Lecture #21 – April 2, 2004

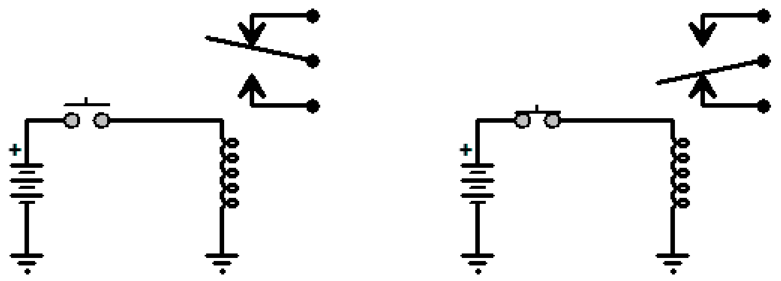
Relays and Adder/Subtractors

In this lecture we look at a real technology for implementing gate circuits, as well as a more-or-less complete design for a general purpose adder/subtractor circuit. The particular technology we will examine is that of the ***electromechanical relay***. Although obsolete by today's standards it is very easy to understand how relays work, and by extension it is easy to convince people that while newer technologies are faster, smaller, cheaper, and more reliable they are mathematically equivalent to relay devices.

Relays

A relay is no more than an electromagnet and a mechanically coupled switch. When the electromagnet is off (no power to the coil), the switch is in its natural relaxed state. Applying power to the coil creates a powerful magnetic field, which pulls in the switch. Removing power allows the switch to snap back to its relaxed state. Many relays use a movable armature and a spring, with the armature *relaying* the switch from the normally closed setting to the normally open setting.

In the diagrams below, the relay is represented by the coil of wire and the single-pole, double-throw switch. There is also a battery and a push-button controlling the power to the coil. (Both the negative pole of the battery and the bottom end of the relay coil are connected to *ground*, which is a convenient notation for showing a common reference point for voltages without having to draw wires all over the place.) In the left view the push-button is open, the relay coil isn't energized, and the switch is in its relaxed state. In the right view the push-button has been pushed, the circuit is complete (through ground), the relay coil is energized, and the switch has been pulled in.

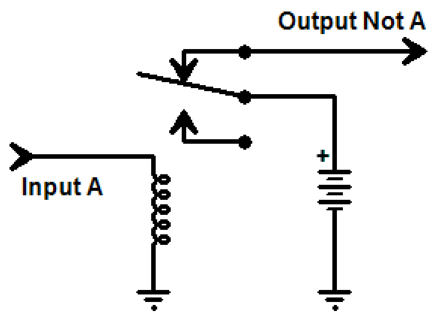


Relay Gates

Creating simple gates with relays requires one relay for each input, with the corresponding switches wired for the desired function. Thus, a half-adder, which normally requires an `OR`-gate, two `AND`-gates, and a `NOT`-gate, must be built using seven individual relays. Each relay may have more than one switch operating in tandem; with two switches per relay and some clever wiring a (two-input) half-adder requires only two relays. With four switches per relay a (three-input) full-adder requires only three relays.

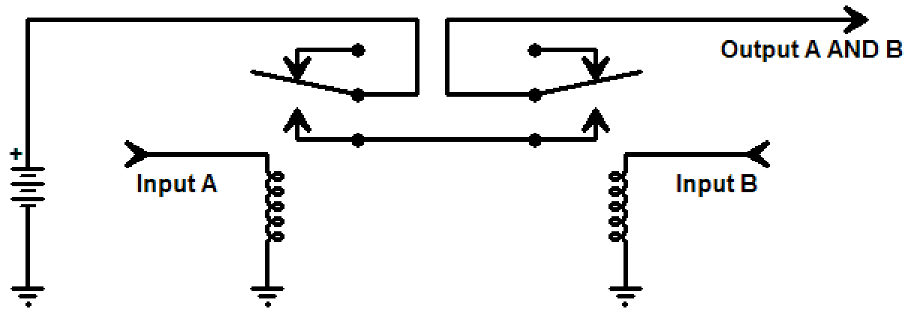
Relay NOT

The simplest relay circuit is the NOT-gate, which requires only a single relay, as shown below. With no power applied to the input, the switch is in its relaxed state, wired so that the output *is* powered through the normally closed contact. This configuration corresponds to input=0, output=1. Applying power to the input activates the relay coil, opening the switch and removing power from the output. This configuration corresponds to input=1, output=0.



