~~PART II~~ ?

CHAPTER FOUR

The Switching Circuits Used in Digital Computers

"Off again, on again, gone again,
Finnigin. —" S. Gillilan (?)

"...One if by land, and two, if by sea;
And I on the opposite shore will be,
Ready to ride and spread the alarm..."
H. W. Longfellow (1xxx-1xxx)

"Logic is the art of convincing us of
some truth." J. Bruyere (1645-1696)

"... convert them into quantitative
symbols, susceptible only to the values
0 and 1." George Boole (18xx-1xxx)

"Two is company, three is a crowd."
T. Fuller (1608-1661)

Whereas we looked downward in chapter 2 from the more comprehensive to the simpler components, developments have necessarily proceeded in the opposite direction, i.e. "bottom up." Most of the major breakthroughs in computing have actually involved the very bottom level, i.e. whether switching circuits were implemented using electromechanical relays, vacuum tubes, discrete transistors or with various levels of integrated circuits. To talk about these developments in a meaningful way, we must first be able to talk about switching circuits. Fortunately, in contrast to analog circuits, switching circuit concepts are easy to understand.

We will digress in this chapter therefore to discuss switching circuits. Our discussions have brought us to the

phase of historical development where switching circuits began to play a major role in computer history.

ELECTRICAL SWITCHES

Switching circuits are electrical circuits with switches in them like the one in the table lamp in your living room. (We will take licence to treat the electrical power coming through a wall socket as though it were direct rather than alternating current. The power supply in a computer performs this conversion.) The switch in such a circuit provides control over the electric current and voltage that flows to the light bulb so that there will only be power for the light when the switch is "on" as shown in figure 4.1. In this diagram there is voltage at the switch

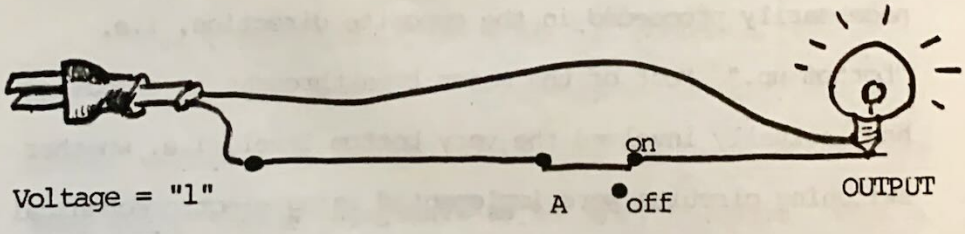


Figure 4.1: Architype "Switching Circuit"

whenever the circuit is plugged in, but there will only be a non-zero voltage at the OUTPUT (the light bulb) if the switch labelled "A" is in the "on" position. The light is an indication of whether the switch is open or closed. If we consider the voltage at the switch as being 1.0 Volts,

then the voltage at the light will be either 1.0 or 0.0 Volts. But since we are only interested in voltage in the sense of its being "on" or "off" rather than its precise numerical value, there are other equivalent ways of considering voltages in switching circuits. We can for example consider the value of OUTPUT as the truth or falsehood of the assertion:

"THE SWITCH IS ON!"

In this case OUTPUT and A are called "logical variables" whose values are either TRUE or FALSE, depending on the switch setting.

as related to a set of assertions like the preceding one.

ENCODING SWITCHES WITH MEANING

Encoding on/off switches with true/false information was not new; Paul Revere would have been a natural at digital design. On or off indications are used in much the same way in digital computers as he applied them in his famous ride. His use of "one if by land, and two, if by sea" rather than "zero and one" provided for guaranteed synchronization and failure detection. This required an extra switch (two lamps) but it included an extra "bit" to provide fault tolerance!

In computers switches control voltages which affect lights or other devices which collectively provide the information which is the ultimate implication of flipping the particular switches. Setting switches need not involve toggleing switches manually, however, although in the early days of computing it meant just that. Nowadays, when you type a character on a keyboard for example, the keyboard electronics will set eight "switches" as a coded indication of which key was depressed. Typing the capital letter, "A" for example, will typically result in the second and eighth of these switches being turned "on" or "TRUE," the other six will be set to "off" or "FALSE." (In the old days one might have flipped the eight switches to the right code and then toggled a ninth one to indicate that the switches now have meaning.) As the computer processes the fact that you typed an "A," it will set and reset a series of "internal" switches on its own in coming to a conclusion based on your having typed an "A."

Yes, computers flip switches too! Computer history would be short indeed if their switching circuits were all controlled manually. The implications of your having typed "A" may be far reaching within the computer, a cascading of millions of switches being toggled before the computer responds a second later. But before we discuss how combined switch settings can be used to effect complex computations, let's discuss the methods that computers use in turning their own circuits on and off.

RELAYS

We have mentioned electromechanical devices that employ electricity to move magnets which in turn trip levers which effect some mechanical function, even like flipping a switch. Switches that are controlled by such means are called electromagnetic "relays." The Z2, Z3 and Z4 and MARK I were implemented using relays, but since then vacuum tubes and transistors have become much more effective in terms of cost, speed, size, power and reliability. But the function being performed is the same and in fact hydraulic switches have even been shown to work. Since relays accomplished the function by the physical movement of a switch very much like using your finger, their operation is easy to understand. For this reason (as well as historical chronology) we will discuss switching circuits from this viewpoint. The few essential differences when vacuum tubes and transistors are employed are discussed when those devices are introduced.

In figure 4.2(a) the essentials of a circuit like the one in figure 4.1 are shown, but with the addition of an electromagnet (the coil of wire with a movable magnetic core indicated by the dark bar) to provide control of the switch. Elecrical components and the power supply are chosen to guarantee that compatibility relations hold such that when one volt is applied at A, the electromagnet will be activated and the switch will open by magnetic attraction.

