Computer Architecture Exercises

These exercises are grouped into three parts. Part I contains routine exercises to help you understand the ideas directly presented in lectures. The exercises in Part II are designed to grow and deepen your understanding of principles that underpin Computer Architecture. These exercises extend the lecture material and invite you to think about 'why' questions and about optimisations to what was lectured as well as alternatives to what was lectured. Part III are open-ended, stretching questions that you could tackle in your mini research project.

I hope you enjoy working through these questions, puzzles and research questions!

John Fawcett, July 2023

Part I

1. An original computer that was invented in Cambridge used a DRAM memory with a 1 MHz bus, meaning that the CPU could send it 1,000,000 requests per second. The dynamic memory needed to refresh every millisecond, to avoid losing the stored data. Refreshes took 1us. If the CPU made a request when the DRAM was busy doing a refresh, the request had to wait for the refresh to complete. Calculate the effective (average) access time for the memory.
2. How many 4-byte integers can we store in a byte-addressed memory with a 32-bit address? How many 4-byte integers could we store in an word-addressed memory with a 32-bit address (assume the word size is 32 bits)?
3. The complete works of William Shakespeare would use 5.5 MiB of memory so why do we need 4 GB of memory or even more in modern computers?
4. Draw a timing diagram to show how we write to a DRAM.
5. We said that a byte-addressed memory only needs a 30-bit address to access 32-bit integers. How many bits would be required to access:
    a. 64-bit long integers on the same byte-addressed memory?
    b. 32-bit and 64-bit integers on a word-addressed memory (assuming the word size is 32 bits)?
6. Give two advantages and two disadvantages of having more registers inside a CPU.
7. The lecture used 7 bits for the opcode. What limit does that impose? Why not devote more than 7 bits to the opcode?
8. The lectures laid out instructions as 32-bit values with the opcode at the most-significant end. Does the order of the opcode and the operands matter? Why (not)?
9. The lectures did not provide an encoding for the HALT instruction. Provide one.
10. The lectures allocated opcode for some instructions that we didn't use. What do SHL (shift left), ASR (arithmetic shift right) and LSR (logical shift right) do? What about AND, OR and NOT?
11. Provide suitable encodings for the following two instructions, which are found in some CPUs.
    a. BIC: bit-clear works as follows. X = Y BIC Z. X and Y are register numbers; Z can be a register number or an immediate. BIC sets X to a copy of Y but zeros out any bits where there is a 1 at the corresponding position in Z. i.e. X = Y AND (NOT(Z))
    b. FACT: factorial. X = FACT(Y) sets Y to X! (the factorial of X). Y can be a register number or an immediate.
12. The instruction format we used in lectures needed two instructions to put an arbitrary value into a register: we subtracted a register from itself to set it to zero, then we added on the

value we wanted to store. Extend the Instruction Set Architecture to make this possible in a single instruction.

13. Work out the machine code encoding for the following instructions:
    a. SHL R0 R1 #19
    b. OR R9 R0 R9
    c. JMP #65537
    d. JMP R1 #65537
14. Does any instruction have more than one valid encoding?
15. Does any encoding represent more than one instruction?
16. What could you do if you are writing a program and need to jump to somewhere that is too far away for the immediate in a JMP instruction to represent?
17. Suppose the instruction cache (I-cache) can hold 64 instructions and replaces the least recently used instructions when it has to load more of the program from memory.
    a. Would it provide any benefit for loops with more than 64 instructions?
    b. What benefit would it provide for the following program with nested loops?

```
BEGIN_1:
        20 instructions
        BEGIN_2:
                39 instructions
                BNE BEGIN_2
        9 instructions
        BNE BEGIN_1
```

18. Write a program in assembly code to be stored in memory starting at address 1024. The program should add up the 32-bit numbers in its own encoding and leave the result in R0.
19. Write a program in assembly code that scans through memory starting at the address held in R0. It should count the number of **integers** it has to skip over before it finds the first zero, and leave that count in R0.
20. Write a program in assembly code that scans through memory starting at the address held in R0. It should count the number of **bytes** it has to skip over before it finds the first zero byte, and leave that count in R0.

Part II

1. The lectures showed the “anatomy of a simple computer” with separate memory buses and Peripheral I/O buses. We didn’t talk much about the Peripheral I/O bus in lectures. What purpose does it serve?
2. Look up and describe how DRAM memory circuits work. What are the *worldline* and *bitline* for? How many transistors are required per bit of data stored?
3. Look up and describe how SRAM memory circuits work. How many transistors are required per bit of data stored? Why is it faster than DRAM? Why do SRAMs use more power than DRAMs?
4. Some CPUs allow operands to be in memory: we do not need to load values into registers before we start using them. What advantage does this offer? What disadvantages?
5. An alternative to 2’s complement format is sign-and-magnitude representation: the most-significant bit indicates whether a represented value is positive (bit=0) or negative (bit=1). Find two advantages of 2’s complement over this format.
6. We saw “little endian” format to represent 32-bit values in a byte-addressed memory. How is “big endian” different? Is it better? Worse?

Part III

1. The lectures showed the “anatomy of a simple computer” with separate memory buses and Peripheral I/O buses. Why separate these? Is there ever a reason to merge them into one? Is there ever a reason to split them further (e.g. two/more memory buses or two/more I/O buses)?
2. I left the representation of floating point numbers as an interesting exercise for a mini research project. IEEE754 is an international standard and specifies (among many things) one way to represent floating point numbers. Find out how it works and, besides numbers, what other “mathematical objects” it can represent. What happens in IEEE754 if you ask the CPU to do something that is not mathematically defined, such as to calculate the square root of -2 or to find the arcsin of 14?
3. We did not consider a data cache in the lectures.
    a. Why might it be more difficult to build a data cache (D-cache) than an I-cache?
    b. Under what circumstances would a D-cache speed up a program?
    c. How could you decide when to replace an entry in a D-cache?