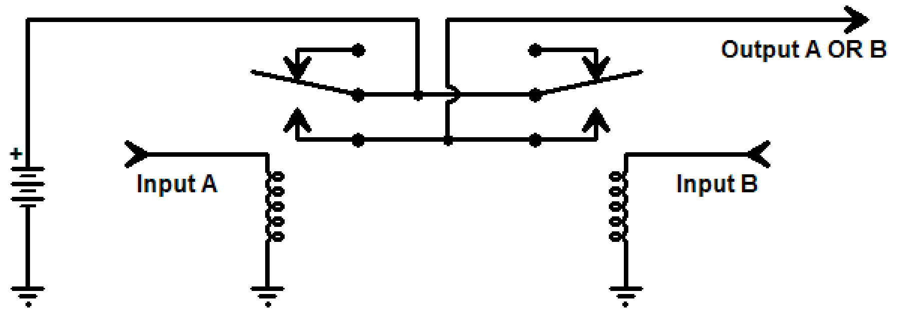
Relay AND

For the AND-gate (and all other two-input examples which require two relays) I've mirrored the right-hand relay left-right for symmetry purposes; this also brings the two sets of switch contacts close to each other so that the wiring is neat and simple. The only way that the output line will be powered is if both relays are activated. Since the contacts are wired in series, the output circuit will be broken if either relay is inactive. A three-input AND-gate requires a third relay, with its normally open switch contacts wired in series with the other two.



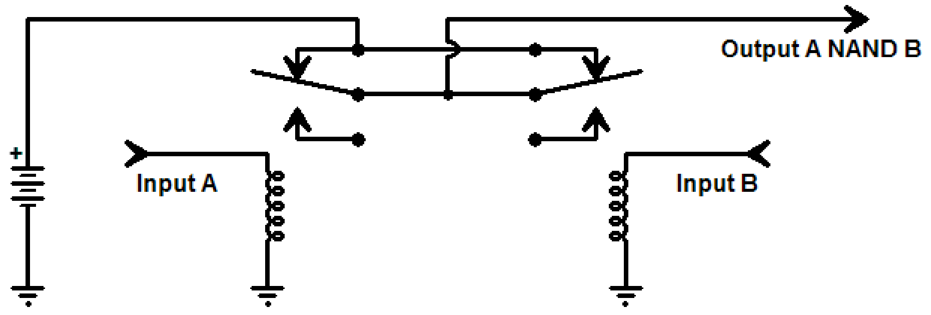
Relay OR

For the OR-gate the normally-open contacts are wired in parallel. Activating either relay (or both) sends power to the output. A three-input OR-gate also requires a third relay, with its normally open switch contacts wired in parallel with the other two.



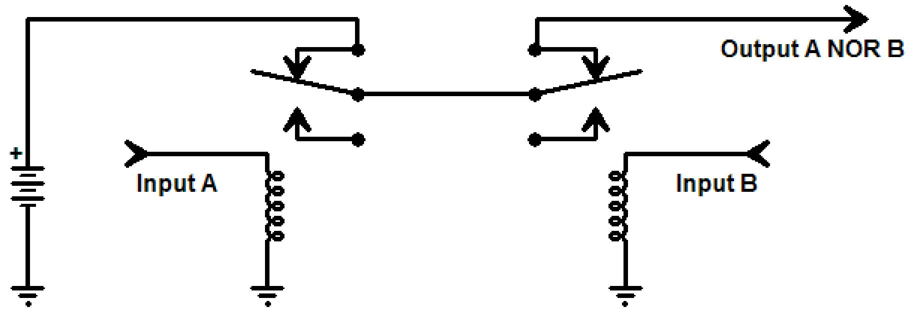
Relay NAND

The NAND-gate is similar to the OR-gate in that the contacts are wired in parallel, but this time it is the normally closed contacts which are used. The output will be powered as long as at least one relay is *un*-powered. The output circuit is broken only if all relays are activated, opening all parallel switches.



Relay NOR

The NOR-gate is similar to the AND-gate in the same way that the NAND-gate is similar to the OR-gate. As with the AND-gate the switches are wired in series, but it is the normally closed contacts which are used. The output is broken if *any* relay is activated.



