A thesis submitted in partial satisfaction of the requirements for the degree of Master of Science in Mathematics

by

John Stuart Collins

June, 1974
The thesis of John Stuart Collins is approved:

California State University, Northridge
June, 1974
To Albert and Martha Collins
They know why
ACKNOWLEDGMENT

I wish to express my gratitude to Mr. John W. McGhee for his ideas, suggestions and encouragement in the preparation of this manuscript and to Dr. Viggo (Pete) Hansen for his inspiration and friendship. Thanks are also due to my wife Rose Mary for her patience, understanding and the use of her typewriter.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>x</td>
</tr>
<tr>
<td>Abstract</td>
<td>xi</td>
</tr>
<tr>
<td><strong>PART I - HISTORY</strong></td>
<td>1</td>
</tr>
<tr>
<td>Chapter 1 - The History of Numbers and Numerals</td>
<td>2</td>
</tr>
<tr>
<td>Section 1 - Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Section 2 - Origins of numeration</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1 - Magic square</td>
<td>4</td>
</tr>
<tr>
<td>Section 3 - Origins of zero</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2 - Greek numeration system</td>
<td>9</td>
</tr>
<tr>
<td>Figure 3 - Sumerian numeral</td>
<td>9</td>
</tr>
<tr>
<td>Section 4 - Development of numeration systems</td>
<td>10</td>
</tr>
<tr>
<td>Chapter 2 - The History of Computation</td>
<td>19</td>
</tr>
<tr>
<td>Section 1 - Computations origin</td>
<td>19</td>
</tr>
<tr>
<td>Section 2 - Computation by the Abacus</td>
<td>20</td>
</tr>
<tr>
<td>Section 3 - Multiplication and division</td>
<td>22</td>
</tr>
<tr>
<td>Section 4 - Origins of symbols for operations</td>
<td>24</td>
</tr>
<tr>
<td>Section 5 - Prelude to modern mathematics</td>
<td>25</td>
</tr>
<tr>
<td>Section 6 - Development of computing machines</td>
<td>27</td>
</tr>
<tr>
<td><strong>PART II - LOGIC</strong></td>
<td>33</td>
</tr>
<tr>
<td>Chapter 3 - The Binary Concept</td>
<td>34</td>
</tr>
<tr>
<td>Section 1 - Introduction</td>
<td>34</td>
</tr>
<tr>
<td>Section 2 - Counting in the binary system</td>
<td>34</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Figure/Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Positional value of a binary number</td>
<td>37</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Binary, decimal equivalents</td>
<td>38</td>
</tr>
<tr>
<td>Section 3</td>
<td>Conversion of decimal to binary</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Subtraction of powers method</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Repeated division method</td>
<td>38</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Decimal fractions to binary fractions</td>
<td>39</td>
</tr>
<tr>
<td>Section 4</td>
<td>Addition and subtraction of binary numbers</td>
<td>40</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Addition (base two)</td>
<td>40</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Subtraction (base two)</td>
<td>40</td>
</tr>
<tr>
<td>Section 5</td>
<td>Multiplication and division of binary numbers</td>
<td>41</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Multiplication (base two)</td>
<td>41</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Division (base two)</td>
<td>41</td>
</tr>
<tr>
<td>Section 6</td>
<td>Trinary (ternary) numeration system</td>
<td>42</td>
</tr>
<tr>
<td>Section 7</td>
<td>Octal numeration system</td>
<td>42</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Repeated division method</td>
<td>43</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Decimal, octal, binary equivalents</td>
<td>43</td>
</tr>
<tr>
<td>Section 8</td>
<td>Alphanumeric codes</td>
<td>45</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Binary code</td>
<td>45</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Octal code</td>
<td>47</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Boolean Algebra</td>
<td>49</td>
</tr>
<tr>
<td>Section 1</td>
<td>Introduction</td>
<td>49</td>
</tr>
<tr>
<td>Section 2</td>
<td>Definition of a Boolean Algebra</td>
<td>50</td>
</tr>
<tr>
<td>Section 3</td>
<td>Venn diagrams</td>
<td>54</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Figure 1 - A set &amp; its complement</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2 - Intersecting sets</td>
<td>55</td>
</tr>
<tr>
<td>Figure 3 - Union of sets</td>
<td>56</td>
</tr>
<tr>
<td>Figure 4 - Union &amp; intersecting sets</td>
<td>56</td>
</tr>
<tr>
<td>Section 4 - Truth table construction and use</td>
<td>57</td>
</tr>
<tr>
<td>Figure 5 - Universal table (base two)</td>
<td>58</td>
</tr>
<tr>
<td>Figure 6 - Proof of distributive property</td>
<td>59</td>
</tr>
<tr>
<td>Section 5 - Boolean Algebra to switching circuits</td>
<td>59</td>
</tr>
<tr>
<td>Figure 7 - Switching circuits</td>
<td>60</td>
</tr>
<tr>
<td>Figure 8 - Logic symbols (gates)</td>
<td>61</td>
</tr>
<tr>
<td>Figure 9 - Truth table (Boolean operators)</td>
<td>62</td>
</tr>
<tr>
<td>Figure 10 - Gating identities</td>
<td>62</td>
</tr>
<tr>
<td>Figure 11 - Table of functions, symbols, &amp; expression</td>
<td>63</td>
</tr>
<tr>
<td>Chapter 5 - The Computer</td>
<td>66</td>
</tr>
<tr>
<td>Section 1 - Introduction</td>
<td>66</td>
</tr>
<tr>
<td>Figure 1 - Comparison (analog to digital)</td>
<td>67</td>
</tr>
<tr>
<td>Section 2 - Parts of the digital computer</td>
<td>68</td>
</tr>
<tr>
<td>Figure 2 - Typical units of computer</td>
<td>68</td>
</tr>
<tr>
<td>Figure 3 - Comparison (series to parallel signal)</td>
<td>73</td>
</tr>
<tr>
<td>Section 3 - The computer word</td>
<td>73</td>
</tr>
<tr>
<td>Figure 4 - Typical computer word</td>
<td>75</td>
</tr>
<tr>
<td>Section 4 - Basic elements of control unit</td>
<td>77</td>
</tr>
<tr>
<td>Figure 5 - Control unit</td>
<td>78</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 5 - Address selection</td>
<td>80</td>
</tr>
<tr>
<td>Figure 6 - Memory storage unit</td>
<td>81</td>
</tr>
<tr>
<td>Section 6 - Operation of a typical arithmetic unit</td>
<td>81</td>
</tr>
<tr>
<td>Figure 7 - Arithmetic unit</td>
<td>83</td>
</tr>
<tr>
<td>Figure 8 - Full serial adder</td>
<td>85</td>
</tr>
<tr>
<td>Figure 9 - Half adder (1)</td>
<td>86</td>
</tr>
<tr>
<td>Figure 10 - Half adder (2)</td>
<td>86</td>
</tr>
<tr>
<td>Figure 11 - Half adder (3)</td>
<td>86</td>
</tr>
</tbody>
</table>

**PART III - PROGRAMMING** | 90   |
| Chapter 6 - Flowcharting | 91   |
| Section 1 - Introduction | 91   |
| Section 2 - Some figures and what they mean | 91   |
| Section 3 - Straight line programming | 93   |
| Section 4 - Program branching | 94   |
| Chapter 7 - A Computer Language (BASIC) | 97   |
| Section 1 - Introduction | 97   |
| Section 2 - Commands | 97   |
| Section 3 - Operators | 99   |
| Section 4 - Statements | 101  |
| Section 5 - Special characters | 107  |
| Section 6 - Functions | 108  |
| Section 7 - Matrices | 109  |
# Table of Contents

**PART IV - APPLICATIONS**

<table>
<thead>
<tr>
<th>Chapter 8 - Number Theory</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1 - Introduction</td>
<td>121</td>
</tr>
<tr>
<td>Section 2 - Divisors and division algorithm</td>
<td>121</td>
</tr>
<tr>
<td>Figure 1 - Division algorithm flowchart</td>
<td>123</td>
</tr>
<tr>
<td>Program 1 - Division algorithm</td>
<td>124</td>
</tr>
<tr>
<td>Section 3 - Prime numbers</td>
<td>126</td>
</tr>
<tr>
<td>Program 2 - Goldbach's Conjecture</td>
<td>127</td>
</tr>
<tr>
<td>Program 3 - Prime factorization</td>
<td>131</td>
</tr>
<tr>
<td>Section 4 - Greatest common factor</td>
<td>132</td>
</tr>
<tr>
<td>Figure 2 - Euclidean Algorithm</td>
<td>133</td>
</tr>
<tr>
<td>Program 4 - Greatest common factor</td>
<td>134</td>
</tr>
<tr>
<td>Section 5 - Least common multiple</td>
<td>135</td>
</tr>
<tr>
<td>Program 5 - Least common multiple</td>
<td>136</td>
</tr>
<tr>
<td>Section 6 - The value of number theory</td>
<td>137</td>
</tr>
</tbody>
</table>

**Chapter 9 - Game Simulation**

<table>
<thead>
<tr>
<th>Section 1 - Introduction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 2 - Background</td>
<td>140</td>
</tr>
<tr>
<td>Program 6 - Viking</td>
<td>141</td>
</tr>
<tr>
<td>Program 7 - Viking solution</td>
<td>147</td>
</tr>
<tr>
<td>Program 8 - Viking solution</td>
<td>150</td>
</tr>
</tbody>
</table>

**BIBLIOGRAPHY**

**INDEX**
PREFACE

Since the advent of modern mathematics in the late 1800's, no one element has so revolutionized the course of mathematical thought as did the invention of the computer. The modern day computer has taken the tedious rigor of trial and error checks out of the present day mathematician's life. A young student of mathematics might ask how the computer evolved and where its origins lay? He might also ask how the computer functions and how the mathematician uses it as a tool in his work? It is the purpose of this text to answer these questions. I cannot hope to completely cover a topic so vast in content as this is. I do hope the student will, through this text, obtain a basic understanding and appreciation of numbers and computers, and that this understanding will provide a solid foundation for further studies in the field of computers and their association with mathematics.

J.S.C.

Northridge, California

June, 1974
ABSTRACT

COMPUTERS:
EVOLUTION AND APPLICATIONS
A Textbook
by
John Stuart Collins
Master of Science in Mathematics
June, 1974

This is a textbook for use in secondary education to introduce the history, development, concept and use of computers to students desiring a basic understanding of computers and their relationship to mathematics.

The text is broken into four main parts. Part I deals with the history of computers and how they evolved from our basic numeration system. Part II deals with the logical mathematical system of which the basic computer is comprised. This part also describes a basic digital computer and its internal units. Part III deals with programming, including flowcharts and a computer language (BASIC). Part IV presents some computer applications in number theory and game simulation, which includes a fascinating application to space travel.

The text offers exercises at the end of each chapter and a complete bibliography for the student wishing to pursue any subject matter further.
PART I

HISTORY

Mathematics is the alphabet with which God has written the Universe.

Galileo Galilei
1564-1642
CHAPTER 1
The History of Numbers and Numerals

Introduction. Most progressive developments of mankind have come about through a specific design, seldom by accident. Either the need or desirability, in most cases, dictated the invention. Progress is the never ending creation of new developments based in part or in whole upon previous products and results. The computer is just another link in this progression. Its development would not have been possible had it not been for previous technological advances and these advances were themselves a result of prior scientific endeavors.

The logical place to begin in tracing the history of the computer is with the events and knowledge that led up to the development of our own numeration system. We shall see how this eventually led to the development of the computer.

Origins of numeration. It is commonly recognized that our present day numeration system is of a comparatively recent origin. Its roots can be found in ancient Babylon, Greece, and India.

Prior to the advent of Christianity the number of numeration systems varied as did the languages of that period. These systems were primarily concerned with arithmetical computations involved in daily living such as time and dates, business transactions, surveying land, weights and measures, construction and most important of all, at that time anyway, the accounting of one's wealth. Most of these systems were non-positional, meaning the
order of the symbols did not affect the quantity involved.

The Babylonian sexagesimal system was one of the few positional systems in existence but did not possess the concept of zero. The early Greeks chose the Babylonian sexagesimal system to express their fractions, rather than adopting the unit-fraction system used by the Egyptians.

Many modern scholars rarely look beyond Greek mathematics which spanned a time interval from at least 600 B.C. to at least 600 A.D. and traveled from Ionia to the toe of Italy, from Athens to Alexandria, as well as other parts of the civilized world. The Greeks were the teachers during this period of time, the rest of the population of the Western World were mostly students of the Greeks. Throughout its long history, ancient Rome contributed little to science or philosophy and even less to mathematics. Our present numeration system did not evolve out of ancient Rome nor did it evolve from Greek mathematics, though there is some thought that the ancient Greeks did influence its evolution.

The question arises, at this point, as to where our present day numeration system came from, if not the Greco-Roman world? The commonly used title of the numeration system provides the answer and to explore the question we have to journey to China and India where the Hindu-Arabic numeration system appears to have originated. It should be noted that there still exists a great deal of debate among scholars about the influence of Babylonian and Greek mathematics on China and India.
The earliest recorded Chinese mathematical work has a much disputed date of 300 B.C. placed upon it and appears to be authored by several Chinese scholars. The text concerns astronomical observations and calculations, a subject of much study by early mathematicians.

The Chinese were also especially fond of patterns. Therefore, it is not surprising that the first record (of ancient but unknown origin) of a magic square appeared there. The square, shown in Figure 1, was supposedly brought to man by a turtle from the River Lo in the days of the legendary Emperor Yii, reputed to be a hydraulic engineer.

![Figure 1](image)

China may have significantly modified the development of mathematics but for some unfortunate breaks in the stream of their scholastic thought. In 213 B.C., for example, the Chinese emperor ordered the burning of all books. Even so it would be a pretty fair conjecture to say that before 400 A.D. more mathematics came out of China then went in. For later periods this question becomes more difficult to answer.

1 Histories of mathematics generally devote little space to Chinese contributions. Some exceptions are D.E. Smith, History of Mathematics, and J.E. Hofmann, Geschichte der Mathematik.

Not long after the era of the Egyptian pyramid builders, India was occupied by Aryan invaders who introduced and developed the Sanskrit literature. This form of literature was used by the great religious teacher, Buddha, who was active in India at about the same time that Pythagoras (ca. 580-500 B.C.) is said to have visited there. Conjecture on what they may have learned or passed on to each other would be highly unfounded at this time. Some archaeologists and historians are still pondering this question.

The fall of the Western Roman Empire is traditionally placed in the year 476 A.D.: It was in this year that Aryabhata, author of one of the oldest Indian mathematical texts, was born. It can be assumed that mathematical activity most probably was taking place in India long before his time.

The establishment of the dynasty of King Gupta (290 A.D.) marked the beginning of a Renaissance in the Sanskrit culture, and the Siddhāntas, or systems (of astronomy), seem to have been an outcome of this revival. Five different versions of the Siddhāntas are known by name, one of which is still extant. The Pauliśha Siddhānta, which dates from about 380 A.D., was summarized by the Hindu mathematician Varahamihira (505 A.D.) and was frequently referred to by the Arabic scholar Al-Biruni (973-1048 A.D.), who suggested a Greek origin or influence. The symbols used by Varahamihira are referred to as the Brahmi.

An English translation by Burgess and Waitney, together with extensive notes, was published in the Journal of the American Oriental Society (1860), pp. 141-498.
characters, which resembled the alphabetic cipherization in the Greek Ionian system. One might wonder if it was only coincidence that the change in India took place shortly after the period in Greece when the Herodianic numerals were replaced by the Ionian.

Our present day notation for integers are direct descendents of the Brahmi ciphered numerals. Through the use of the positional principle, the ciphers for the first nine units can also serve as the ciphers for the corresponding multiples of ten. The original Brahmi ciphers went beyond the first nine. Unfortunately, it is not known when the reduction to nine ciphers occurred. It appears that the change took place in India, but the inspiration for the change has not been resolved. The really sad part about the new Hindu system is that they did not apply the new numeration for integers to the realm of decimal fractions. Hence, the chief potential advantage of the change from Ionian notation was lost.

It should be noted that the reference to nine symbols, rather than ten, implies that the Hindus arrived at the point where there was a need for the introduction of a notation for a missing position, a zero symbol. The earliest reference to zero in India was found in an inscription of 876 A.D., more than two centuries after the first reference to the other nine numerals. It is possible that zero originated in the Greek world and was transmitted to India after the decimal positional system had already been established there. With this tenth numeral in the Hindu notation

the modern system of numeration for integers was completed. Although these ancient forms differ considerably from ours, the principles were established.

There were three basic principles, all of ancient origin: (1) a decimal base; (2) a positional notation; and (3) a ciphered form for each of the ten numerals. Not one of these three were due originally to the Hindus, but it is presumed that they are responsible for linking the three to form the modern system of numeration.

About 766 A.D. the Hindu system was brought to Baghdad and adopted by the Arabs, who played an important role in familiarizing other parts of the world with it. The work the Arabs brought was known as the Sindhind, another astronomical-mathematical treatise. It is generally thought that this was one of the Hindu Siddhântas. This work was translated into Arabic about 775 A.D. The Arabs never claimed they originated the Hindu system but always recognized their indebtedness to the Hindus both for the numeral forms and for the distinguishing feature of place value. Many authors of the past half century did not research enough to note this fact and therefore believed that the Arabs invented the numeration system we use. These authors referred to the symbols as Arabic numerals and to the system as the Arabic numeration system. Fortunately, modern day texts have corrected this oversight.

The famous Persian mathematician al-Khowarizmi, writing about 825 A.D., gave an account of the system in which he specifically ascribed it to the Hindus. Unfortunately, the original Arabic
work by al-Khowarizmi is lost; a twelfth-century Latin translation, probably by Adelard of Bath an English monk, is extant. The opening words of the translation "algoritmi dixit" (al-Khowarizmi says), gave rise to the term "algorithm" or "algorism" for various sorts of computational processes.

Origins of zero. The Classical Greeks never recognized zero as a number. Their alphanumeric numerals began with one, to which their number lore assigned such attributes as "male," "reason," "the essence of number," "the origin of all things," and "the divine principle." In the Greek system shown in Figure 2, apostrophes were often used to show that the letter was to be interpreted as a numeral. Obviously this is a base-ten system, yet it is non-positional. If the number 248 is represented from the table, the representation could be $\sigma\mu\eta$ (200+40+8). Since this system is purely additive, it could be represented just as easily as $\eta\mu\sigma$ (8+40+200), understanding that addition is commutative. In fact there are six different ways to represent the number.

Representing the number 305 illustrates the lack of a zero place-holder. Obviously there are two ways to write this number: $\epsilon\tau$ (5+300) or $\tau\epsilon$ (300+5). Is there need for a zero symbol in non-positional numeration?

Positional or place-value numeration could not function without a symbol for an empty place or position. In pre-Babylonian

5

Sumer (about 3,500 B.C.) a positional system with base sixty was in use. An empty place was indicated by actually leaving an empty place in the numeral. Therefore the Sumerian numeral shown in Figure 3 might stand for \((4 \times 60) + (5 \times 1) = 245\). It could just as easily stand for \((4 \times 60^2) + (5 \times 1) = 14,405\). Only a careful study of the context might reveal the intended meaning.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>₁</td>
<td>₂</td>
<td>₃</td>
<td>₄</td>
<td>₅</td>
<td>₆</td>
<td>₇</td>
<td>₈</td>
<td>₉</td>
<td>₀</td>
</tr>
<tr>
<td>₁₀</td>
<td>₂₀</td>
<td>₃₀</td>
<td>₄₀</td>
<td>₅₀</td>
<td>₆₀</td>
<td>₇₀</td>
<td>₈₀</td>
<td>₉₀</td>
<td>₀₀</td>
</tr>
<tr>
<td>₁₀₀</td>
<td>₂₀₀</td>
<td>₃₀₀</td>
<td>₄₀₀</td>
<td>₅₀₀</td>
<td>₆₀₀</td>
<td>₇₀₀</td>
<td>₈₀₀</td>
<td>₉₀₀</td>
<td>₁₀₀₀</td>
</tr>
</tbody>
</table>

Figure 2.

The first reference to zero in 876 A.D. by a Hindu mathematician named Mahavira showed an error in judgement. In his book *Ganita-Sara-Sangraha* (The Compendium of Calculation), he stated: "A number multiplied by zero is zero, and that number remains unchanged which is divided by, added to, or diminished by zero." Mahavira recognized the concept of zero as the identity for addition but incorrectly assumes division by zero has the same

---

effect as addition and subtraction of zero. This misconception was corrected by the great Hindu mathematician Bhaskara (1114-ca. 1185) who stated: "A definite number divided by cipher (zero) is a submultiple of naught." He further illustrated (modern notation here), "10÷0=10/0," and "3÷0=3/0," and says, "These fractions of which the denominator is cipher are termed infinite quantities."

The Hindus marked a blank by calling it *Sunya*, meaning "void" or "empty." This word was translated in Arabic to *Sifr*, meaning "vacant." The Latin translation of the Arabic word (about 1200) kept the sound but not the sense of the word, resulting in *zephirum* or *zephyrum*. Progressive changes in languages derived from Latin led to the development of our words "cipher" and "zero."

**Development of numeration systems.** The basic elements of our modern day numeration system have been traced through Babylon, Greece, China and finally India, where they were picked up by a growing Islamic empire and spread to the Western World. The basic ingredients of this system is an arbitrary integer greater than 1 selected as a base, B; a set of B distinct digits including

---


zero; and multiplicative and additive concepts. The multiplicative concept is that each digit is to be multiplied by the power of the base corresponding to the position in which the digit is written. The additive concept is that the number represented by these symbols will be the sum of these products. The last thing we need is a method for marking the position of the "units digit." This is accomplished by raising the base to the power zero. An example of this concept is letting $B=10$ (our number system), and observing that $6,782=6\times10^3+7\times10^2+8\times10^1+2\times10^0$.

As has already been stated the use of decimal fractions was not a part of the original Hindu development. The first systematic treatment of these was given in 1585 by Simon Stevin.

The transition from Roman and Greek symbols to the Hindu-Arabic symbols did not take place immediately in the Western World. Though the Islamic empire occupied portions of the European coast along the Mediterranean Sea, Western and Central Europe were still under the influence and control of the Roman Catholic Church and the Pope in Rome. The Church was at war with the Arab World for hundreds of years (Crusades, 11th, 12th, and 13th centuries). To acknowledge a system of notation considered Arabic was a sacrilege. This conflict brings to mind a parable about a "New Notation" of long ago.

Many years ago a Roman civil engineer, who was a high official

in Alexandria, was approached by a young Arabian mathematician with an idea which the Easterner believed would be of much value to the Roman Government in their road-building, navigating, tax-collecting, and census-taking activities. As the Arab explained in his manuscript, he had discovered a new type of notation for number writing, which was inspired from some Hindu inscriptions. The Roman official presumably studied this manuscript very carefully for several hours, then wrote the following reply:

Your courier brought your proposal at a time when my duties were light, so fortunately I have had the opportunity to study it carefully, and am glad to be able to submit these detailed comments.