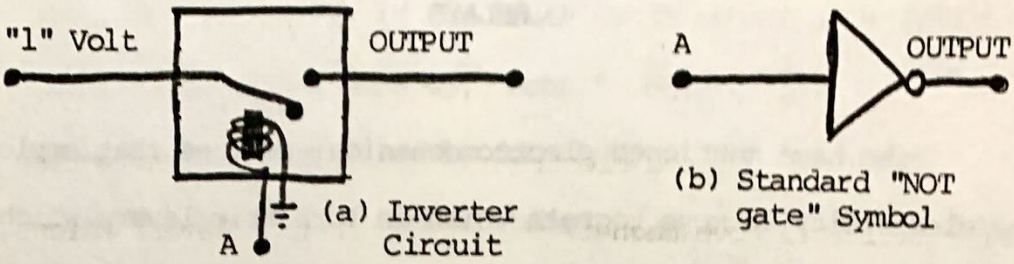


Figure 4.2: Electromagnetic Relay Switch

OUTPUT will change voltage from "1" to "0". If the applied voltage at A is "0", then the switch will relax to its closed position and the OUTPUT will be "1".

Such a device is called a "gate." (The suitability of this name should be apparent by inspecting its "opening" and "closing" features in the figure.)

STANDARD LOGIC GATES

The OUTPUT voltage in the particular switch that is shown in figure 4.2(a) always takes on the opposite characteristics to that of the switch voltage, A. This circuit is sometimes referred to as an "inverter circuit" for this reason. It is more commonly called a "NOT gate," however, for which the special symbol shown in figure 4.2(b) is used. This opposition of A and OUTPUT are expressed as:

OUTPUT = NOT A

This is easy to understand if you think of A as the truth or falsehood of an assertion, such as, "John F. Kennedy died of

old age!" If OUTPUT is the truth or falsehood of a directly opposing assertion such as, "John F. Kennedy was assassinated," then clearly for A and OUTPUT the above relationship holds. Both statements cannot be true or false at the same time. When we "assert OUTPUT" for example, we can be stating a truth or a falsehood; which can be determined by its relationship to other statements whose truth or falsehood we know. It is a matter of logic, hense the name "logic circuits" and "logic gates."

In the NOT and subsequent standard gate symbols, the input power ("1" Volt in all our examples) is omitted. Electrical power and the incidental requirements for resistors and capacitors become underlying assumptions which are not typically shown explicitly in logic diagrams employing such standard gate symbols.

A different switch implementation for which the gate remains open when A=0, can be used to create extremely useful logic circuits when replicated in series. This combination of switches is shown in figure 4.3(a); it is an "AND gate". In the figure, the relays have been stylized, but they assume the same basic electromechanical implementation as shown in figure 4.2(a). Combinations of switches as shown require that all of the switches be closed for OUTPUT to obtain voltage. In this case we can say OUTPUT = TRUE if and only if all switches are closed, which is characterized by A=TRUE and B=TRUE and C=TRUE. This is the same as saying:

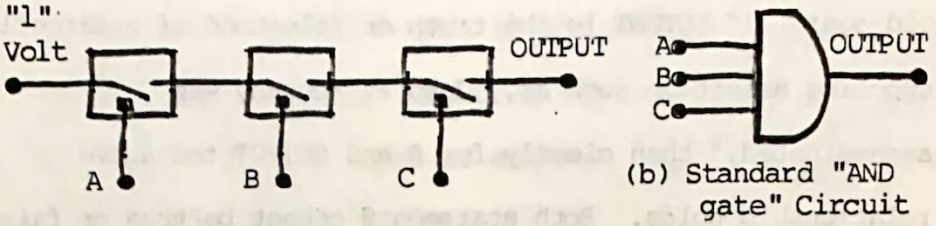


Figure 4.3: (a) Conjunction of Switches

OUTPUT = A AND B AND C

where it is understood that A, B and C are logical variables, i.e. the truth or falsehood of assertions like those we used in describing the previous gate. If A stands for the assertion, "the creature is green," B for the assertion, "it has a hard shell," and C for, "it walks slowly on four legs," then OUTPUT can be the statement, "the creature is a turtle!" If any of the statements A, B and C are not all true, then the creature is not a turtle! The standard symbol for an "AND gate" is shown in figure 4.3(b).

In figure 4.4(a) another useful combination of switches is shown which performs the logical function known as "OR".

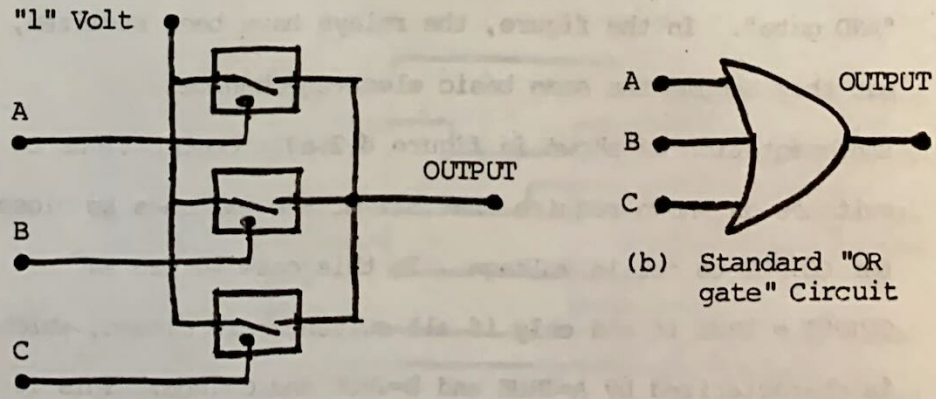


Figure 4.4: (a) Switches Representing A OR B OR C

In this case: OUTPUT = TRUE if and only if at least one of the switches is closed (on, or TRUE), i.e. A=TRUE or B=TRUE or C=TRUE. You can think of examples to express as:

OUTPUT = A OR B OR C

The standard "OR gate symbol is shown in figure 4.4(b).

In figure 4.5 we show a graph of logical variables A, B and C as well as NOT A, A AND B AND C, and A OR B OR C. The graph is an example of a timing diagram like those used in designing digital logic circuits for computers. Each has two possible values, high/low, true/false, one/zero, etc..

