Malvino

DIGITAL COMPUTER ELECTRONICS

AN INTRODUCTION TO MICROCOMPUTERS

SECOND EDITION

Digital Computer Electronics

An Introduction to Microcomputers

Second Edition

Albert Paul Malvino, Ph.D.

Gregg Division
McGraw-Hill Book Company

New York Atlanta Dallas St. Louis San Francisco
Auckland Bogotá Guatemala Hamburg Johannesburg
Lisbon London Madrid Mexico Montreal New Delhi
Panama Paris San Juan São Paulo Singapore Sydney
Tokyo Toronto

Sponsoring Editor: Paul Berk
Editing Supervisors: Tim Perrin and Larry Goldberg
Design and Art Supervisors: Nancy Axelrod and Meri Shardin
Production Supervisor: Priscilla Taguer

Text Designer: Ampersand Studio
Cover Designer: Ampersand Studio
Cover Illustrator: Jon Weiman
Technical Studio: Fine Line, Inc.

Library of Congress Cataloging in Publication Data

Malvino, Albert Paul.
Digital computer electronics.

Includes index.
1. Electronic digital computers.
2. Microcomputers. 3. INTEL 8085 (Computer)
I. Title.
TK7888.3.M337 1982 621.3819'58 82-8952
ISBN 0-07-039901-8 AACR2

Digital Computer Electronics:
An Introduction to Microcomputers, Second Edition

2 3 4 5 6 7 8 9 0 SEMBKP 8 9 8 7 6 5 4 3

ISBN 0-07-039901-8

Digital Computer Electronics

ALSO BY ALBERT P. MALVINO

Electronic Principles

Experiments for Electronic Principles (with G. Johnson)

Transistor Circuit Approximations

Experiments for Transistor Circuit Approximations

Resistive and Reactive Circuits

Electronic Instrumentation Fundamentals

Digital Principles and Applications (with D. Leach)

PREFACE

Textbooks on microprocessors and microcomputers are very often hard to understand. Sometimes it seems as if something important had been left out of the discussion; this book is my attempt at putting everything back in.

The early chapters of *Digital Computer Electronics*, Second Edition, cover digital theory and devices. In later chapters this information is applied to microprocessors, and finally, you will learn about the construction and operation of microcomputer systems. The only prerequisite to using this textbook is an understanding of diodes and transistors.

I have featured the 8085 microprocessor (an enhanced version of the 8080) because this 8-bit device is an ideal subject of study for a fundamental microcomputer textbook. Once you understand the 8085, you pass a major hurdle and things begin to make sense in the microcomputer world.

To help you master the 8085, you will first study an educational computer called SAP (simple-as-possible). This computer has three generations: SAP-1, SAP-2, and SAP-3. SAP-1 is a bare-bones computer built with TTL chips. You will see every wire, every signal, and every circuit used in this elementary computer. This will reinforce your grasp of digital electronics and prepare you to understand the more advanced computer concepts in SAP-2 and SAP-3. Many of the operational details of the 8080 and 8085 microprocessors are covered in SAP-2 and SAP-3. After studying these you will have learned almost the entire 8080/8085 instruction set.

The later chapters discuss advanced microcomputer topics such as handshaking, interrupts, memory shadows, and D/A and A/D conversion. When finished with this book, you will have a deep and solid understanding of microcomputer basics. With that kind of foundation you will find it relatively easy to branch out to systems that use other 8-bit, as well as 16-bit, microprocessors.

A correlated laboratory manual, *Experiments for Digital Computer Electronics* by Michael A. Miller, is available for use with this textbook. Early experiments cover the basics of digital electronics: gates, adders, flip-flops, and more. Later experiments are about program counters, instruction decoders, and accumulators. In the final experiments you assemble and program a SAP-1 computer.

During the preparation of this textbook, many people made valuable suggestions. I want to thank Charles Counts of Intel Corporation, Michael A. Miller of the DeVry Institute of Technology, William H. Murray of Broome Community College, Richard Raines of Shasta College, Michael Slater of Logical Services Incorporated, and the staff of the Sylvania Technical School.

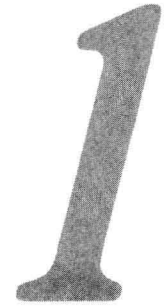
Albert Paul Malvino

A man of true science uses but few hard words,
and those only when none other will answer his purpose;
whereas the smatterer in science thinks that
by mouthing hard words he understands hard things.