Your new notation may have a number of merits, as you claim, but it is doubtful whether it ever would be of any practical value to the Roman Empire. Even if authorized by the Emperor himself, as a proposal of this magnitude would have to be, it would be vigorously opposed by the populace, principally because those who had to use it would not sympathize with your radical ideas. Our scribes complain loudly that they have too many letters in the Roman Alphabet as it is, and now you propose these ten additional symbols of your number system, namely 1,2,3,4,5,6,7,8,9, and your 0. It is clear that your 1-mark has the same meaning as our mark-I, but since this mark-I already is a well-established character, why is there any need for yours?

Then you explain the last circle-mark, like our letter o,
as representing "an empty column", or meaning nothing. If it means nothing, what is the purpose of writing it? I cannot see that this is serving any useful purpose; but to make sure, I asked my assistant to read this section, and he drew the same conclusion.

You say that the number 01 means the same as just 1. This is an intolerable ambiguity and could not be permitted in any legal Roman documents. Your notation has other ambiguities which seem even worse: You explain that the mark-1 means ONE, yet on the very same page you show it to mean TEN in 10, and ONE HUNDRED in your 100. If my official duties had not been light while reading this, I would have stopped here; you must realize that examiners will not pay much attention to material containing such obvious errors.

Further on, you claim that your system of enumeration is much simpler than with Roman Numerals. I regret to advise that I have examined this point very carefully and must conclude otherwise. For example, counting up to FIVE, you require five new symbols whereas we Romans accomplish this with just two old ones, the mark-I and the mark-V. At first sight the combination IV (meaning ONE before FIVE) for four may seem less direct than the old IIII, but note that this alert representation involves LESS EFFORT, and that gain is the conquering principle of the Empire.

Counting up to twenty (the commonest counting range among the populace), you require ten symbols whereas we now need
but three: the I, V, and X. Note particularly the pictorial suggestiveness of the V as half of the X. Moreover, it is pictorially evident that XX means ten-and-ten, and this seems much preferred over your 20. These pictorial associations are very important to the lower classes, for as the African says, "picture tells thousand words."

You claim that your numbers, as a whole, are briefer than the Roman Numerals, but this is not made evident in your proofs. Even if true, it is doubtful that this would mean much to the welfare of the Empire, since numbers account for only a small fraction of the written records; and in any case there are plenty of slaves with plenty of time to do this work.

When you attempt to show that you can manipulate these numbers much more readily than Roman Numerals, your explanations are particularly bad and obscure. For Example, you show in one addition that 2 and 3 equal 5, yet in the case which you write as: 79
\[
\begin{array}{c}
+16 \\
\hline
95
\end{array}
\]
this indicates that 9 and 6 also equal 5. How can this be? While that is not clear, it is evident that the other part is in error, for we know that 7 and 1 equal 8, not 9.

Your so-called "repeating and dividing" tables also require much more explanation, and possibly correction of errors. I can see that your "Nine Times" table gives sets which add up to nine, namely 18, 27, 36, 45, 54, 63, 72, 81, and 90 but
I see no such useful correlation in the "Seven Times" Table.
For example, since we have Seven, not nine days in the Roman week, it seems far more important to have a system that gives more sensible combinations in this "Seven Times" Table (he could have had his "sets of sevens" with the octinary system, harmonious with the musical octave, the volumetric English measures, the electron octet, and 4-dimensional relativistic Quantum Theory!).

All in all, I would advise you to forget this overly ambitious proposal, return to your sand piles, and leave the number reckoning to the official Census Takers and Tax Collectors. I am sure that they give these matters a great deal more thought than you or I can.

It should be noted by any student of mathematics that the Roman's argument for Roman Numerals is what makes the case for the Arab's system.

The Greeks, Romans, Arabs, and Hindus were not the only societies experimenting and working with various numeration systems, an anthropological study of Western tribes of the Torres Straits, published in 1889, tells of a tribe which had only two number words, urapun and okosa, for 1 and 2 respectively. Out of these two words, by grouping, they manufactured names for 3, 4, 5, and 6 (okosa urapun, okosa okosa, okosa okosa urapun and okosa okosa okosa okosa). The one exception to this base two

---

numeration system was ras, which was used for any number greater than 6.

The first publication describing a binary numeration system appeared in 1703. This was written by Gottfried Wilhelm von Leibniz, the famous German mathematician who, along with Newton, developed the calculus. Leibniz was partially motivated by an attempt to explain semimystical symbols found in an ancient Chinese work. He wrote a generalized discussion of the manner in which one could use any arbitrarily chosen number as a base for the development of a numeration system. His discussion outlined how any integers as large as you wish could be written using only two symbols, 0 and 1. Leibniz saw a parallel to the story of the creation of the Universe in Genesis, in which we learn that God, whom he associated with 1, created the Universe out of nothing, or a void, which he associated with 0. In his Characteristica Generalis he illustrated a universal mathematics that later blossomed into the symbolic logic of George Boole (1815-1864), and still later, in 1910, into the great Principia Mathematica of Whitehead and Russell.

We are now ready to see how this relatively new numeration system is used for computation and how computation evolved through history.

Exercises:
1. What were early mathematical systems mostly concerned with?
2. What does it mean for a numeration system to be non-positional?
3. The Babylonian system was a positional sexagesimal system. What base is indicated by this description?
4. Where do most scholars agree that our present numeration system evolved from (area of the World)?
5. Page 4 shows an example of a magic square. Can you give a different example?
6. Pythagoras is considered the father of mathematics and music (he invented the musical scale). His secret society identified each other through the use of a geometric drawing called a pentagraph. Can you draw this figure?
7. Most ancient mathematical texts were concerned with a particular topic. Identify this topic.
8. Identify the three basic principles of a base 10 positional numeration system.
9. Identify the source of the often used present day word "algorithm."
10. Using Figure 2 write the numbers 487 and 52 in ancient Greek symbols.
11. What is the definition of the answer to a number divided by zero?
12. On page 11 you will note that the number 6,782 is written in what we call expanded notation. Express the number 11,642 in a similar way.
13. In the parable on pages 12, 13, 14 and 15 the Roman points to the "simple" way of marking 1, 4, and 5 using Roman numerals. Express 6, 7, and 8 the way he would.

14. How would the Roman in the parable express the problem and answer (20+6=26) in Roman numerals?
CHAPTER 2

The History of Computation

"In view of the great usefulness of the decimal division, it would be a praiseworthy thing if the people would urge having this put into effect so that in addition to the common divisions of the measures, weights and money, that now exist, the state would declare the decimal division of the large units legitimate to the end that he who wished might use them. It would further this cause also, if all new money should be based on this system of primes, seconds, thirds, etc. (tenths, hundreds, thousands). If this is not put into operation as soon as we might wish, we have the consolation that it will be of use to posterity, for it is certain that if men of the future are like men of the past, they will not always be neglectful of a thing of such great value." These were the words of Simon Stevin in 1585.

Computation's origin. To compute is the same as to calculate. Within the word calculate lies much of the history of computation because the word is derived from the Latin word caculus and is related to the Greek word chalix, both of which mean a small stone or pebble. The Greek historian Herodotus (fl. 5th Century B.C.) said: "The Greeks write, and calculate with pebbles, by moving the hand from left to right; the Egyptians do

1 Translation by V. Sanford, in D.E. Smith, A Source Book in Mathematics.
Finger numbers were widely used by early civilizations as a numerical language. It wasn't even necessary to understand one another's language to conduct business transactions. The numbers were indicated by means of different positions of fingers and hand. Since man was created with ten fingers, the choice of a base for counting usually fell upon the number ten. This base was widely used by primitive people, although not exclusively. The word digit is itself derived from the Latin word digitus, which means finger.

The Babylonian mathematicians were sophisticated enough to develop a multiplication table and a table of reciprocals for their number system. The only trouble was that these tables were recorded on cumbersome and highly perishable clay tablets. Although their system was accurate, another method of calculation was needed. Computation by the abacus. The earliest device known for carrying out arithmetic operations is the abacus. Its name is derived from the Greek word abax, or abakion, which means a board strewn with dust or sand for reckoning or drawing figures. As Herodutus said, pebbles were often used in computation. The abacus was primarily a tool of the merchants and tradesmen, and it could be used throughout the ancient world regardless of differences in languages and numbers. This explains the close resemblance

2

between the Roman abaci, the Chinese suan phan, the Japanese soroban, and the Russian tschotu.

The only preserved Greek abacus was found on the island of Salamis and is a marble counting board dating probably from the fourth century B.C..

The Arabic abacus has ten balls on each wire and no center bar, whereas the Chinese has five lower and two upper counters on each wire, separated by a bar. Each of the upper counters on a wire of the Chinese abacus is equivalent to five on the lower wire. In both the Roman abaci and the Japanese soroban there are four counters in the lower panel and one in the upper corresponding to each line. Those in the upper panel have five times the value of those in the lower.

One major weakness of the abacus is that each step erases the preceding step, so that there is no way to check the answer except by recomputing. Even so, it is still used in many parts of the World today, especially in the Orient. In 1946, for example, there was a contest between a Japanese champion operator of the soroban, and Thomas N. Wood, an American said to be one of the most expert operators of the electrical desk calculator in Japan, at that time. Matsuzaki, the Japanese champion, defeated Wood in operations of addition, subtraction, and division. Wood

It should be noted that there exists considerable question about the skill of the desk-calculator operator.
As knowledge of how to calculate using the Hindu-Arabic numerals spread, the abacus gradually gave way to more advanced methods. Early arithmeticians, recognizing the simplicity of Hindu-Arabic numerals, set to work to devise methods for the multiplication and division of numbers.

**Multiplication and division.** The Hindus experimented with multiplication and division, but their vague mysticism and obscure ways of recording their computation in verse greatly hindered general understanding and acceptance. It was really not until the end of the fifteenth century that arithmetic began to assume our modern form. It took four hundred years, from the time the work of al-Khowarizmi was widely disseminated by a twelfth century Latin translation, for the algorists, as the advocates of the new system were called, to win out over the abacists. In another hundred years the abacists were almost forgotten.

In 1484 there appeared an arithmetic work known as *Triparty en la science des nombres*, which was written by the French mathematician Nicolas Chuquet. Little is known about the man except that he was born in Paris, received his bachelor's degree in medicine, and practiced at Lyons. The only writers the author mentions in the *Triparty* are Boethius and Campanus. There is evidence of Italian influence, which probably resulted from an acquaintance with Fibonacci's *Liber abaci*, written almost three centuries earlier.
The first of the "Three Parts" concerns the rational arithmetic operations on numbers, including an explanation of the Hindu-Arabic numerals. The second part concerns irrational numbers, and the third with the theory of equations. Chuquet recognized positive and negative integral exponents and syncopated some of his algebra. His work was too advanced to exert much influence on his contemporaries, but it should be noted that Etienne de la Roche virtually reprinted the *Tripartie* in his own *Larismethique nouvellement composee* in 1520, and again in 1538. Hence the *Tripartie* was not without effect.

Ten years after Chuquet published his work in France there appeared the first printed edition of the *Summa de arithmetica, geometrica, proportioni et proportionalita*, usually referred to briefly as the Sūma, of the Renaissance mathematician friar Luca Pacioli (ca. 1445-1509) in Italy. The presentation contains eight plans for the performance of multiplication. The algebra in the Sūma is syncopated by the use of such abbreviations as \( p \) (from *piu*, "more") for plus, \( m \) (from *meno*, "less") for minus, \( c o \) (from *cosa*, "thing") for the unknown \( x \), \( ce \) (from *censo*) for \( x^2 \), \( cu \) (for *cuba*) for \( x^3 \), and \( cece \) (from *censocenso*) for \( x^4 \).

Equality is sometimes indicated by \( ae \) (from *aequalis*). Pacioli traveled extensively and taught as well as wrote a number of other works, not all of which were printed. In 1509, 

\[ x^4 \]

he published *De divina proportione*, which contains figures of the regular solids thought to have been drawn by Leonardo da Vinci. This work also contains the ratio later known as "the golden section" which was widely used in early Greek architecture.

**Origins of symbols for operations.** The first printed appearance of our present day + and - signs was by Johann Widman in Leipzig in 1489. These signs were also used as symbols of algebraic operations in 1514 by the Dutch mathematician Vander Hoecke. Widman, incidentally, possessed a manuscript copy of the *Algebra* of al-Khowarizmi, whose works were well known to most German mathematicians. One such mathematician was Adam Riese (1492-1559) whose *Die Coss* was most influential in replacing the old computation (in terms of counters and Roman numerals) by the newer method (using the pen and Hindu-Arabic numerals). *Die Coss* was one of the earliest printed works to make use of decimal fractions, although the invention of decimal fractions is usually given to Simon Stevin for his *La Disme* published in 1585.

First use of the symbol $\times$ for multiplication is attributed to William Oughtred in his *Clavis mathematicae* (1631) and in an anonymous appendix to an earlier book (1618). Leibniz objected to the use of Oughtred's symbol because of possible confusion with the letter $x$. In 1698 he suggested the use of a dot as a

---

symbol for multiplication.

Fibonacci symbolized division in fraction form with the use of the horizontal bar. In his *Arithmetica integra* (1544) Michael Stifel employed the arrangement $8)24$ to mean $24$ divided by $8$. The symbol $\div$ was first used to signify division by the Swiss born Johann Heinrich Rahn in an algebra text published in 1659.

**Prelude to modern mathematics.** Most of Western Europe was involved in the advance of mathematics during the sixteenth century, but the central figure in the transition was a Frenchman, Francois Viete (1540-1603). He was born in Fontenay and died in Paris. Viete's most famous work is his *In artem*, which did much for the development of symbolic algebra. In this work Viete introduced the practice of using the vowels to represent unknown quantities and the consonants to represent known ones. This system was turned around by Descartes in 1637.

There is an interesting story about Viete concerning his ability to break the secret Spanish code containing several hundred characters, and thereby enabling France to conduct a more successful campaign in its war against Spain. So certain was King Philip II that the code could not be deciphered that he complained to the Pope that the French were employing magic against his country, "contrary to the practice of the Christian faith."

Viete showed how to compute the roots of equations of higher degrees than two and how to approximate numerically the trisections,
quinquisections, etc., of angles. In his work, published in 1646, he outlined formulas for calculating the chords of multiple arcs and also for finding the chords of fractional arcs. This was expanded upon later by Newton and Leibniz.

It can be said, with good justification, that Viète was the father of a generalized analytic approach to trigonometry. Viète, in the Canon mathematicus (1579), prepared extensive tables of all six functions for angles to the nearest minute. He urged the use of decimal, rather than sexagesimal, fractions.

Like Viète, John Napier (1550-1617), a Scotsman, was not a professional mathematician. In a commentary on the Book of Revelations, for example, he argued that the Pope was the anti-Christ. His interest in mathematics was concerned chiefly with computation and trigonometry. His "rods" or "bones" were a form of lattice multiplication. His analogies and rules of circular parts concerned spherical trigonometry.

In 1614 Napier published his first work, entitled Mirifici logarithmorum canonis descripto ("A Description of the Wonderful Law of Logarithms") which included a table of the logarithms of natural sines. This work was instrumental in reducing the time and increasing the accuracy of carrying out the numerical computations of multiplication, division, and the extractions of roots.

Henry Briggs published the first tables of logarithms to the base ten in 1624. His tables were carried to fourteen places of the logarithms of numbers between 1 and 20,000 and from 90,000
to 100,000. Adriaen Vlacq completed the table in the second edition of Briggs' tables by filling the gap 20,000 to 90,000, but only carried out to ten places.

D.E. Smith concluded that "it is unquestionably true that the invention of logarithms had more to do with the use of decimal fractions than any other single influence."

In summarizing the mathematical achievements of the sixteenth and early seventeenth century, we can say that symbolic algebra was well started, computation with Hindu-Arabic numerals became standardized, decimal fractions were developed, the cubic and quartic equations had been solved and the theory of equations advanced, negative numbers were being accepted, trigonometry was perfected and systematized, and some excellent tables were computed. We are now ready for the most outstanding age in history in the field of mathematics.

**Development of computing machines.** The first calculating machine was the abacus, which has already been discussed. The second was a device created by Napier called "Napier's Bones," or "Napier's Rods." In his work entitled *Rabdologia* (from the Greek word *rabdos* meaning "rods"), published in 1617, Napier describes how multiplication can be achieved by simple addition once the rods are arranged in the proper way. These rods were later placed on parallel cylinders so that they could more easily be put into place for multiplication.

In 1620 Edmund Gunter, an associate of Briggs, constructed
a logarithmic scale and multiplied numbers by means of a pair of dividers. This was the forerunner of our present day slide rule. William Oughtred improved on Gunter's idea in 1622 when he replaced the dividers by two logarithmic scales, one of which would slide by the other. Newton carried it a step further in 1675 when he suggested a runner for the slide rule. This idea was not actually put into practice until 1775 by John Robertson. Many modifications since then have contributed to give us our present day slide rule.

The first calculating machine was invented by the French mathematician Blaise Pascal in 1642 to assist his father in the auditing of the government accounts at Rouen. In fact, Pascal built and sold some fifty machines. His calculator was able to handle numbers not exceeding six digits and could perform addition and subtraction operations. It consisted of a large rectangular box with six wheels on top. The wheels were connected to recording drums, which in turn activated numerical wheels that could be read through holes on top of the machine.

Later in the century, Leibniz (1671) in Germany and Sir Samuel Morland (1673) in England invented machines that multiplied. An important innovation introduced by Leibniz was the sliding carriage. In 1820, Thomas de Colmar transformed a Liebniz type of machine into one which could perform subtractions and divisions. In 1875, Frank S. Baldwin, an American, patented the first calculating machine that could perform the four fundamental
operations of arithmetic without resetting the machine. W.T. Odhner (Swedish) was granted a U.S. patent in 1878 on a machine similar to that of Baldwin. Today's electrically operated desk calculators are basically the same as the Baldwin machine.

About 1812, the English mathematician Charles Babbage (1792-1871) began to consider one of the most ambitious projects undertaken during the nineteenth century, the construction of a machine to aid in the calculation of mathematical tables. By 1823, Babbage had lost his own personal fortune in the venture, but secured financial aid from the British government in the amount of 17,000 pounds sterling. The machine envisioned by Babbage would have performed all arithmetic operations and would have stored information for recall, using an elaborate pattern of gears, wheels, and levers. His "engine," a digital computer, was never completed. In 1842, the government cut off funds, but Babbage was not to be discouraged. He enlarged his ideas and began work on what he called his "analytic engine," which was intended to automatically execute a whole series of arithmetical operations assigned to it at the start by the operator. This machine also was never finished, but his son, H.P. Babbage, completed part of the engine in 1906 and published 25 multiples of \( \pi \) to 29 figures as an example of its work.

Babbage didn't fail because of design, but more as a result of a lack of machine tools, electrical circuits, and metal alloys that are essential in modern computers.
In 1888, William S. Burroughs designed a calculator that printed its figures. Henry Pottin patented a similar machine in 1883 in England, and in 1885 in the U.S., that printed totals and subtotals.

In 1880, Herman Hollerith invented a machine for sorting cards. The cards contained holes which permitted their distribution by electromagnetic relays activated by contacts made through the holes. IBM adopted this principle for its early computers in 1929.

The modern era of mechanical computation may be said to have started in 1925 at the Massachusetts Institute of Technology by Vannevar Bush (1890-1974) and his associates who built a large-scale analog calculator, powered by electric motors, but otherwise mechanical.

The first direct descendent of the analytic engine of Babbage is the IBM Automatic Sequence Controlled Calculator (ASCC) completed at Harvard University in 1944. The machine is 51 feet long, 8 feet high, with two panels 6 feet long, and weighs about 5 tons. Before its completion it had been outmoded by plans for ENIAC (Electronic Numerical Integrator and Calculator). This was a vacuum tube computer completed in 1945 at the University of Pennsylvania. This machine requires a 30 by 50 foot room, contains 19,000 vacuum tubes, and weighs about 30 tons. It is now relegated to the Smithsonian Institution in Washington, D.C..

In 1948 the invention of the transistor was announced at Bell
Laboratories. This small crystalline device revolutionized
electronics and especially the computer. Compared to the vacuum
tube it is much smaller, has a longer life, uses much less
current, and consequently generates almost no heat.

Computers today have become so vast and intricate that they
surpass even the wildest dreams of Babbage, who lived a century
before his time. Kepler said, the invention of logarithms doubled
the life of an astronomer. Today's computers are still being
assessed as to their impact on the expanded careers of scientists
and mathematicians. With its increased power has come applications
in new fields of mathematics such as linear programming, game
theory, operations research, and many more.

Von Neumann and Oskar Morgenstern teamed up to produce the
Theory of Games and Economic Behavior which brought finite
mathematics into the social sciences. Norbert Wiener (1894–
1964) published his Cybernetics, a book establishing a new subject
devoted to the study of control and communication in animals
and machines. Both Von Neumann and Wiener were involved in
quantum theory and contributed to set theory, group theory,
operational calculus, probability, and mathematical logic and
foundations.

Computers, in fact, have so revolutionized the quantity of
knowledge that if we could stack in one pile all the accumulated
knowledge of mankind from inception to about 25 years ago and
in another pile stack the newly acquired knowledge from 25 years
ago to present, the latter pile would be larger.
Exercises:
1. Where is the word calculate derived from and what does its derivation translate to?
2. What is one major weakness of the abacus?
3. Define algorist.
4. Define abacist.
5. Oughtred used the symbol X for multiplication, Leibniz objected to this use. What did Leibniz suggest instead?
6. Who introduced the use of vowels and consonants for unknown and known quantities respectively and who reversed this order?
7. What are "Napier's Bones" used for?
8. Logarithms had a tremendous affect on mathematics. What was that affect?
9. Who invented the first calculating machine?
10. Who invented the forerunner of the modern IBM card?
11. What invention revolutionized both electronics and the development of computers?
PART II

LOGIC

Taking mathematics from the beginning of the world to the time of Newton, what he has done is much the better half.

Gottfried Wilhelm von Leibniz
1646-1716
CHAPTER 3

The Binary Concept

Introduction. Decisions are most often based on a number of yes or no type of conditions. "Yes," the height is too high. "No," I don't want to get hurt. "Yes," I've decided not to jump. A light switch is either ON or OFF, it either passes electrical current or it doesn't. The light switch is ON, so "no," I won't repair the light socket. The light switch is OFF, so "yes," I will repair the socket. These are both examples of two state conditions. The latter decision involved the state of a switch, either ON or OFF, "yes" or "no." The computer functions in much the same way and this is why the binary numeration system is so useful. It has only two numerals, 0 and 1, which correspond to OFF and ON respectively in modern digital computers. Let's have a closer look at this interesting system.

Counting in the binary system. Counting in the binary numeration system is quite simple to understand once you understand the concept of how our commonly used base 10 system works.

In the decimal numeration system we use every day, the value of a numeral depends upon the numeral's position in a number. As has been shown in Chapter 1, 6,782=6X10^3+7X10^2+8X10^1+2X10^0. The quantity to the right of the equals sign is expressed in what is commonly known as expanded notation using base 10. Each numeral in the number to the left of the equals sign has a name.
corresponding to the position it occupies. Since this is a base 10 number the names for each position are just powers of 10 beginning with the zero power and ascending upward. Stated simply, the unit's position, ten's position, hundred's position, thousand's position, etc., since \(10^0=1\), \(10^1=10\), \(10^2=100\), and \(10^3=1000\). Therefore, 2 occupies the unit's position, 8 the ten's position, 7 the hundred's position, and 6 the thousand's position. For larger numbers it's simple enough to go to the next position, ten thousand's, and so on.