All this would not be very significant if it were not for the provable fact that all mathematical expressions can be reduced to equivalent combinations of the operations NOT, AND and OR operating on logical variables whose values are

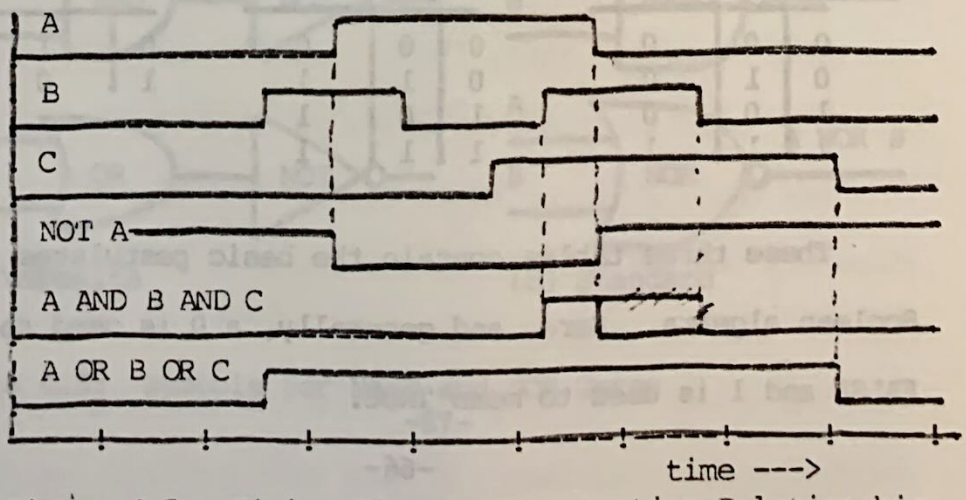


Figure 4.5: Timing Diagram Illustrating Relationship Between Logical Variables and Functions

simply TRUE or FALSE. This implies of course that any computation can be reduced to digital logic using switching circuits no more sophisticated than those which we have described. This is the ultimate theoretical basis of digital computers!

Lord Bertrand Russell and Alfred North Whitehead proved this in their monumental treatise: Principia Mathematica in 1910. This work which Russell describes as having caused him an "intellectual hernia," summed up the study of logic introduced originally by George Boole in his "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probability" published in 1847. Boolean algebra derived its name from Boole, who introduced among other notations, the use of "Truth Tables". Truth tables make it easy to evaluate the possibilities of logic functions as shown for OUTPUT = NOT A, OUTPUT = A AND B and OUTPUT = A OR B below:

Logical AND

A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT

A	OUTPUT
0	1
1	0

These three tables contain the basic postulates of Boolean algebra. Here, and generally, a 0 is used to mean FALSE and 1 is used to mean TRUE.

In 1938 Claude Shannon explained the application of Boolean algebra to the design of digital logic circuits. Actually, not all three of the NOT, AND and OR functions are required to implement any conceivable mathematical expression. The NOT operation in conjunction with either the AND or the OR operation are sufficient in themselves to implement any computation. This gives rise to a more efficient set of standardized gates, the NAND and NOR gates which stand respectively for NOT-AND and NOT-OR as follows:

A NAND B = NOT (A AND B)

A NOR B = NOT (A OR B)

In figure 4.6, the composite and standard symbols for the NAND and NOR gates are shown.

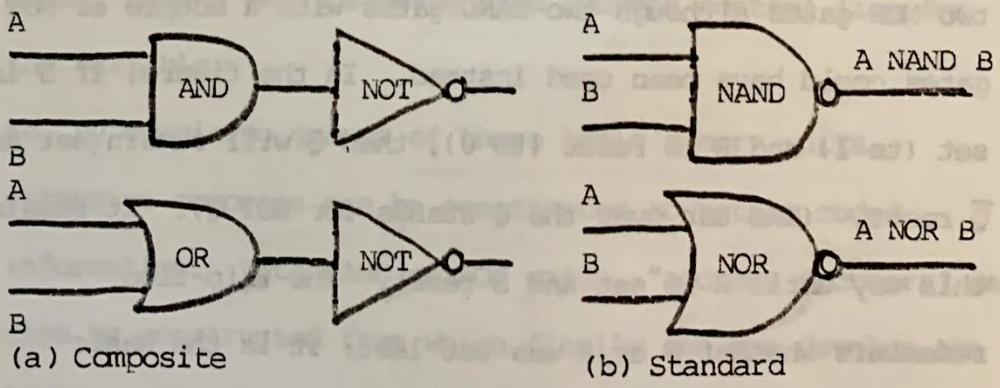


Figure 4.6: Symbols for NAND and NOR Gates

The use of logic gates to implement storage for numeric values and the circuits which perform arithmetic on these values is the essential first step in computer development. The standard gate symbols that we have introduced can be used to design a flip-flop as shown in figure 4.7. The simple RS flip-flop (the RS stands for Reset/Set) involves

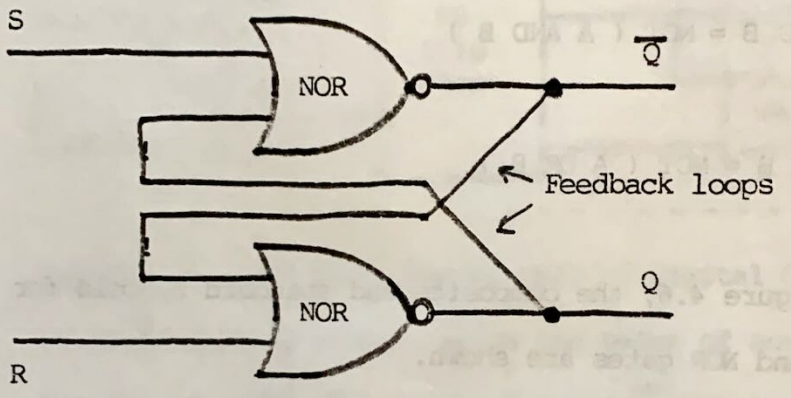


Figure 4.7: RS Flip-Flop Made From NOR Gates

two NOR gates although two NAND gates with a couple of NOT gates could have been used instead. In the figure, if S is set (to 1) and R is reset (to 0), then Q will remain set and $\overline{Q}$ reset. (The bar over the Q stands for NOT Q). It remains this way until R is set and S reset. The flip-flop remembers whether S or R was set last; it is the basic storage unit of computer memories.

