INTRODUCTION TO COMPUTING
TOPIC 6: CIRCUITS

PAUL L. BAILEY

1. Circuits

1.1. **What is a Circuit?** A circuit is a collection of connected *electrical components*, such as wires, voltage sources, resistors, transistors, and so forth. Electricity flows through the circuit like water through an irrigation system.

The words *voltage* and *current* have precise scientific meanings; for our purposes, with our irrigation system analogy, we may think of these words as follows:

- Voltage: the force behind the flow of electricity
- Current: the rate of flow of electricity

The electrical components control the voltage and current on each wire.

1.2. **Physical Circuits.** Physically, a circuit may be constructed in several ways.

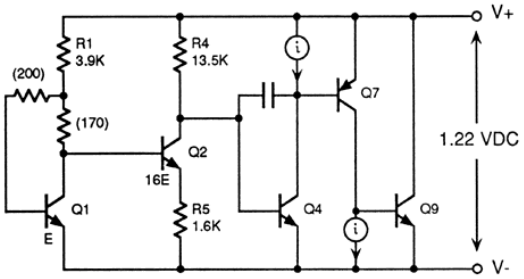
- The circuit may be a loose tangle of wires and components.
- the circuit may be placed on a piece of plastic, called a circuit board. The wires are thin strips of metal embedded in the board, and the components such as resistors and transistors are soldered onto the board.
- The circuit may be compressed into a single microchip, uses layers of a semiconductor such as silicon. Billions of components may be etched into one chip.

In any case, the ideas behind the design are similar; it is how the components connect with each other logically which determines the behavior of the circuit.

1.3. **Circuit Diagrams.** Circuit diagrams describe the circuit by indicating which components are connected together in the circuit. The diagram does not dictate how the circuit is realized, but it does determine how it will behave.

Circuits may be connected together to form larger circuits; this pieces of the larger circuit may be called *subcircuits*.

Here is an example of a circuit diagram for a subcircuit known as a "band gap reference".



Date: March 11, 2008.

1.4. **Electrical Components.** The electrical components of a circuit have standardized symbols which represent them. The following components play a role in our later discussion.

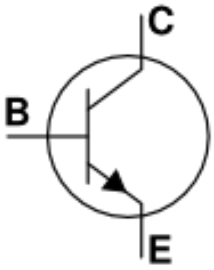
Voltage **Ground** **Resistor** **Transistor**

A *voltage source* create voltage. The electrons in the wires are attracted to the + node of the voltage source; the traditionally, current indicates the "flow" of positive charge, away from the + node.

A *ground* is a return path for the current; any wire connected directly to a ground has zero voltage.

A *resistor* slows down the current.

A *transistor* acts as a switch for voltage. Each transistor has three connections, known as the *base*, the *collector*, and the *emitter*.



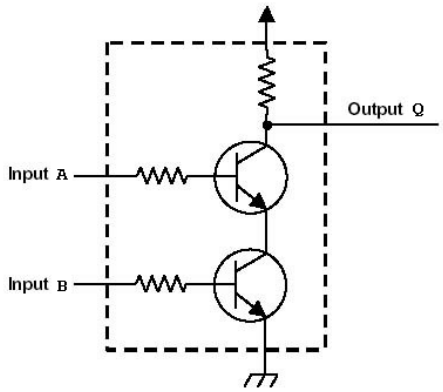
A small current applied to the base allows the large current to flow through the transistor from the collector to the emitter.

1.5. **Digital Circuits.** In digital circuits, the presence or absence of voltage translates to binary digits (bits). Only two positive voltages exist on a wire in the circuit. For example, perhaps 5 volts denotes "on" and 0 volts denotes "off". The OFF state represents the binary digit 0, and the ON state represents the binary digit 1.

```
zero volts:  0V = OFF = binary 0 = logical FALSE
five volts: +5V = ON  = binary 1 = logical TRUE
```

A gate is a digital circuit which detects input voltage (OFF or ON) to control output voltage (OFF or ON). Gates are used as subcircuits of digital circuits to perform standard logical operations. Each logical operator has a corresponding gate represented by its own symbol.

1.6. **Construction of a NAND Circuit.** The following circuit detects the voltage on `A` and `B`, and produces voltage on `Q` exactly when there is no voltage on either `A` or on `B`. We say that `Q=1` if and only if `A=0 AND B=0`.