Herman Melville

CONTENTS

Number Systems and Codes



Modern computers don't work with decimal numbers. Instead, they process *binary numbers*, groups of 0s and 1s. Why binary numbers? Because electronic devices are most reliable when designed for two-state (binary) operation. This chapter discusses binary numbers and other concepts needed to understand computer operation.

1-1 DECIMAL ODOMETER

René Descartes (1596–1650) said that the way to learn a new subject is to go from the known to the unknown, from the simple to the complex. Let's try it.

The Known

Everyone has seen an odometer (miles indicator) in action. When a car is new, its odometer starts with

00000

After 1 mile the reading becomes

00001

Successive miles produce 00002, 00003, and so on, up to

00009

A familiar thing happens at the end of the tenth mile. When the units wheel turns from 9 back to 0, a tab on this wheel forces the tens wheel to advance by 1. This is why the numbers change to

00010

Reset-and-Carry

The units wheel has reset to 0 and sent a carry to the tens wheel. Let's call this familiar action *reset-and-carry*.

The other wheels also reset and carry. After 999 miles the odometer shows

00999

What does the next mile do? The units wheel resets and carries, the tens wheel resets and carries, the hundreds wheel resets and carries, and the thousands wheel advances by 1, to get

01000

Digits and Strings

The numbers on each odometer wheel are called *digits*. The decimal number system uses ten digits, 0 through 9. In a decimal odometer, each time the units wheel runs out of digits, it resets to 0 and sends a carry to the tens wheel. When the tens wheel runs out of digits, it resets to 0 and sends a carry to the hundreds wheel. And so on with the remaining wheels.

One more point. A *string* is a group of characters (either letters or digits) written one after another. For instance, 734 is a string of 7, 3, and 4. Similarly, 2C8A is a string of 2, C, 8, and A.

1-2 BINARY ODOMETER

Binary means two. The binary number system uses only two digits, 0 and 1. All other digits (2 through 9) are thrown away. In other words, binary numbers are strings of 0s and 1s.

An Unusual Odometer

Visualize an odometer whose wheels have only two digits, 0 and 1. When each wheel turns, it displays 0, then 1, then

back to 0, and the cycle repeats. Because each wheel has only two digits, we call this device a *binary odometer*.

In a car a binary odometer starts with

0000 (zero)

After 1 mile, it indicates

0001 (one)

The next mile forces the units wheel to reset and carry; so the numbers change to

0010 (two)

The third mile results in

0011 (three)

What happens after 4 miles? The units wheel resets and carries, the second wheel resets and carries, and the third wheel advances by 1. This gives

0100 (four)

Successive miles produce

0101 (five)
0110 (six)
0111 (seven)

After 8 miles, the units wheel resets and carries, the second wheel resets and carries, the third wheel resets and carries, and the fourth wheel advances by 1. The result is

1000 (eight)

The ninth mile gives

1001 (nine)

and the tenth mile produces

1010 (ten)

(Try working out a few more readings on your own.)

You should have the idea by now. Each mile advances the units wheel by 1. Whenever the units wheel runs out of digits, it resets and carries. Whenever the second wheel runs out of digits, it resets and carries. And so for the other wheels.

Binary Numbers

A binary odometer displays binary numbers, strings of 0s and 1s. The number 0001 stands for 1, 0010 for 2, 0011 for 3, and so forth. Binary numbers are long when large amounts are involved. For instance, 101010 represents decimal 42. As another example, 111100001111 stands for decimal 3,855.

Computer circuits are like binary odometers; they count and work with binary numbers. Therefore, you have to learn to count with binary numbers, to convert them to decimal numbers, and to do binary arithmetic. Then you will be ready to understand how computers operate.

A final point. When a decimal odometer shows 0036, we can drop the leading 0s and read the number as 36. Similarly, when a binary odometer indicates 0011, we can drop the leading 0s and read the number as 11. With the leading 0s omitted, the binary numbers are 0, 1, 10, 11, 100, 101, and so on. To avoid confusion with decimal numbers, read the binary numbers like this: zero, one, one-zero, one-one, one-zero-zero, one-zero-one, etc.

1-3 NUMBER CODES

People used to count with pebbles. The numbers 1, 2, 3 looked like ●, ●●, ●●●. Larger numbers were worse: seven appeared as ●●●●●●●.

