

# Binary Conversion; Propositional Logic

## CS 350, Lab 3 Due Fri Feb 10 (2400 hrs)

### A. Why?

- Octal and hexadecimal numbers are useful for abbreviating long bitstrings.
- Propositional logic and truth tables are the basic ways to represent truth functions.

### B. Outcomes

At the end of today, you should:

- Have written a program that converts from decimal to binary and hexadecimal.
- Be able to give the truth table for a logical expression.
- Be able to write the DNF (Disjunctive Normal Form, a.k.a. Sum of Products) expression for a truth table.

### C. Problems [100 points total]

#### Part 1 [40 points]

1. [10 pts] Write out a truth table for  $(X + Y) Z + (\neg X + Y) \neg Z$ .
2. [10 pts] Write out a truth table for  $(U + V)(\neg V + W) + \neg W(\neg Z + \neg U)$
3. [10 pts] Write out a truth table for  $((X \rightarrow Y) \rightarrow Z) \leftrightarrow (X \rightarrow (Y \rightarrow Z))$ . Is it a tautology?
4. [10 pts] The truth table shown to the right describes Z as a function of U, V, X, and Y. Write a full DNF expression for Z. (Full DNF: Don't simplify it)

U	V	X	Y	Z
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

#### Part 2 [60 points]

- For this problem, write a C program that reads in an integer, calculates and prints its 2's complement bitstring, and then calculates and prints the hex string for the bitstring. It should repeat this process until the user enters zero.
- For the 2's complement representation, use the minimum number of bits (don't forget to include the sign bit). E.g.,  $1_{10}$  translates to  $01_2$ , and  $-1_{10}$  translates to  $11_2$ .
- For the hex representation, you may need to sign-extend the bitstring to get a string whose length is a multiple of 4. E.g., once  $-1_{10}$  is translated to  $11_2$ , you extend this to  $1111_2$ , which translates to  $F_{16}$ .
- If your input is out-of-range, say so and skip its processing. (If you have 32-bit integers, you'll probably not be able to handle  $-2^{31}$  and possibly  $-2^{31} + 1$ .)

**Point Breakdown [60 points total]**

- [5 points] Prompt for an integer, repeat the below until the integer is 0.
- Convert the integer to its 2's complement representation:
  - [5 points] Detect values that are out-of-range for you; say they are and skip them.
  - [5 points] Handle the translation of negative integers.
  - [15 points] Handle the translation of positive integers.
  - [10 points] Be sure to get the bitstring to read left-to-right.
  - [3 points] Don't forget the sign bit!
  - [2 points] Print the bitstring.
- Convert the bitstring to hex.
  - [10 points] If the length of the bitstring isn't a multiple of 4, handle the leftmost hex digit by sign-extending the bitstring.
  - [8 points] Handle all the other hex digits.
  - [2 points] Print the hex string.

**Sample Output**

Here's an example of some possible sample output (yours doesn't have to be formatted exactly like this).

```

Enter an integer (0 to quit): 1
Value: 1 Binary: 01 Hex: 1
Another integer (0 to quit): -1
Value: -1 Binary: 11 Hex: F
Another integer (0 to quit): 2147483647
Value: 2147483647 Binary: 01111111111111111111111111111111 Hex: 7FFFFFFF
Another integer (0 to quit): 2147483648
Value: -2147483648 Sorry, -2147483648 is out of range
Another integer (0 to quit): -2147483647
Value: -2147483647 Binary: 10000000000000000000000000000001 Hex: 80000001
Another integer (0 to quit): -2147483648
Value: -2147483648 Sorry, -2147483648 is out of range
Another integer (0 to quit): 255
Value: 255 Binary: 01111111 Hex: 0FF
Another integer (0 to quit): 256
Value: 256 Binary: 0100000000 Hex: 100
Another integer (0 to quit): -255
Value: -255 Binary: 100000001 Hex: F01
Another integer (0 to quit): -256
Value: -256 Binary: 1100000000 Hex: F00
Another integer (0 to quit): 17
Value: 17 Binary: 010001 Hex: 11
Another integer (0 to quit): -17
Value: -17 Binary: 101111 Hex: EF
Another integer (0 to quit): 0

```

**Extra Credit**

- [5 points] When you print the binary and hex strings, print a space between each group of 4 characters (as seen from right-to-left.)