Decimal fractions also can be named according to the position of the numerals, but utilize negative powers of 10 instead of the positive powers used in the whole numbers. For example, 
\[
27.86 = 2 \times 10^1 + 7 \times 10^0 + 8 \times 10^{-1} + 6 \times 10^{-2}.
\]
As you move to the left you are moving in the direction of ascending powers of 10 and as you move to the right you move along descending powers of 10. In this context I use ascending to mean larger and descending to mean smaller. Refer to the number line where any number is larger than any number on its left and smaller than any number on its right. In the preceding example, 8 occupies the one-tenth position, 6 the one-hundredth, since \(10^{-1}=1/10\) and \(10^{-2}=1/100\).

For counting in our base 10 system we use the set of ten numerals \(0, 1, 2, 3, 4, 5, 6, 7, 8, 9\), begin with 0 and ascend at increments of 1 each. We refer to this infinite set as the set of whole numbers. Each position goes through a repetitious cycle of numerals, 0-9, then back to 0 to repeat the same cycle. The
unit's position ascends at increments of 1 each, the ten's position at increments of 10 each, the hundred's at 100 each, and so on. As each position completes its cycle at the numeral 9 and reverts back to 0 it causes the position on its left to go to the next numeral in its cycle. Continuous counting will cause a chain reaction to occur for as long as one wishes to count. This process is easily compared with the odometer which records distance traveled in an automobile. The most common odometer records tenths of a mile, or kilometer, and makes increments of 1/10 each. The same cycling takes place. If it is desired, increments of 1/100 could be designed into the odometer with the same cycling process. In fact it doesn't matter what position you start in as long as you remember you are making increments corresponding with that position.

In the binary numeration system the rules are the same, only the base has been changed. In the base 2 system our set of numerals is (0,1) instead of the ten numerals used in base 10. The positions of the numerals are denoted in the same manner as in base 10, but instead of 10 to some power it's 2 to a power. Figure 1 illustrates the positional values for both integers and fractional quantities of the binary numeration system.

Each 1 or 0 of the binary number is called a bit. Therefore the quantity 1101.0101₂ is an eight-bit number. I will discuss bits and how they are used later in the text.

The example in Figure 1 shows us a binary number composed
of eight numerals and a binary point. This is fine, but what does it mean? The answer lies on the right hand side of Figure 1.

Using the expanded notation of the base 2 number we can add these quantities and we will arrive at a base 10 number.

$$1101.0101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 8 + 4 + 0 + 1/4 + 0 + 1/16$$

or 13 5/16 or $$13.3125_{10}$$. Take note of the subscripts indicating the base.

Counting in binary numbers is the same as counting in base 10 except the cycle is 2 instead of 10. Again as each position finishes its cycle (1 in binary) and goes back to 0 it causes the position on the immediate left to increment to the next numeral in its cycle. Figure 2 illustrates the binary and decimal
equivalents of numbers 0-10 (base 10).

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>*</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>*</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>*</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>*</td>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.

Conversion of decimal to binary. There are two commonly used methods for converting decimal numbers to binary equivalents. The first is the subtraction of powers method. The second is the division method.

To convert any decimal number to its binary equivalent by the subtraction of powers method, subtract the largest possible power of 2 from the decimal number, and place a 1 in the appropriate weighting position of the partially completed binary number (see Figure 3). Continue this procedure until the decimal number is reduced to 0. If, after the first subtraction, the next smaller power of 2 cannot be subtracted, place a 0 in the appropriate weighting position. Let's look at an example: \(42_{10} = ? \text{ binary}.

\[
\begin{array}{c|c|c}
42 & 10 & 2 \\
-32 & -8 & -2 \\
10 & 2 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
32 & 16 & 8 & 4 & 2 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

Figure 3.
Therefore, \( 4Z_{10} = 101010_2 \).

To convert a decimal number to binary by the division method, divide the decimal number by 2. If there is a remainder, place a 1 in the unit's position of the partially formed binary number. If there is no remainder, put a 0 in the unit's position of the binary number. Divide the quotient from the first division by 2, and repeat this process, putting a 0 or 1 in the next position (21=2 positions) depending on whether or not you get a remainder (see Figure 4). Continue until the quotient has been reduced to 0.

Example: \( 47_{10} = ? \) binary.

\[
\begin{array}{c|c|c}
\text{quotient} & \text{remainder} \\
\hline
2) 47 & 23 & 1 \\
2) 23 & 11 & 1 \\
2) 11 & 5 & 1 \\
2) 5 & 2 & 1 \\
2) 2 & 1 & 0 \\
2) 1 & 0 & 1 \\
\end{array}
\]

Figure 4.

Therefore, \( 47_{10} = 101111_2 \).

Fractional quantities expressed in the decimal system can be converted into a binary number by repeated multiplication by 2. The fractional part in the result of each multiplication is again 1.

Thanks to Digital Equipment Corporation's Introduction to Programming for their illustrations on pp. 1-9 to 1-10.
multiplied by 2. This process is repeated until further multiplications by 2 produce only zeros in the result. In the result of each multiplication the digit (0 or 1) to the left of the decimal point becomes one of the bits of the binary number being sought, the first bit so obtained being the most significant bit.

Example: \(0.3125_{10} = \) binary.

\[
\begin{array}{c|c|c|c|c}
\times 2 & 0.3125 & 0.6250 & 0.2500 & 0.5000 \\
\hline
0.6250 & 1.2500 & 0.5000 & 1.0000 \\
\end{array}
\]

Therefore, \(0.3125_{10} = 0.0101_{2}\).

Addition and subtraction of binary numbers. Because the binary system employs only two symbols, addition is a simple process. It follows three basic rules. They are:

\[
\begin{array}{c|c|c|c|c}
0 & + 0 & 0 & + 1 & 1 \\
\hline
0 & 0 & 1 & + 1 & 10 \\
\end{array}
\]

Figure 6.

Note that 1 plus 1 yields a sum of 0 and a carry of 1 into the next column. Similarly, 1+1+1 yields a sum of 1 and a carry of 1 into the next column.

Subtraction of binary numbers follows four basic rules. They are:

\[
\begin{array}{c|c|c|c|c}
0 & - 0 & 1 & - 1 & 10 \\
\hline
0 & - 0 & 0 & - 0 & - 1 \\
\end{array}
\]

Figure 7.

Note that it is necessary to borrow in the last case, but what
is borrowed is from the $2^1$ column, moving it back 1 increment in its cycle (in this case, to 0), and is therefore equivalent to the quantity $2_{10}=1_2+1_2$, since this is from the two's position. When $1_2$ is subtracted from this quantity, $1_2$ is all that is left.

**Multiplication and division of binary numbers.** Multiplication with binary numbers follows the same procedures as multiplication of decimal numbers. There is only one basic difference, the partial products are added following base 2 addition rules (see Figure 8): There are only two possible results in the product of two binary bits, 0 or 1. In the final product, after the partial products have been added, there can be only two distinct symbols, 0 and 1. Example: $11_2 \times 1101_2 = ?$ binary.

```
   1101
X 11
```

```
1101
```

```
100111_2 ← final product
```

Figure 8.

Therefore $11_2 \times 1101_2 = 100111_2$.

Division of binary numbers also follows the same procedures as division of decimal numbers. Again, remember that the numbers being computed are base 2 and therefore cannot have any numerals in the computation that are not 0 or 1. Example: $101010_2 \div 110_2 = ?$ binary.

```
111
110)101010
-110
1001
-110
110
-110
```

Figure 9.
Therefore $101010_2 + 1102 = 1112_2$.

Note in the first division by the divisor $110_2$ that the first three figures of the dividend are not large enough, therefore the first four figures of the dividend are necessary. Also note that the maximum number of times the divisor can possibly go into these first four figures is the largest numeral of the number system being used, in this case it's $1_2$.

**Trinary (ternary) numeration system.** I'm only going to discuss the trinary number system briefly by stating that such a system of notation could be mechanized in a computer by any device with three stable states, a magnetic core is a good example. A core that is unmagnetized could be represented by $0$, one that is negatively polarized by $1$, and positively polarized by $2$. The trinary number system uses three numerals $(0,1,2)$ and follows essentially the same rules as the binary except here the cycle is three numerals in length instead of two. Also the largest numeral is $2$ and in all computations involving base 3 numbers (trinary system) no computation can have a numeral larger than $2$.

In some texts the ternary system is explained using the three symbols $-,0,$ and $+$, where $-=-1$, $0=0$, and $+=+1$.

**Octal numeration system.** The octal numbering system came into being as a result of difficulty in dealing with long strings of 2

---

binary zeros and ones and converting them into decimals. When testing computers, sample problems must be hand-fed into it, which may involve binary numbers with as many as 40 digits each. These long strings of binary numbers are not only awkward to read and handle, but in order to find out what these binary digits stand for in terms of decimal numbers you have to go through a lengthy conversion process using powers of two, as has already been shown.

A system that permits direct substitution of its values into binary digits and vice versa must have a base whose powers are in direct proportion to those of the binary system. It must also have an equivalent numeral for every possible combination of a group of binary digits. The octal numeration system is such a system. It uses eight numerals (0,1,2,3,4,5,6,7) and has a cycle of length eight (0-7) in each position when counting.

To convert from a decimal number to an octal number the easiest method is the division method (see Figure 10). Example: 483₁₀ =? octal.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>8) 483 = 60</td>
<td>3 8₀</td>
</tr>
<tr>
<td>8) 60 = 7</td>
<td>4 8¹</td>
</tr>
<tr>
<td>8) 7 = 0</td>
<td>7 8²</td>
</tr>
</tbody>
</table>

Figure 10.

Therefore 483₁₀ = 743₈.
Let's look at this example as a binary number. If you use the subtraction of powers method you will find that \(483_{10} = 111100011_{2}\). This is a fairly large base 2 number so let's see how the octal system will simplify this long string of ones and zeros. Figure 2 provides a clue to how this can be done when you observe that for decimal numbers 0-7 the binary equivalents are 0-111\(_2\) or, for our problem, 000-001-010-011-100-101-110-111\(_2\). In the octal system we move by successive powers of eight for each position, while in the binary system we move by successive powers of two. However, any group of three successive binary bits also ascends by powers of eight. It is this simple relation that makes conversion from one notation into the other such a simple matter. Therefore, \(483_{10} = 111100011_{2} = 743_{8}\), as has been shown. If, when grouping in threes from right to left, you end up with a remainder of one or two bits instead of three, just add zeros to the left side of the binary number to get a group of three (see Figure 11). Binary fractions can be handled by grouping in threes from left to right from the binary point.

Arithmetic operations are done in the octal system using the same methods already outlined in the binary system. Remember that computation in any number system depends on its base. In the octal system any computations done will involve only the numerals 0-7. The number of symbols used to represent any single numeral of any commonly used place value numeration system can never exceed the base.
A comparison of decimal, octal, and binary systems is shown in the table below.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>001 000</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>001 001</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>001 010</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>001 011</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>001 100</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>001 101</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>001 110</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>001 111</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>010 000</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>001 000 000</td>
</tr>
<tr>
<td>128</td>
<td>200</td>
<td>010 000 000</td>
</tr>
<tr>
<td>256</td>
<td>400</td>
<td>100 000 000</td>
</tr>
</tbody>
</table>

Figure 11.

Alphanumeric codes. Data processing machines use binary codes to represent alphabetic information as well as numerical data. One such alphanumeric code is shown in Figure 12. Each digit, alphabetic character, punctuation mark, etc. is represented by a unique six-bit combination. The digit 7, for example, is 000111₂, the letter X is 010111₂, the letter P is 100111₂, and the letter G is 110111₂. There are a total of 64 six-bit combinations, more than enough to represent the letters, digits, and some punctuation marks and special symbols.

In order to give the alphanumeric code an error-detecting characteristic, a seventh bit may be added to each six-bit group. This extra bit, usually placed in the position farthest left, is referred to as a check bit or parity bit. This bit is selected
so that every seven-bit combination will have an odd number of 1's. The letter X, for example, would be 10101112. Computers using such codes can be designed to recognize circuit failures which produce seven-bit combinations containing an even number of 1's. Some computers use an even-parity check.

<table>
<thead>
<tr>
<th>Binary coding system</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0001</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>J</td>
</tr>
<tr>
<td>0010</td>
<td>*</td>
<td>2</td>
<td>S</td>
<td>K</td>
</tr>
<tr>
<td>0011</td>
<td>*</td>
<td>3</td>
<td>T</td>
<td>L</td>
</tr>
<tr>
<td>0100</td>
<td>*</td>
<td>4</td>
<td>U</td>
<td>M</td>
</tr>
<tr>
<td>0101</td>
<td>*</td>
<td>5</td>
<td>V</td>
<td>N</td>
</tr>
<tr>
<td>0110</td>
<td>*</td>
<td>6</td>
<td>W</td>
<td>O</td>
</tr>
<tr>
<td>0111</td>
<td>*</td>
<td>7</td>
<td>X</td>
<td>P</td>
</tr>
<tr>
<td>1000</td>
<td>*</td>
<td>8</td>
<td>Y</td>
<td>Q</td>
</tr>
<tr>
<td>1001</td>
<td>*</td>
<td>9</td>
<td>Z</td>
<td>R</td>
</tr>
<tr>
<td>1010</td>
<td>*</td>
<td>0</td>
<td>!</td>
<td>?</td>
</tr>
<tr>
<td>1011</td>
<td>*</td>
<td>#</td>
<td>$</td>
<td>*</td>
</tr>
<tr>
<td>1100</td>
<td>*</td>
<td>%</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1101</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>1110</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>1111</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Figure 12.

Another alphanumeric code is an octal coding system referred to as the "8-bit ASCII code," which is an abbreviation for USA Standard Code for Information Interchange.

Figure 13 is only a partial table, but does suffice to illustrate how an octal system can be used to code characters necessary in programming a computer. The fact that the code for octal digits 0 through 7 is the sum of that digit plus 2608 should be observed.

It has been shown how binary symbols can be used to represent the elements of our English language including the Hindu-Arabic symbols as well as operations involving these symbols. In the next chapter we shall see how Boolean algebra assembles the binary system into the electronic circuits of a computer.

---

4

See Digital Equipment Corporation's Introduction to Programming, p. 5-5.
Exercises:
1. Write 8,726 in expanded notation.

2. Write 3.1415 in expanded notation.

3. How many symbols would there be in a base 7 numeration system?

4. How many bits are there in $101.011_2$?

5. Write $101.1_2$ in expanded notation.

6. Write $19_{10}$ as a base 2 numeral.

7. Write $56_{10}$ as a base 2 numeral.

8. Write $2.56_{10}$ as a base 2 numeral.

9. Add $1.011_2 + 10.01_2$.

10. Subtract $1110_2 - 111_2$.

11. Multiply $1110_2 \times 111_2$.

12. Divide $1011111_2 \div 101_2$.

13. Write $867_{10}$ as a base 8 numeral.

14. Write $1011110101_2$ as a base 8 numeral.

15. Using Figure 12 on page 46, decode the following message:
    010011111001110010100100100101100100001110010100100100011100111001101
    0010010011010100010011110110110011001110110101001100110110101111
    1010 .

16. Using Figure 13 on page 47, decode the following message:
    324310311323255311323255301255302305324324305322255303317315320325
    324305322255303317304305256 .
CHAPTER 4

Boolean Algebra

Introduction. George Boole (1815-1864) is given much of the credit for the idea of laying down postulates for the manipulation of abstract symbols, not necessarily numbers, although work in this area of thought was taking place in England at about the same time by other mathematicians. Boole published his ideas in 1847 in a pamphlet entitled Mathematical Analysis of Logic. In 1854 he published An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities or, more simply stated, the Laws of Thought. This last publication was written along the lines of Leibniz' "universal characteristic." This was later expanded upon by Gottlob Frege in 1884 and culminated in the Principia mathematica (1910-13) of Bertrand Russell and A.N. Whitehead and present-day mathematical logic.

Boole used lowercase letters such as \( x,y,z \), to denote sets. Today we often use uppercase \( A,B,C \), to denote sets. Boole used the number 0 for the null or empty set, whereas we commonly use \( \emptyset \). We frequently use \( U \) or \( I \) for the universal set, whereas Boole used 1. Boole wrote the intersection of two sets \( x \) and \( y \) (\( A \) and \( B \)) as \( xy \) or \( x \cdot y \) (now frequently as \( A \cap B \), called "A cap B") as consisting of all elements that are in both sets. The logical sum of two sets \( A \) and \( B \) is denoted by \( A \cup B \) and called "A cup B"
and consists of the set whose members are members of the set \( A \) or the set \( B \) or both. Boole's "Logical Sum" was a little different and difficult to interpret, we will rely on our present day interpretation. The complement of a set \( A \) we will denote \( A' \) (called "A prime"), it is also commonly denoted by \( I-A \), \(-A\), and \( \bar{A} \) (called "not A"). By definition, \( A \cup A'=I \), and \( A \cap A'=0 \) or \( \emptyset \).

Definition of a Boolean algebra. A Boolean algebra is a 6-tuple \((B, \cup, \cap, ', 0, 1)\), where \( B \) is a set, \( \cup \) is a binary operation (variously called "cup," "union," or "join") in \( B \), \( \cap \) is a binary operation (variously called "cap," "intersection," or "meet") in \( B \), ' is a binary relation in \( B \) having \( B \) as its domain, 0 and 1 are distinct elements of \( B \).

A set \( B \), on which binary operations \( \cup \) and \( \cap \) are defined, is called a Boolean algebra provided the following axioms (or postulates) hold:

(i) \( \cup \) and \( \cap \) are associative, i.e., for all \( a, b, c, \in B \) (\( \in \) means "in" or "belonging to"), \( a \cup (b \cup c) = (a \cup b) \cup c \) and \( a \cap (b \cap c) = (a \cap b) \cap c \).

(ii) \( \cup \) and \( \cap \) are commutative, i.e., for all \( a, b, c, \in B \), \( a \cup b = b \cup a \) and \( a \cap b = b \cap a \).

(iii) Each operation is distributive with respect to the other, i.e., for all \( a, b, c, \in B \), \( a \cup (b \cap c) = (a \cup b) \cap (a \cup c) \) and \( a \cap (b \cup c) = (a \cap b) \cup (a \cap c) \).

(iv) \( B \) contains an identity element 0 with respect to \( \cup \) and an identity element 1 with respect to \( \cap \), i.e., for all \( a, b, c, \in B \), \( a \cup 0 = a \) and \( a \cap 1 = a \).
(v) For each \( a \in B \) there exists an \( a' \in B \) such that \( a \cup a' = 1 \) and \( a \cap a' = 0 \).

In the above axioms \( \cup \) and \( \cap \) are used instead of the more familiar + and \( \cdot \). We use 0 to denote the empty set \( \emptyset \) and 1 to denote the universal set \( U \). In this respect we can define Boolean functions. If in the expression \( x' \cup (y \cap z) \) we replace \( \cup \) by + and \( \cap \) by \( \cdot \) to obtain \( x' + y \cdot z \), it seems natural to call \( x' \) and \( y \cap z \) monomials and the entire expression \( x' \cup (y \cap z) \) a polynomial.

Also, by a constant we shall mean any symbol, as 0 and 1, which represents a specified element of \( B \) and by a variable we shall mean a symbol which represents an arbitrary element of \( B \). Any expression consisting of combinations by \( \cup \) and \( \cap \) of a finite number of elements of a Boolean algebra \( B \) will be called a Boolean function. The number of variables in any function is the number of distinct letters appearing without regard to whether the letter is primed or unprimed. Thus, \( a \cup a' \) is a function of one variable \( a \) while \( a \cap b' \) is a function of two variables \( a \) and \( b \).

It should be noted that there is one basic difference between ordinary algebra and Boolean algebra. In ordinary algebra every integral function of several variables can always be expressed as a polynomial (including 0) but cannot always be expressed as a product of linear factors. In Boolean algebra Boolean functions can generally be expressed in polynomial form (including 0 and 1), i.e., a union of distinct intersections, and in factored form, an intersection of distinct unions.
The principle of duality may be described by stating that any theorem deducible from the axioms (i)-(v) of a Boolean algebra remains valid when the operation symbols $\cup$ and $\cap$ and the identity elements 0 and 1 are interchanged throughout. Observe that each axiom is a dual pair of statements, with (v) regarded as self-dual. Therefore, any theorem of Boolean algebras has a dual theorem, the duals of the steps appearing in the proof of the theorem providing a proof of the dual. This principle yields a free theorem for each theorem obtained, unless that theorem happens to be its own dual. The following theorem is a result of axioms (i)-(v).

Theorem 4.1. In each Boolean algebra $(B, \cup, \cap, ', 0, 1)$ the following hold:

(vi) The elements 0 and 1 are unique.

(vii) Each element has a unique complement.

(viii) For each element $a$, $(a')' = a$.

(ix) $0' = 1$ and $1' = 0$.

(x) For each element $a$, $a \cup a = a$ and $a \cap a = a$.

(xi) For each element $a$, $a \cup 1 = 1$ and $a \cap 0 = 0$.

(xii) For all $a$ and $b$, $a \cup (a \cap b) = a$ and $a \cap (a \cup b) = a$.

(xiii) For all $a$ and $b$, $(a \cup b)' = a' \cap b'$ and $(a \cap b)' = a' \cup b'$.

Proof: (vi) Assume that $0_1$ and $0_2$ are elements of $B$ such that $a \cup 0_1 = a$ and $a \cup 0_2 = a$ for all $a$. Then $0_2 \cup 0_1 = 0_2$ and $0_1 \cup 0_2 = 0_1$. $0_2 \cup 0_1 = 0_1 \cup 0_2$ by axiom (ii). Therefore $0_2 = 0_1$. Thus 0 is a unique element in $B$. The dual of this proof will determine
the uniqueness of 1.

Proof: (vii) Assume \( a_1 \) and \( a_2^1 \) are both complements of \( a \). Then

\[
\begin{align*}
\text{Proof: (vii)} & \quad \text{Assume } a_1^1 \text{ and } a_2^1 \text{ are both complements of } a. \text{ Then} \\
& \quad a_1^1 = a_1^1 \cup 0, \text{ by (iv)}; \\
& \quad = a_1^1 \cup (a \cap a_2^1), \text{ since } a \cap a_2^1 = 0; \\
& \quad = (a_1^1 \cup a) \cap (a_1^1 \cup a_2^1), \text{ by (iii)}; \\
& \quad = (a_1 a_1^1) \cap (a_1^1 \cup a_2^1), \text{ by (ii)}; \\
& \quad = 1 \cap (a_1^1 \cup a_2^1), \text{ since } a \cup a_1^1 = 1; \\
& \quad = (a_1^1 \cup a_2^1) \cap 1, \text{ by (ii)}; \\
& \quad = a_1^1 \cup a_2^1, \text{ by (iv)}.
\end{align*}
\]

Similarly, we prove \( a_2^1 = a_2^1 \cup a_1^1 \). Therefore, by (ii), \( a_1^1 = a_2^1 \).