Codes

From the earliest times, people have been creating codes that allow us to think, calculate, and communicate. The decimal numbers are an example of a code (see Table 1-1). It's an old idea now, but at the time it was as revolutionary; 1 stands for ●, 2 for ●●, 3 for ●●●, and so forth.

Table 1-1 also shows the binary code. 1 stands for ●, 10 for ●●, 11 for ●●●, and so on. A binary number and a decimal number are equivalent if each represents the same amount of pebbles. Binary 10 and decimal 2 are equivalent because each represents ●●. Binary 101 and decimal 5 are equivalent because each stands for ●●●●●.

TABLE 1-1. NUMBER CODES

Decimal	Pebbles	Binary
0	None	0
1	●	1
2	●●	10
3	●●●	11
4	●●●●	100
5	●●●●●	101
6	●●●●●●	110
7	●●●●●●●	111
8	●●●●●●●●	1000
9	●●●●●●●●●	1001

Equivalence is the common ground between us and computers; it tells us when we're talking about the same thing. If a computer comes up with a binary answer of 101, equivalence means that the decimal answer is 5. As a start to understanding computers, memorize the binary-decimal equivalences of Table 1-1.

EXAMPLE 1-1

Figure 1-1*a* shows four light-emitting diodes (LEDs). A dark circle means that the LED is off; a light circle means it's on. To read the display, use this code:

(a) (b)

Fig. 1-1 LED display of binary numbers.

LED	Binary
Off	0
On	1

What binary number does Fig. 1-1*a* indicate? Fig. 1-1*b*?

SOLUTION

Figure 1-1*a* shows off-off-on-on. This stands for binary 0011, equivalent to decimal 3.

Figure 1-1*b* is off-on-off-on, decoded as binary 0101 and equivalent to decimal 5.

EXAMPLE 1-2

A binary odometer has four wheels. What are the successive binary numbers?

SOLUTION

As previously discussed, the first eight binary numbers are 0000, 0001, 0010, 0011, 0100, 0101, 0110, and 0111. On the next count, the three wheels on the right reset and carry; the fourth wheel advances by one. So the next eight numbers are 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111. The final reading of 1111 is equivalent to decimal 15. The next mile resets all wheels to 0, and the cycle repeats.

Being able to count in binary from 0000 to 1111 is essential for understanding the operation of computers.

TABLE 1-2. BINARY-TO-DECIMAL EQUIVALENCES

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

Therefore, you should memorize the equivalences of Table 1-2.

1-4 WHY BINARY NUMBERS ARE USED

The word "computer" is misleading because it suggests a machine that can solve only numerical problems. But a computer is more than an automatic adding machine. It can play games, translate languages, draw pictures, and so on. To suggest this broad range of application, a computer is often referred to as a *data processor*.

Program and Data

Data means names, numbers, facts, anything needed to work out a problem. Data goes into a computer, where it is processed or manipulated to get new information. Before it goes into a computer, however, the data must be coded in binary form. The reason was given earlier: a computer's circuits can respond only to binary numbers.

Besides the data, someone has to work out a *program*, a list of instructions telling the computer what to do. These instructions spell out each and every step in the data processing. Like the data, the program must be coded in binary form before it goes into the computer.

So the two things we must input to a computer are the program and the data. These are stored inside the computer before the processing begins. Once the computer run starts, each instruction is executed and the data is processed.

Hardware and Software

The electronic, magnetic, and mechanical devices of a computer are known as *hardware*. Programs are called *software*. Without software, a computer is a pile of "dumb" metal.

An analogy may help. A phonograph is like hardware and records are like software. The phonograph is useless without records. Furthermore, the music you get depends on the record you play. A similar idea applies to computers. A computer is the hardware and programs are the software. The computer is useless without programs. The program stored in the computer determines what the computer will do; change the program and the computer processes the data in a different way.

Transistors

Computers use *integrated circuits* (ICs) with thousands of transistors, either bipolar or MOS. The parameters (β_{dc}, I_{CO}, g_m, etc.) can vary more than 50 percent with temperature change and from one transistor to the next. Yet these computer ICs work remarkably well despite the transistor variations. How is it possible?

The answer is *two-state* design, using only two points on the load line of each transistor. For instance, the common two-state design is the cutoff-saturation approach; each transistor is forced to operate at either cutoff or saturation. When a transistor is cut off or saturated, parameter variations have almost no effect. Because of this, it's possible to design reliable two-state circuits that are almost independent of temperature change and transistor variations.