Conclusions about Relays

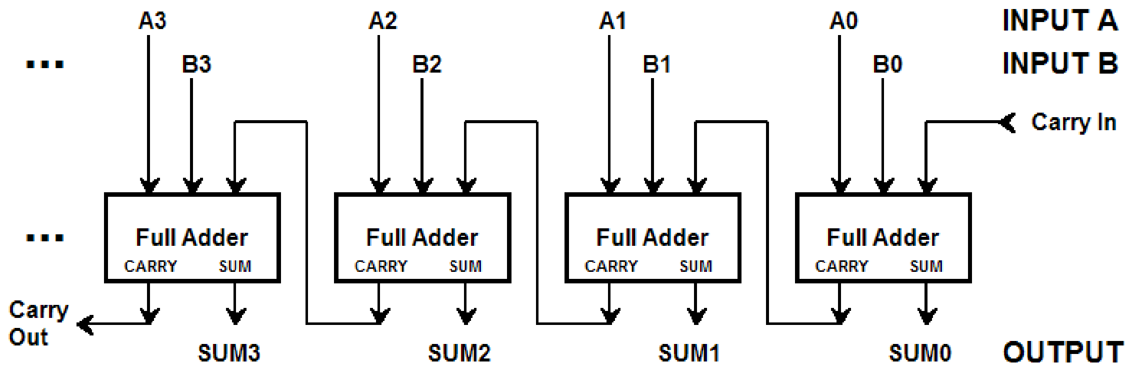
Relays are slow, mechanical devices that perform switching by moving an armature from one contact to another. Their advantages are in their technological simplicity; relays can be constructed by hand from nothing more than insulated wire and a few bits of iron and copper. The earliest electrical and electronic computers used many relays, as did much of the early telephone switching gear.

Relay circuits are easily built and tested by people with beginner-level electronics skills. I am reminded of many post-apocalyptic, time-travel, or alternative-history science fiction novels, where people with contemporary knowledge and skills are forced to "make do" with primitive equipment and techniques, yet come through in the end with some bit of technological wizardry. A classic example is the episode of the original Star Trek titled "The City on the Edge of Forever," where Mr. Spock builds a computer interface to his tricorder with relays and vacuum tubes.

Once you believe that computer circuits can be constructed with relays, it takes very little stretch of the imagination to infer that similar circuits may be built with more advanced technologies.

Adder-Subtractor

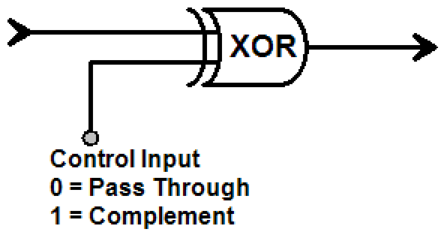
As we saw in the previous lecture, one of the most important circuits in any computer (whether built with relays or something more modern) is the adder. What we will see in this section is that an adder may also be used as a subtractor with minimal additional circuitry. Recall how a ripple-carry adder is constructed from full-adders:



Normally for addition the Carry In bit is 0. For us to use the adder as a subtractor we need only form the two's complement of input B. This can be done by complementing all the input bits of B (forming the one's complement), and setting the Carry In bit to 1.

An `XOR`-gate (exclusive-`OR`) can be used as a "selective complement" device. If one input is considered to be a control input, then the output will match the other input if the control input is 0, and will be the complement of the other input if the control input is