Proof: (viii) By definition of the complement of \( a \), \( a \cup a' = 1 \) and \( a \cap a' = 0 \). Therefore, by (ii), \( a' \cup a = 1 \) and \( a' \cap a = 0 \), i.e.,

\[
\begin{align*}
(a')' & = a, \text{ by (vii)}.
\end{align*}
\]

Proof: (ix) \( 1 = a \cup a' \), by (v);

\[
\begin{align*}
& = 0 \cup (a \cup a'), \text{ by (iv)}; \\
& = 0 \cup 1, \text{ by (v)}; \\
& = 0 \cup 0', \text{ by (vi) and (vii)}.
\end{align*}
\]

Therefore \( 0' = 1 \). The dual gives us \( 1' = 0 \).

Proof: (x) \( a \cup a = (a \cup a) \cap 1 \), by (iv);

\[
\begin{align*}
& = (a \cup a) \cap (a \cup a'), \text{ by (v)}; \\
& = a \cup (a \cap a'), \text{ by (iii)}; \\
& = a \cup 0, \text{ by (v)}; \\
& = a, \text{ by (iv)}.
\end{align*}
\]

The dual gives us \( a \cap a = a \).

Proof: (xi) \( a \cap 0 = 0 \cup (a \cap 0) \); 

\[
\begin{align*}
& = (a \cap a') \cup (a \cap 0), \quad \text{(iv)}; \\
& = (a \cap a') \cup (a \cap 0), \quad \text{(v)} ;
\end{align*}
\]
\[= a \cap (a' \cup 0), \quad \text{(iii)};\]
\[= a \cap a', \quad \text{(iv)};\]
\[= 0, \quad \text{(v)}.\]

Duality gives us \(a \cup \bar{i} = 1.\)

Proof: (xii) \(a \cup (a \cap b) = (a \cap 1) \cup (a \cap b), \quad \text{(iv)};\)
\[= a \cap (1 \cup b), \quad \text{(iii)};\]
\[= a \cap 1, \quad \text{(xi)};\]
\[= a, \quad \text{(iv)};\]
\[a \cap (a \cup b) = a, \quad \text{(duality)}.\]

Proof: (xiii) \((a \cup b) \cup (a' \cap b') = ((a \cup b) \cup a') \cap ((a \cup b) \cup b'), \quad \text{(iii)};\)
\[=((a \cup a') \cup b) \cap (a \cup (b \cup b')), \quad \text{(i)};\]
\[=(1 \cup b) \cap (a \cup 1), \quad \text{(v)};\]
\[=1 \cap 1, \quad \text{(xi)};\]
\[=1, \quad \text{(x)};\]
\[(a \cup b)' = a' \cap b', \quad \text{(vii)};\]
\[(a \cap b)' = a' \cup b', \quad \text{(duality)}.\]

Venn diagrams. John Venn (1834-1923), a contemporary of Boole's and also an Englishman, invented a way of representing Boolean expressions by drawing a fence around all members of a set so as to exclude all nonmembers. The "area" common to both of two sets can be represented by shading that region different from the rest (see Figure 2). This shaded region represents the logical product of the two sets. The logical sum can be represented similarly (see Figure 3). Similar diagrams had been invented independently by Leonhard Euler (1707-1783) and by Augustus De
Consider the Venn diagram of Figure 1. The set $A$ is represented by the circle. The entire rectangle (including $A$) is equal to 1. Since the circle represents $A$, the remainder of the rectangle is $A'$, and $A \cup A' = 1$.

![Figure 1.](image)

The Venn diagram in Figure 2 shows us the possibilities of the logical product of sets $A, B,$ and $C$. These possibilities are often referred to as AND functions.

![Figure 2.](image)

The logical sum can be represented by the complete shading-in of the circles which represent the sets involved. In the first part of Figure 3, illustrating the function $A \cup B$, the $A$ and $B$ circles are completely shaded. Later on in the text, when we
discuss switching circuits, this indicates that transmission of current will occur if A is present, or if B is present, or if both A and B are present (hence, the name OR function). Notice that a portion of circle C is shaded. This can be interpreted to mean that transmission will occur with the presence of C only if it is accompanied by A or B or by A and B. Signal C itself has nothing to do with transmission, as indicated by the fact that the portion of circle C which is not intersected by any other circle is left unshaded.

\[
\begin{align*}
\text{A} \cup \text{B} & \quad \text{A} \cup \text{C} & \quad \text{A} \cup \text{B} \cup \text{C} \\
\text{Figure 3.} & \quad & \quad \\
\text{Figure 4 illustrates how the binary operations } \cup \text{ and } \cap \text{ can be diagramed to show that } (\text{A} \cup \text{B}) \cap (\text{A} \cup \text{C}) = \text{A} \cup \text{B} \cap \text{C}. \text{ The crosshatched area in the third part of the diagram indicates the function } (\text{A} \cup \text{B}) \cap (\text{A} \cup \text{C}). \text{ This is redrawn in the fourth part and reinterpreted as the relationship } \text{A} \cup (\text{B} \cap \text{C}).
\end{align*}
\]

\[
\begin{align*}
\text{A} \cup \text{B} & \quad \text{A} \cup \text{C} & \quad (\text{A} \cup \text{B}) \cap (\text{A} \cup \text{C}) & \quad \text{A} \cup (\text{B} \cap \text{C}) \\
\text{Figure 4.} & \quad & \quad & \quad \\
\end{align*}
\]
The possibilities of any function's involving three or fewer sets can be illustrated by Venn diagrams. When more than three sets are involved, the Venn diagram technique can become quite difficult and cumbersome. Some interesting possibilities do exist and I encourage the reader to investigate some of these.

Truth table construction and use. After Boolean functions have been derived by the application of the various axioms and theorems already stated, the functions may be analyzed for all possible input conditions by use of a special chart. The chart can list all the variables horizontally and all possible 0 and 1 combinations vertically. In this way, 1 and 0 are substituted for each variable so that all possible combinations are included. Such a chart may be called the universal table.

The number of possible combinations for the states 1 and 0 is determined by the formula \( C = S^n = 2^n \) where \( C \) = number of possible combinations, \( n \) = number of variables being considered, and \( S \) = number of possible states (which is 2, since 1 and 0 are the only possible states being considered). Two variables would, therefore, have four possible combinations. Three variables would have eight possible combinations. A universal table can be made for any 1 number of variables. Figure 5 is a universal table for two and three variables.

Truth-value assignments can be summarized in truth tables.

---

1 See A.C. Gillie, *Binary Arithmetic and Boolean Algebra*, pp.95-96.
where you can display the assignments for all possibilities of truth values to the variables and functions being considered. It should be noted that the universal table is the same for any function having three variables, but the output of the truth table is determined by the Boolean function of the particular arrangement under analysis.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.

In Figure 6 a proof of the first distributive axiom is illustrated in the first part of the diagram, a proof of the second distributive axiom is illustrated in the second part. Since both of these functions have three variables, their universal tables are identical. The fourth and sixth column of the table are the truth tables of the particular function under analysis.

This table (Figure 6) is an excellent example of what happens in switching circuits. Imagine 0 representing no current and 1 representing current. Note on the table that all 0's in gives 0 out and all 1's in gives 1 out. Note also that 1 on A, 0 on B and C, give 1 out on the top half of the table but 1 on A, 0 on B and C, give 0 out on the bottom half. The reason is due...
to the difference in the two Boolean functions under analysis.

Switching circuits will be more thoroughly explained in the next section.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>(A \cup (B \cap C))</th>
<th>out</th>
<th>((A \cup B) \cap (A \cap C))</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>(A \cap (B \cup C))</td>
<td>out</td>
<td>((A \cap B) \cup (A \cap C))</td>
<td>out</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----------------</td>
<td>-----</td>
<td>----------------</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6.

Boolean algebra to switching circuits. The most interesting recent development in connection with Boolean algebra is its application to the design of electronic computers through the interpretation of Boolean combinations of sets as switching circuits. The logical product of two sets corresponds to a circuit with two switches connected in series. Current flows in such a circuit only if both the first and second switches are closed. The logical sum of two sets corresponds to a circuit with two switches in parallel. Current flows in this circuit if either one or the other or both switches are closed. Figure 7 illustrates various combinations.

The last two circuits are equivalent (they correspond to identical sets by the distributive law), but the components necessary for...
the first of these, $A \cap (B \cup C)$, makes that one simpler and probably cheaper to construct. Naturally, economy is a consideration in constructing computers. Recent progress in economizing computers has brought prices on some models to as low as $2,000 from highs in the hundreds of thousands of dollars twenty years ago.

\[
\begin{align*}
\text{A} \cap \text{B} & \quad (\text{in}) \quad /A \quad /B \quad (\text{out}) \\
\text{A} \cup \text{B} & \quad (\text{in}) \quad /A \quad /B \quad \quad (\text{out}) \\
\text{A} \cap (\text{B} \cup \text{C}) & \quad (\text{in}) \quad /A \quad /B \quad /C \quad (\text{out}) \\
(\text{A} \cap \text{B}) \cup (\text{A} \cap \text{C}) & \quad (\text{in}) \quad /A \quad /B \quad /A \quad /C \quad (\text{out}) 
\end{align*}
\]

Figure 7.

Utilizing the information in Figure 6, note that in the first part the output is zero in both circuits when all switches are open or switch A is open and either B or C is open. The output is 1 when switch A is open and both switches B and C are closed or switch A is closed, regardless of the condition of switches B and C.

There are names for these different switch combinations as they are used in a computer. The first switch combination diagramed in Figure 7 is called an AND gate ($A \cap B$). The second diagram is called an OR gate ($A \cup B$). A variation of the OR gate is called the exclusive OR and is similar to the inclusive OR shown with the exception that one set of conditions for A and B are excluded.
This exclusion can be symbolized in the circuit diagram by connecting the two switches mechanically together. This connection, if done so the switches oppose each other, makes it impossible for the switches to be closed simultaneously, although they may be open simultaneously or individually. Thus, the circuit is completed when \( A=1 \) and \( B=0 \), and when \( A=0 \) and \( B=1 \). The complements of the OR and AND gates are referred to as NOR and NAND gates respectively. This is accomplished by placing what is known as an INVERTER in the circuit, which means a 1 into the INVERTER gives a 0 out and vice versa. This is often referred to as negating the gate. Boolean algebra functions can be negated in the same way by the complement of the variables and operations of the function.

The standard logic symbols for the construction of AND, OR, INVERT gates, etc. have been presented. Each symbol has a set of specific relations between input and output conditions. These same logic functions, or operators, can be represented using algebraic signs. The signs used to express the logic relations between Boolean variables that are commonly used are shown in Figure 8.

1) AND: \( \cdot \) (dot);
2) OR: \( + \) (plus sign);
3) EXCLUSIVE OR: \( \pm \) (plus or minus sign);
4) NEGATION: \( ' \) (prime sign);
5) EQUALITY: \( \equiv \) (equivalence sign).

Figure 8.
Truth tables can be set up to illustrate these relations. These truth tables of all the basic Boolean operations are shown in Figure 9.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A·B</th>
<th>A+B</th>
<th>A⇒B</th>
<th>A'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AND  OR  X-OR  EQUAL  NEGATE

Figure 9.

The axioms of Boolean algebra that have already been covered can be illustrated using the algebraic signs. That is, the associative, commutative, distributive, identity, and complement axioms (i-v). All you have to do is substitute $\cdot$ for $\cup$ and $+$ for $\cup$. It should be noted that the first part of the distributive axiom is not valid in ordinary algebra, although, if you make the symbol substitution, it appears to indicate that it is. Remember, we are dealing with Boolean algebra, the one basic difference between it and ordinary algebra has already been stated.

There are nine basic identities for the gates illustrated using one variable and one constant and using only AND, OR, and INVERT operators. These are shown in Figure 10.

1) $A''=A$, (viii) \text{ INVERT identity}
2) $A\cdot1=A$, (iv)
3) $A\cdot0=0$, (xi) \text{ AND identities}
4) $A\cdot A=A$, (x)
5) $A\cdot A'=0$, (v)

Figure 10.
These gating networks that have been outlined have logical symbols for use in diagrams to illustrate their function without drawing tedious electronic schematics. Figure 11 lists the logical function, symbol, and corresponding Boolean expression. The exclusive OR and equivalence functions are made up of a combination of these logic symbols and are, therefore, not shown in this particular Figure.

<table>
<thead>
<tr>
<th>Function</th>
<th>Logic Symbol</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A - B</td>
<td>A·B</td>
</tr>
<tr>
<td>OR</td>
<td>A - B</td>
<td>A+B</td>
</tr>
<tr>
<td>NAND</td>
<td>A - B</td>
<td>(A·B)'</td>
</tr>
<tr>
<td>NOR</td>
<td>A - B</td>
<td>(A+B)'</td>
</tr>
<tr>
<td>INVERTER</td>
<td>A - A' or A'</td>
<td>A' or A</td>
</tr>
<tr>
<td>negated input AND</td>
<td>A - B</td>
<td>A'·B'</td>
</tr>
<tr>
<td>negated input OR</td>
<td>A - B</td>
<td>A'+B'</td>
</tr>
</tbody>
</table>

Figure 11.
We are now at the point where we are ready to discuss the computer. In the next chapter we will see how binary arithmetic is coupled with Boolean algebra and electronics to form the digital computer.
Exercises:

1. In contrast to George Boole who used 0 and 1 for the empty set and universal set respectively, what are the present day symbols commonly used for these two sets?

2. What is the one basic difference between ordinary algebra and Boolean algebra?

3. What is the principle of duality and what is its chief benefit?

4. Simplify: \((x \cap y) \cup ((x \cup y') \cap y)'\).

5. Prove: If \(a \cap b = a \cap c\) and \(a \cup b = a \cup c\) for any \(a, b, c \in \mathbb{B}\), then \(b = c\).

6. Simplify: \((a \cup b) \cap a' \cap b'\).

7. Draw two identical Venn diagrams showing two intersecting sets in a universal set. Label the intersecting sets A and B in each diagram. Shade in \((A \cup B)'\) in one diagram and \(A' \cap B'\) in the other. What do you observe?

8. Draw two identical Venn diagrams showing three intersecting sets in a universal set. Label the intersecting sets A, B, and C in each diagram. Shade in \(A \cap (B \cup C)\) in one diagram and \((A \cap B) \cup (A \cap C)\) in the other. What do you observe? What conclusion can you draw from your observation?

9. Construct a universal table utilizing four variables.

10. Construct a switching circuit to represent the Boolean function \((a \cap c) \cup (b \cap (b' \cup c)) \cap (a' \cup (b \cap c'))\). (Hint: this function can be simplified).

11. Using the logic symbols shown in Figure 11 on page 63 show the Boolean function \((A \cdot B) + (C \cdot D)\).
CHAPTER 5

The Computer

Introduction. Modern computers are of two general types, analog or digital. Analog computers process analog signals and digital computers process digital signals. An analog computer generally measures on a continuous scale much like an automobile speedometer. It can take on any one of an infinite number of possible values by translating physical conditions such as flow, temperature, pressure, angular position, or voltage into related mechanical or electrical quantities. The analog computer is interested in the value of a signal. It uses mechanical or electrical equivalent circuits as a representation for the physical phenomenon being investigated. It can represent numbers by physical quantities such as by lengths, as in a slide rule, or by voltage currents. The analog computer is employed primarily in the fields of engineering and science. In contrast, digital computers are much more versatile and are used extensively in the fields of physics, mathematics, and business and, more recently, law enforcement. Digital computers are also used in connection with analog computers for automatic machine control.

Modern digital computers are extremely fast, so fast that their speed is almost beyond the grasp of human understanding. Some can now perform over one billion operations per second. Problems that formerly required years to solve can presently be solved in days, hours, even minutes by digital computers. These computers can
exchange information with each other, some at the rate of 75,000 words per minute. Our space program is a good example of computers working together.

Both analog and digital computers have advantages in different applications. Figure 1 illustrates some comparisons.

<table>
<thead>
<tr>
<th>Analog Computer</th>
<th>Digital Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets up analogy of problem.</td>
<td>Breaks problem down into arithmetic.</td>
</tr>
<tr>
<td>Represents physical variable by continuous measurement of analogous quantity.</td>
<td>Represents numbers by discrete, coded pattern.</td>
</tr>
<tr>
<td>Basic operation performed by relatively few single-purpose devices.</td>
<td>Operations performed by relatively many interchangeable arithmetic devices.</td>
</tr>
<tr>
<td>Relatively few devices needed; hence low cost and ease of programming.</td>
<td>Many devices needed; hence, high cost and harder to program.</td>
</tr>
<tr>
<td>Distinct elements used for each operation.</td>
<td>Identical elements used in sequence.</td>
</tr>
<tr>
<td>Accuracy limited to about 1 part in 100,000.</td>
<td>Unlimited accuracy.</td>
</tr>
<tr>
<td>Data storage (memory) dispersed in various non-interchangeable devices.</td>
<td>Data storage concentrated in space, interchangeable and unlimited in duration.</td>
</tr>
<tr>
<td>Analog computer serves as model and mirrors relations of actual system; operations usually carried out in actual (real) time of physical system.</td>
<td>Digital computer compounds arithmetic data, unrelated to system it represents. Time of operations usually does not correspond to real time.</td>
</tr>
</tbody>
</table>

Figure 1.

Analog Computer

- Represents physical or mathematical quantities.
- Best suited to represent measurable quantities and simulate response of physical systems by mathematical analogies.

* Digital Computer

- Can represent numbers, as well as letters and other symbols.
- Best suited to handle discrete random processes, statistical data, and numerical problems of business and scientific nature.

Figure 1 (continued).

Since most computers are digital or predominantly digital, the remainder of this chapter is concerned only with digital computers.

Parts of the digital computer. The basic units of a digital computer are the following: (1) the input unit, for translating into usable form all input information, (2) the arithmetic and logic unit that performs the actual arithmetic operations, (3) the memory or storage unit that retains the program (instructions), problem, and solution, (4) the control unit, which directs the computer operations, and (5) the output unit that translates the computer output into a form understandable by the operator (see Figure 2).
The input unit supplies the computer with basic data and also passes on the instructions or program from the operator that tell it what to do with the basic data. A manually operated keyboard may be used to accomplish this. Some computers require faster inputs and, therefore, use punched cards for data inputs. Input may also be accomplished via a punched paper tape or magnetic tape. Input has also been developed through the use of TV-type cathode-ray tubes and audio input is being perfected with a limited vocabulary. Frequently, the input is in analog form and is converted into a digital format by an analog-to-digital converter. The input system converts this information into a series of signals. Each signal is merely the presence or absence of a voltage or electric current. A signal is either there or not there at a particular time. When the information is put into this form, the computer is able to work with the information and process it through a series of logical operations.

The data or instructions, now in the form of signals, are next sent to the computer memory, or storage. This unit stores information until it is needed. It contains information for the control unit (instructions) and for the arithmetic unit (data). Some computer texts refer to external units as storage, such as magnetic tapes and disks, and to internal units as memory, such as magnetic cores. The memory unit is, in a sense, passive in that it merely receives data, stores it, and gives it up on demand. Each location within the memory unit has a unique address where the information can be
Instructions and data are written into or read out from internal storage without discriminating between them. The only way instructions can be distinguished from working data is by the routing of signals from internal storage. Signals routed to the control unit are interpreted as instructions, while those routed to the arithmetic unit are considered data. If data is sent, by mistake, to the control unit instead of the arithmetic unit it is treated as instructions, this will undoubtedly result in confusion and an error alarm will be activated and the computer will halt. The order of the data being placed into or taken out of memory is governed by the instructions contained in the program.

The next element, the control unit, is one of the most active parts of the computer. It selects information from the memory in the proper sequence and sends the information to other elements. This unit is the master dispatching station and clock of the computer. It directs data through the system and controls the sequence of operations. It must interpret the coded instructions contained in the program and initiate the proper commands (signals) to the various computer units. This succession of signals flows along control lines, opening or closing switches (gates) that connect specific registers to other units of the computer, and thereby controls the flow of data according to the sequence of the program. The control unit automatically times the computer and makes sure it acts as a fully integrated system.

The arithmetic unit receives the information and commands from
the control unit. Here the information, still in the form of symbolic signals or "words," is analyzed, broken down, combined, and rearranged in accordance with both the basic rules of logic designed into the computer and the commands received from the control unit. This unit performs the actual work of computation and calculation. It carries out its job by counting series of signals or by the use of logic circuits. Most modern computers use components such as transistors and integrated circuits. Subtraction can be carried out by adding complements, multiplication by repeated addition and shifts, and division by repeated subtractions and shifts. The arithmetic unit also makes various logical decisions, such as choosing between two alternatives, the familiar AND, OR, NOT (negate), and other operations already covered. The arithmetic unit must distinguish between positive and negative numbers and modify its behavior accordingly. For example, a number from the memory can be placed in the addition portion of the arithmetic unit. Upon receipt of an order such as AND, a number will be added to the one already in the unit, this sum will then be placed in the "accumulator." This sum can be transferred to the memory or can be retained by the accumulator for further operations. In the case of a negative number, the unit will recognize the negative sign. For subtracting, the unit finds the difference and attaches the proper sign. If the data is in the form of letters (alphanumeric), a logical comparison may be required to determine which word is earlier or later in a sequence. The sequence of
operations is a program, the individual steps are instructions.

When the signals have passed all the way through the arithmetic unit, they are no longer in the form of a problem but are now in the form of an answer or response to the program. This response is then passed through the memory unit and on to an output unit. This output unit accomplishes the reverse of the input unit, converting the new sequence of signals back into a form that is understood by the operator or by other computers or machines. It is usually in the form of a permanent typewritten record or a visual display of the response. The response may be (and often is) printed by the same keyboard or teletype used to input instructions and data.

Information passes through a computer in a systematic manner by timing all operations in terms of a basic cycle. This cycle consists of the time elapsed between the issuing of a command and the complete execution of this command by the computer. We will see in Part IV of this text how this basic cycle can play an important part in some programs. During a cycle the command will be carried out through a series of pulses (voltage-no voltage) occurring in fractions of a microsecond. The fastest rate at which pulses are transmitted and used throughout the computer is known as the repetition rate. The flow of information through the computer may be either sequential, through one channel at a time (called series operation), or it may occur simultaneously through several channels (parallel operation) as shown in Figure 3 for a signal involving six binary bits. Parallel is faster than series
since all digits are operated on at once, but requires more elaborate equipment.

![Series and Parallel Data Representation]

Figure 3.

The computer word. In digital computers, a word applies to any group of digits (bits) which are handled as a single group and not as individual units. One computer may use eight digits in a word, while another may be designed to have fewer or even more digits in a word. It is desirable, from a cost point of view, to minimize the number of bits required in the computer word.

It has already been mentioned that storage is broken into a number of addresses or locations. At each of these addresses a group of digits is stored and is handled by the computer as a unit. The group of digits stored at each address in memory is usually referred to as a word. Each address is assigned a number and the address is then referred to by that number. We say that address
15 contains the number 496 (binary representation, of course), or that address 25 contains an instruction word. Notice that the memory stores both number words and instruction words. Most computers can store instruction words and number words in the same locations (though not simultaneously). This makes the memory unit more flexible.