Transistor Register

Here's an example of two-state design. Figure 1-2 shows a transistor register. (A *register* is a string of devices that store data.) The transistors on the left are cut off because the input base voltages are 0 V. The dark shading symbolizes the cutoff condition. The two transistors on the right have base drives of 5 V.

The transistors operate at either saturation or cutoff. A base voltage of 0 V forces each transistor to cut off, while a base voltage of 5 V drives it into saturation. Because of this two-state action, each transistor stays in a given state until the base voltage switches it to the opposite state.

Another Code

Two-state operation is universal in digital electronics. By deliberate design, all input and output voltages are either low or high. Here's how binary numbers come in: low voltage represents binary 0, and high voltage stands for binary 1. In other words, we use this code:

Voltage	Binary
Low	0
High	1

For instance, the base voltages of Fig. 1-2 are low-low-high-high, or binary 0011. The collector voltages are high-high-low-low, or binary 1100. By changing the base voltages we can store any binary number from 0000 to 1111 (decimal 0 to 15).

Bit

Bit is an abbreviation for binary digit. A binary number like 1100 has 4 bits; 110011 has 6 bits; and 11001100 has 8 bits. Figure 1-2 is a 4-bit register. To store larger binary numbers, it needs more transistors. Add two transistors and you get a 6-bit register. With four more transistors, you'd have an 8-bit register.

Nonsaturated Circuits

Don't get the idea that all two-state circuits switch between cutoff and saturation. When a bipolar transistor is heavily saturated, extra carriers are stored in the base region. If the base voltage suddenly switches from high to low, the transistor cannot come out of saturation until these extra carriers have a chance to leave the base region. The time it takes for these carriers to leave is called the *saturation delay time* t_d. Typically, t_d is in nanoseconds.

In most applications the saturation delay time is too short to matter. But some applications require the fastest possible

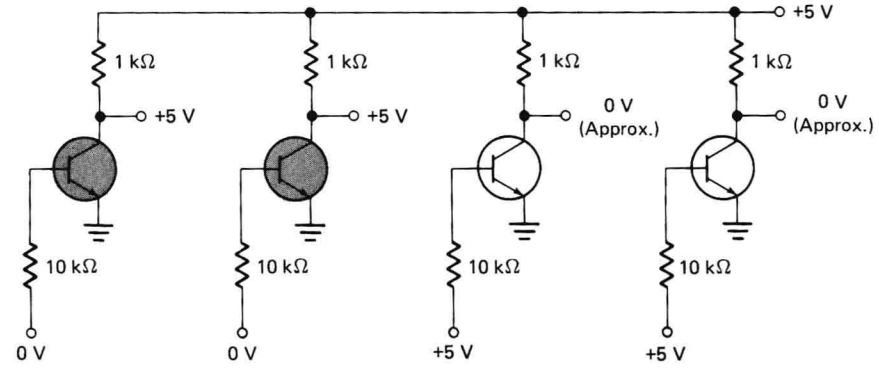


Fig. 1-2 Transistor register.

switching time. To get this maximum speed, designers have come up with circuits that switch from cutoff (or near cutoff) to a higher point on the load line (but short of saturation). These nonsaturated circuits rely on clamping diodes or heavy negative feedback to overcome transistor variations.

Remember this: whether saturated or nonsaturated circuits are used, the transistors switch between distinct points on the load line. This means that all input and output voltages are easily recognized as low or high, binary 0 or binary 1.

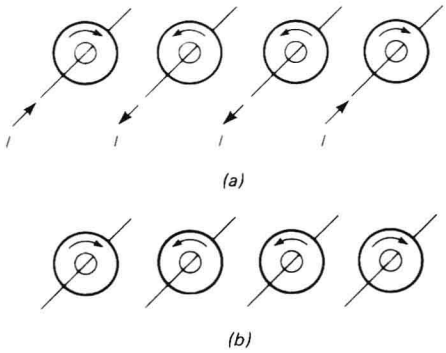


Fig. 1-3 Core register.

Magnetic Cores

In some digital computers, magnetic cores store binary data. Figure 1-3*a* shows a 4-bit core register. With the right-hand rule, you can see that conventional current into a wire produces a clockwise flux; reversing the current gives a counterclockwise flux. (The same result is obtained if electron-flow is assumed and the left-hand rule is used.)

