                    // result
loop:
        beq t0, s2, end_loop
        sw zero, 0(s0)              // a control 0 is sent
        sw t1, 0(s0)                // a control 1 is sent
        slli a0, a0, 1              // a0 is left shifted one position, to make room for the new bit
        lw a1, 0(s1)                // the bit is read from the data address
        or a0, a0, a1               // the read bit is added to the number
        addi t0, t0, 1              // index is increased
        j loop                      // loop is repeated

end_loop:
        // Epilogue
        //...
        ////////////////////////
```

Answer the following questions:

1. **ASM [1pt]** Write the prologue and the epilogue of the provided function.
2. **ASM [0.5pt]** Regarding the previous question, what would change if the **read_sensor** function called another function?
3. **ASM [0.5pt]** If the provided function was called from the main program, in what register would the result be returned? Does this follow the RISC-V function call convention?
4. **CPU [1.5pt]** Assuming one iteration of the loop (8 instructions) in the pipelined RISC-V, provide the chronogram indicating the data and control hazards.
5. **CPU [0.5pt]** Can we eliminate any of the hazards by reordering the code? Which one/s?
6. **CPU [1pt]** Assuming that the whole loop is executed (with all the iterations) in the single-cycle, multicycle and pipelined RISC-V processors, how many cycles are needed to execute it in each case?
7. **CPU [1pt]** Calculate the CPI for the three processors. **NOTE**: If you did not find out the correct number of cycles in the previous question, you may assume it is 187 for the single-cycle processor, 765 for the multicycle processor and 271 for the pipelined processor.
8. **CPU [1pt]** Assuming that the clock cycle is 27.6 ns for the single-cycle processor, 9.8 ns for the multicycle processor and 10.5 ns for the pipelined processor, how much time does each processor take to execute the whole loop?
9. **MEM [1pt]** Assuming that the processor has a 1MB main memory and a 128B cache with 16B blocks, indicate the cache memory address format.
10. **MEM [1pt]** The full loop with all the iterations is executed in the RISC-V processor with the cache mentioned in the previous question. If the **beq** instruction is in address 0x50, what is the cache miss rate?
11. **MEM [1pt]** Discuss what would happen if the **beq** instruction was placed in address 0x80. Would there be more or fewer cache misses? Explain why.