The time it takes to obtain a word from the memory unit is called the access time. Access time is an important factor in the speed of a computer. Until recently this time element impeded the construction of high-speed computers. Magnetic cores have proven to be the most reliable devices for a short access time plus the added ability to store information for an indefinite period of time.

The computer word must include, as a minimum, a command or instruction of what to do and the memory address of the information upon which the command is to be performed. The memory address of the next instruction to be performed may be specified in the word, if it is not the next memory address in sequence after the current instruction. Word length can be fixed or variable. Data and instructions can be combined in one word or handled as separate words.

A computer can be organized to operate with one word, containing both data and instruction, or two words, one for instructions and another for data. Using a single-word format, one part of each word always signifies the command. Using separate words, the only method
of distinguishing between an instruction and data may be by the location of the word within the memory unit.

Figure 4 shows three examples of possible word formats. The first example is 22 bits long, and the other two are 20 bits long. The first example shows a single-word format. This is divided into three parts: an operation code or instruction, a memory address, and the operand or number to be operated on. The operation code specifies a certain command from the computer command signals, i.e., an operation the computer can perform. In this example, the operation code is represented by the first 3 binary bits of the word. This example automatically limits the maximum possible number of different commands which the computer can execute to eight \(2^3\).

![Diagram of word formats]

Figure 4.

In Figure 4A, bits 4 to 10 identify the memory-address location. With this example \(2^7=128\) different words can be stored in the computer memory unit. The memory address can be used to specify the address of the next instruction to be performed or the address where the data being processed should be stored. The operand, or
data portion of the word, is given by bits 11 to 22. This would allow data words up to 12 bits, or 11 bits plus sign, to be handled by the computer. Using a series of such words, data can be stored, withdrawn, operated on, and the results restored for future use.

Figure 4B and C are examples of 20-bit fixed-word formats in which separate data words and command words are used in conjunction with one another. The words are distinguished from one another by their location. The command word contains instruction data, and the data word contains the number to be operated upon. Double-word format provides flexibility that is not attainable with the single-word format because it allows greater command-data capability and minimizes word inefficiencies.

The command word in Figure 4B has a 6-bit operation code. This allows \(2^6 = 64\) different command functions. The 14 bits of memory address indicate a memory capacity of up to \(2^{14} = 16,384\) words.

The data word in Figure 4C is a 20-bit binary number. A sign bit indicates whether the number represented by the remaining 19 bits is positive or negative. For example, 0=negative, 1=positive, or vice versa, depending on the configuration of the computer.

The number of divisions in the basic computer instruction word is primarily determined by the number of addresses which are referred to. Many computers have multiple-address instruction words. For example, a two-address computer will have a basic instruction word with three sections, the first consisting of the operation code and the second and third sections each containing
a particular address in memory.

Different computers use addresses differently. For example, both addresses in a two-address computer may specify operands, and the result will be stored at the first of the addresses. Another alternative is to have the first address specify the location of the word to be used, and the second address the location in memory of the next instruction to be used.

Word destination is determined by the control unit. Data words containing operand (numerical) data are always sent to one of the registers in the arithmetic unit upon the command of the control unit. In contrast, instruction words are sent to the memory buffer register in the control unit. They are held in the control unit until the operation code of the instruction is decoded. The function of the memory unit for either type of word is to store the information until needed and then supply the proper word at the proper time to the correct register, upon command of the control unit. Let's have a closer look at how this is accomplished.

Basic elements of control unit. The control elements of modern computers varies from eight for very small computers, to over 180. Obviously, in this text only the fundamentals of control elements can be described. To start with, some sort of basic timing signals must be provided, and it is the function of the control element to route these signals. Also, the operation code section of the instruction word must be decoded. The control unit must also have some means of communicating with memory for each instruction word
and calling out the operands from memory.

Figure 5 illustrates the control registers of a simplified single-address computer, along with indications of how information flows between registers.

From memory

Memory buffer register

Add 1 to counter

Operation register

Address register

Instruction counter

Memory address register

To operation decoder

To memory

Figure 5.

The basic registers of the control unit for our generalized computer are the instruction counter, the operation register, and the address register.

The instruction counter is a counter of the same length as the address section of the instruction word. The counter can be cleared, or 1 added to the contents of the counter upon receipt of each instruction. The contents of the counter can also be transferred to
the memory address register and used to locate a word in memory. Normally, the counter will be increased by 1 during the performance of each instruction and transferred into the memory address register. The contents of the counter will indicate the address of the instruction to be performed at the beginning of each instruction timing signal.

The operation register stores the operation-code digits which determine the operation to be performed. If the computer word has an operation code with a length of six bits, the operation register will be six binary bits in length. When the instruction word indicated by the counter is located, the operation-code bits of the word will be read into the operation register. The operation-decoder decodes the bits stored in the operation register and releases the basic timing signals that will guide the computer according to that instruction.

The address register contains the address in memory of the operand to be used, or the address at which the accumulator contents are to be stored, or the number of places a number is to be shifted, etc. The address section of the computer word (see Figure 4) is transferred from memory into the address register at the beginning of each instruction. In our example the address register can be transferred to either of two places: (1) the memory address register, after which the memory will locate the desired word or store information into the address selected, or (2) if the instruction is a branch-type instruction, the contents of the address register
can be transferred into the instruction counter.

**Address selection.** We have discussed how the control unit brings information from and sends information to the memory unit by means of the memory buffer register and memory address register. To accomplish this the control unit must know where to find or store data in the memory unit. This function is performed by the address selector. There are different types of storage systems, and a different method of address selection must be used for each of them. One method is magnetic-core memories, which fits in nicely with what has already been discussed so this method is the one we will concern ourselves with.

Magnetic-core memories are used when information is stored in a fixed location. Figure 6 illustrates a possible logic set-up for address selection in a fixed location. Individual AND gates control the selection of each row (X-rows) and column (Y-columns). The logic for the columns is not shown, since it is identical with that used for the rows.

Three binary bits, A, B, C, or a three-element octal digit, have eight possible combinations of truth values, and consequently any one of eight memory rows can be selected. A second octal digit selects any one of eight memory columns, so that the intersections with the selected row determine 64 possible memory locations.

Note the negated inputs on the AND gates. Inverters could have been used, but the diagram is simpler as shown. Each gate can respond to only one combination of truth values. The selection of
the proper address combination is determined by a control signal sent to the address register. Additional address digits are required if the memory contains more than eight rows and columns in a single plane or matrix.

Figure 6.
Operation of a typical arithmetic unit. The arithmetic unit of a digital computer consists of a number of registers in which information can be stored, and a set of logical circuits which make it possible to perform certain transformations on the information stored in the registers and to perform certain
operations between registers (see Figure 7). The sequence of operations are broken into small steps by the logic circuits of the arithmetic control. Coded commands from the control unit to the arithmetic unit may include the following:

1. Clear all registers for new information.
2. Read in new data.
3. Read out results.
4. Add.
5. Subtract (add complements).
6. Shift right for division and multiplication.
7. Shift left
8. Shift data out of accumulator register and reinsert into input.
10. Round off numbers to specified number of digits.
11. Double precision—join Q and A-registers for computations with double-length words.

Figure 7 shows a functional block diagram of a typical arithmetic unit that would receive these commands. A minimum of three registers are required to perform arithmetical or logical operations. Two to receive and temporarily store numbers (operands), and a third (accumulator) to store the results before transfer to memory unit.
A more refined computer would include a fourth register for multiplication and division (Q-register) even though these operations could be performed by the first two. The adder and accumulator register are often combined into one register called
the A-register. An auxiliary register, called the B-register, holds one of the arithmetic operands extracted from the memory unit and delivers it, when required, to the A-register. This operand could be any one of an addend, subtrahend, multiplicand, or a divisor. For example, in addition the auxiliary register-B delivers the required addend directly to the adder in register-A. The other operand (augend) is supplied by the accumulator, having been previously stored there. The adder accomplishes its task and the result is stored back in the accumulator.

Since the accumulator is not performing operations, it doesn't matter what is in it when information is put there. Data will be the same when it leaves the accumulator as it was when it went in. New data and operations in other registers will cause the accumulator to attain partial or final results as information goes in and is called out of this very active element. For multiplication and division, one of the operands (multiplicand or divisor) is delivered to the accumulator by the B-register, while the other is supplied by the Q-register.

![Figure 7.](image-url)
It is not the purpose of this text to show a complete design of a digital computer, but only to acquaint the reader with the basic functions internal to the computer so you may be able to use it more efficiently. For this purpose I have chosen to illustrate how serial addition is accomplished in the arithmetic unit.

Serial addition is accomplished by the use of two half adders an OR circuit and a 1-pulse delay element. This latter element retards the signal one binary bit later coming out then when it went in. Figure 8 shows a block diagram of how a serial adder might be constructed in some computers.

The addend and augend data are delivered, upon command of the control unit, from memory to registers A and B. Both registers shift the addend and augend digits to the right, bit-by-bit, to the first half-adder, where they are added immediately. Clock pulses from the control unit accomplish this bit-by-bit shift. The first pair of digits to enter the adder are from the lowest order column of each of the two binary numbers. Output of the first half-adder consists of the sum of the two digits and a possible carry (to be added to the next higher order column). The sum is sent to the second half-adder, the carry bit (if any) is passed through an OR gate to a delay element, where it is delayed by one pulse interval (a bit later).

Upon command of the next clock pulse, another pair of digits are shifted from registers A and B and added by the first half-adder. The resulting sum is again sent to the second half-adder, while any possible carry is sent to the delay element. Simultaneously the
second half-adder, on this second clock pulse, adds the two bits at its gates (the sum of the first two bits and 0, since the carry bit is being delayed). This sum is fed back through an OR gate to the input of the accumulator, whose left-hand portion has been emptied by the shift to the right. Any possible carry of the second half-adder is sent through an OR gate to the delay element. This process is repeated until all the digits of the two numbers have been added and the sum is in the accumulator, while the B-register is empty. A sensing device will tell control the sum has been found and the information can be stored in memory or added to another. Note that the output of the delay element is automatically 0 on the first clock pulse.

Figure 8.
There are many possible ways of implementing a half-adder, the following are three examples. Half-adder 1 implements the expression:

$$\text{Sum} = A \cdot B' + A' \cdot B \quad \text{and} \quad \text{Carry} = A \cdot B$$

Figure 9.

Half-adder 2 implements the expression

$$\text{Sum} = (A + B) \cdot (A' + B') \quad \text{and} \quad \text{Carry} = (A' + B')'$$

Figure 10.

Half-adder 3 implements the expression

$$\text{Sum} = (A \cdot B)' \cdot (A + B) \quad \text{and} \quad \text{Carry} = A \cdot B$$

Figure 11.
Note, in Figure 9, the commutative principle (illustrated in Chapter 4), allows the sum to be expressed in various orders. The same principle can be applied in Figures 10 and 11 to give a different order to the sum functions. This difference in the way the sum functions can be expressed is not to be confused with a difference in the signal at the outputs of the half-adder. The signal remains the same no matter how many different ways one might choose to express the sum, i.e., 1 or 0. Thus, although the logic of the three figures is different, all three half-adders produce the same result.

The arithmetic unit working in conjunction with the computer's memory unit can carry out a long string of calculations without the need to obtain more information from the operator of the input unit. To do this automatically the computer must continually instruct itself what to do next to solve a particular problem. All necessary instructions must be prepared in advance in the form of a complete program that is stored, step-by-step, in the computer's memory. This program was devised and placed in memory by the operator to accomplish the task of solving certain types of problems. Without instructions the computer cannot function, it is not capable of thinking (contrary to some beliefs). These instructions are selected in the proper sequence from memory by the control unit, which co-ordinates all operations. The next part in this text will take up the subject of programming.
Exercises:

1. There is one main difference between the analog and digital computer in the types of signals each processes. What is this difference? Define this difference in easy to understand terms.

2. Which computer is generally faster, the analog or the digital?

3. Which computer is generally more accurate, the analog or the digital?

4. Which computer is generally more expensive, the analog or the digital?

5. List the basic units of the digital computer and their function.

6. What happens to an analog signal being put into a digital computer? Describe this signal after it is in the computer.

7. Where is the signal from question 6 first sent?

8. Where are instructions routed within the digital computer?

9. Where is data routed within the digital computer?

10. What unit of the computer acts as a timing device?

11. What unit performs the job of computation and calculation?

12. How is subtraction generally performed by the computer? Try and explain your answer using base two numbers.

13. What is the difference between a program and instructions?


15. Which operation is faster, series or parallel?

16. Define a computer word.

17. Define: access time.

18. List three types of computer words and describe each.
19. If the memory address of a computer word had space for 5 bits of information how many different word locations would this provide?

20. List the basic elements of the control unit.

21. Describe the function of the address register within the control unit.

22. What is the minimum number of registers required to perform arithmetic operations within the arithmetic unit? Describe how they function.

23. Using Figure 9 on page 86, if a 1 is sent in A and a 1 is sent in B, what is A'·B? What is A'·B+B'·A?
PART III

PROGRAMMING

Mathematics— the unshaken
Foundation of Sciences, and the
plentiful Fountain of Advantage
to human affairs.

Isaac Barrow

1630-1677
CHAPTER 6

Flowcharting

Introduction. Any individual who has ever had to assemble something from a set of plans should have no trouble understanding the purpose and usefulness of a flowchart. A flowchart is a graphical representation of a given problem, indicating the logical sequence of operations that the computer is to perform. Drawing a flowchart is a good way to express on paper the types, as well as sequence, of operations necessary to solve a problem. Although some flowcharts are quite complicated it is not necessary for students to master sophisticated programming or flowcharting techniques to solve problems. Flowcharts are a means of helping solve problems, not an end.

The flowchart is basically a collection of geometric figures with statements or questions written inside them, lines connecting the figures, and arrows indicating direction through the flowchart.

The following examples should serve to acquaint the student with the flowchart and how it is assembled and used.

Some figures and what they mean.

This figure represents the terminal, where every program starts and finishes.
This figure represents a predefined process, or subprogram, that is necessary to solve some of the more complex programs presented.

This figure represents an operation that is to be performed, or one step in solving a problem.

This diamond shaped figure represents a decision point in the program (branching). Direction of flow is determined by the answer to the question, either yes or no.

This quadrilateral type of figure, sometimes referred to as a trapazoid, represents a print out of information at the terminal.

This figure is a connector. It is used to connect part of the flowchart to another part on the same page or even another page. The number 10 means that is the number of the instruction the program is to branch to.
Straight-line programming. This is a flowchart that follows a step by step straight sequence (no branching) for finding the sum of three numbers. The accumulator will contain the sum when all three numbers have been added. The accumulator will retain the sum until cleared.
Program branching. This program has two loops in it. First it waits for a non-negative number to be entered. Then it types out that number, followed by all the smaller positive integers and zero, counting backwards. The first loop has two instructions, and the second loop has four instructions. The start and end instructions are not part of either loop.

This flowchart is a very important illustration of the way to use loops. Note in the second loop that A changes each time it is executed. Here is a program that lets you type in positive numbers and decrease them by one until you reach the value of zero. Without the decision operation the loop would be infinite. That is, there would be no escape from it. It is always important to provide a test of the contents of a loop so that the program might be completed.

The next flowchart is very important in the understanding of nested loops (one within the other). We will show how this is
accomplished when we discuss the computer language BASIC in the next chapter. The reader should understand that the sequence of operation follows the same pattern already described except that when instructed to leave the inner loop the flow of operation finds itself in an outer loop briefly then back into the inner loop for another sequence of operations.

This is a flowchart for printing the whole numbers from 0 to 99.
Exercises:

1. Draw a flowchart for finding the largest of three integers (Warning: consider negative quantities).

2. Draw a flowchart for finding the square root of any perfect squares inputed considering all possible numerical inputs (Warning: don't forget rational numbers).
CHAPTER 7
A Computer Language—BASIC

Introduction. BASIC (Beginners All-purpose Symbolic Instruction Code) is a combination of simple English and algebra. It was originally developed at Dartmouth College. It is similar in some respects to an older computer language called FORTRAN but is much simpler to learn and understand. The main reason for its simplicity is that the language is very conversational. It allows a conversation to take place between the operator and the computer in ENGLISH, as long as the operator follows certain rules. It is the purpose of this chapter to outline the rules and their meaning.

Commands. Commands are executed immediately, they do not require statement numbers. Statement numbers are essential in all other operations or statements. Statements may be entered in any order. The computer keeps them in numerical order no matter how they are entered and operates according to this numerical order.

Commands tell the computer what to expect or what the operator expects from the computer. The following is a list of commands and a brief definition of what they accomplish.

APPEND—Appends the named program to current program.

BYE—Log off. Disconnects teletype from computer on time-share system.

CATALOG—Produces a listing of user library program names and length in two-character words.

DELETE—Deletes all statements after and including the specified one
or between and including the specified ones.

ECHO-Permits use of a half duplex coupler, entered after logging in.

GET-Retrieves the specified program from the user's library and makes it the current program.

GET-$-Retrieves the named program from the system library.

HELLO-Log on. User needs I.D. code and password.

KEY-Returns control to keyboard after TAPE inputs.

KILL-Deletes the specified program from the user's library (does not modify the current program).

LENGTH-Produces a listing of the current program length in two-character words.

LIBRARY-Produces a listing of system library program names, and size in two-character words.

LIST-Produces a listing of current program.

NAME-Assigns specified name to the current program, name may be 1 to 6 characters in length and must include only printing characters.

PUNCH-Punches current program to paper tape.

RENUMBER-Renumber program from 10 in multiples of 10.

RUN-Starts program execution.

SAVE-Saves the current program in user's library under the specified name.

SCRATCH-Erases current program (but not program name).

TAPE-Informs computer that following input is from paper tape.

TIME-Produces a listing of terminal and account time.
Operators. In BASIC the computer accepts the symbols +, -, *, and / for the operations of addition, subtraction, multiplication, and division, respectively. There are no symbols such as X and ↑, and the computer, programmed in BASIC, wouldn't understand them if they did exist on the teletype keyboard. The following is a listing of the BASIC operators and their meaning:

= Assignment operator; assigns a value to a variable. May also be used with LET.

↑ Exponentiation (as in X^2, written X↑2).

* Multiplication symbol.

/ Division symbol.

+ Addition symbol.

- Subtraction symbol. This symbol is also used as a sign for negative numbers.

It is important to note that in BASIC statements involving mixed arithmetic operations there exists an order of precedence. The order of precedence (hierarchy) is:

1. Paranthetic values operated on first. The computer will always operate on problems inside parenthesis first. It is good practice to group arithmetic operations with parenthesis when unsure of the exact order of precedence.

2. Exponential (^) quantities are evaluated.

3. Multiplication and division problems are accomplished.

4. Addition and subtraction problems are accomplished.

Note: operators on the same level of priority are acted upon from left to right in a statement.
# Expression does not equal (#) expression.
<> Expression does not equal (<>) expression.
> Expression is greater than (>) expression.
< Expression is less than (<) expression.
>= Expression is greater than or equal to (>=) expression.
<= Expression is less than or equal to (<=) expression.
AND - Expression 1 and expression 2 in statement must both be true for statement to be true.
OR - If either expression 1 or expression 2 is true, statement is true.
NOT - Statement is true when expression is false.
Note: the numerical values used in the logical evaluation of the operators just listed are--true =any nonzero number; false =0.
These operators are often used in branching instructions. For example:
10 If A<>B THEN 40
means if expression A is not equal to expression B then go to statement 40.
20 IF (C=D) AND (D<0) THEN 50
means if C equals D and D is negative then go to statement 50.
The last two operators we need to know are:
MAX - Evaluates for the larger of the two expressions. Often used with a LET statement (15 LET C=E(1) MAX E(2)).
MIN - Evaluates for the smaller of the two expressions.
Statements. Statements are used as instructions to the computer. They are ordered according to their statement numbers, and executed according to this order. Statements cannot be executed without running a program. They tell the computer what to do while a particular program is running. The command RUN initiates a program and the statements in the program take over at this point. The only interruptions made by the operator are statements requiring a response from the operator or X OFF key on the teletype, which will initiate a break in program operation with STOP, and error messages which are coded according to manufacturers design. It is not necessary for the student to memorize every detail in the statements section of this text. By application of simple programs (to start) the student will gradually obtain a working knowledge of computer programming. The following is a listing of statements and their purpose, including examples.

DATA-Specifies data, read from left to right. Often used in conjunction with READ.

Example: 10 READ A
40 DATA 16

READ-Reads information from DATA statement. When looping, more than one bit of data may be used. When the first READ statement is encountered, the pointer (which keeps track of data in BASIC) indicates that the first item in the first DATA statement (lowest statement number) is to be read, the pointer then moves to the next bit of data.

Example: 10 READ B
50 DATA 16,-5,0,4
If there are more loops in the program than bits of data the computer will print an error code or "out of data." The pointer remains at the last bit of data until the program ends or is re-run. To re-read the data in a program it is necessary to reset the pointer. A RESTORE statement moves the pointer back to the first data item. RESTORE—Permits re-reading data without re-running the program.

Often the RESTORE statement will follow a NEXT statement in a FOR-NEXT loop.

Example: 
10 FOR I=1 TO 5
20 READ A
40 NEXT I
50 RESTORE

The student should note that there would exist operational statements that utilize the data. These statements would be located between statements 20 and 40 in the above example.

FOR...NEXT—Often referred to as nesting loops. Executes statements between FOR and NEXT the specified number of times (a loop), and in increments of the size indicated after STEP. STEP and step size may be omitted which will automatically cause the increments to be one (1). Note the previous example.

Example: 
10 FOR A=1 TO 5
20 FOR B=2 TO N-1
30 FOR C=X TO Y STEP 3
40 40 40 40
90 NEXT C
100 NEXT B
110 NEXT A

The student should take very careful note that the range of loop C is from statement 30 to 90, the range of loop B is from statement
20 to 100, and the range of loop A is from statement 10 to 110. The range of FOR...NEXT loops may not overlap, they are nested within another loop or are the last or only loop in the program. An example of an incorrectly nested loop would be:

```
5 FOR A=1 TO 5
10 FOR B=2 TO N-1
 .
 .
80 NEXT A
90 NEXT B
```

The respective ranges of the loops overlap which will cause an error code message to be printed.

IF...THEN-Logical test of specified condition, transfers control if true. This statement is sometimes described as a conditional transfer. If the specified condition for transfer is false, the program will continue to the next statement in sequence.

Example: 10 READ A,B
20 IF A > B THEN 50
30 LET C=B↑2-A↑2
40 GO TO 60
50 LET C=A↑2-B↑2
60 PRINT A,B,C
70 GO TO 10
80 DATA 5,10,14,5,7,8,41,33
90 END

In this example several statements have been used that haven't, as yet, been explained. Let the following suffice:

LET -Assigns a variable a value.
GO TO -Transfers control (jumps) to specified statement number.
PRINT -Prints the specified values, five fields per line when commas are used as separators, up to 12 fields per line when semicolons are used as separators. PRINT by itself causes the
teletype to advance one line (used for spacing).

END -Terminates the program. This is always the last statement in a program and must be there for the program to be complete.

There are a few more statements to cover that are not as frequently used in the initial stages of learning BASIC but are essential for more advanced programming.