The cores have rectangular hysteresis loops; this means that flux remains in a core even though the magnetizing current is removed (see Fig. 1-3*b*). This is why a core register can store binary data indefinitely. For instance, let's use the following code:

Flux	Binary
Counterclockwise	0
Clockwise	1

Then, the core register of Fig. 1-3*b* stores binary 1001, equivalent to decimal 9. By changing the magnetizing currents in Fig. 1-3*a* we can change the stored data.

To store larger binary numbers, add more cores. Two cores added to Fig. 1-3*a* result in a 6-bit register; four more cores give an 8-bit register.

The *memory* is one of the main parts of a computer. Some memories contain thousands of core registers. These registers store the program and data needed to run the computer.

Other Two-State Examples

The simplest example of a two-state device is the on-off switch. When this switch is closed, it represents binary 1; when it's open, it stands for binary 0.

Punched cards are another example of the two-state concept. A hole in a card stands for binary 1, the absence of a hole for binary 0. Using a prearranged code, a card-punch machine with a keyboard can produce a stack of cards containing the program and data needed to run a computer.

Magnetic tape can also store binary numbers. Tape recorders magnetize some points on the tape (binary 1), while leaving other points unmagnetized (binary 0). By a prearranged code, a row of points represents either a coded instruction or data. In this way, a reel of tape can store thousands of binary instructions and data for later use in a computer.

Even the lights on the control panel of a large computer are binary; a light that's on stands for binary 1, and one that's off stands for binary 0. In a 16-bit computer, for instance, a row of 16 lights allows the operator to see the binary contents in different computer registers. The operator can then monitor the overall operation and, when necessary, troubleshoot.

In summary, switches, transistors, cores, cards, tape, lights, and almost all other devices used with computers are based on two-state operation. This is why we are forced to use binary numbers when analyzing computer action.

EXAMPLE 1-3

Figure 1-4 shows a strip of magnetic tape. The black circles are magnetized points and the white circles unmagnetized points. What binary number does each horizontal row represent?

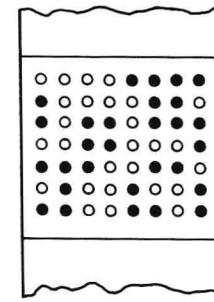


Fig. 1-4 Binary numbers on magnetic tape.

SOLUTION

The tape stores these binary numbers:

Row 1	00001111	Row 5	11100110
Row 2	10000110	Row 6	01001001
Row 3	10110111	Row 7	11001101
Row 4	00110001		

(*Note:* these binary numbers may represent either coded instructions or data.)

A string of 8 bits is called a *byte*. In this example, the magnetic tape stores 7 bytes. The first byte (row 1) is 00001111. The second byte (row 2) is 10000110. The third byte is 10110111. And so on.

A byte is the basic unit of data in computers. Most computers process data in strings of 8 bits or some multiple (16, 24, 32, and so on). Likewise, the memory stores data in strings of 8 bits or some multiple of 8 bits.

1-5 BINARY-TO-DECIMAL CONVERSION

You already know how to count to 15 using binary numbers. The next thing to learn is how to convert larger binary numbers to their decimal equivalents.

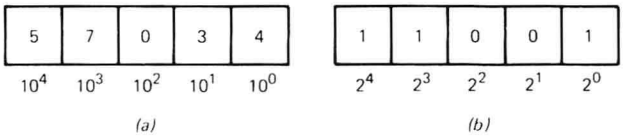


Fig. 1-5 (*a*) Decimal weights; (*b*) binary weights.

Decimal Weights

The decimal number system is an example of positional notation; each digit position has a *weight* or value. With decimal numbers the weights are units, tens, hundreds, thousands, and so on. The sum of all digits multiplied by their weights gives the total amount being represented.

For instance, Fig. 1-5*a* illustrates a decimal odometer. Below each digit is its weight. The digit on the right has a weight of 10^0 (units), the second digit has a weight of 10^1 (tens), the third digit a weight of 10^2 (hundreds), and so forth. The sum of all units multiplied by their weights is

$$(5 \times 10^4) + (7 \times 10^3) + (0 \times 10^2) + (3 \times 10^1) + (4 \times 10^0) = 50{,}000 + 7000 + 0 + 30 + 4 = 57{,}034$$

Binary Weights

Positional notation is also used with binary numbers because each digit position has a weight. Since only two digits are used, the weights are powers of 2 instead of 10. As shown in the binary odometer of Fig. 1-5*b*, these weights are 2^0 (units), 2^1 (twos), 2^2 (fours), 2^3 (eights), and 2^4 (sixteens). If longer binary numbers are involved, the weights continue in ascending powers of 2.