The use of flip-flops to store and retrieve data requires additional logic if more than a single logical variable is to be stored in the entire system. For one thing there must be some designation of which stored variable is being referred to. This means that there must be a unique "address control line" for each stored variable

which acts like the key for a locker. When the address control line is set, then the flip-flop value can be changed by W and $\overline{W}$ (this operation is called "Write") or its value can be determined by R and $\overline{R}$ (this operation is called "Read"). This is shown in figure 4.8, where the RS Flip Flop is just the circuit shown in figure 4.7. Now whenever

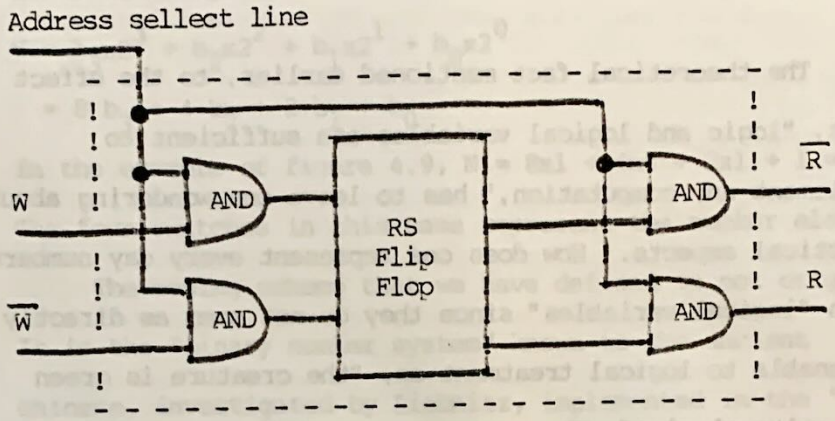


Figure 4.8: The Basic Memory Unit of Digital Computers

it is desired to Read or Write the value of any stored logical variable, the value of W and $\overline{W}$ must be set up appropriately before, or the values of R and $\overline{R}$ determined after, having set the designated address control line for the variable.

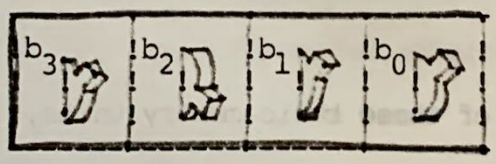
So, by combining groups of these basic memory units, extensive memories can be constructed to retain coded information. With NAND or NOR gates, "address decoders" can also be constructed from which finally one can develop the logic to set the appropriate address sellect line for each stored variable. The entire scheme discussed in reference

to figure 2.6 can then be developed as we will see a little further on. But before this can become completely clear however, the coding scheme for information stored and manipulated by these devices must be discussed.

BINARY NUMBERS

The theoretical fact mentioned earlier, to the effect that, "logic and logical variables are sufficient to implement any computation," has to leave one wondering about practical aspects. How does one represent every day numbers with "logical variables" since they do not seem as directly ammenable to logical treatment as, "the creature is green and walks slowly!" The answer, as we will see, is, "In binary!"

Let's go back to lamp switches! In figure 4.9 there are four ordinary switches drawn. You notice that each of



Number Represented

1 0 1 1

$2^3 + 2^1 + 2^0 = 11$

Figure 4.9: Switch Settings to Represent Numbers

the switches can be up or down (2 positions for each) so that there are a total of $2 \times 2 \times 2 \times 2 = 2^4 = 16$ possible combinations for the four switches. (If you doubt this, you can very easily try all the combinations and count them.) We can therefore code the switches to represent the numbers 0 to 15 (or any other sixteen numbers) any way we want to. Being familiar with decimal notation predjudices us however in favor of a positional notation wherein each switch has a

smaller value than the one to its left. If we give our four switches the names: b_0, b_1, b_2, and b_3 starting at the right, we can consider each to be a sort of digit. These "digits" will be logical variables however, with 0 and 1 (rather that 0 to 9) representing their range of possible values. Then the value, N of the number represented by our switch setting is:

$$N = b_3x2^3 + b_2x2^2 + b_1x2^1 + b_0x2^0$$

$$= 8\,b_3 + 4\,b_2 + 2\,b_1 + b_0$$

In the example of figure 4.9, N = 8x1 + 4x0 + 2x1 + 1 = 11. The four switches in this case represent the number eleven.

The coding scheme that we have defined is not original! It is the "binary number system" known to the ancient chinese, investigated by Liebnitz, implemented in the "Z" series computers by Zuse and finally suggested to us by von Neumann. You can see how ideally binary numbers are suited to switching circuits in particular and digital computers in general.

The following table shows the coding scheme for each of the sixteen possible switch settings. Remember 0 is "off"

BINARY	DECIMAL	BINARY	DECIMAL
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

or "down", and 1 is "on" or "up" so in the "BINARY" column, the four bits form an image of the particular setting of the switches shown in figure 4.9.

BINARY ARITHMETIC

There are some more questions to be answered before the theoretical proofs of Principia Mathematica can have practical significance. For example, how are logic operations like those that we implemented with relays able to perform arithmetic operations on numbers coded into switch settings according to this scheme? To answer this question, let's first look at what it means to add two binary numbers. Since in decimal 1+1=2, we must have that 0001+0001=0010 in binary. If you try a few examples of such reasonable expectations using the table above, you'll notice that whenever a sum of "bits" exceeds one, we must carry values in excess of 1 to the left. This is what we're used to except that carry occurs when a digit exceeds 9 in decimal.

DIGITAL ELECTRONIC ADDING MACHINES

A "single bit adder" takes two input signals, A and B and interprets their voltage values as a bit in a binary sequence representing a number. The logic designed into the adder is shown in figure 4.10. The performance of this

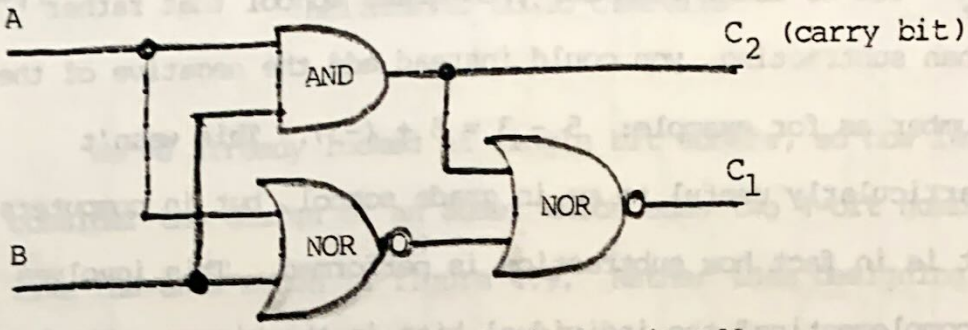


Figure 4.10: Logic Design of Single Bit Adder

single bit addition unit is what you would expect from such a function. The result involves two bits rather than just one, like when you add single digits 7 and 6 to get a two digit result, 13. The results are as described in the following truth table:

TRUTH TABLE FOR SINGLE BIT ADDER

A	B	C_2	C_1	A+B	decimal
0	0	0	0	00	= 0
0	1	0	1	01	= 1
1	0	0	1	01	= 1
1	1	1	0	10	= 2

Arithmetic performed by the Arithmetic Logic Unit of a digital computer involves primarily combinations of single bit adders like the one shown above which involves about three gates.

