Brandon Smith   Winter 2018
Professor: Joseph Vibihal

COMP 273 Assignment 1

Question 1: Given 8 random bits forming a byte, can you determine whether the byte represents an integer or a character? Why?

Answer: No it is not possible to determine whether or not the byte is a character or an integer. This is because a byte is just an 8 digit binary number. If we wanted to differentiate between an integer and a character for any given byte we would have to reserve one bit of that byte to represent the state of the byte, effectively cutting our range of possible characters in half (256 down to 128). We, as programmers, keep track of what's a character and what's an integer by remembering where we address the data.

Question 2: Assume we have CPU instructions that look like this:

load register, address
save register, address

Where the instruction **load** saves the data pointed to by the address into the register and the instruction **save** writes the data from the register into the location pointed to by address.

Assume at address 52 is the beginning of a string "ABC" and at address 1 is the printer.

Write pseudo-C code that uses the above CPU instructions to print the string to the printer (move each byte to address 1). You can use variables in place of register and address. You can assume the string ends with a NULL.

Answer: **C code so all numbers can easily be written as decimal numbers and the compiler will handle conversions to binary**

Effect: print a string from memory
Input: N/A
Output: printer's representation of the data

```
pointer_to_string ⟵ 52
pointer_to_printer ⟵ 1
while(pointer_to_string does not point to NULL) do {
    load data, pointer_to_string
    save data, pointer_to_printer
    pointer_to_string ⟵ pointer_to_string + 1
}
```

Question 3: Perform the following conversion:

a. 1023_{10} to Binary to Octal
b. 10110110_2 to Decimal to Hex

Answer:
a.

$$\frac{1023_{10}}{2} = 511_{10}\ R\ 1$$
$$\frac{511_{10}}{2} = 255_{10}\ R\ 1$$
$$\frac{255_{10}}{2} = 127_{10}\ R\ 1$$
$$\frac{127_{10}}{2} = 63_{10}\ R\ 1$$
$$\frac{63_{10}}{2} = 31_{10}\ R\ 1$$
$$\frac{31_{10}}{2} = 15_{10}\ R\ 1$$
$$\frac{15_{10}}{2} = 7_{10}\ R\ 1$$
$$\frac{7_{10}}{2} = 3_{10}\ R\ 1$$

$$\frac{3_{10}}{2} = 1_{10}\ R\ 1$$
$$\frac{1_{10}}{2} = 0_{10}\ R\ 1$$

So by the method of remainders (build the number by the remainders going from bottom to top) $1023_{10} = 1111111111_2$.

1111111111_2 can be split into segments of 3 digits and to convert to octal you must simply convert each group of 3 digits. We must do this from right to left so as to avoid any problems with leading zeroes. $111_2 = 7_8$, etc...

$$001\ 111\ 111\ 111_2 = 1\ 7\ 7\ 7_8$$

So $1023_{10} = 1111111111_2 = 1777_8$.

b.

$$10110110_2 = 1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$$
$$10110110_2 = 182_{10}$$

The above is an example of the summation rule which all numbers of any base follow. Now perform the method of remainders on the decimal number to convert to Hexadecimal.

$$\frac{182_{10}}{16} = 11_{10}\ R\ 6$$
$$\frac{11_{10}}{16} = 0_{10}\ R\ B$$

So $10110110_2 = 182_{10} = B6_{16}$.

Question 4: Perform the following binary operation assuming our resultant has a maximum of 8 bits:
$00110110_2 + 01111001_2 - 00001100_2$
Was there an overflow, a signed overflow, or no overflow?

Answer: **Assume negative numbers are being represented by two's compliment. NOTE that if a number has an exponent on it in the addition that means there was a 1 carried from the right.**

$$\begin{array}{cccccccccc} & 0^1 & 0^1 & 1^1 & 1 & 0 & 1 & 1 & 0_2 \\ + & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1_2 \\ \hline & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1_2 \end{array}$$

As of this point, there has been a signed overflow because under the two's complement method of implementing negative numbers the second half of the binary numbers available (1 in the leftmost bit) are what are used to represent negative numbers. Hence, we have just added two positive numbers and resulted in a negative number. However there has not been an arithmetic overflow (carry-out).

Now determine the negative of 00001100_2 using two's complement.

$$\begin{array}{ccccccccc} & 1 & 1 & 1 & 1 & 0 & 0^1 & 1^1 & 1_2 \\ + & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1_2 \\ \hline & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0_2 \end{array}$$

No arithmetic overflow (carry-out) in this procedure. Finally add the sum of the first two numbers with the negative of the third to reach the final answer.

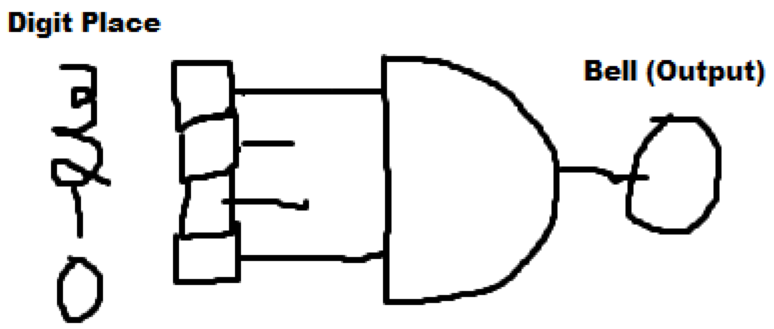
$$\begin{array}{ccccccccc} & 1^1 & 0^1 & 1^1 & 0^1 & 1^1 & 1 & 1 & 1_2 \\ + & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0_2 \\ \hline \underline{1} & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1_2 \end{array}$$

In this final step there was an arithmetic overflow but there was no signed overflow. The final answer is

$$10100011_2$$

and there was overall a signed overflow and an arithmetic overflow for the entire process.

Question 5: Draw, on paper, the circuit for the following problem: Given a 4-bit absolute value integer number as input, the circuit has a single output wire that rings a bell when it is set to one. The output turns to one when the number is odd and greater than seven.



Answer: