John Carroll
Jcc0044
902521946

HOMEWORK 1

§1.3 (Data Representation) – Binary representation; bits, LSB, MSB, binary addition. Storage sizes. Hexadecimal representation. Two's complement representation. Conversion of integers between representations. Maximum and minimum representable values. ASCII and Unicode representation of character strings.

§2.1 (General Concepts in Microcomputer Design) – CPU: registers, clock, control unit, ALU, memory storage unit. Data, I/O, control, and address buses. Clock cycles. Instruction execution cycle: instruction pointer/program counter; fetch-decode-execute (FDX) cycle, including fetching operands and storing output operands. Procedure for reading from memory; cache memory. Processes; tasks; multitasking; scheduler; task switching; preemptive vs. non-preemptive multitasking.

Supplemental – Executables (.exe files) vs. DLLs vs. static-link libraries; PE file format: text, data, idata sections; loaders; relationship between the text and data sections of a PE file and the code and data segments of a process

Directions: *This assignment is due by* ***2:00 p.m. on Friday, September 5****. Submit your answers as a PDF in Canvas; see the syllabus for detailed instructions and a description of the late policy. Paper submissions will* ***not*** *be accepted.*

Scoring (45 total points): *#1–5: 3 points each • #6–10: 4 points each • #11–12: 5 points each*

1. Consider the 8-bit binary number 00110001.

   (a) What are the values of bits 2 through 4? It would be 1,0,0, or just 1.

   (b) What is the value of the MSB? It is 0.

   (c) What is the hexadecimal representation of this number? It is 31h.

2. Add the following unsigned binary numbers, showing carries if applicable:

Carries: 00001110	Check
00000101	5
+ 11100111	+ 231
11101100	236

3. What is the decimal representation of each of the following 8-bit *signed* (two's complement) binary integers? *Show your work.*

(a)	(b)	(c)
10000100	01000000	11111111
01111011		00000000
01111100		00000001
Ans: -124	Ans: 64	Ans: -1

4. What is the 8-bit *signed* (two's-complement) *binary* representation of each of the following decimal integers? Show your work.

(a)	(b)	(c)
–42	42	–128
42		128
00101010	00101010	10000000
00101001		01111111
11010110		10000000
Ans:11010110	00101010	10000000

5. If a processor is clocked at 3.2 GHz, what is the length of one clock cycle, in nanoseconds? *Show your work.*

1 nanosecond = 10^{-9} seconds

$(1)/(3.2*10^{-9}) = x*10^{-9}$

$10^{9}*((1)/(3.2*10^{9})) = (x*10^{-9})*10^{9}$

x = 1/3.2 = 0.3125ns

6. What is the decimal representation of each of the following 8-bit *signed* (two's complement) hexadecimal integers? *Show your work.*

(a)	(b)
9Ch	31h
10011100	00110001
01100011	00110001
01100100	
Ans: -100	Ans: 49

7. What is the 8-bit *signed* (two's-complement) *hexadecimal* representation of each of the following decimal integers? Show your work.

(a)	(b)
–47	47
00101111	00101111
11010000	
11010001	
Ans: D1h	Ans: 2Fh

8. Suppose a computer's memory contains the following four bytes: 4Eh 6Fh 21h 00h.

(a) Assume these bytes represent an ASCII string. What string do they represent?
Ans: "No!"

(b) Would the UTF-8 encoding of this string be the same as, or different from, the ASCII encoding? Explain.
Ans: Every ASCII string is also a valid UTF-8 representation of the same string due backward compatibility.

9. What is the range of values that can be represented by

(a) a 73-bit unsigned integer?
Ans: Range is [0 to 2^{73}-1], which is also [0 to 9,444,732,965,739,290,427,392]

(b) a 73-bit signed integer?
Ans: Range is [-2^{72} to 2^{72}-1]

10. Can the number –10 be represented as a 4-bit signed integer? If so, show its 4-bit signed (two's complement) representation. If not, explain why; describe how many bits are required, and show its representation using that many bits.
Ans: No, this is because the lowest number that can be represented in a 4-bit signed representation is -8 and the largest is 7. To properly represent it would need 5 bits and be represented as: 10110.