To complement a number you just reverse all the bit settings in the number (zeros to ones and ones to zeros) and add a one to the least significant bit. For example, the complement of 0110. you reverse bits to get 1001, then add 0001 to obtain 1010. Notice that when the number and its complement are added, the result is 0000 with a carry bit lost to the left.

You no doubt remember from grade school that rather than subtracting, you could instead add the negative of the number as for example: 5 - 3 = 5 + (-3). This wasn't particularly useful to me in grade school, but in computers it is in fact how subtraction is performed. This involves "complementing" the individual bits in the binary subtrahend and adding. Multiplication also is performed by addition with shifting just as you learned in grade school. Division ~~also~~, since it involves sequences of subtraction, can be implemented with binary adders combined in a logical structure as well.

HIERARCHICAL DESIGN

We will discuss how combinational logic can be used in a hierarchy to easily implement the major functions of digital computers discussed in chapter 2. Single bit memories and adders can be combined by simple design techniques to achieve the tremendous capabilities currently typical of these devices. Digital computers involve two major components whose design will be discussed: Memories and central processing units.

We'll look first at the Arithmetic Logic Unit, one of whose major components is the adder. Then we'll look at what is involved in the storage devices employed by digital computers.

We've already looked at single bit adders, so now lets consider the design of an adder which adds two 4-bit numbers like the ones shown in figure 4.9. Rather than designing a 4-bit adder from "scratch," a designer would select a single bit adder like the one shown in figure 4.10 as a basic component of his design and proceed from there. Having solved the single bit addition problem once, he would not worry about how each separate bit position in the 4-bit numbers gets added. He would concentrate rather on the carry operation, arranging his components to effect it properly as shown in figure 4.11. In this adder there are thirteen components, 10 single bit adders (each labeled 1BA) and 3 OR gates. But each of these components are constructed from many switches, i.e. the OR gates each require 2 switches, and the single bit adders each require three gates and a total of 8 switches. So altogether there are 86 switches required to perform a 4-bit addition.

An 8-bit adder can be constructed from seven components: Two 4-bit adders, four single bit adders and one OR gate, requiring a total of 206 switches. Figure 4.12 illustrates the design which uses the components defined in figures 4.11, 4.10, 4.6, 4.4, 4.3 and 4.2 in a complete hierarchy. In the figure we have used solid arrows to indicate multiple bit interfaces; this symbology is used for

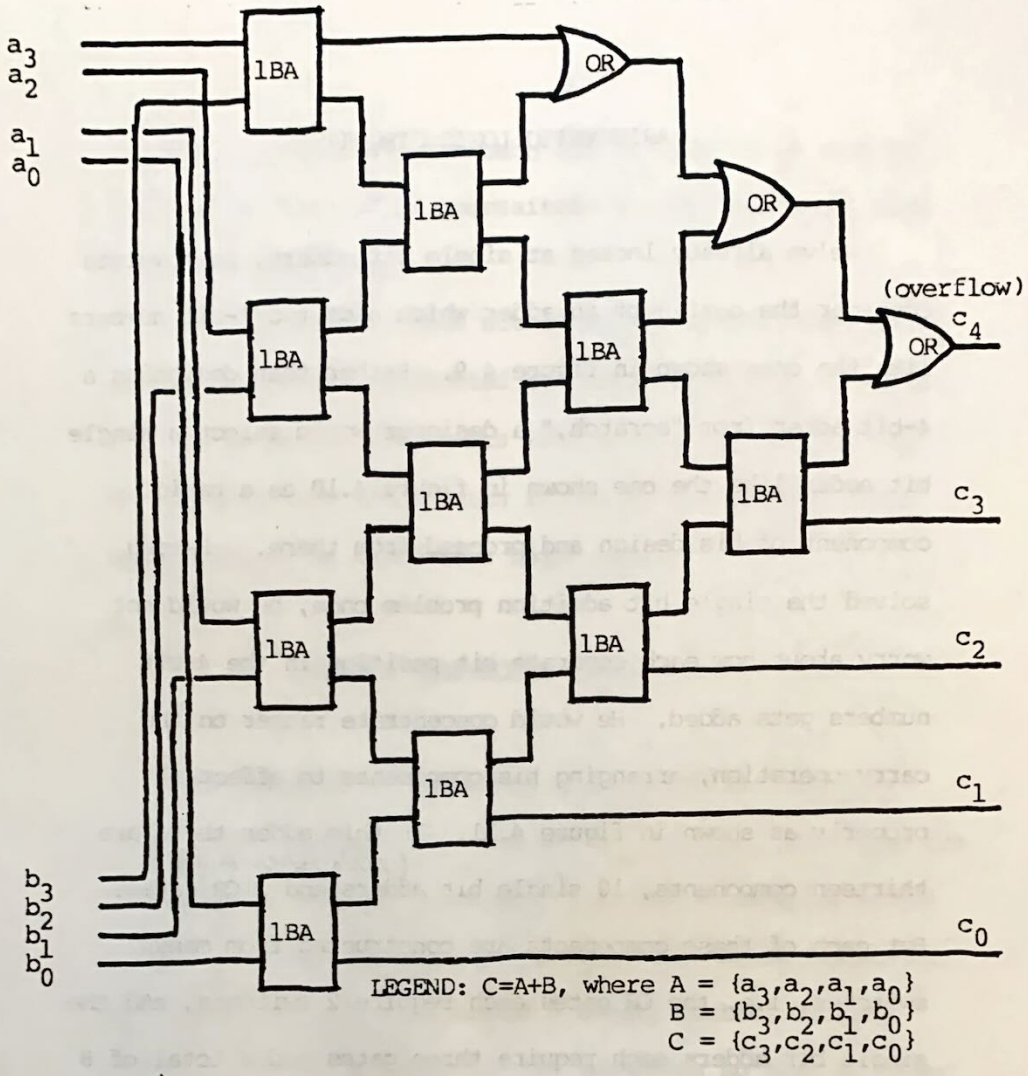


Figure 4.9: Four-bit Adder (Using Single Bit Adders and OR Gates)