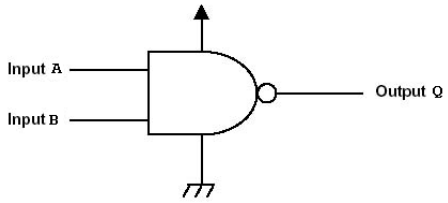
The resistors on the inputs limit the base-emitter current to just enough to turn the transistors on. The top transistor is on if and only if `A=1`, and the bottom transistor is on if and only if `B=1`.

If both transistors are on, the output will be connected to ground at 0V, dropping the voltage on the output, thus setting `Q=0`. If either transistor is off, the output is connected through the resistor to +5V, so `Q=1`.

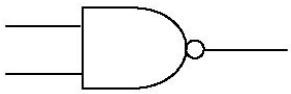
Thus this circuit produces the effect of logical NAND:

```
A NAND B = NOT(A AND B) = NOT(A) OR NOT(B).
```

The entire circuit is represented by the following symbol:



Since all digital circuits have the same +5V and 0V power supply connections, we can eliminate them from the symbol:



The circuit represented by this symbol is known as a NAND gate.

2. Logical Gates

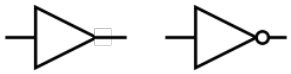
Each of the primarily logical operators is physically realized by a corresponding *logical gate*. A logical gate detects voltage on one or two inputs, and allows voltage on one output according to the logical operation.

Logical gates are digital circuits; the voltages on the connecting wires are exactly one of two possible values; these values are interpreted as the binary digits 0 or 1, corresponding to logical FALSE or TRUE. The gate performs an operation on the input values in the manner dictated by the logical operation to which it corresponds.

Each type of logical gate has a standardized symbol representing the subcircuit which implements the gate in a circuit diagram. We now list these symbols, and show the truth tables which describe the behavior of each gate.

2.1. **Unary Gates.** A unary gate takes one input (denoted `A`) and produces one output (denoted `Q`). The unary gates are called YES and NOT. A YES gate is logical identity, and NOT gate reverses the input.

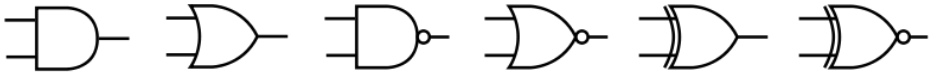
YES **NOT**



A	YES	NOT
0	0	1
1	1	0

2.2. **Binary Gates.** A binary gate takes two inputs (denoted `A` and `B`) and produces one output (denoted `Q`). The binary gates are called AND, OR, NAND, NOR, XOR, and XNOR (XNOR is logical IFF).

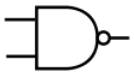
AND **OR** **NAND** **NOR** **XOR** **XNOR**



A	B	AND	OR	NAND	NOR	XOR	XNOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

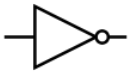
2.3. **Gate Construction.** Each of the logical gates can be constructed from the NAND gate. We review each gate symbol and its truth table, and we show how to construct it from NAND gates or previously constructed gates.

2.4. **NAND Gate.**

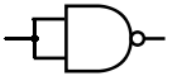


A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

2.5. **NOT Gate.**

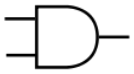


A	Q
0	1
1	0

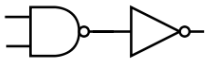


NOT A = A NAND A

2.6. **AND Gate.**

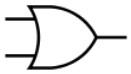


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

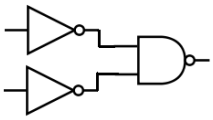


A AND B = NOT(A NAND B)

2.7. **OR Gate.**

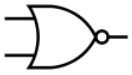


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

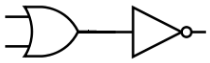


A OR B = NOT(A) NAND NOT(B)

2.8. **NOR Gate.**



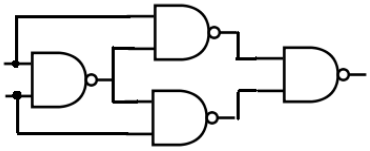
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



A NOR B = NOT(A OR B)

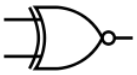
2.9. **XOR Gate.**

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

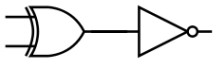


A XOR B = (A NAND (A NAND B)) NAND ((A NAND B) NAND B)

2.10. **XNOR Gate.**



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0



A XNOR B = NOT(A XOR B)

3. Adder Circuits

An *arithmetic circuit* implements an arithmetic operation, such as addition, subtraction, or multiplication. We demonstrate how to build a circuit which performs addition.