INPUT -Allows data to be entered from teletype while a program is running. The program will halt when it encounters an INPUT statement and prints a question mark. The program will not continue until the input requirements are satisfied.

Example: 10 PRINT "WHAT VALUE DO YOU WANT MULTIPLIED?"
20 INPUT A
30 PRINT "WHAT DO YOU WANT TO MULTIPLY BY?"
40 INPUT B
50 LET C=A*B
60 PRINT "THE PRODUCT OF""A""AND""B""IS""C
70 END

In this program the PRINT statements in 10, 30, and 60 are a bit different than previously described and illustrated. When the computer encounters a PRINT statement, as in 10 and 30 above, where there exists a statement in quotes following the PRINT instruction the computer simply prints verbatim whatever is within quotes after the PRINT statement. In statement 60 the computer prints exactly what is within the quotes but when it encounters the variables A, B, and C it prints the quantity that has been computed or stored for that variable.

Example: In the above program, if you input the number 7 when the computer asks you-WHAT VALUE DO YOU WANT MULTIPLIED?-and 3 when the computer asks-WHAT DO YOU WANT TO MULTIPLY BY?-
the computer will then type-THE PRODUCT OF 7 AND 3 IS 21.

A sophisticated programmer would refine the above program so that the INPUT question mark will assume the position of the question mark in the questions of statements 10 and 30. This is accomplished by following the closing quote of each of the two statements with a semicolon. Ending a PRINT statement with a semicolon causes subsequent output to be printed on the same line, rather than generating a return-linefeed on the teletype after the statement is executed.

The following statements are useful to an advanced programmer for repetitive operations such as programmer defined functions (standard mathematical and trigonometric functions are convenient timesavers for programmers at any level).

GOSUB-Begins executing the subroutine at specified statement. Used in conjunction with a RETURN statement.

RETURN-Transfers control to statement following its GOSUB.

GOSUB...RETURN statements eliminate the need to repeat frequently used groups of statements in a program. RETURN statements may be used at any desired exit point in a subroutine. When transferring to a subroutine a RETURN statement is essential in order to get back to the main body of the program. GOSUB...RETURN's may be nested similarly to the FOR...NEXT loops.

Example:

10 INPUT A
20 GOSUB 80
  
  80 IF A>0 THEN 100
  90 LET A=-A
100 GOSUB 180
110 RETURN

LET A=A↑2
190 LET B=A↑2
200 RETURN
210 END

Note the order in which the program is executed is: (When A>0), 10,20,80,100,180,190,200,110, and statements after 20; (When A<0), 10,20,80,90,100,180,190,200,110, and statements after 20.

One other statement is usually used in conjunction with GOSUB... RETURN, that is, a STOP statement.

STOP-Terminates the program. The STOP statement may be used anywhere within a program.

Subroutines are usually placed just before the END statement and the STOP statement just before the subroutines so that the program flow will not carry into the subroutines unless instructed to do so.

The final statements that you should be aware of are the REMark, DIMension, and TAB.

REM-Inserts non-executable remarks in a program. REM statements are printed only when the program is listed or punched, they are ignored when the program is in operation.

DIM-Specifies maximum string or matrix size.

Example: 10 DIM A (60)
20 DIM B (10,10)
30 DIM C (30), D (4,8)

Due to configuration and storage capacity there is usually a maximum integral value that an array of elements or matrix may be
dimensioned. The student should consult the respective computer manual to see what the configuration allows. The integers refer to the number of array elements, for one integral value, or the number of rows and columns respectively, if two integral values are given. If not dimensioned, arrays will automatically assume a dimension set by the configuration of the computer (again, consult manual).

Statement 10, on page 106, says array A is a string of 60 elements. Statement 30 says array C is a string of 30 elements and matrix D has 4 rows and 8 columns.

**TAB**—Used only in a PRINT statement. Causes the teletype to move to the space number specified (BASIC prints 72 spaces to a line), then prints what the PRINT statement calls for.

**Special Characters.** In this section we are going to cover, briefly, certain characters found on the teletype and a shorthand notation referred to as E notation. Both of these topics are essential to a programmer.

**ALT-MODE**—Deletes a line being typed.

**ESC**—Accomplishes the same as ALT-MODE.

**BREAK**—Terminates a running program, listing, or punching.

**LINEFEED**—Causes teletype to advance one line.

**RETURN**—Must follow every command or statement. Causes the teletype to return to the first print position.

→—Backspace. Deletes as many preceding characters as →'s are typed in.

**E notation**—means of expressing numbers in the form of a decimal number raised to some power of 10.
Example: \(1.00000E+06 = 1 \times 10^6 = 1,000,000\)
\(1.02000E+04 = 1.02 \times (10 \text{ to the fourth power}) = 10,200\)
\(1.02000E-04 = 1.02 \times 10^{-4} = .000102\)

**Functions.** The general mathematical functions listed below may be used as expressions, or as parts of expressions in statements.

**DEF FN**—Allows the programmer to define functions. FN must be followed by a function label (letter A to Z) and then the argument or function defined.

Example:
20 DEF FNB (X) = A^2 + 2*A
30 DEF FNC (A) = X/6
40 PRINT FNC (A)

The numerical value for A may be contained elsewhere in the program or may be inserted through the use of an INPUT statement.

In the previous example, if A was assigned a value of 4, when FNC (A) is called for in statement 40, the function defined for FNC in statement 30 is used to determine the value printed. The student should be careful to avoid defining functions in a manner that will cause circular definitions within a program. This will cause infinite looping.

**ABS** (X)—Gives the absolute value of the function in place of X.

**EXP** (X)—Gives the constant \(e\) (irrational number with decimal approximation of 2.7182818, carried to seven places) raised to the power of the value (X). If X were 2, EXP (2) would mean \(e^{\uparrow}2\).

**INT** (X)—Gives the largest integer <= the value for (X).

**LOG** (X)—Gives the natural logarithm of (X). (X) must have a positive value.
RND (X)-Generates a random number between 0 and 1 in place of the dummy variable (X).
SQR (X)-Gives the square root of (X). (X) must have a positive value.
SIN (X)-Gives the sine of (X). (X) must be real and in radians.
COS (X)-Gives the cosine of (X). (X) must be real and in radians.
TAN (X)-Gives the tangent of (X). (X) must be real and in radians.
ATN (X)-Gives the arctangent of (X). (X) must be real and in radians.
LEN (X)-Gives the current length of a string, that is, number of characters.
TYP (X)-If argument (X) is negative, gives the type of data in a file as: 1=number; 2=string; 3="end of file"; 4="end of record." If argument (X) is positive, gives the type of data in a file as: 1=number; 2=string; 3="end of file."
For sequential access to files—skips over "end of records."
If argument (X)=0, gives the type of data in a data statement as: 1=number; 2=string; 3="out of data."
SGN (X)-Assumes value of 1 if (X)>0, 0 if (X)=0, and -1 if (X)<0.
Matrices. In this section we will endeavor to explain matrix manipulation. It is the intention of the author to show the matrix capabilities of BASIC and assume that the reader has some knowledge of matrix theory. A matrix may be defined as an array, in BASIC as an ordered collection of numeric data (numbers).
MAT IDN-Establishes an identity matrix (with all ones down the diagonal). A new working size (m,n) may be specified.
Example: 10 MAT C=IDN (4,4)
In the example on page 109 \((m,n)=(4,4)\). The IDN matrix must be two-dimensional and square.

MAT ZER-Sets all elements of the specified matrix equal to 0.
Example: 20 MAT D=ZER (5,4)

A new working size \((m,n)\) may be specified after ZER.

MAT CON-Sets all elements of the specified matrix equal to 1.
Example: 30 MAT E=CON (6,3)

A new working size \((m,n)\) may be specified after CON.

INPUT X \((m,n)\)-Allows input from the teletype of a specified matrix element at location \((m,n)\) of matrix X.
Example: 40 INPUT A (4,3)

This example means the operator may input an element into the 4th row 3rd column of matrix A.

MAT INPUT Y \((m,n)\)-Allows input of a matrix from the teletype. A new working size may be specified by \((m,n)\).

MAT PRINT-Prints the specified matrix at the teletype.
Example: 15 MAT PRINT B
20 MAT PRINT B (4,3)

In this example both matrix B and the element in the 4th row 3rd column of matrix B will be printed.

MAT READ-Reads matrix from DATA statements.
Example: 10 MAT READ C
20 MAT READ B (4,4)
30 READ B (2,3)

In this example matrix C is read, matrix B is read, if it is a 4 by 4 matrix, and the element in the 2nd row of the 3rd column of matrix B is read.
MAT +  -Matrix addition. Both matrices **must** be the same size.

MAT -  -Matrix subtraction. All matrices **must** be the same size.

MAT *  -Matrix multiplication. The number of columns in first matrix of product **must** equal the number of rows in second matrix of product.

MAT =  -Establishes equality of two matrices. Assigns values of one matrix to another.

Example:  10 MAT A=B+C
          20 MAT D=E-F
          30 MAT G=H^J
          40 MAT K=L

        In this example B and C **must** have the same size, as well as D, E and F, and matrix H **must** have the same number of columns as there are rows in matrix J.

        The reader might note that in following the alphabet in the above example, the letter I was skipped. This is not an oversight but done intentionally. It is often helpful to reserve the letter I for the identity matrix and the letter Z for the zero matrix to avoid confusion when setting up a program.

        The last two statements we will concern ourselves with in this section are:

MAT TRN - Transposes an m by n matrix to an n by m matrix.

Example:  40 MAT A=TRN (B)

MAT INV - Inverts a square matrix into a square matrix of the same size. Matrix may be inverted into itself.

Example:  60 MAT A=INV (A)
           70 MAT B=INV (C)
This concludes the discussion on BASIC. There are other aspects of the language not covered here. The student is encouraged to consult a text that deals exclusively with the computer language BASIC in order to obtain a more thorough knowledge of how to use it.
Exercises:

The best way to learn BASIC is by actually working with the computer. Even if you don't have access to a computer terminal that is programmed in BASIC you should, by analysis and the information in this chapter, still be able to answer the following questions.

After each command or printed statement type the return key. This will cause the computer to either act upon the command immediately or accept and store the statement if it has been correctly written, an error message will be printed if the information has been incorrectly typed.

Each exercise will have the word type. This is only an instruction to the student and not information to be put into the computer.

1. Type: SCRATCH

What happened?

2. Type: 10 PRINT 9+3
   20 END
   RUN

What happened?

3. Type: 10 PRINT 9-3
   20 END
   RUN

What happened?

4. Type: 10 PRINT 9/3
   20 END
   RUN

What happened?

5. Type: 10 PRINT 9*3
   20 END
   RUN
What happened?

6. Type: 10 PRINT 9↑2
20 END
RUN

What happened? ↑ is obviously the symbol for what?

7. Type: 10 PRINT 9+3
20 PRINT 9-3
30 PRINT 9/3
40 PRINT 9*3
50 PRINT 9↑2
60 END
RUN

What happened? Is there a significant order?

8. Type: SCR

What happened?

Type: 10 INPUT A,B
20 LET C=A+B*3
30 PRINT C
40 END
RUN

The computer will type "?" and wait. What is it waiting for?

Type: 6,2

Did the computer type 24 or 12? Did the computer add or multiply first?

9. Type: 20 LET C=(A+B)*3
LIST

What happened to line 20 typed in exercise 8?

10. Type: RUN
6,2

Did the computer add or multiply first?

11. Type: 20 LET C=A/B*3
RUN
6,2

Did the computer multiply or divide first?
12. Type: 20 LET C=A/(B*3)  
         RUN  
         6,2  

Did the computer multiply or divide first?

13. Type: SCR  
        10 PRINT "HELLO"  
        20 GO TO 10  
        30 END  
        RUN  

You may stop this program by typing the break key. Why did the program repeat HELLO? Does GO TO 10 refer to 10 PRINT "HELLO"? This is called a program loop. It will continue to print HELLO until you stop it.

14. Type: 10 INPUT A,B  
        20 IF A > B THEN 100  
        30 IF A < B THEN 150  
        40 IF A=B THEN 20  
        100 PRINT A "IS GREATER THAN" B  
        110 GO TO 210  
        150 PRINT B "IS GREATER THAN" A  
        160 GO TO 210  
        200 PRINT A "EQUALS" B  
        210 END  
        RUN  
        10,20  

        RUN  
        75,15  

(Note that the program was not retyped)

What happened?

15. A line such as 30 PRINT C is called a statement. 30 is called a statement number. How many statements could possibly precede statement 30 in a program? What statement or instruction would follow the largest statement number in a program?
16. Type: SCR  
10 INPUT A  
20 LET C=SQR(A)  
30 PRINT C  
40 END  
RUN  
9  
What happened? SQR must be the abbreviation of what mathematical function? If you can't answer this question then follow the next instructions.  
Type: RUN  
25  
RUN  
49  
RUN  
81  
Answer the previous questions.  
17. In a statement containing exponents, addition and subtraction, and multiplication and division, what operation is performed first? Second? Last?  
18. Type: SCR  
10 FOR A=1 TO 10 STEP .5  
20 PRINT A  
30 NEXT A  
40 END  
RUN  
In this program the variable A will take in turn how many different values? Statement 10 defines the first value of A to be what? How much will A be increased each increment? What was the purpose of statement 30? What happens if 1 TO 10 is changed to 5 TO 9? What happens if .5 is changed to .8? What happens if STEP .5 is omitted in statement 10?
19. Type: SCR
   10 LET A=SIN (1)
   20 PRINT A
   30 END
RUN

The sine of 1 should have been printed. Take note that this is obviously not the sine of 1 degree. It is the sine of how many radians?

20. Type: 15 LET B=COS (1)
   25 PRINT B
   LIST

LIST is typed to tell the computer to list the entire program. How did the print out differ from the program in exercise 19? Program statements may be changed by retyping the statement.
Type: RUN
This should define the cosine and sine of 1 radian. Which is typed first?

21. Repeat exercise 20 for arctangent, exponent, natural log, absolute value, and square root. COS (1) would be replaced by:

   ATN (1)  
   EXP (2) where \( e^2 = 2.71282^2 \)  
   LOG (2) which is the natural log of \( X \), \( \log_e X \)  
   ABS (-2.4)  
   SQR (81)

22. SIN, COS, ATN, EXP, LOG, ABS, and SQR are functions.

Try this complex function.

Type: SCR
   10 FOR N=1 TO 10
   20 LET X=RND (A)
   30 PRINT X
   40 NEXT N
   50 END N
RUN

Did BASIC print numbers at random? What does RND mean?
Why would you use this feature? How many random numbers were printed? What step of the program defined the quantity of random numbers? What do you guess the probable range of random numbers to be? How would you change program statement 20 to cause random numbers in the range of 0 to 9 to be printed?

23. How would you utilize the complex function INT (integer) in conjunction with RND to print whole numbers in the range of 0 to 9? (Hint: fill in the blank)

20 LET X=INT( __ *10) 
RUN

24. Type: SCR
10 READ A,B,C
20 PRINT A,B,C, A↑2, B↑2
30 DATA 4,5,6,7,8,9,10,11,12,13,14,15
35 GO TO 10
40 END
RUN

Are READ and DATA dependent on each other for this program? What data in statement 30 is treated as the variable A? As the variable B? As the variable C? How do READ and DATA differ from INPUT?

25. Type: SCR
10 REM THIS IS A PROGRAM TO TYPE HI!
15 PRINT "HI!"
20 END
RUN

LIST

What is the function of statement 10?

26. Type: SCR
10 LET S=INT (6.75334*10↑3)/10↑3
20 PRINT S
30 END
RUN
What was printed by the computer? Why wasn't only the integer 6 printed? Could the $10^3)/10^3$ portion of the statement allow selective rounding? What happens when $10^3)/10^3$ is replaced by $10^2)/10^2$? By $10^4)/10^4$?

27. Write a program that will produce two single digit random numbers. Call one X and the other Y.

28. Expand the capability of your program in exercise 27 to allow a student to input the sum of the two random numbers generated. Call the student's answer by the variable A. Compare $X+Y-A=0$.

If true, create two new values of X and Y. If false, type YOU BLEW IT, TRY AGAIN and reprint the original values of X and Y.
PART IV

APPLICATIONS

Mathematics is the queen of the sciences and number theory the queen of mathematics.

Carl Friedrich Gauss

1777-1855
CHAPTER 8
Number Theory

Introduction. Numbers can fall into many categories depending on their individual properties. We shall define the set of natural numbers, denoted \( N \), as the numbers in the set \( \{1,2,3,4,\ldots \} \). From this basic definition we can define the other categories that numbers fall into according to the following:

(natural numbers) \( \cup \) (zero) = (whole numbers);

(whole numbers) \( \cup \) (opposites or negatives of \( N \)) = (integers);

(integers) \( \cup \) (quotients of integers, \( a/b, b \neq 0 \)) = (rational numbers);

(rational numbers) \( \cup \) (irrational numbers) = (real numbers).

We shall denote the set of whole numbers by \( W \), integers by \( I \), and rational numbers by \( R \). This text will not be dealing with the set of irrational numbers as such.

Divisors and the division algorithm. Let us begin with the following definition:

Definition 8.1. If \( a, b \in I \) and \( b \neq 0 \), we say that \( b \) is a divisor (or factor) of \( a \) if there exists \( c \in I \) such that \( a = c \cdot b \).

In this context, we say that \( b \) divides \( a \) or that \( a \) is divisible by \( b \) or that \( a \) is a multiple of \( b \).

In symbolic notation we shall denote \( b \) divides \( a \) by \( b \mid a \) and \( b \) does not divide \( a \) by \( b \nmid a \). It is to be understood that a divisor is always different from zero, i.e., if \( b \mid a \) then \( b \neq 0 \).

The following are simple facts that are a result of our definition
and will be stated without proof. Verification by the reader should be a simple exercise.

(i) If \( b \mid a \) and \( a \mid c \), then \( b \mid c \).

(ii) \( b \mid a \) if and only if \( b \mid (-a) \).

(iii) \( b \mid a \) if and only if \( (-b) \mid a \).

(iv) \( \pm 1 \mid a \) for every integer \( a \).

(v) If \( b \pm 1 \), then \( b = \pm 1 \).

(vi) \( b \mid 0 \) for every \( b \neq 0 \).

(vii) If \( a \mid b \) and \( b \mid a \), then \( a = \pm b \).

(viii) If \( a \neq 0 \) and \( b \mid a \), then \( |b| \leq |a| \). Moreover, if \( b \mid a \) and \( b \neq \pm a \), then \( |b| < |a| \).

(ix) If \( b \mid a \) and \( b \mid c \), then \( b \mid (ax + cy) \) for arbitrary integers \( x \) and \( y \).

A useful result of (ix) is that if \( b \mid a \) and \( b \mid c \), then \( b \mid (a+c) \) and \( b \mid (a-c) \). Also, if \( a+c = d \) and \( b \) divides any two of the integers \( a, c, \) and \( d \), it also divides the third.

In our discussion of the divisibility properties of integers we shall limit ourselves primarily to positive divisors of positive integers. There is no loss of generality, due to (ii) and (iii).

Division algorithm 8.2. If \( a \) and \( b \) are integers with \( a \geq b > 0 \), there exist unique integers \( q \) and \( r \) such that

\[
a = qb + r, \quad 0 \leq r < b.
\]

Here \( a \) is arbitrary, \( b \) is a positive integer \( \leq a \), \( q \) is called the quotient and \( r \) the remainder in the division of \( a \) by \( b \). Note that \( b \) is a divisor of \( a \) if and only if \( r = 0 \).

Let's see how this useful information can be applied in a computer.
First we need a flowchart to analyze our problem and provide decisions and answers. In later problems we will dispense with the flowchart and go directly into the program. The reader should acknowledge use of the flowchart as a helpful means of setting up a program but not always a necessity.

![Flowchart Diagram]

Figure 1.
Program 1:

5 INPUT A
10 INPUT B
15 LET Q=INT(ABS(A)/ABS(B))
20 LET R=ABS(A)-ABS((Q)*B))
25 IF R=0 THEN 35
30 PRINT"THE REMAINDER IS" R
35 PRINT"THE QUOTIENT IS" SGN(A)*SGN(B)*Q
40 END

Program 1 output:

RUN

?15
?5
THE QUOTIENT IS 3

DONE

RUN

?46
?8
THE REMAINDER IS 6
THE QUOTIENT IS 5

DONE

RUN

?16
?20
THE REMAINDER IS 16
THE QUOTIENT IS 0

DONE

Note the absolute value function was necessary in statements 10 and 20 in order to handle negative integer inputs. The correct sign of the quotient is picked up again in statement 35 with the sign functions.

This program can be refined to avoid incorrect inputs, such as B=0, by the insertion of a PRINT instruction. For example:

4 PRINT "WHAT DIVIDEND DO YOU DESIRE?"
Further refinement might make use of a logical test of the inputs such as:

11 IF B=0 THEN 13
12 GO TO 15
13 PRINT "CHECK ALGORITHM 8.2 (B#0)."
14 GO TO 4

Program 1:

LIST

4 PRINT "WHAT DIVIDEND DO YOU DESIRE?"
5 INPUT A
9 PRINT "WHAT DIVISOR DO YOU DESIRE?"
10 INPUT B
11 IF B=0 THEN 13
12 GO TO 15
13 PRINT "CHECK ALGORITHM 8.2 (B#0)."
14 GO TO 4
15 LET Q=INT(ABS(A)/ABS(B))
20 LET R=ABS(A)-ABS(Q*B)
25 IF R=0 THEN 35
30 PRINT "THE REMAINDER IS"R
35 PRINT "THE QUOTIENT IS"SGN(A)*SGN(B)*Q
40 END

Program 1 output:

RUN

WHAT DIVIDEND DO YOU DESIRE?
?14
WHAT DIVISOR DO YOU DESIRE?
?0
CHECK ALGORITHM 8.2 (B#0).
WHAT DIVIDEND DO YOU DESIRE?
?14
WHAT DIVISOR DO YOU DESIRE?
?6
THE REMAINDER IS 2
THE QUOTIENT IS 2

DONE

Note when the revised program was listed that the space following
the closing quote in statements 30 and 35 of the original input was ignored by the computer. Also it should be noted that in the original program the statements were spaced at intervals of 5. This is a good policy for any programmer so that should refinement be necessary, as in statements 4, 9, 11, 12, 13, and 14, there will be room to insert necessary statements.

Prime numbers. In discussing prime numbers we will restrict ourselves to the set of whole numbers. Moreover, unless stated otherwise, we shall use the word "divisor" to mean "positive divisor." For example, we say that the only divisors of 8 are 1, 2, 4, and 8.

Definition 8.3. A positive integer p≠1 is said to be a prime number if its only divisors are 1 and p. A positive integer, other than 1, is said to be composite if it is not a prime.

In general, a prime number is a whole number which is greater than 1 and which has no whole number factor other than itself and 1. We shall refer to the set of primes as P. It should be observed that, according to the definition, 1 is neither prime nor composite. Also, a positive integer n is composite if and only if n=a·b, 1<a<n and 1<b<n. The first few prime numbers are 2, 3, 5, 7, 11, ...

A whole number which is divisible by the prime number 2 is called an even number. Thus, 6 is an even number because 6=3·2. Two itself is even, since 2=1·2. Zero is an even number, since 0=0·2. The first few even numbers are 0, 2, 4, 6, 8, ... The remaining whole numbers, 1, 3, 5, 7, 9, ..., are called odd numbers. An odd number is a whole
number that is not divisible by 2.

Theorem 8.4. The product of two even integers is even, the product of an even integer and an odd integer is even, and the product of two odd integers is odd.

Proof: (even)·(even)=(even), by 8.1 and (i);
(even)·(odd)=(even), by 8.1 and (i);
(odd)·(odd)=(odd), by 8.1 and dual of (i).

Goldbach's Conjecture. Every even number $E$ that is greater than 4 is the sum of two odd prime numbers. This is one of the famous unproved theories in mathematics. The question was raised in correspondence between Christian Goldbach (1690-1764) and Leonhard Euler (1707-1783), who had proved that every positive integer is the sum of not more than four squares. Many people have since searched for a counter-example to disprove Goldbach's conjecture. The following is a program which will print each even number $E$, where $6 \leq E \leq 100$, and two odd prime numbers whose sum is $E$. Does this program "prove" or "disprove" the conjecture?

Program 2:

```
5 FOR E=6 TO 100 STEP 2
10 FOR P=3 TO E-3
15 FOR F=2 TO P-1
20 IF P/F=INT(P/F) THEN 60
25 NEXT F
30 FOR N=3 TO E-3
35 FOR D=2 TO N-1
40 IF N/D=INT(N/D) THEN 55
45 NEXT D
50 IF P+N=E THEN 75
55 NEXT N
60 NEXT P
65 PRINT "CONJECTURE FALSE FOR"E
70 GO TO 85
75 PRINT E"="P"+"N
```
80 NEXT E
85 END

Program 2 output:

RUN

6   =  3   +  3
8   =  3   +  5
10  =  3   +  7
12  =  5   +  7
14  =  3   + 11
16  =  3   + 13
18  =  5   + 13
20  =  3   + 17
22  =  3   + 19
24  =  5   + 19
26  =  3   + 23
28  =  5   + 23
30  =  7   + 23
32  =  3   + 29
34  =  3   + 31
36  =  5   + 31
38  =  7   + 31
40  =  3   + 37
42  =  5   + 37
44  =  3   + 41
46  =  3   + 43
48  =  5   + 43
50  =  3   + 47
52  =  5   + 47
54  =  7   + 47
56  =  3   + 53
58  =  5   + 53
60  =  7   + 53
62  =  3   + 59
64  =  3   + 61
66  =  5   + 61
68  =  7   + 61
70  =  3   + 67
72  =  5   + 67
74  =  3   + 71
76  =  3   + 73
78  =  5   + 73
80  =  7   + 73
82  =  3   + 79
84  =  5   + 79
86  =  3   + 83
88  =  5   + 83
90  =  7   + 83
92  =  3   + 89
Statements 10 through 25 find a prime P. Statements 30 through 45 find a second prime N. Statement 50 tests conjecture. Statement 55 finds a different second prime N. Statement 60 finds a different first prime P. Statements 75 and 80 prints the even number \( E = P + N \) for the values of \( E, P, \) and \( N \), then continues with the next even number.

The process of finding all the prime factors of a natural number is called factoring over the set of prime numbers. When a number is expressed as the product of its prime factors it is called the prime factorization or a complete factorization of the number.

**Theorem 8.5.** Except for the order of the factors, each composite number greater than 1 has exactly one prime factorization.

This property is called the unique factorization property of natural numbers.

**Proof:** By definition any composite number \( n = a \cdot b \), \( 1 < a < n \) and \( 1 < b < n \).

If \( a \) and/or \( b \) is also composite then another application of this definition will further break these factors of \( n \) into their respective factors. This process is repeated until all factors are listed as a product of prime factors and the product of these factors equals \( n \). By definition a prime number has only two factors, 1 and itself, therefore the factorization is unique.

An example of the prime factorization of a number can be seen with
the number 12. Since 12 is an even number, 12 = 2 \cdot 6. 6 is a composite number equal to 2 \cdot 3, therefore 12 = 2 \cdot 2 \cdot 3. The set of prime factors of the number 12 is \((2, 3)\). Note that in the prime factorization of 12 it is necessary to list the prime factor 2 twice in order to achieve the product 12.

Finding prime factors of numbers greater than 100 can sometimes be complex and even tedious. There is a method that can simplify this process. If a number \(n\) has a prime factor, then the prime \(p|n\). If \(p \not| n\), then \(n\) is a composite number. If \(p|n\), \(p \not| n\), then \(p \leq \frac{1}{3}n\).

Using this information to determine the prime factors of a number \(n\), we only have to check the primes less than or equal to one half of \(n\) to see if they divide \(n\). This brings us to the next theorem.

**Theorem 8.6.** If there exists no prime numbers less than or equal to prime \(p\) that are factors of \(n\) and \(p^2 \geq n\), then \(n\) is a prime number.

The proof is obvious from definition 8.1, the division algorithm and theorem 8.5. If there exists a prime greater than a given prime \(p\) that is a factor of \(n\), then \(p^2 \not| n\). This follows from theorem 8.5 where every composite number has a unique prime factorization. If a number \(n\) has no prime factors less than or equal to a given prime \(p\) and \(p^2 \geq n\) it cannot possibly have any prime factors greater than \(p\).

Let's see how we might set up a computer program to factor composite numbers into a product of primes and identify prime numbers.
Program 3:

1 REM--THIS IS A PRIME FACTORIZATION PROGRAM.
2 REM--IT WILL ACCEPT ANY INTEGRAL VALUES GREATER THAN 1 AT INPUT.
5 PRINT "WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT"
10 PRINT "OF PRIMES?";
15 INPUT N
20 IF N <= 1 THEN 30
25 GO TO 40
30 PRINT "YOUR INPUT IS NOT ACCEPTABLE, CHECK DEF.8.2 AND THM.8.4."
35 GO TO 5
40 PRINT "THE PRIME FACTORIZATION OF"N"IS";
45 IF N=2 THEN 65
50 FOR D=2 TO N-1
55 IF N/D=INT(N/D) THEN 75
60 NEXT D
65 PRINT "NOT POSSIBLE SINCE"N"IS PRIME."
70 GO TO 100
75 LET N=N/D
80 IF N=1 THEN 95
85 PRINT D"*";
90 GO TO 55
95 PRINT D
100 END

Program 3 output:

RUN

WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT
OF PRIMES??28
THE PRIME FACTORIZATION OF 28 IS 2 * 2 * 7
DONE

RUN

WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT
OF PRIMES??258
THE PRIME FACTORIZATION OF 258 IS 2 * 3 * 43
DONE

RUN

WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT
OF PRIMES??61
THE PRIME FACTORIZATION OF 61 IS NOT POSSIBLE SINCE 61 IS PRIME.
DONE
WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT OF PRIMES??

YOUR INPUT IS NOT ACCEPTABLE, CHECK DEF.8.2 AND THM.8.4.

WHAT NUMBER DO YOU WISH TO HAVE FACTORED INTO A PRODUCT OF PRIMES??

THE PRIME FACTORIZATION OF 2 IS NOT POSSIBLE SINCE 2 IS PRIME.

DONE

Greatest common factor. In general, a common factor of two or more whole numbers is a whole number which is a divisor of each of the given numbers.

Definition 8.7. The greatest common factor (GCF) of k positive integers n (k ≥ 2) is the unique positive integer d with the following two properties:

(i) d|n_i (i=1,2,...,k).

(ii) If c|n_i (i=1,2,...,k), then c|d.

If you have the prime factorizations of two numbers, then you can determine their greatest common factor by inspection. For example, we know that 60=2·2·3·5 and 126=2·3·3·7. Also we know that if 60 and 126 have common factors, then these prime factors must be divisors of d by definition. In fact, the greatest common factor turns out to be the product of these common prime factors. In this case the GCF (60, 126) is 2·3=6. Another example:

2310=2·3·5·7·11 and 858=2·3·11·13.

The GCF (2310, 858)=2·3·11=66.

Definition 8.8. If k positive integers n (k ≥ 2) have no common prime factors, i.e., the GCF=1, then we say the numbers are relatively prime.
Over 2000 years ago, the Greek mathematicians discovered a very simple method, called the Euclidean Algorithm, for finding the GCF of two numbers. Using this method, the GCF of very large numbers may be found quite easily. Using other methods may require hours of time consuming arithmetic. The Greek method is best understood by using some geometric reasoning.

Suppose we wish to find the GCF of 259 and 888. We first construct two line segments, both of equal units, one 259-units long and the other 888-units long. We are going to find the largest segment that is contained a whole number of times in both segments.

The 259-unit segment is contained in the 888-unit segment three times, with a 111-unit segment remaining (see Figure 2).

Any segment that is contained a whole number of times in both the 888-unit segment and the 259-unit segment is contained a whole number of times in the 111-unit segment. So we repeat the process for the 259-unit and 111-unit segments, and we keep repeating this process until "it comes out even," i.e., there exists no remaining segment.

1.

2.

3.

4.

Figure 2.
Finally, in Step 4 of Figure 2, we see that 37-units are contained a whole number of times in both the 259-unit and 888-unit segments.

This geometric argument can be reduced to a series of division problems:

A. $888 \div 259 = 2$ remainder 111
B. $259 \div 111 = 2$ remainder 37
C. $111 \div 37 = 3$ remainder 0

Notice how steps A through C compare with Steps 1 through 3 in Figure 2.

The Euclidean Algorithm is the basis for a computer program that will find the GCF of any two positive integers.

Program 4:

```
1 REM--PROGRAM FOR FINDING THE GCF OF ANY TWO POSITIVE INTEGERS.
5 PRINT "TYPE THE TWO NUMBERS YOU WISH TO FIND THE GCF OF."
10 PRINT "SMALLEST NUMBER FIRST. EXAMPLE: 6,9 ."
15 INPUT A,B
20 IF A>B THEN 10
25 PRINT "THE GCF OF"A"AND"B"IS";
30 LET Q=INT(B/A)
35 LET R=B-Q*A
40 IF R=0 THEN 60
45 LET B=R
50 LET A=R
55 GO TO 30
60 PRINT A
65 END
```

Program 4 output:

RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE GCF OF.
SMALLEST NUMBER FIRST. EXAMPLE: 6,9 .
?6,9
THE GCF OF 6 AND 9 IS 3
DONE
RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE GCF OF.
SMALLEST NUMBER FIRST. EXAMPLE: 6,9.
?64,48
SMALLEST NUMBER FIRST. EXAMPLE: 6,9.
?48,64
THE GCF OF 48 AND 64 IS 16
DONE

RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE GCF OF.
SMALLEST NUMBER FIRST. EXAMPLE: 6,9.
?78,164
THE GCF OF 78 AND 164 IS 2
DONE

Least common multiple. In general, a common multiple of two or more
given whole numbers is a whole number which is a multiple of each of
the given numbers. The least common multiple (LCM) of the numbers is
the smallest nonzero common multiple of the given numbers.

Definition 8.9. The least common multiple (LCM) of \( n_k \) positive
integers \( (k \geq 2) \) is the unique positive integer \( m \) with
the following two properties:

(i) \( n_i \mid m \ (i=1,2,\ldots,k) \)

(ii) If \( n_i \mid c \ (i=1,2,\ldots,k) \), then \( m \mid c \).

As in the case of the GCF, if you have the prime factorization of
the given set of numbers, the LCM is easy to obtain. For example, we
know that \( 45=3 \cdot 3 \cdot 5 \) and \( 63=3 \cdot 3 \cdot 7 \). We also know that any common
multiple of these two numbers will contain the same prime factors.
Namely 3, 5, and 7. Since the prime factor 3 occurs twice in each of
the prime factorizations of these two numbers, it would have to occur
exactly twice in the least common multiple of the numbers. Therefore
the LCM (45,63) is $3\cdot3\cdot5\cdot7=210$.

Let's look at another example in order to illustrate this point further.

$$27=3\cdot3\cdot3 \quad \text{and} \quad 36=2\cdot2\cdot3\cdot3$$

The LCM of $(27,36)=2\cdot2\cdot3\cdot3\cdot3=108$. Note that the prime factor 2 must occur twice in the LCM because it occurs twice in the prime factorization of 36. Also note that the prime factor 3 occurs three times in the LCM and in the factorization of 27 but only twice in the factorization of 36. This is because $3\cdot3=9$ is a factor of $3\cdot3\cdot3$, or 27, but the reverse is not true.

This brings us to a computer program for finding the LCM of two numbers.

Program 5:

1 REM-PROGRAM FOR FINDING THE LCM OF ANY TWO POSITIVE INTEGERS.
5 PRINT "TYPE THE TWO NUMBERS YOU WISH TO FIND THE LCM OF--"
10 PRINT "SMALLEST NUMBER FIRST. EXAMPLE: 7,15 ."
15 INPUT A,B
20 IF A>B THEN 10
25 FOR N=1 TO A
30 LET M=N*A
35 IF M/A=INT(M/A) THEN 45
40 NEXT N
45 PRINT "THE LCM OF"A"AND"B"IS"M
50 END

Program 5 output:

RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE LCM OF--
SMALLEST NUMBER FIRST. EXAMPLE: 7,15 .
66,78
THE LCM OF 66 AND 78 IS 858

DONE
RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE LCM OF--
SMALLEST NUMBER FIRST. EXAMPLE: 7,15.
77,15
THE LCM OF 7 AND 15 IS 105

DONE
RUN

TYPE THE TWO NUMBERS YOU WISH TO FIND THE LCM OF--
SMALLEST NUMBER FIRST. EXAMPLE: 7,15.
76,8
THE LCM OF 6 AND 8 IS 24

DONE

The value of number theory. An immediate, valuable consequence of
number theory is in dealing with fractions. When adding or
subtracting fractions it is sometimes necessary to find the least
common denominator. It turns out that the least common denominator
is the least common multiple of the denominators given. Also, when
reducing proper fractions to lowest terms, the greatest common
factor of both the numerator and denominator will reduce the fraction
to lowest terms in one step. If the numerator and denominator of a
proper fraction are relatively prime, then the fraction is in its
lowest terms.

All mathematicians seek ways to simplify their work when dealing
with involved arithmetic, number theory is just one of the many tools
they use, the computer is another.
Exercises:

1. List the categories real numbers fall into.

2. Prove (i) - (ix) on page 122.

3. In program 1 if input A is 13 and input B is 6, what will Q be? What will R be? What will ABS(A) be? What will ABS((Q)*B)) be?

4. In program 2 what statement would you modify in order to check the even numbers from 6 to 500 for Goldbach's conjecture?

5. If you ran program 2 you would find that there would be a timely delay in the printout with the larger even numbers, what causes this delay? What is it called?

6. Is the factorization of any composite number unique? Why?

7. In program 3 if input N is 15, what will D be in statement 55 when it first detects an integer? What will N be in statement 75 after the first integer is detected?

8. In program 3 explain the purpose of statement 80.

9. In program 4 if input A is 14 and input B is 35, what will Q be on the first loop? The second loop?

10. In program 4 if input A is 12 and input B is 32, what will R be on the first loop? The second loop?

11. In program 5 if input A is 4 and input B is 6, what is M when M/A=INT(M/A)?
CHAPTER 9

Game Simulation

Introduction. In the infinite vastness of space lay the mysteries of man's future. The Apollo flights to the Moon, the Mariner and Viking flights to Mars are only the first in an infinite series of explorations conducted by mankind. Unmanned probes into space are the predecessors to man himself. It must be recognized that there are dangers ahead for man in space and these dangers have to be minimized before we can venture forth.

The unmanned space probes as well as the manned missions will be controlled and guided by computers. Not only computers on the probe itself but back on Earth. If man is riding within the spacecraft, he will definitely have a computer on board to assist him in his mission. If man is not aboard then a computer may not be necessary on the craft itself but definitely will be controlling and guiding the craft from Earth. The computer is a great help in minimizing the dangers of space travel. It is a tool in which we must rely very heavily upon if we are to unravel the mysteries of space.

The computer as a tool is only as reliable as the people who designed, built, and programmed it, not to mention the individuals using it. We control the computers, they don't control us. Though the computer is able to make decisions based on collected and evaluated data, it was a human being that decided what data would be collected, what evaluations would be done, and what decisions would be made.
These bits of information were programmed into the computer by a human mind. If an error exists, it was directly or indirectly placed there by a person. I distinguish between error and malfunction in that by error I mean within the computer itself. A malfunction would be a breakdown in the computer support systems (such as power).

A space traveler must sleep, breath, and eat in order to survive. The computer eliminates the first two necessities but does consume power. The computer is the untiring, uncomplaining, ever working co-pilot of man on every space mission he takes.

The following game simulation illustrates the relationship between a person (the operator) and this fascinating machine that was designed for the purpose of relieving him of tedious arithmetic operations and the gathering of various types of data.

Before we list the program I feel a little background information is in the best interest of the reader.

Background. Simulation is a major part of every space mission. Before the actual flight takes place it has been simulated 100, maybe even 1000 times in the laboratory. In simulation all factors are brought into play and any anticipated or unanticipated dangers are simulated in order to see their affects and make any corrections or adjustments necessary.

Two Viking landers are scheduled to reach Mars in 1976. Launched from an orbiting mother spacecraft, they will depend on parachutes and retro rockets to bring their fragile cargo of instruments to a safe and gentle landing. Their planned mission will be to scoop up
Martian soil with ten-foot retractable arms. Inside the spacecraft, miniature laboratories will analyze the samples. In just one cubic foot of the spacecraft have been compressed three automated chemical laboratories complete with a computer, tiny ovens, counters for radioactive tracers, filters, a lamp to simulate Martian sunlight, and a gas chromatograph to identify chemical substances. Other parts of Viking will analyze the atmosphere, air pressure, temperature, and wind velocity, identify minerals in the Martian soil, and detect volcanic and seismic activity. A camera system will take stereo pictures in color and infrared as well as in black and white.

All pictures and data will be transmitted back to Earth by telemetry in much the same way as Mariner transmitted its information.

All this will be accomplished with 50 watts of power from radioisotope thermoelectric generators—hardly more than the amount of electricity used by the light bulb in your refrigerator!

Simulations of these systems are presently going on in Colorado gravel pits as well as the laboratories of TRW Inc., in Redondo Beach, California.

Let's see what the program for our simulation looks like.

Viking.

Program 6:

5 PRINT "EARTH CONTROL CALLING VIKING MODULE..."
10 PRINT "YOU ARE NOW ON MANUAL CONTROL. YOU MAY USE ANY BURN"
15 PRINT "RATE OF 0 THROUGH 200 LBS./SEC ON YOUR RETRO ROCKET."
20 PRINT "YOUR TASK IS TO LAND GENTLY ON THE PLANET MARS."
25 PRINT "GOOD LUCK!"
30 PRINT
35 PRINT
40 PRINT "SEC","MI+FT","MPH","LBS FUEL","BURN RATE";
45 PRINT
50 LET L=0
55 LET A=120
60 LET V=1
65 LET M=3.3500E+04
70 LET N=1.8000E+04
75 LET G=2.3000E-03
80 LET Z=2.8
85 PRINT L,INT(A);INT(5280*(A-INT(A))),3600*V,M-N,
90 INPUT K
95 LET T=10
100 IF K>=0 THEN 115
105 PRINT "BURN RATE >=0 ONLY"
110 GO TO 90
115 IF K<200 THEN 130
120 PRINT "BURN RATE < 200 ONLY"
125 GO TO 90
130 IF M-N<1.00000E-03 THEN 185
135 IF T<1.00000E-03 THEN 85
140 LET S=T
145 IF M>=N+S*K THEN 155
150 LET S=(M-N)/K
155 GOSUB 380
160 IF I<0 THEN 310
165 IF V<0 THEN 175
170 IF J<0 THEN 340
175 GOSUB 280
180 GO TO 130
185 PRINT "FUEL OUT AT"L" SEC"
190 LET S=(-V+SQR(V*V+2*A*G))/G
195 LET V=V+G*S
200 LET L=L+S
205 LET W=3600*V
210 PRINT "ON MARS AT"L" SEC--IMPACT VELOCITY"W" MPH"
215 IF W>10 THEN 250
220 PRINT "GOOD LANDING...TIME 13:45 ON A 24HR.39MIN.35SEC. MARTIAN"
225 PRINT "DAY...LOCATION--LATITUDE 0--LONGITUDE 120...HIGHEST POINT"
230 PRINT "ON MARTIAN SURFACE--NIX OLYMPICA--BEARING 30 DEGREES N-NW"
235 PRINT "AT A DISTANCE OF 820.6 STATUTE MILES..OUTSIDE TEMPERATURE"
240 PRINT "A BALMY 30 DEGREES FAHRENHEIT ON MARTIAN SURFACE."
245 STOP
250 IF W>60 THEN 270
255 PRINT "BAD JOLT!!!--INSTRUMENTATION DAMAGED--BACK-UP SYSTEMS"
260 PRINT "MAL-FUNCTIONING--MISSION IN DANGER!!!--"
265 STOP
270 PRINT "NO SURVIVORS--BLASTED NEW MARTIAN CRATER"W*.27 FT DEEP."
275 STOP
280 LET L=L+S
285 LET T=T-S
290 LET M=M-S*K
295 LET A=I
300 LET V=J
305 RETURN
310 IF S<5.00000E-03 THEN 205
315 LET D=V+SQR(V*V+2*A*(G-Z*K/M))
320 LET S=2*A/D
325 GOSUB 380
330 GOSUB 280
335 GOTO 310
340 LET W=(1-M*G/(Z*K))/2
345 LET S=M*V/(Z*K*(W+SQR(W*W+V/Z)))+5.00000E-02
350 GOSUB 380
355 IF I<0 THEN 310
360 GOSUB 280
365 IF J>0 THEN 130
370 IF V>0 THEN 340
375 GOTO 130
380 LET Q=S*K/M
385 LET J=V+Q*S+Z*(-Q*Q/2-Q*3/3-Q*4/4)
390 LET I=A-G*S*S/2-V*S+Z*S*(Q/2+Q*2/6+Q*3/12+Q*4/20)
395 RETURN
400 END

In order to get a quick output completed on this program we are going to allow the module to free fall towards the planet. This will give us some useful information about the program and also the gravitational pull of the planet Mars.

Program 6 output:

RUN

EARTH CONTROL CALLING VIKING MODULE...
YOU ARE NOW ON MANUAL CONTROL. YOU MAY USE ANY BURN RATE OF 0 THROUGH 200 LBS./SEC ON YOUR RETRO ROCKET.
YOUR TASK IS TO LAND GENTLY ON THE PLANET MARS.
GOOD LUCK!

<table>
<thead>
<tr>
<th>SEC</th>
<th>MI+FT</th>
<th>MPH</th>
<th>LB FUEL</th>
<th>BURN RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>120</td>
<td>3600</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>10</td>
<td>109</td>
<td>3682.8</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>20</td>
<td>99</td>
<td>3765.6</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>30</td>
<td>88</td>
<td>3848.4</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>40</td>
<td>78</td>
<td>3931.2</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>50</td>
<td>67</td>
<td>4014</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>60</td>
<td>55</td>
<td>4096.8</td>
<td>15500</td>
<td>?0</td>
</tr>
<tr>
<td>Statement</td>
<td>44</td>
<td>1927</td>
<td>4179.6</td>
<td>15500</td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>80</td>
<td>32</td>
<td>3379</td>
<td>4262.4</td>
<td>15500</td>
</tr>
<tr>
<td>90</td>
<td>20</td>
<td>3616</td>
<td>4345.2</td>
<td>15500</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>2639</td>
<td>4428</td>
<td>15500</td>
</tr>
</tbody>
</table>

ON MARS AT 106.866487 SEC--IMPACT VELOCITY 4484.85451 MPH
NO SURVIVORS--BLASTED NEW MARTIAN CRATER 1210.91071 FT. DEEP.
STOP

Let's follow the computer flow through the program in order to see how the parameters printed in the output are obtained. We will pick up at statement 90 which is where the operator inputs a burn rate (0 in this case). The computer flow is: 90--95--100--115--130--135--140--145--155--380--385--390--395--160--165--170--175--280--285--290--295--300--305--180--130--135--85--then back through the same series repeatedly until after the completion of the 100 sec. printout on the output we have shown. At this point the flow (following a 0 operator input) is: 95--100--115--130--135--140--145--155--380--385--390--395--160--310--315--320--325--380--385--390--395--330--280--285--290--295--300--305--310--315--320--325--380--385--390--395--330--280--285--290--295--300--305--310--205--210--215--270--275.

If you study this flow on program 6 you will better understand the program and also get a better feel for how BASIC works. For example: You will notice that the return statements in the subroutines (280-305 and 380-395) always cause the computer to loop back to the statement following the GOSUB instruction (155-subroutine-160 and 175-subroutine-180). The two types of flow illustrated here are the same up to statement 160 then the second flow goes through a continuous looping of statements 310-335 with two loops through each
of the subroutines until statement 310 recognizes $S < 0.005$ then the final printout is obtained. Of course a careful study of the program will show that $S$ is a variable of time, $A$ and $I$ are variables of distance, and $V$ and $J$ are variables of speed. Subroutine 380 determines values for $I$ and $J$ and subroutine 280 assigns those values, and others, to variables used in print statement 85.

If a student of programming wishes to set up a statement number flowchart similar to the two I have illustrated, all he has to do is place print statements between each statement of any program. In program 6, for example:

4 PRINT "5"
9 PRINT "10"
14 PRINT "15"

Also, each statement number following GO TO, IF...THEN, and GOSUB will have to be decreased by 1. The reason should be obvious.

Another interesting aspect is in our program 6 output. Since we have allowed the module to free fall into the planet (no retro rocket burn) we can compute the constant acceleration of the module due to the gravitational attraction of the planet Mars. We only have to use the formula $v = v_0 + a \cdot t$ where the acceleration "a" is the constant rate of change of velocity, or the change per unit time. The term "$a \cdot t$" is the product of change in velocity per unit time, $a$, and the duration of the time interval, $t$. Therefore it equals the total change in velocity. The velocity $v$ at the time $t$ then equals the velocity $v_0$ at the time $t=0$ (initial velocity), plus the change in velocity $a \cdot t$. 

Knowing the initial velocity, final velocity and the time from program 6 output, we can compute "a." I leave it to you to compute this value in feet per second$^2$ and compare it with textbook estimates of Mars gravitational attraction.

Finding a possible solution for the burn rate for Viking is not as difficult as one might think. First of all, avoid the trial and error approach that takes up much computer time and may not produce a solution anyway and, also, produces little in educational benefits. Let's utilize the computer and write a program that will narrow our choices for a constant burn rate.

By analyzing our program 6 output we see that free fall takes over 105 seconds. This tells us that a safe landing will probably take more than 110 seconds from the initial burn rate. A maximum burn rate of 200 LBS./SEC on each burn will last only 77.5 seconds and 155 LBS./SEC will last only 100 seconds. This probably indicates a burn rate much less than 150 LBS./SEC.

The objective is to bring the distance and velocity to simultaneous values of 0 MI 0 FT for distance and approximately 1 MPH for velocity. In our previous discussion of the computer flow we observed that the values for distance and speed were generated in subroutine 380, utilizing I for distance and J for velocity. Q is a variable that is only used within the subroutine but the values of the other variables are predetermined, initially in statements 55 through 80 then subroutine 280.

If we could write a program using subroutine 380 and the initial
values of the variables concerned, then generate various values for
distance and velocity dependent on time (S) and burn rate (K), we
could get a rough idea of where the range of a constant burn rate
might be.

Viking solution.

Program 7:

5 LET A=120
10 LET V=1
15 LET M=3.35000E+04
20 LET G=2.30000E-03
25 LET Z=2.8
30 PRINT"SEC","MI+FT","MPH","BURN RATE"
35 FOR S=125 TO 250 STEP 25
40 FOR K=10 TO 150 STEP 10
45 LET Q=S*K/M
50 LET J=V+G*S+Z*(Q-Q*Q/2-Q*Q*Q/3-Q*Q*Q*Q/4)
55 LET I=A-G*S*S/2-V*S+Z*S*(Q/2+Q*Q/2+Q*Q*Q/3+Q*Q*Q*Q/4)
60 PRINT S,INT(I);INT(5280*(I-INT(I)));3600*J,K
65 NEXT K
70 NEXT S
75 END

Statements 5 through 25 are variable values that are extracted
from program 6 statements 55-80 with the exception of statement 70,
which isn't needed. Statement 30 is only needed to identify the
parameters. Statements 35 and 40 assign values for time and burn rate
respectfully. Statements 45-55 are extractions from subroutine 380 in
program 6 and statement 60 prints the parameters for each loop.

Program 7 output:

RUN

<table>
<thead>
<tr>
<th>SEC</th>
<th>MI+FT</th>
<th>MPH</th>
<th>BURN RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>-17</td>
<td>3399</td>
<td>4251.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>-44</td>
<td>4198</td>
<td>1350.23</td>
</tr>
</tbody>
</table>
The negative velocity rates are due to an excess burn rate causing the module to accelerate away from the planet. The negative distances are a result of passing the planet surface level (0 MI+FT). I have not included the entire printout but only the pertinent parameters that came closest to 0 MI+FT and 1 MPH simultaneously. This turned out to be approximately 225 seconds and a burn rate of approximately 60 lbs/sec. Careful analysis tells us that time is probably between 210 and 220 seconds and the burn rate is probably between 62 and 68 lbs/sec. This calls for a simple modification of program 7 in statements 35 and 40.

Program 7 (modification 1):

35 FOR S=210 to 220 STEP .5  
40 FOR K=62 TO 68 STEP .5

You should be aware that modifications to the original program 7 only require typing in the modifications over the original program and not re-typing the entire program.

Program 7 (modification 1) output:

<table>
<thead>
<tr>
<th>SEC</th>
<th>MI+FT</th>
<th>MPH</th>
<th>BURN RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>210</td>
<td>-9</td>
<td>4756</td>
<td>405.068</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DONE
Another careful examination of the output will show that the closest values to 0 MI+FT and 1 MPH are at 212 seconds and 65.5 lbs/sec burn rate. A study of the parameters will show that the time is probably between 212 and 213 seconds and the burn rate is probably between 65 and 65.5 lbs/sec.

At this point the student must be careful because, although further refinement of the nested loops (35-(40-65)-70) in program 7 will narrow down the possibilities of a burn rate, the original Viking program involves a few more statements for a safe landing than we have included in program 7. For instance statements 160--165--170 in program 6 are a logical test of parameters I, V, and J respectively. Also, statement 160 (test of I) causes a branching to statement 310 (see page 144) prior to the final printout of the program. If a student is observant of output parameters (the object of program outputs in this chapter), he might note that in program 7 output (225 sec and 60lbs/sec), as well as program 7 (modification 1) output (212 sec and 65.5lbs/sec), that our best readout came about when there was a simultaneous switch from - to + of parameter I (MI+FT) and + to - of parameter J (MPH). This indicates a very important function in statements 160--165--170. We will need a different program to find our burn

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>212</td>
<td>0</td>
<td>3895</td>
<td>-2.51913</td>
<td>65.5</td>
</tr>
<tr>
<td>212</td>
<td>2</td>
<td>241</td>
<td>-55.505</td>
<td>66</td>
</tr>
<tr>
<td>220</td>
<td>8</td>
<td>808</td>
<td>-484.331</td>
<td>68</td>
</tr>
</tbody>
</table>

DONE
rate.

Utilizing the information we already have and further information from page 144 (final flow) we see that statements 310 through 335 of program 6 need to be used in our program. Also, we should note that when \( J < 0 \) (statement 170) the program branches to 340 through 375, a necessary consideration. Both of these branches utilize the perennial subroutines 380 and 280. We can dispense with some of the statements as unnecessary in our search, \( V = J \) and \( T \) is only a countdown. But we mustn't overlook the necessary value of \( V \), what is meant is that we needn't test \( V \). The following program is assembled from the information so far discussed.

Program 8:

```
5 READ L,A,V,M,G,Z
10 RESTORE
15 PRINT "SEC","MPH","BURN RATE"
20 FOR 65 TO 65.5 STEP .01
25 LET S=10
30 GOSUB 145
35 IF J<0 THEN 80
40 IF J<0 THEN 110
45 GOSUB 165
50 IF L<213 THEN 25
55 PRINT L,3600*V,K
60 READ L,A,V,M,G,Z
65 RESTORE
70 NEXT K
75 STOP
80 IF S<.005 THEN 55
85 LET D=V+SQR(V^2+2*A*(G-Z*K/M)
90 LET S=2*A/D
95 GOSUB 145
100 GOSUB 165
105 GO TO 80
110 LET W=(1-M*G/(Z*K))/2
115 LET S=M*V/(Z*K*(W+SQR(W*W+V/Z)))+0.05
120 GOSUB 145
125 IF I<0 THEN 80
130 GOSUB 165
```
135 IF J > 0 THEN 25
140 IF (J < 0) AND (I > 0) THEN 195
145 LET Q=S*K/F
150 LET J=V+G*S+Z*(-Q-Q*Q/2-Q*3/3-Q*4/4)
155 LET I=A-G*S*S/2-V*S+Z*S*(Q/2+Q*2/6+Q*3/12+Q*4/20)
160 RETURN
165 LET L=L+S
170 LET M=M-S*K
175 LET A=I
180 LET V=J
185 RETURN
190 DATA 0,120,1,3.35000E+04,2.30000E-03,2.8
195 END

Notice the test on J and I in statement 140 to see the approximate point when, simultaneously, J is - and I is +. Also, notice that we don't concern ourselves with a distance printout as this value is approaching zero anyway. The read statement is just a simplification by eliminating the need for a few more statements. Let's see what this program produces in the way of further information.

Program 8 output:

RUN

<table>
<thead>
<tr>
<th>SEC</th>
<th>MPH</th>
<th>BURN RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>204.283</td>
<td>203.141</td>
<td>65</td>
</tr>
<tr>
<td>204.723</td>
<td>191.487</td>
<td>65.01</td>
</tr>
<tr>
<td>205.194</td>
<td>179.074</td>
<td>65.02</td>
</tr>
<tr>
<td>205.703</td>
<td>165.687</td>
<td>65.03</td>
</tr>
<tr>
<td>206.26</td>
<td>151.118</td>
<td>65.04</td>
</tr>
<tr>
<td>206.884</td>
<td>134.845</td>
<td>65.05</td>
</tr>
<tr>
<td>207.602</td>
<td>116.225</td>
<td>65.06</td>
</tr>
<tr>
<td>208.458</td>
<td>94.156</td>
<td>65.07</td>
</tr>
<tr>
<td>209.612</td>
<td>64.6115</td>
<td>65.08</td>
</tr>
</tbody>
</table>

DONE

Now we are getting close. The final output at a burn rate of 65.08 indicates that the negative-positive flip for J and I took place between a burn rate of 65.08 and 65.09. The only other thing that could terminate the program is when K reaches 65.5 on the for-next
loop. We need only to modify our program.

Program 8 (modification 1):

20 FOR K=65.08 TO 65.09 STEP .001

Program 8 (modification 1) output:

RUN

<table>
<thead>
<tr>
<th>SEC</th>
<th>MPH</th>
<th>BURN RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>209.619</td>
<td>64.4209</td>
<td>65.08</td>
</tr>
<tr>
<td>209.761</td>
<td>60.7938</td>
<td>65.081</td>
</tr>
<tr>
<td>209.924</td>
<td>56.6466</td>
<td>65.082</td>
</tr>
<tr>
<td>210.1</td>
<td>52.1829</td>
<td>65.083</td>
</tr>
<tr>
<td>210.285</td>
<td>47.4865</td>
<td>65.084</td>
</tr>
<tr>
<td>210.498</td>
<td>42.0771</td>
<td>65.085</td>
</tr>
<tr>
<td>210.728</td>
<td>36.2437</td>
<td>65.086</td>
</tr>
<tr>
<td>211.019</td>
<td>28.8889</td>
<td>65.087</td>
</tr>
<tr>
<td>211.414</td>
<td>18.9484</td>
<td>65.088</td>
</tr>
</tbody>
</table>

DONE

I offer the next modification to the reader to try. If correctly done, a solution to the constant burn rate should appear that will place the velocity within the tolerances of statement 215 of program 6.

A student shouldn't stop once he finds a satisfactory solution to program 6. That wouldn't be in the best interests of the spirit of inquiry. The inquiring student might be interested in applying this program to another planet or moon in our solar system. He would, of course, have to change gravity (G), amount of fuel necessary (M-N), thrust of the retro-rocket (Z), possibly the approach velocity (3600*V), and, depending on needs, the mass of the spacecraft (M). Some of the print statements would also have to be changed. Real imagination would come up with different locations on landing, depending on landing velocity. A consideration to ponder is the affect atmospheric
density has in relation to approach velocity in slowing the spacecraft down, and heating it up. We don't want to fry our astronauts.

The bounds of the uses of a computer, as in the bounds of the applications of program 6, are established by the limits of our imaginations. The computer is simply an asset in furthering our creative abilities.

Q.E.D.
Bibliography

Books


Bibliography


Bulletins and Pamphlets


Index

<table>
<thead>
<tr>
<th>A</th>
<th>Binary numeration system</th>
<th>16, 34-42, 44-46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abaci, abacus, 21</td>
<td>Bit, binary number, 36, 72, 73, 75, 76, 79, 80, 84, 85, 140, 101</td>
<td></td>
</tr>
<tr>
<td>Abacus, 20-22</td>
<td>Boole, mathematician, 16, 49-50</td>
<td></td>
</tr>
<tr>
<td>Access time, 74</td>
<td>Boolean algebra, 49-64</td>
<td></td>
</tr>
<tr>
<td>Address selection, 80</td>
<td>axioms, 50-51</td>
<td></td>
</tr>
<tr>
<td>Al-Biruni, mathematician, 5</td>
<td>functions, 51</td>
<td></td>
</tr>
<tr>
<td>Algorithm, 8, 121-122, 133</td>
<td>monomials, 51</td>
<td></td>
</tr>
<tr>
<td>Al-Khowarizmi, mathematician, 7-8, 22, 24</td>
<td>polynomials, 51</td>
<td></td>
</tr>
<tr>
<td>Alphanumeric, codes, 45-47, 71</td>
<td>constant, 51</td>
<td></td>
</tr>
<tr>
<td>AND gate, 60-63, 71, 80</td>
<td>variable, 51</td>
<td></td>
</tr>
<tr>
<td>Arithmetic unit, 81-87</td>
<td>principle of duality, 52</td>
<td></td>
</tr>
<tr>
<td>Aryabhata, mathematician, 5</td>
<td>theorem, 52-54</td>
<td></td>
</tr>
<tr>
<td>ASCC, calculator, 30</td>
<td>Brahmi, Indian alphabetic character, 5-6</td>
<td></td>
</tr>
<tr>
<td>Axioms, 50-51</td>
<td>Branching, 93-95</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Briggs, mathematician, 26-27</td>
<td></td>
</tr>
<tr>
<td>Babbage, mathematician, 29, 30, 31</td>
<td>Burroughs, mathematician, 30</td>
<td></td>
</tr>
<tr>
<td>Baldwin, mathematician, 28-29</td>
<td>Bush, mathematician, 30</td>
<td></td>
</tr>
<tr>
<td>Barrow, mathematician, 90</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Basic, 97-112</td>
<td>Chalix, &quot;stone or pebble,&quot; 19</td>
<td></td>
</tr>
<tr>
<td>commands, 97-98</td>
<td>Colmar, mathematician, 28</td>
<td></td>
</tr>
<tr>
<td>operators, 99-100</td>
<td>Commands, 97-98</td>
<td></td>
</tr>
<tr>
<td>statements, 101-107</td>
<td>Computer, 66-87, 139, 140, 153</td>
<td></td>
</tr>
<tr>
<td>loop, 101, 102, 103, 144</td>
<td>analog, 66-68</td>
<td></td>
</tr>
<tr>
<td>nested loop, 103</td>
<td>digital, 66-87</td>
<td></td>
</tr>
<tr>
<td>special characters, 107-108</td>
<td>program, 72, 97, 99, 101, 102, 103, 104, 124, 125, 127-128, 131, 134, 136, 141-143, 147, 150-151</td>
<td></td>
</tr>
<tr>
<td>functions, 108-109</td>
<td>instructions, 72</td>
<td></td>
</tr>
<tr>
<td>matricis, 109-111</td>
<td>repetition rate, 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>word, 73-77, 78, 79, 98</td>
<td></td>
</tr>
<tr>
<td>Computer (continued)</td>
<td>Greatest common factor, 132-135</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>access time, 74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>control unit, 77-81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>address selection, 80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arithmetic unit, 81-87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant, 51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control unit, 77-81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cybernetics, text, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Descartes, mathematician, 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digitus, digit, &quot;finger,&quot; 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Divisors, 121-122, 126</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENIAC, computer, 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Euclidean Algorithm, 133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exclusive OR gate, 60-61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fibonacci, mathematician, 22, 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flowchart, 91-95, 123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>branching, 93-95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions, 51, 108-109</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ganita-Sara-Sangraha, Indian text, 9-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Galileo, mathematician, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gauss, mathematician, 120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goldbach's Conjecture, 127-129</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greatest common factor, 132-135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greek numeration, 8-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gunter, mathematician, 27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Herodotus, historian, 19, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hindu-Arabic numeration system, 3, 22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hollerith, mathematician, 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In artem, text, 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions, 72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVERT gate, 61-63, 71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kepler, mathematician, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>La Disme, text, 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Least common multiple, 135-137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leibniz, mathematician, 16, 24, 28, 33, 49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop, 101, 102, 103, 144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magic square, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mahavira, mathematician, 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mars, planet, 139, 140-141, 145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrices, 109-111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monomials, 51</td>
<td></td>
</tr>
<tr>
<td>Morgenstern, mathematician, 31</td>
<td></td>
</tr>
<tr>
<td>Morland, mathematician, 28</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Napier, mathematician, 26, 27</td>
<td></td>
</tr>
<tr>
<td>Nested loop, 103, 149</td>
<td></td>
</tr>
<tr>
<td>Newton, mathematician, 33</td>
<td></td>
</tr>
<tr>
<td>Number theory, 121-137</td>
<td></td>
</tr>
<tr>
<td>divisors, 121-122, 126</td>
<td></td>
</tr>
<tr>
<td>divisor program, 124-125</td>
<td></td>
</tr>
<tr>
<td>prime numbers, 126-127</td>
<td></td>
</tr>
<tr>
<td>Goldbach's Conjecture, 127-129</td>
<td></td>
</tr>
<tr>
<td>prime factorization, 129-132</td>
<td></td>
</tr>
<tr>
<td>greatest common factor, 132-135</td>
<td></td>
</tr>
<tr>
<td>Euclidean Algorithm, 133</td>
<td></td>
</tr>
<tr>
<td>Least common multiple, 135-137</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Octal numeration system, 42-46</td>
<td></td>
</tr>
<tr>
<td>Operators, 99-100</td>
<td></td>
</tr>
<tr>
<td>OR gate, 60-63, 71, 84</td>
<td></td>
</tr>
<tr>
<td>Oughtred, mathematician, 24, 28</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Pacioli, mathematician, 23</td>
<td></td>
</tr>
<tr>
<td>Pascal, mathematician, 28</td>
<td></td>
</tr>
<tr>
<td>Polynomial, 51</td>
<td></td>
</tr>
<tr>
<td>Prime factorization, 129-132</td>
<td></td>
</tr>
<tr>
<td>Prime numbers, 126-127</td>
<td></td>
</tr>
<tr>
<td>Principal of duality, 52</td>
<td></td>
</tr>
<tr>
<td>Program, 72, 97, 98, 101, 102, 103, 104, 124, 125, 127-128, 131, 134, 136, 141-143, 147, 150-151</td>
<td></td>
</tr>
<tr>
<td>Pythagoras, mathematician, 5</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Rahn, mathematician, 25</td>
<td></td>
</tr>
<tr>
<td>Repetition rate, 72</td>
<td></td>
</tr>
<tr>
<td>Riese, mathematician, 24</td>
<td></td>
</tr>
<tr>
<td>Russell, mathematician, 16, 49</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Sanskrit, literature, 5</td>
<td></td>
</tr>
<tr>
<td>Sexagesimal, Babylonian system, 3, 20</td>
<td></td>
</tr>
<tr>
<td>Siddhântas, Indian texts, 5</td>
<td></td>
</tr>
<tr>
<td>Sifr, &quot;vacant,&quot; 10</td>
<td></td>
</tr>
<tr>
<td>Sindhind, Arab text, 7</td>
<td></td>
</tr>
<tr>
<td>Soroban, abacus; 21</td>
<td></td>
</tr>
<tr>
<td>Special characters, Basic, 107-108</td>
<td></td>
</tr>
<tr>
<td>Statements, 101-107</td>
<td></td>
</tr>
<tr>
<td>Stevin, mathematician, 19, 24</td>
<td></td>
</tr>
<tr>
<td>Suan phan, abacus, 21</td>
<td></td>
</tr>
<tr>
<td>Sūma, text, 23</td>
<td></td>
</tr>
<tr>
<td>Sunya, &quot;void,&quot; 10</td>
<td></td>
</tr>
<tr>
<td>Switching circuits, 59-63, 70</td>
<td></td>
</tr>
<tr>
<td>AND gate, 60-63, 71, 80</td>
<td></td>
</tr>
<tr>
<td>OR gate, 60-63, 71, 84</td>
<td></td>
</tr>
</tbody>
</table>
Index

Switching circuits (continued)
  exclusive OR gate, 60-61
  INVERT gate, 61-63, 71

T
  Theorem, 52-54, 127, 129
  Triparty, text, 22
  Trinary numeration system, 42
  Truth table, 57-59
  Tschotu, abacus, 21

V
  Vander Hoecke, mathematician, 24
  Varahmihira, mathematician, 5
  Variable, 51
  Venn diagrams, 54-57
  Viète, mathematician, 25-26
  Viking, 139, 140-141
    Viking program, 141-147
    Viking solution, 147, 150-151
  Von Neumann, mathematician, 31

W
  Whitehead, mathematician, 16, 49
  Widman, mathematician, 24
  Wiener, mathematician, 31

Word, 73-77, 78, 79, 98

Z
  Zephirum, zephyrum, "cipher," 10