The decimal equivalent of a binary number equals the sum of all binary digits multiplied by their weights. For instance, the binary reading of Fig. 1-5*b* has a decimal equivalent of

$$(1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 16 + 8 + 0 + 0 + 1 = 25$$

Binary 11001 is therefore equivalent to decimal 25.

As another example, the byte 11001100 converts to decimal as follows:

$$(1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 128 + 64 + 0 + 0 + 8 + 4 + 0 + 0 = 204$$

So, binary 11001100 is equivalent to decimal 204.

Fast and Easy Conversion

Here's a streamlined way to convert a binary number to its decimal equivalent:

1. Write the binary number.
2. Write the weights 1, 2, 4, 8, . . . , under the binary digits.
3. Cross out any weight under a 0.
4. Add the remaining weights.

For instance, binary 1101 converts to decimal as follows:

1.	1 1 0 1	(Write binary number)
2.	8 4 2 1	(Write weights)
3.	8 4 ~~2~~ 1	(Cross out weights)
4.	8 + 4 + 0 + 1 = 13	(Add weights)

You can compress the steps even further:

1 1 0 1	(Step 1)
8 4 ~~2~~ 1 → 13	(Steps 2 to 4)

As another example, here's the conversion of binary 1110101 in compressed form:

1 1 1 0 1 0 1	
64 32 16 ~~8~~ 4 ~~2~~ 1 → 117	

Base or Radix

The *base* or *radix* of a number system equals the number of digits it has. Decimal numbers have a base of 10 because digits 0 through 9 are used. Binary numbers have a base of 2 because only the digits 0 and 1 are used. (In terms of an odometer, the base or radix is the number of digits on each wheel.)

A subscript attached to a number indicates the base of the number. 100_2 means binary 100. On the other hand, 100_{10} stands for decimal 100. Subscripts help clarify equations where binary and decimal numbers are mixed. For instance, the last two examples of binary-to-decimal conversion can be written like this:

$$1101_2 = 13_{10}$$
$$1110101_2 = 117_{10}$$

In this book we will use subscripts when necessary for clarity.

1-6 MICROPROCESSORS

What is inside a computer? What is a microprocessor? What is a microcomputer?

Computer

The five main sections of a computer are input, memory, arithmetic and logic, control, and output. Here is a brief description of each.

Input This consists of all the circuits needed to get programs and data into the computer. In some computers the input section includes a typewriter keyboard that converts letters and numbers into strings of binary data.

Memory This stores the program and data before the computer run begins. It also can store partial solutions during a computer run, similar to the way we use a scratchpad while working out a problem.

Control This is the computer's center of gravity, analogous to the conscious part of the mind. The control section directs the operation of all other sections. Like the conductor of an orchestra, it tells the other sections what to do and when to do it.

Arithmetic and logic This is the number-crunching section of the machine. It can also make logical decisions. With control telling it what to do and with memory feeding it data, the arithmetic-logic unit (ALU) grinds out answers to number and logic problems.

Output This passes answers and other processed data to the outside world. The output section usually includes a video display to allow the user to see the processed data.

Microprocessor

The control section and the ALU are often combined physically into a single unit called the *central processing unit* (CPU). Furthermore, it's convenient to combine the input and output sections into a single unit called the *input-output* (I/O) *unit*. In earlier computers, the CPU, memory, and I/O unit filled an entire room.

With the advent of integrated circuits, the CPU, memory, and I/O unit have shrunk dramatically. Nowadays the CPU can be fabricated on a single semiconductor chip called a *microprocessor*. In other words, a microprocessor is nothing more than a CPU on a chip.

Likewise, the I/O circuits and memory can be fabricated on chips. In this way, the computer circuits that once filled a room now fit on a few chips.

Microcomputer

As the name implies, a *microcomputer* is a small computer. More specifically, a microcomputer is a computer that uses a microprocessor for its CPU. The typical microcomputer has three kinds of chips: microprocessor (usually one chip), memory (several chips), and I/O (one or more chips).