3.1. **Bit Half-Adder.** Let us review the process of adding to binary numbers. We line up the bits and add vertically; if two of bits are on, this produces a carry bit. The next column now may have three bits. Here is an example.

```
Carry:            111 1
1st Number:      10011011
2nd Number:    + 00111010
               ----------
                 11010101
```

We see that addition, one column of bits at a time, requires three inputs and two outputs.

We call the inputs `A`, `B`, and a carry-in bit `I`. We call the outputs the sum `Q`, and the carry-out `O`, which we may also call the overflow bit.

A *half-adder* is a circuit with two inputs `A` and `B`, and two outputs `Q` and `O`. It adds two bits, and outputs the sum and the overflow

The sum is determined by addition modulo two:

$$0+0=0, \quad 0+1=1, \quad 1+0=1, \quad 1+1=0.$$

Note that addition modulo two is accomplished by the logical operation XOR.

The overflow is on if both inputs are on, which is accomplished by the logical operation AND.

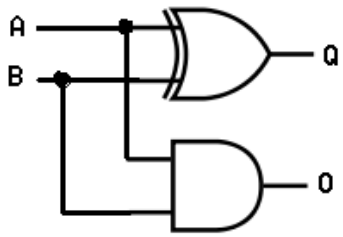
Thus, the half-adder circuit is determined by the rules

- `Q = A XOR B`
- `O = A AND B`

The truth table for this circuit is

A	B	Q	O
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

The diagram for this circuit is



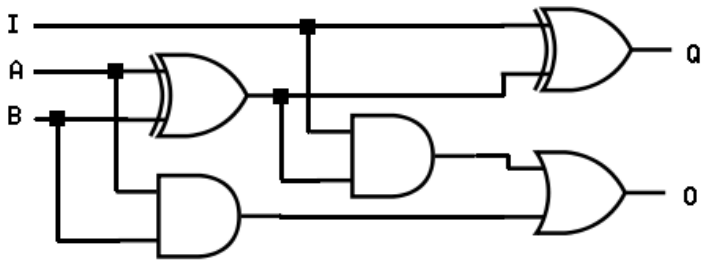
3.2. **Bit Adder.** An *adder* is a circuit with three inputs `A`, `B`, and `I`, and two outputs `Q` and `O`. These are given by the rules

- `Q = A XOR B XOR C` (even is zero, odd is one)
- `O = (A AND B) OR (A AND I) OR (B AND I)` (at least two are on)

The truth table for this circuit is

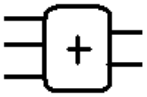
A	B	I	Q	O
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The diagram for this circuit is



3.3. **Nibble Adder.** A *nibble* is four bits, or half of a byte. We show how to build a circuit to add two nibbles. The output is a nibble and an overflow bit. We may generalize this to add words of any bit length.

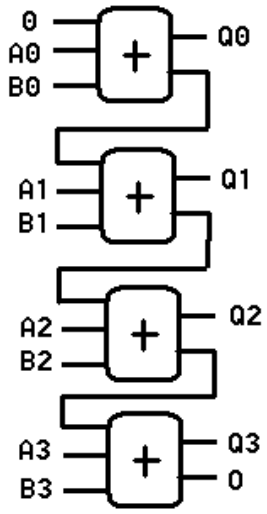
Let the following symbol denote the entire adder circuit:



To construct the nibble adder, take four adder circuit. Lower voltage on the in-carry bit of the first adder. Connect the out-carry of each adder to the in-carry of the next adder, with the last out-carry producing the overflow bit.

Let `A` and `B` denote input nibbles, and let `Q` denote the output nibble. Let `A0,A1,A2,A3` denote the bits of `A`. Let `B0,B1,B2,B3` denote the bits of `B`. Let `Q0,Q1,Q2,Q3` denote the bits of `Q`. Let `O` denote the overflow bit.

The nibble adder circuit diagram is



3.4. **Multiplexors.** A *multiplexor* is a circuit with a multiple input signals and a single output signal; a set of *select signals* to determine which of input signals is output.

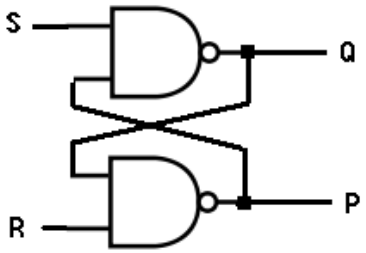
For example, suppose there are three select signals, eight input signals, and one output signal. There are $2^3 = 8$ possible states for the three select signals. We number the input signals 0 through 7; the states of the select signals determines which of the input signals to send on the output line.

