Now, what do we mean by these operations? Regularly the complete meaning of an operation is described by giving a table of the outputs for some of the inputs, and then covering the remaining cases of output-input by general statements. This we have done in Chart 1, in rather condensed language. The lower case (not capital) letters a, b, c, refer to numbers that Simon knows, 0, 1, 2, 3. The lower case letters p, q, r, refer to *truth values:* the truth value of a statement is 1 if the statement is true, and 0 if the statement is false. A primitive way of indicating a truth value is with check marks (√) and crosses (×). But check marks and crosses are not numbers like 1 and 0, and cannot be combined like numbers, and there is no need to bother with them, for using them is like using a crystal set when you could use a console radio. The expression T(...), which is read: "T of ...", where ... is some statement, is a nice short way of writing "the truth value of ...". Truth values are becoming more important all the time as a means of designing and economizing electronic computer circuits. For example, the electronic computer Maddida was designed largely by truth value algebra instead of circuit diagrams.

The capital letters P, Q, R refer to statements of which the truth values are p, q, r, respectively. Statements have to be expressed usually with words, and sometimes may be expressed in other ways. But the little letters p, q, r are truth values and are always 1 or 0. It is readily understood that: $p = T(P)$, $q = T(Q)$, and $r = T(R)$.

For example, let us consider the fourth operation, that of selection. The general statement for this operation is $c = ap + b(1-p)$. What will this rule give us in a concrete case? Suppose a is 2, and b is 3, and p is the truth value of the statement "0 is greater than 1."

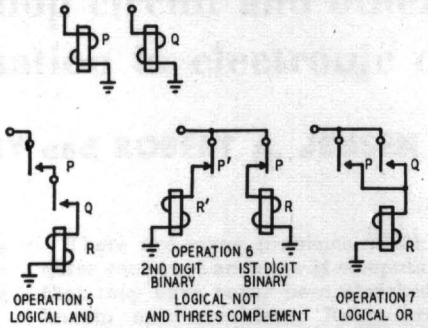

Fig. 4—Logical circuits. "And" circuit outputs a pulse only if both relays are closed; "not" circuit only if both are open; "or" if either or both are closed.

In other words, what we want to do is select 2 if 0 is greater than 1, and select 3 if 0 is not greater than 1. Here the statement that we are interested in is "0 is greater than 1"; this statement is false, and so p, its truth value, is 0. Putting $a = 2$, $b = 3$, and $p = 0$, in the rule,

$$c = ap + b(1-p) = [2 \times 0] + [3 \times (1-0)].$$

The arithmetical result of this is 3, which of course agrees with what our common sense tells us. Why do we resort to all this roundaboutness? Because when you want to tell a mechanical brain to do something, you have to be very explicit.

Operations 8 and 9 need some comment. Their purpose is to enable us to handle numbers of more than two binary digits. If we are adding two four-digit binary numbers, for example, first we attend to the two binary digits at the right, using operation 1, Addition Without Carry. Second we attend to the two binary digits at the left, using operation 8, Addition, Subject to Carry, this process remembering whether or not there was a carry from that first step.

The circuits for operation 1 to 4 are given in Chapter 3 of *Giant Brains*, on pages 36 to 38. The circuits for operations 5 to 7 are given schematically in Fig. 4. We note that "logical NOT" and "threes complement" are the same operation, with the same code 0101. This is because if we use only the right-hand binary digit, we have the logical-NOT operation; and if we use both the left-hand and the right-hand binary digits, we have the threes complement. The circuits for operations 8 and 9 are not hard to design, in one way or another; there is no space to give them here.

Chart 1—Operations of Simon

Operation 1
Addition without carry
$c = a + b$

a \ b	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Operation 2
Subtraction or negation without carry = fours complement
$c = 4p - a$
$p = T(a \text{ is } 1, 2, 3)$

a	p	c
0	0	0
1	1	3
2	1	2
3	1	1

Operation 3
Greater than
$p = T(a \text{ is greater than } b)$

a \ b	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0

Operation 4
Selection
$c = ap + b(1-p)$

p	0	0	0	0	1	1	1	1
b	0	1	2	3	0	1	2	3
a								
0	0	1	2	3	0	0	0	0
1	0	1	2	3	1	1	1	1
2	0	1	2	3	2	2	2	2
3	0	1	2	3	3	3	3	3

Operation 5
Logical "and"
$r = T(P \text{ and } Q)$

p \ q	0	1
0	0	0
1	0	1

Operation 6

Logical "not"
$r = \overline{T}(\text{not P})$

p	r
0	1
1	0

Threes complement
$c = 3 - a$

a	c
0	3
1	2
2	1
3	0

Operation 7
Logical "or"
$r = T(P \text{ or } Q)$

p \ q	0	1
0	0	1
1	1	1

Operation 8
Addition, subject to carry
$c = a + b + p$
$p = T(\text{previous addition was } 1+3, \text{ or } 2+2, \text{ or } 3+2, \text{ or } 3+3)$

p	0	0	0	0	1	1	1	1
b	0	1	2	3	0	1	2	3
a								
0	0	1	2	3	1	2	3	0
1	1	2	3	0	2	3	0	1
2	2	3	0	1	3	0	1	2
3	3	0	1	2	0	1	2	3

Operation 9
Subtraction or negation subject to carry
$c = 3 - a + q(4p - 3)$
$p = T(a \text{ is } 1, 2, \text{ or } 3)$
$q = T(\text{previous negation without carry was of } 0)$

a \ q	p	0	1
0	0	3	0
1	1	2	3
2	1	1	2
3	1	0	1

Programming and timing

The next thing we have to consider is the programming and timing of the machine as a whole.

In Simon, as constructed, we are using only one tape for both instructions and numbers. It proved to be convenient when constructing Simon to cause the tape to be read three times in each complete machine cycle, at time 2, time 5, and time 8. We call these three readings three *entries* of information into Simon. The first entry, at time 2, specifies the register which is to receive the information. When this entry is read, it causes the entrance relay of the receiving register to be energized, and so connects the coils of the receiving register to the bus, and clears out any information previously in it. The second entry, at time 5, puts any number or operation from the tape into input registers 1 and 2, so that the number or operation can be made use of in the machine. The third entry, at time 8, specifies the sending register, i.e., the register from which information is to be transmitted, and at the same time causes the information in that register to pass through the bus into the receiving register.

In some electric brains, we would have to consider carefully the subsequences of the timing of the routines of such computing operations as multiplication and division. This is not the case however in Simon, because every computing operation is completed in the machine cycle following the designating of the operation in computer register 4. If we should wish to do multiplication or division on Simon, we would need to program it by means of the instruction tape, and the use of arithmetical and logical operations.

For example, what would we punch in the tape if we wanted the operation "selection" (code 0011) to go into computer register 4 (code 1011), the register which chooses the operation the computer is to utilize? We would punch, in three successive lines of tape:

Entry	Punched Holes	Meaning
1	11011	Get ready to receive in CR 4, and clear IR 1 and 2.
2	00011	Put operation selection in IR 1.
3	00000	Transfer out of IR 1.

This description of Simon and its operations is incomplete. We have not touched on the wiring of the front panel, so that different types of automatic or manual operations are possible. We have not covered the two other