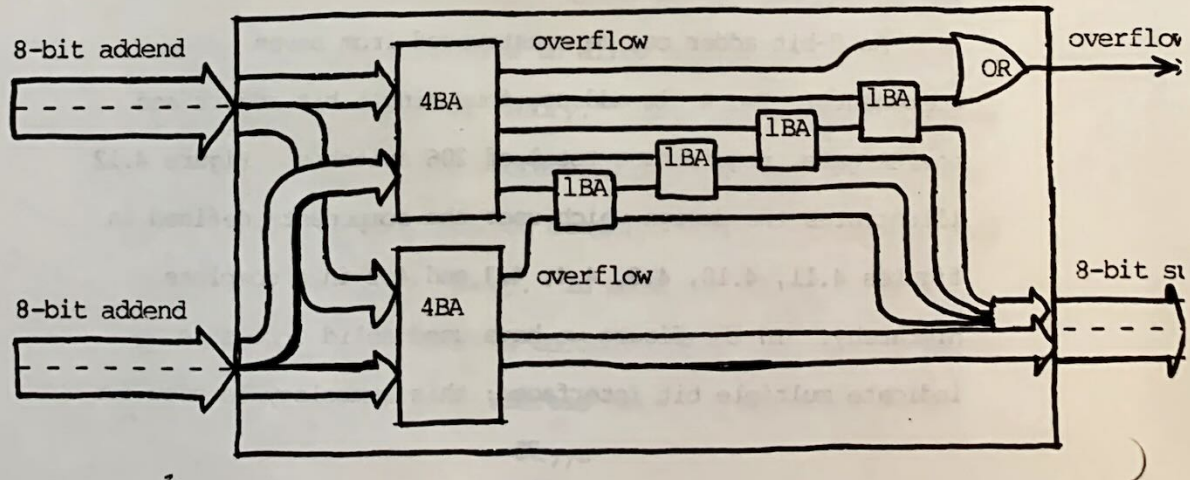


Figure 4.10: 8-Bit Adder Implementation

"busses," applicable when each bit is used in an orderly fashion which does not necessitate an elaboration of each bit interconnection.

To perform a sixteen bit addition which is typical of integer arithmetic supported by current machines, involves the use of eleven components with a total of 478 switches. This then is the adder which is one of the several major components of an Arithmetic Logic Unit.

DIGITAL ELECTRONIC STORAGE DEVICES

The property of computers that separates them from calculators is that they have memory. We have mentioned the significance of flip-flops as being able to store a single logical variable, and shown a design which would accommodate the storage of a single bit in a binary number. Binary numbers involve concatenated bit sequences so that in order to store an entire number, there must be a way of combining a set of flip-flops as a unit and a way of distinguishing between these defined units to allow access to one rather than another location of memory. These requirements involve the organization of bit storage in a two-dimensional array; its extent in each direction comprizes the "word length" and "number of words" of memory. In addition there is the requirement for an "address decoder" which can use a numeric value to determine which address select line to activate.

MEMORY REGISTERS

The organization of logic circuits like the one shown in figure 4.8 to effect storage for a set of switch indications representing a four-bit number is shown in figure 4.13.

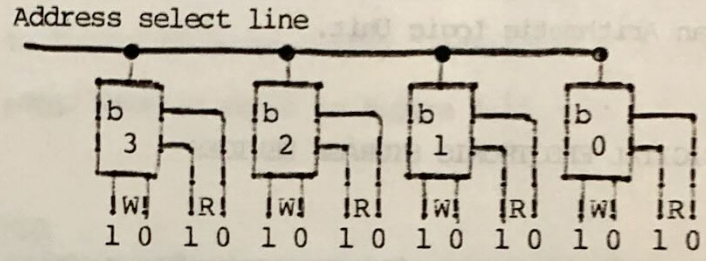


Figure 4.13: Single Four-Bit Register (or Word) of Memory

Such concatenated bits are usually referred to as "registers" or "words" of memory. You will notice that each bit in a word responds to the same address select line; this enables the entire set of four bits to all be activated at the same time as a single unit. Each bit can be set by activating the appropriate WRITE line. (The 1 and 0 associated with each W are used to indicate the set and reset capabilities shown in figure 4.9.) The READ lines are to be interpretted similarly. Usually registers are indicated merely by the juxtaposition of boxes as shown in figure 4.14 with the bit numbers assumed to range as indicated across the top. Particular bit values are indicated in the boxes.

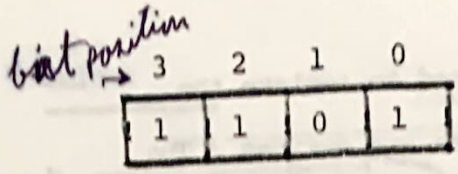


Figure 4.14: Standard 4-Bit Register Symbol Containing Value (13)

MEMORY ARRAYS

The organization of many registers or words of memory to form a block with many "locations" for storing numbers or other data is shown in figure 4.15. In this diagram, the

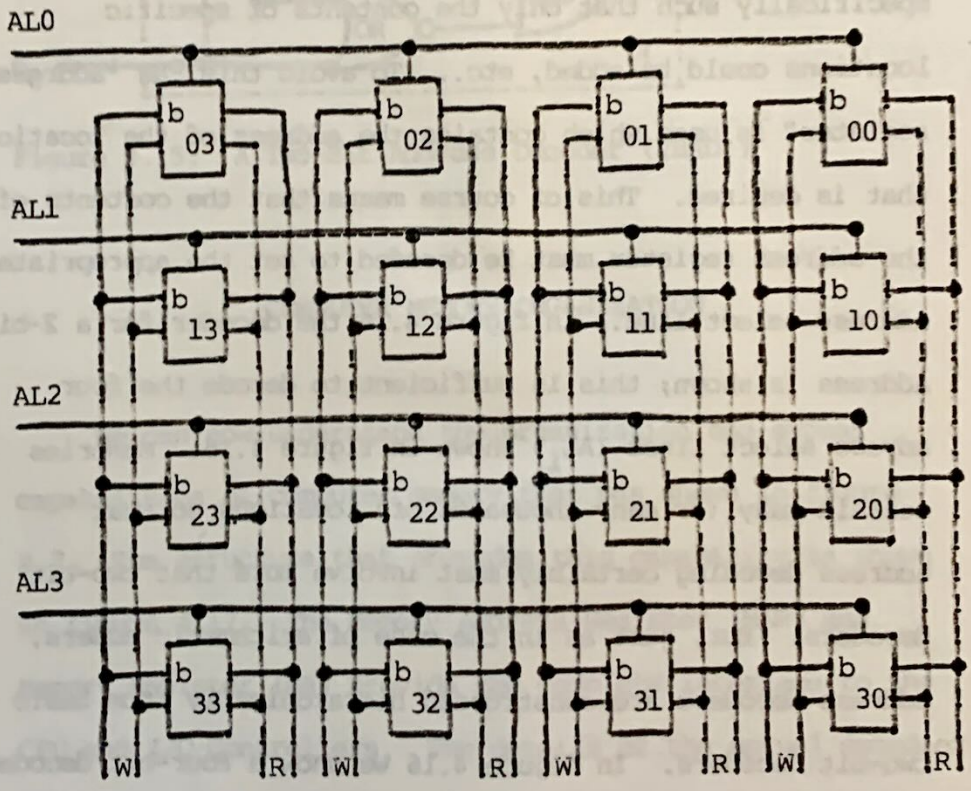


Figure 4.15: Memory Organization With Four Locations

address select lines (AL_i) are used to isolate the particular location whose contents are desired. The interface to reading and writing the contents of specific memory locations is exactly as described for the register shown in figure 4.13.

ADDRESS DECODING

A computer would not be a very flexible device if each location of memory had to be accessed (read or written) specifically such that only the contents of specific locations could be added, etc.. To avoid this, an "address register" is used which contains the address of the location that is desired. This of course means that the contents of the address register must be decoded to set the appropriate address select line. In figure 4.16 the decoder for a 2-bit address is shown; this is sufficient to decode the four adress select lines (AL_i) shown in figure 4.15. Memories contain many (or many thousands of) locations so that address decoding certainly must involve more that two-bit decoders. But, just as in the case of arithmetic adders, address decoders are constructed hierarchically from basic two-bit decoders. In figure 4.16 we show a Four-bit decoder constructed from five two-bit decoders. This decoder is able to address sixteen locations (or alternatively sixteen lower level decoders). An eight-bit decoder can be built from 17 four-bit decoders which can address 256 locations.

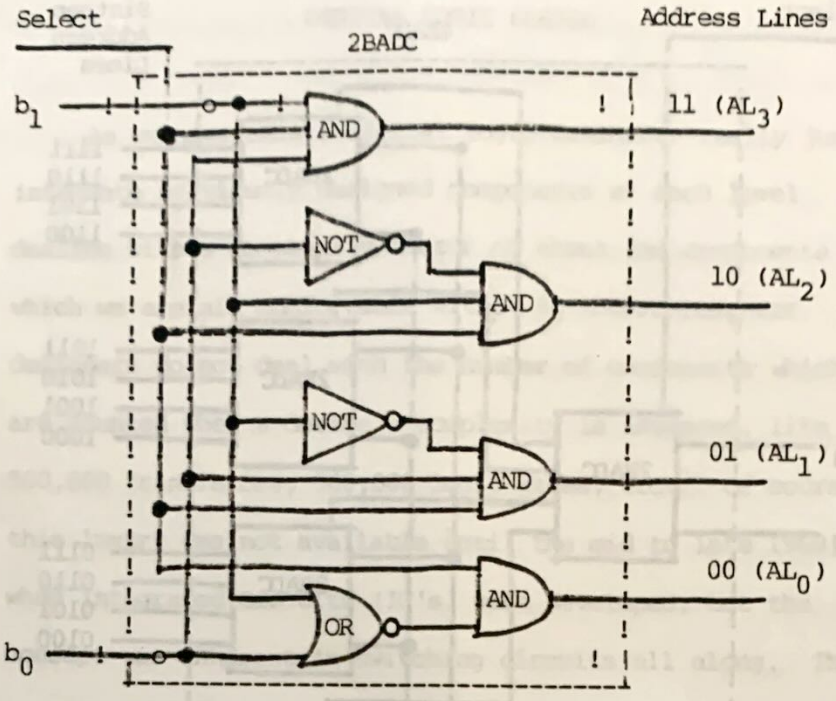


Figure 4.15: A Two-Bit Address Decoder (2BADC)

COMPUTER MEMORY ORGANIZATION

We can now understand the organization and access capabilities of computer memory that was shown in figure 2.7. The structure that provides this capability is shown in figure 4.17. The Memory Address Register (MAR) and Memory Register (MR) provide the complete interface to the CPU and I/O Controllers. The details of the actual decoding and access operations are handled independent of those devices.

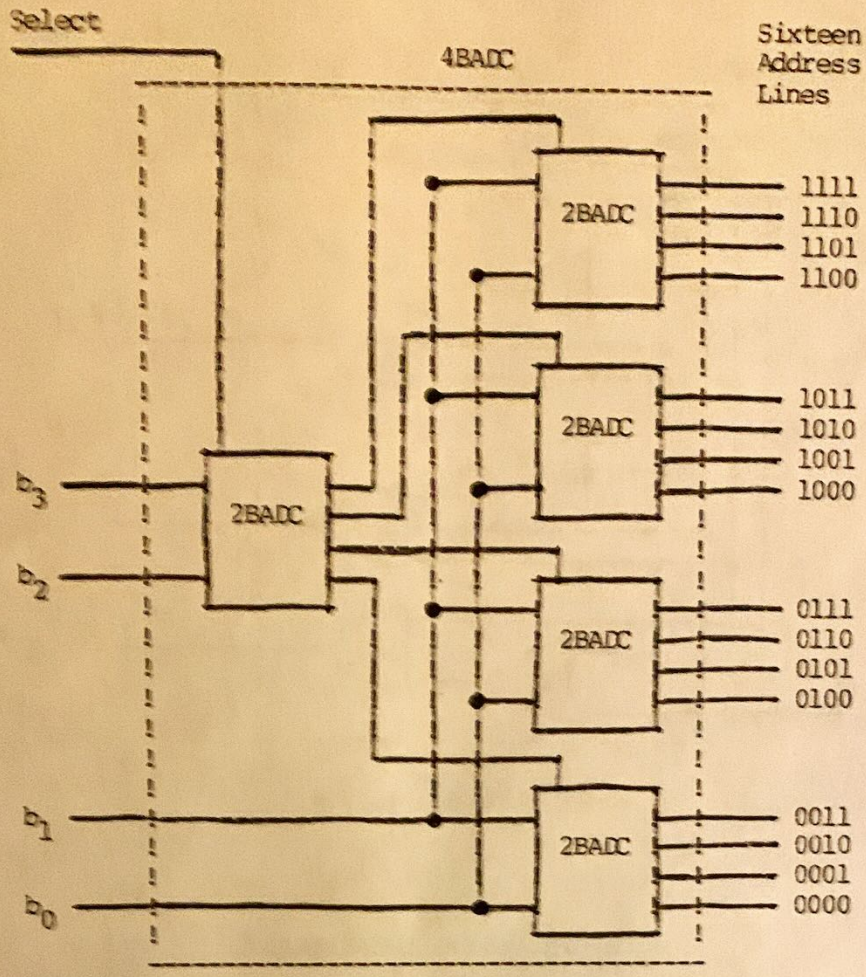


Figure 4.16: Four-Bit Address Decoder For 16 Locations

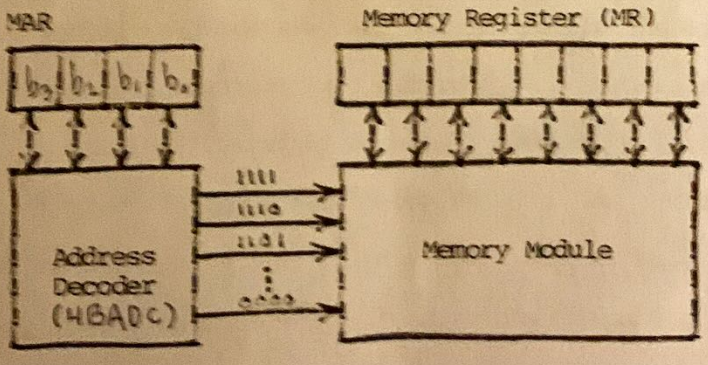


Figure 4.17: Computer Memory Organization

DIGITAL LOGIC DESIGN

As we have shown, digital logic designers really just integrate previously designed components at each level dealing with a complexity factor of about ten components which we are all comfortable with. So modern computer designers do not deal with the number of components which are counted when a device's complexity is assessed, like 500,000 transistors, 300,000 logic gates, etc.. Of course this luxury was not available until the mid to late 1960's when Integrated Circuits (IC's) were developed, but the concept was inherent in switching circuits all along. The mere practicalities that had to be overcome were primarily size and reliability. Size because of the difficulties in manipulating gigantic assemblies and reliability because of the required replaceability of failed components. Now we have gone full circle, scrapping an entire computer if a single bit fails! So you can see that cost of the component is also involved, and to a lesser extent power requirements. Solutions to all of these practical issues are the major breakthroughs that have made the current situation in computing.

DIGITAL DESIGN CONSIDERATIONS

The purpose and use of switching circuits is basically very different than the electric circuits used in analog

computers. In switching circuits, as we have seen, voltages have typically only two possible values: "high" and "low", "on" and "off", "1" and "0", or "true" and "false" depending on how they are interpretted. Of course during rapid transitions between states, there are continuous variations in voltages as in an analog circuit, but if values are not sampled immediately after transitions, this particular difficulty goes away. See figure 4.18 for a more precise illustration than that which we showed in figure 4.5 of what actually transpires in a circuit when a switch opens or closes. How long the delay must be to avoid these

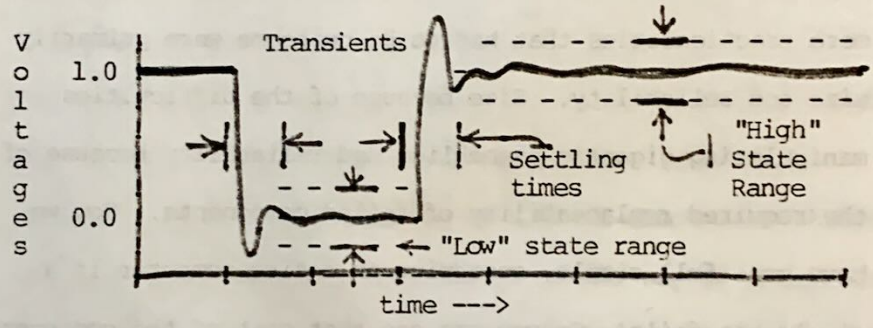


Figure 4.18: Wide Tolerances in Digital Signals

"transients" is a function of the technology employed. Electromagnets were slow; transistors employed in today's integrated circuits are fast, requiring as little as a few "nanoseconds" (billionths of a second) to attain equilibrium values.

An "oscillator" which generates an alternating "high" and "low" signal at a precise rate was developed and used as

a clock to "trigger" the simultaneous operation of all the switching circuits in a system so that this sampling function could be synchronized.

PRECISION WITHOUT COMPONENT CONSTRAINTS

The equilibrium voltages, even though conceptually two discrete possibilities ("1" and "0" throughout our earlier discussion of figure 4.5 and more recently 4.18), might also vary by fairly large amounts without making it difficult to distinguish between states as shown also in this figure. A voltage value between 0.83 and 1.37 might be sellected as representing one and values between -0.31 and 0.42 to represent zero for example, providing a tolerance that could be achieved by very inexpensive components. Thus the precise engineering discipline of digital electronic design was born which required very little in the way of precision from the components (other than the frequency of the clock). This is because voltages are sampled periodically rather than continuously and because wide tolerances are allowed on sampled voltage values.

The advantage of these digital rather than analog circuits like those employed by analog computers is much like the advantage of using an abacus over the use of lengths of rope to perform summations of large numbers: When lengths of rope, the components in a slide rule, or the current in an analog circuit are used, the precision of the

resulting arithmetic is limited by the precision of components. However, there is no relationship between the precision with which the beads or pebbles in an abacus are made and the precision of the arithmetic performed on the abacus. The same is true in digital computers.

In the next chapters we will witness the proliferation of these digital electronic devices.