CS 240 Lab 2
Introduction to Linux and Binary Representation

- **Git/CodeTub**
- **Review of Digital Logic**
- **Binary and Hex Numbers**
- **Exclusive Or**
- **Bit Puzzles**

Review of Digital Logic

Truth tables specify the output for all the given input combinations of a function.

An input combination can be expressed by ANDing together the inputs (each input or its' complement is used in the expression, depending upon which combination is being expressed)

A function can then be expressed as a **sum-of-products** by ORing together the input combinations which make the function true.

A	B	A'B'	A'B	A'B' + A'B
0	0	1	0	1
0	1	0	1	1
1	0	0	0	0
1	1	0	0	0

A	B	A'	A'B	A' B'	A'+A'B+A'B'
0	0	1	0	1	1
0	1	1	0	0	1
1	0	0	0	0	0
1	1	0	0	0	0

F = A'B' + A'B Q = A' + A'B + A'B'

F and Q are equivalent (produce the same function) when they have the same truth table.

Equivalency can also be proved using the Boolean identities.

Universal Gates

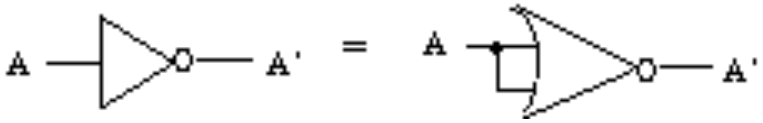
Any Boolean function can be constructed with only NOT, AND, and OR gates

But also with either only NAND or only NOR gates = **universal gates**

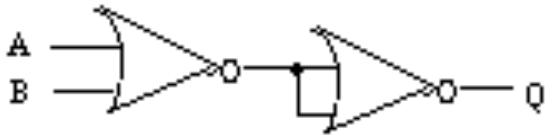
DeMorgan's Law shows how to make **AND** from NOR (and vice-versa)

AB = (A' + B')' (**AND** from NOR)
A + B = (A'B')' (**OR** from NAND)

NOT from a NOR



OR from a NOR



To implement a function using only NOR gates:

- apply DeMorgan's Law to each AND in the expression until all ANDs are converted to NORs
- use a NOR gate for any NOT gates, as well.
- remove any redundant gates (NOT NOT, may remove both)

Implementing the circuit using only NAND gates is similar.

Binary and Hexadecimal Numbers

Hex	Binary			
	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

Binary can be converted to decimal using positional representation of powers of 2:

$$0111_2 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0, \quad \text{result} = 7_{10}$$

Decimal can be also be converted to binary by finding the largest power of 2 which fits, subtract, and repeat with the remainders until remainder is 0 (assigning 1 to the positions where a power of 2 is used):

$$6_{10} = 6 - 2^2 = 2 - 2^1 = 0, \quad \text{result} = 0110_2$$

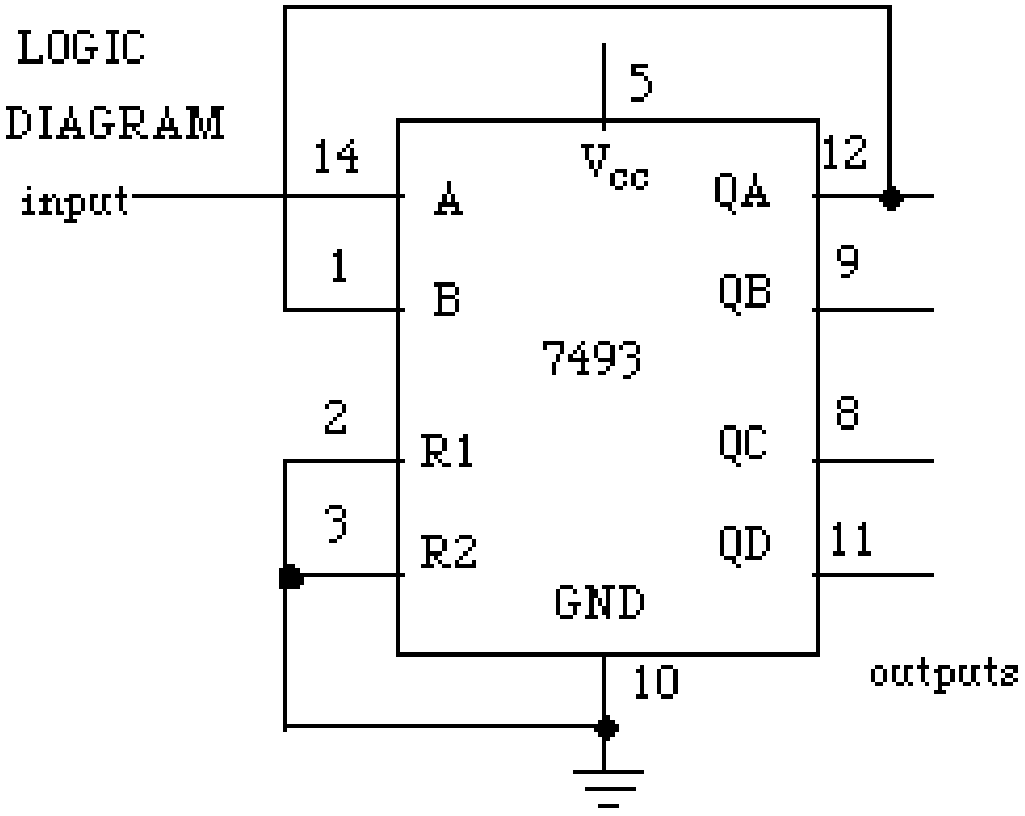
Hex can be converted to binary and vice versa by grouping into 4 bits.

$11110101_2 = F5_{16}$ $37_{16} = 00110111_2$

Logic Diagrams

Not the same as pin-outs! Show information about the logical operation of the device.

Inputs on left side of diagram
Outputs on right
Voltage shown on top
Ground shown on bottom



Exclusive Or
Useful for comparisons

A **parity bit** is an extra bit of information which is sent when data is transmitted, to check for errors in transmission. For a given set of bits, the number of bits whose value is 1 is counted. The parity bit is an extra bit which is also sent with the original data. The party bit is set to 0 or 1 to make the total number of 1 bits even.

A	B	C	P_{even}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Bit Puzzles

Example:

```
/* isPower2 returns 1 is x is a power of 2, and 0 otherwise
   isPower2(5) = 0, isPower2(8) = 1, isPower2(0) = 0

    No negative value is a power of 2

    Legal operations: ! ~ & ^ | + << >>

    Max operations: 20

    Rating: 4
*/

int isPower2(int x) {
  return 2
}
```

You must write C code to return the correct value for a given input

Constants ,must not be larger than 0xFF (decimal 256)

You may not use conditionals or loops

Tips

Although integers are 32-bit values in this program, assume a smaller number of bits in your handwritten examples to make your binary numbers easier to work with

Handwrite some specific binary values and manipulate them with boolean operators.

Here are some simple manipulations and tips which may help you find a solution:

- Complement the number
- Add and/or subtract 1
- Mask (bitwise AND with a mask value to isolate bits)
- Shift left and then right again (mo)
- Use Exclusive OR to compare values
- Bitwise AND a general solution with a special case (such as 0)
- !(0) = 1, but !(any other number) = 0