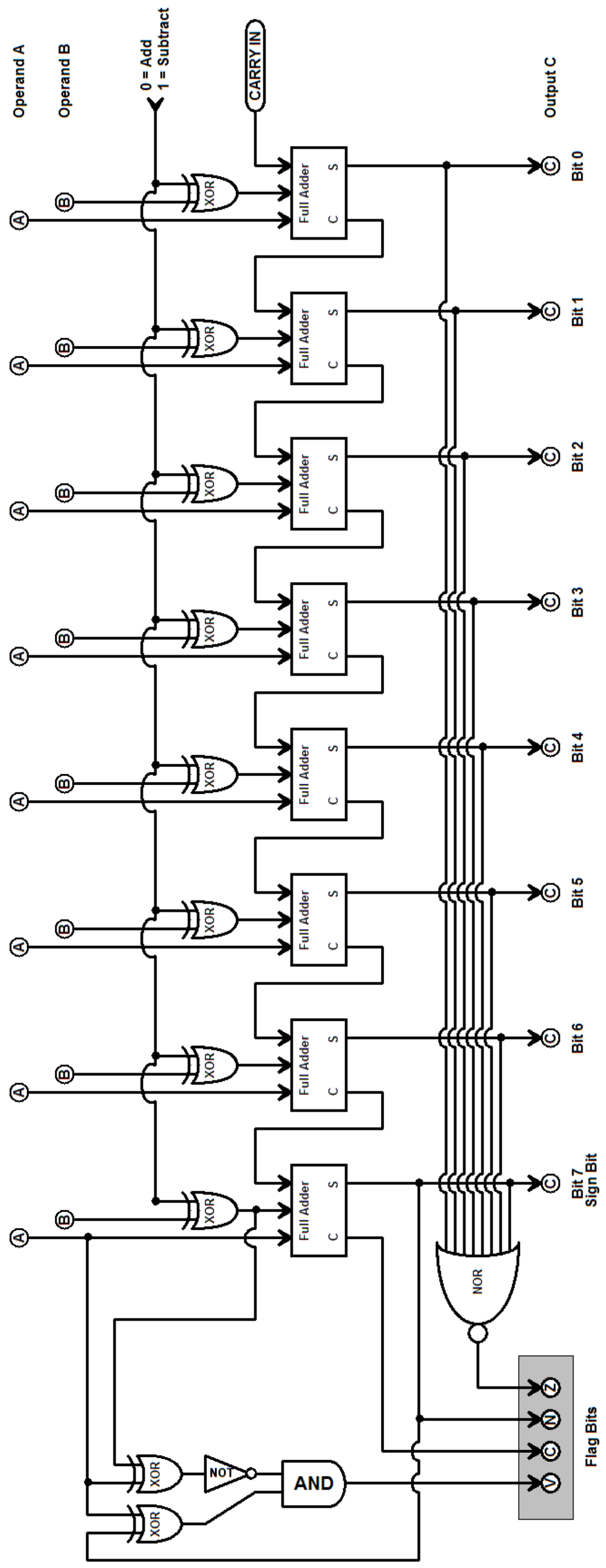
1. By putting one of these XOR-gates on every B input line and tying all the control inputs together, the common control input is used to select whether B or its one's complement is sent to the adder. This line can also be tied to the spare input (the Carry In) of the rightmost full-adder, so that when complementing is selected an extra 1 is also added to the result, thus creating the two's complement of B. If the control line is 0 addition is selected, and if the control line is 1 subtraction is selected.



We can also generate all of the common status flags through the addition of a few extra gates. The carry out of the leftmost full-adder becomes the new value of the C (carry) bit. The N (negative) flag is simply the leftmost sum bit; if 1 the result was negative, and if 0 the result was positive. The Z (zero) flag is the NOR of all output bits. Generating the V (overflow) bit is a bit more complicated. There are four cases to consider: adding two positives and getting a negative result, adding two negatives and getting a positive result, subtracting a negative from a positive (i.e., adding two positives) and getting a negative result, and subtracting a positive from a negative (i.e., adding two negatives) and getting a positive result. In my solution (shown on the next page), the extra circuitry consists of two XOR-gates, an AND-gate, and a NOT-gate.

The circuit on the next page is the complete design of an 8-bit adder/subtractor, such as might be found on a 6502 processor. Examine it closely to insure that you understand how each section works. It should be pretty obvious how to extend the circuit to any number of bits.

A few points about the flag bits are in order. First, the Carry Out line is not simply looped around into the Carry In line; there is a single bit of storage interposed between the two so that there is no infinite feedback loop. This bit of storage, in the ***program status register***, is updated as the addition or subtraction instruction is executed; the C bit's value is used as part of the *input* to the instruction being executed, and the C bit is updated with a new value from the *output* of the instruction. Second, the new values of the flag bits are always computed, whether or not they will be tested by subsequent instructions; if the numbers are treated as signed then you write instructions that test the N and V bits, but if the numbers are considered unsigned you simply ignore any updates to those bits. Finally, on processors such as the 6502 and 8088 the new values of the flag bits are *always* written into the processor status register automatically; on the ARM new flag values go into the status register only if the "S" suffix is added to instructions (e.g., writing ADDS instead of ADD).


Operand A
Operand B
0 = Add
1 = Subtract
CARRY IN
Output C
A
B
XOR
Full Adder
C
S
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Sign Bit
NOR
NOT
AND
Flag Bits
Z
N
C
V