11. Although it has virtually no practical applications, it is possible to represent numbers in base –2; these are called *negabinary* numbers. Negabinary numbers use the digits 0 and 1, but they are converted to decimal by multiplying the digits by powers of –2. For example,

$$\begin{aligned} 00111001_{-2} &= 1 \times (-2)^5 + 1 \times (-2)^4 + 1 \times (-2)^3 + 1 \times (-2)^0 \\ &= -32 + 16 + -8 + 1 \\ &= -23. \end{aligned}$$

What is the range of values that can be represented by a 16-bit unsigned negabinary integer? Explain your answer.

(Hint: Try solving the problem for 4-bit unsigned negabinary integers first.)

Solving for 4-bit unsigned negabinary integers,

$$\begin{aligned} 1111_{-2} &= 1 \times (-2)^3 + 1 \times (-2)^2 + 1 \times (-2)^1 + 1 \times (-2)^0 \\ &= -8 + 4 + -2 + 1 \end{aligned}$$

$= -5$
The range for 4-bit is [-10 to 5]

Solving for 16-bit unsigned negabinary integers,
$1111111111111111_{-2} = 1 \times (-2)^{15} + 1 \times (-2)^{14} + 1 \times (-2)^{13} + 1 \times (-2)^{12} \times 1 \times (-2)^{11} + 1 \times (-2)^{10} + 1 \times (-2)^{9} + 1 \times (-2)^{8} \times 1 \times (-2)^{7} + 1 \times (-2)^{6} + 1 \times (-2)^{5} + 1 \times (-2)^{4} \times 1 \times (-2)^{3} + 1 \times (-2)^{2} + 1 \times (-2)^{1} + 1 \times (-2)^{0}$
$= -32768 + 16384 + -8192 + 4096 + -2048 + 1024 + -512 + 256 + -128 + 64 + -32 + 16 + -8 + 4 + -2 + 1$

From adding all of the negative numbers and then adding the positives together, but separately from each other we get the range [-43690 to 21845].

12. Consider the fetch-decode-execute cycle. Suppose a computer's instruction set is designed such that every instruction will either *fetch operands,* or *store output operands,* but never both. Joe Bob decides that having separate *fetch operands* and *store output operand* steps is wasteful, since both of them access memory and it will never be necessary to perform both for the same instruction. So, he wants to get rid of the *store output operand* step; instead, he'll replace the *fetch operands* step with a single *fetch-operands-or-store-output-operand* step. This means the FDX cycle will look like this:

```
while (true) {
        Fetch next instruction
        Advance the instruction pointer
        Decode the instruction
        If memory operands involved, fetch operands or store output operand (Joe's step)
        Execute the instruction
}
```

Is there a problem with his design, or will this work? Explain.

Ans: The normal FDX cycle is below:

Fetch:
The control unit fetches the next instruction from the instruction queue and increments the instruction pointer (IP). The IP is also known as the program counter

Decode:
The control unit decodes the instruction's function to determine what the instruction will do. The instruction's input operands are passed to the ALU, and signals are sent to the ALU indicating the operation to be performed.

Fetch operands:
If the instruction uses an input operand located in memory, the control unit uses a read operation to retrieve the operand and copy it into internal registers. Internal registers are not visible to user programs.

Execute:
The ALU executes the instruction using the named registers and internal registers as operands and sends the output to named registers and/or memory. The ALU updates status flags providing information about the processor state.

Store output operand:
If the output operand is in memory, the control unit uses a write operation to store the data.

I do not think this design should work well. An instruction will not have to write to memory prior to executing. Generally speaking, it will need to write to memory after executing and that is where this design is lacking – it lacks an option to write to memory by memory operand after execution of the instruction.