Serial transmission of binary information occurs over one wire, one bit at a time. The voltage is raised and lowered at fixed intervals to signal the next bit.

Parallel transmission of binary information occurs over multiple wires, so an entire nibble, byte, or word may be transmitted.

We may use a multiplexor to convert parallel information to serial information. For example, the results of a parallel adder, as in the previous section, could in this way be converted into serial output.

3.5. **RS Nand Latch.** How does a computer store a bit? A simplified circuit to do this is known as the *RS Nand Latch*. The RS stands for Reset/Set. The circuit for an RS Nand Latch is



Let us break this down.

The circuit stores a bit: if `Q=0`, the bit is off, and if `Q=1`, the bit is on.

To understand the circuit, observe the following.

```
Q = (S NAND P) = NOT(S AND P) = NOT(S) OR NOT(P),
P = (R NAND Q) = NOT(R AND Q) = NOT(R) OR NOT(Q).

S=0  =>  NOT(S)=1  =>  Q=1
R=0  =>  NOT(R)=1  =>  P=1

S=0 AND R=1  =>  Q=1 AND R=1  =>  NOT(P)  =>  P=0
R=0 AND S=1  =>  P=1 AND S=1  =>  NOT(Q)  =>  Q=0

R=1 AND S=1  =>  Q=NOT(P) AND P=NOT(Q)
```

The truth table for the circuit is

S	R	Q	P
0	0	1	1
0	1	1	0
1	0	0	1
1	1	NOT P	NOT Q

The procedure for using the circuit is

- Start `R`, then `S`. Now `R=1`, `S=1`, `Q=0` and `P=1`.
- Keep `R` and `S` on.
- To set `Q=1`, set `S=0`, then back to 1. (`S` stands for "set")
- To set `Q=0`, set `R=0`, then back to 1. (`R` stands for "reset")

4. Von Neumann Architecture

4.1. **Memory.** Recall that a computer's *memory* is a sequence of bits, broken up into groups of bits known as *words*. Each word is itself a sequence of bits, and the architecture of the particular computer determines how many bits are in a word. A *byte* is eight bits, and a typical computer word length is a multiple of eight bits, so that each word contains several bytes. Most IBM PC compatible computers of the early twenty first century use a 32 bit word, which contains 4 bytes.

We view the memory as a sequence of words. Each word has a location in this sequence; the location is called the *address* of the word, and the bits stored at this address are called its *contents*. Addresses and contents are again, to the computer, a sequence of bits.

The chart below indicates this for an eight bit word.

Address	Contents
10000000	01001111
10000001	01001110
10000010	01010101
10000011	00001000
10000100	00000000
10000101	00010001
⋮	⋮

Notice that whereas the addresses are sequential, the contents can be anything.

4.2. **Instructions.** A computer consists of an enormous circuit, which is a collection of interconnected subcircuits. The circuitry itself determines the operations which the computer can perform.

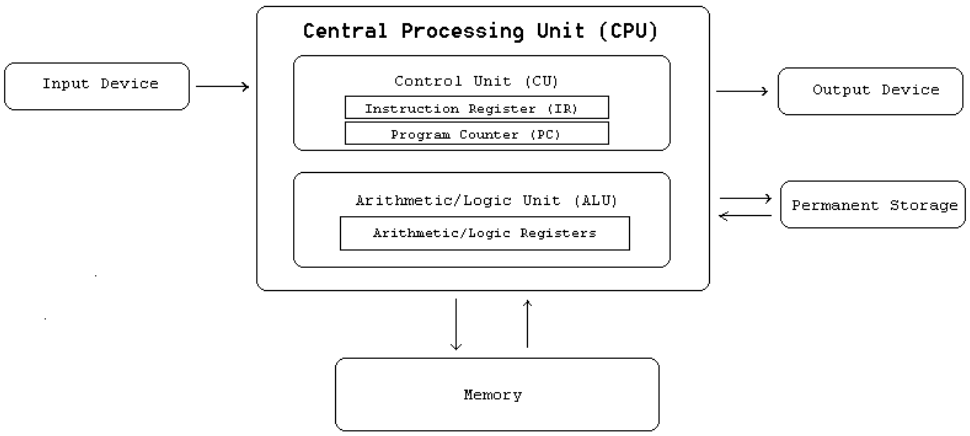
A single operation is known as an *instruction*. An instruction consists of a single word; it is a collection of bits. The circuitry determines the behavior of a given instruction. We give some simple examples, in an imaginary computer.

