Systems-I — Sample final exam questions

Note that these samples are for material from the latter part of the course. The sample and real mid-term exams are ample evidence of the questions formats for the course's earlier matierals.

1. Consider the classic *Fibonacci sequence:* $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$. That is:
   - $fib(n) = n$ *iff* $n = 0$ or $n = 1$
   - $fib(n) = fib(n-2) + fib(n-1)$ *iff* $n \geq 2$

   Using the mini-k ISA, write an assembly code program that calculates the n^{th} Fibonacci number. Assume that address $0xfe$ contains the value of n, and that the result should be stored in $0xff$.

2. Provide short answers to each of the following:
   (a) What motivated the change in basic processor design from straightforward 1- or 2-cycle designs to pipelines?
   (b) What is *virtual memory*? Name two ways in which it improves the task of programming for a given computer.
   (c) Explain the sequence of events that occurs when a key is pressed on a keyboard? That is, how does the hardware, and the program running on the processor, respond to that event and copy the byte of information generated by the key-press into RAM?

3. Assume a processor that implements an ISA with a 16-bit word size. For this processor, show the design of a *2-way set associative hardware cache with 64-bit cache lines (data only; add bits for the metadata).* Specifically, show the contents of each cache entry, including the number of bits per element of the entry. Moreover, show only the circuitry and structure required to read from this hardware cache; do not worry about writing values into it.

4. Consider the following instruction:

   ```
   TSTSET %rB
   ```

   Here, `TESTSET` is the *test-and-set* instruction, and the operand is a main memory address. This instruction loads the value from the main memory address stored in the given register, placing it into the accumulator, and then writes the value 1 into that main memory location.[1] It is essential that this task be performed **as a single instruction**.
   Show the modifications necessary to your datapath and control to add this intruction to your ISA. You do not need to show your complete datapath and control, but rather only the elements that needs to be modified to support this instruction. It should be clear, from what you show, how this instruction would be implemented by your circuitry.
   (This *atomic test-and-set* instruction, which could also be called "load-and-set" is commonly offered by ISA's to allow for the "locking" of memory locations that more than one program may try to access at the same time. Without it, strange things can happen when two programs, try to perform this operation using more than one instruction.)

[1] The value sitting in the accumulator can then be treated like the result of a comparison instruction, since it is either *set* or *clear*, and can be stored into a named register and then used as the test register on a branch instruction.