If a small memory is acceptable, a manufacturer can fabricate all computer circuits on a single chip. For instance, the 8048 from Intel Corporation is a one-chip microcomputer with an 8-bit CPU, 1,088 bytes of memory, and 27 I/O lines.

Powers of 2

Microprocessor design started with 4-bit devices, then evolved to 8- and 16-bit devices. In our later discussions of microprocessors, powers of 2 keep coming up because of the binary nature of computers. For this reason, you should study Table 1-3. It lists the powers of 2 encountered in microcomputer analysis. As shown, the abbreviation K stands for 1,024 (approximately 1,000).† Therefore, 1K means 1,024, 2K stands for 2,048, 4K for 4,096, and so on.

Some personal microcomputers have 64K memories that can store up to 65,536 bytes.

TABLE 1-3. POWERS OF 2

Powers of 2	Decimal equivalent	Abbreviation
2^0	1	
2^1	2	
2^2	4	
2^3	8	
2^4	16	
2^5	32	
2^6	64	
2^7	128	
2^8	256	
2^9	512	
2^{10}	1,024	1K
2^{11}	2,048	2K
2^{12}	4,096	4K
2^{13}	8,192	8K
2^{14}	16,384	16K
2^{15}	32,768	32K
2^{16}	65,536	64K

† The abbreviations 1K, 2K, and so on, became established before K- for *kilo-* was in common use. Retaining the capital K serves as a useful reminder that K only approximates 1,000.

1-7 DECIMAL-TO-BINARY CONVERSION

Next, you need to know how to convert from decimal to binary. After you know how it's done, you will be able to understand how circuits can be built to convert decimal numbers into binary numbers.

Double-Dabble

Double-dabble is a way of converting any decimal number to its binary equivalent. It requires successive division by 2, writing down each quotient and its remainder. The remainders are the binary equivalent of the decimal number. The only way to understand the method is to go through an example, step by step.

Here is how to convert decimal 13 to its binary equivalent.
Step 1. Divide 13 by 2, writing your work like this:

$$\begin{array}{rl} 6 & 1 \rightarrow \text{(first remainder)} \\ 2\overline{)13} & \end{array}$$

The quotient is 6 with a remainder of 1.
Step 2. Divide 6 by 2 to get

$$\begin{array}{rl} 3 & 0 \rightarrow \text{(second remainder)} \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array}$$

This division gives 3 with a remainder of 0.
Step 3. Again you divide by 2:

$$\begin{array}{rl} 1 & 1 \rightarrow \text{(third remainder)} \\ 2\overline{)3} & 0 \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array}$$

Here you get a quotient of 1 and a remainder of 1.
Step 4. One more division by 2 gives

$$\begin{array}{rl} & \text{Read down} \\ 0 & 1 \\ 2\overline{)1} & 1 \\ 2\overline{)3} & 0 \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array} \quad \Big\downarrow$$

In this final division, 2 does not divide into 1; therefore, the quotient is 0 with a remainder of 1.

Whenever you arrive at a quotient of 0 with a remainder of 1, the conversion is finished. The remainders when read downward give the binary equivalent. In this example, binary 1101 is equivalent to decimal 13.

Double-dabble works with any decimal number. Progressively divide by 2, writing each quotient and its remainder. When you reach a quotient of 0 and a remainder of 1, you are finished; the remainders read downward are the binary equivalent of the decimal number.

Streamlined Double-Dabble

There's no need to keep writing down 2 before each division because you're always dividing by 2. From now on, here's how to show the conversion of decimal 13 to its binary equivalent:

$$\begin{array}{rl} 0 & 1 \\ \overline{)1} & 1 \\ \overline{)3} & 0 \\ \overline{)6} & 1 \\ 2\overline{)13} & \end{array} \quad \Big\downarrow$$

EXAMPLE 1-4

Convert decimal 23 to binary.

SOLUTION

The first step in the conversion looks like this:

$$\begin{array}{rl} 11 & 1 \\ 2\overline{)23} & \end{array}$$

After all divisions, the finished work looks like this:

$$\begin{array}{rl} 0 & 1 \\ \overline{)1} & 0 \\ \overline{)2} & 1 \\ \overline{)5} & 1 \\ \overline{)11} & 1 \\ 2\overline{)23} & \end{array} \quad \Big\downarrow$$

This says that binary 10111 is equivalent to decimal 23.