- `00000101` means "copy the contents of memory location 00000101 into the accumulator"
- `00100011` means "perform a bitwise AND of the accumulator with location 00000011"
- `00011101` means "add the contents of the accumulator to location 00001101"
- `01001010` means "jump to memory location 00001010"
- `11000001` means "shift the bits of the accumulator to the right"

Note how basic the instructions are. Programs are built from thousands, or possibly millions, of such instructions.

The *stored program concept* proceeds from the idea that the instructions and the data of a computer application should reside in the same place, the computer's memory.

4.3. **Von Neumann Architecture.** The mathematician John von Neumann designed the following architecture for an abstract computer.



This design has the following features.

- The memory unit holds both the instructions and the data.
- The input unit moves data from the outside world into the computer.
- The output unit moves data from the computer to the outside world.
- The arithmetic/logic unit performs the arithmetic and logical operations on words.
- The control unit manages the process through the fetch/execute cycle.

The *central processing unit* (CPU) is the combination of the control unit and the arithmetic/logic unit; these work together. The arithmetic/logic unit (ALU) determines what the computer can do, and the control unit (CU) determines what the computer actually does.

4.4. **Registers.** *Registers* are special memory locations which reside inside the CPU.

The arithmetic/logic unit typically contains one or more registers, often referred to as *accumulators*, which hold intermediate results during computations.

The control unit contains two important registers which drive the fetch/execute cycle:

- the *Instruction Register* (IR) contains the instruction which is currently being executed
- the *Program Counter* (PC) contains the memory address of the next instruction to be executed

4.5. **Fetch/Execute Cycle.** The fetch/execute cycle consists of these steps.

- Fetch the next instruction.
  - The contents of the memory address indicated by the program counter is loaded into the instruction register.
  - The program counter is incremented, so that it points to the next instruction.
- Decode the instruction.
- Get data if needed.
  - Fetch information from memory, if the instruction so indicates.
- Execute the instruction.
  - Signal the arithmetic/logic unit to perform an operation, if indicated.
  - Modify the program counter to cause a jump to an instruction elsewhere in memory, if indicated.

4.6. **Summary.** The computer has a timer which drives the fetch/execute cycle.

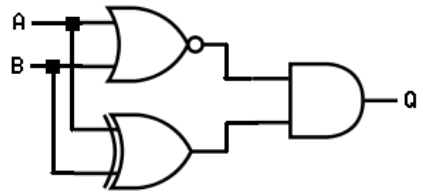
The instruction is fetched, and place in the instruction register. The circuits of the control unit cause this instruction to control the flow of electricity through the computer. Memory is modified, output may occur.

When the timer indicates, the next instruction is fetched, electricity flows through the circuit according to a different course, determined by this new instruction. Memory is modified, output may occur.

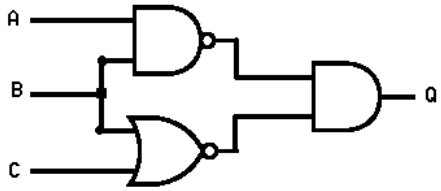
Then the next instruction is fetched on time, and so forth. This cycle repeats millions or billions of times each second. The cycle never stops while the computer is on; even when it appears to be doing nothing, it is looping, waiting for some input which will alter its course.

5. Problems

Problem 1. Write a logical expression and a truth table for the following circuit diagram.



Problem 2. Write a logical expression and a truth table for the following circuit diagram.



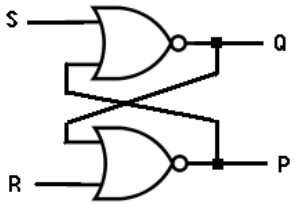
Problem 3. Write a logical expression and a circuit diagram for the following truth table.

A	B	Q
0	0	1
0	1	1
1	0	0
1	1	1

Problem 4. Write the truth table and draw a circuit diagram for the following logical expressions.

(a) `NOT(A) OR B`
(b) `(A OR B) NAND (A AND B)`
(c) `A XOR (B AND C)`
(d) `(A AND B) OR (A AND C)`

Problem 5. Analyze the *RS NOR Latch* below.



Department of Mathematics & Computer Science, Southern Arkansas University
E-mail address: plbailey@saumag.edu