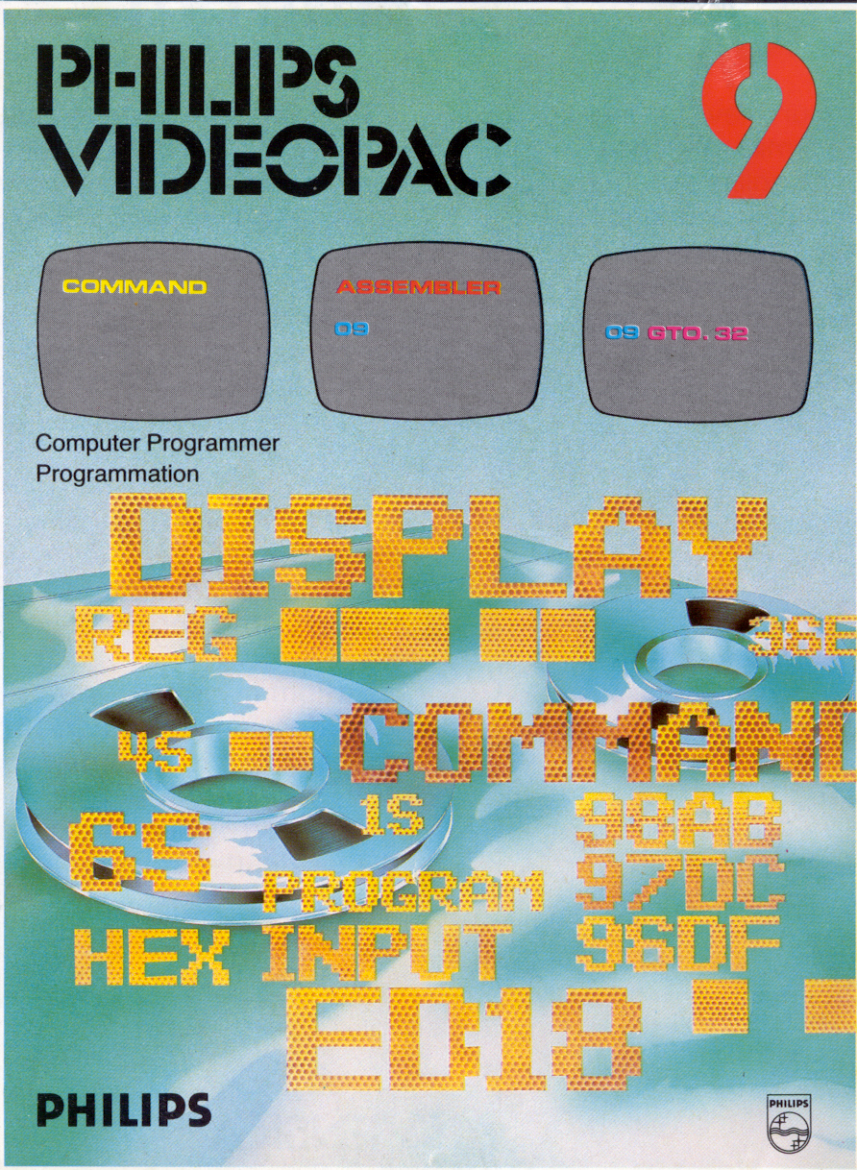
PHILIPS VIDEOPAC



CONTENTS

Each time a game is finished, press RESET (Δ). COMMAND is shown on the screen. Now:

(a) Select another game.

Or (b) Select another Videopac. Remove existing Videopac by placing one hand next to it, and pulling handle upwards. Replace Videopac in its box. Refer now to Instructions for Use of the next Videopac.

Or (c) Plug aerial back into TV, and unplug the Videopac Computer from the mains.

Check procedure

If you suspect a fault in the equipment follow this procedure (with a Videopac installed). Press RESET (Δ). The TV will emit a short sound, and COMMAND should appear on your TV screen. If not, ensure that the equipment is set up properly as detailed in the Instructions for Use (both of the equipment, and of the Videopac used). If the fault remains, take both the equipment and Videopac to your dealer.

History

Since the dawn of history, man has been developing methods of counting, calculating, and storing information. Primitive man used his fingers, bones, pebbles, and shells to count and record his possessions, the phases of the moon, the passing of time, and the changing of the season. As his needs grew and his abilities evolved, he developed more sophisticated calculating methods and storage devices. Written languages were developed, number systems were created, and more convenient and mobile storage systems evolved - from drawing on the walls of caves, to hieroglyphics on tablets of clay and stone, and finally, to the stage of complex modern languages on the printing press.

The abacus is perhaps the oldest known computing device, first used in China in the Sixth Century B.C. In later years, it also was used by the Latins and Greeks, and even today, the abacus is used in restaurants, homes, and schools.

By the Seventeenth Century, the use of numbers had increased in importance due to great advances in astronomy, navigation, and the other sciences. It was necessary that calculations be more accurate and that past information be stored and later re-evaluated in light of new data. In the 1830's, Charles Babbage designed what today we would call a digital computer. This analytical engine would have been able to perform any arithmetic or logical operation. It was designed to be programmable, a computer rather than a mere calculator. Babbage's idea was to use punched cards for entering data and instructing the machine with mathematical commands! Though Babbage died before the machine could be built, his elaborate drawings represent the foresight he had into programmable machines.

Though numerous business which could calculate and print were designed and built in the Nineteenth Century, it was not until the 1940's that electronic components were first used to build computers. The first generation of electronic computers was built with vacuum tubes. Magnetic drums were used for main storage, while external storage was on punched cards or magnetic tape. All programming was done in laborious machine (numeric) language.

The second generation of computers built in the early 1960's used transistors instead of vacuum tubes, and, as a result, were cheaper, faster, and more reliable. While magnetic tubes and disks were used for external storage, the magnetic drum was replaced by a magnetic core for internal storage. High-level or source languages such as Cobol and Fortran were developed, along with assembler language to make programming easier.

In the early 1960's, when the transistor was replaced in the computer by the monolithic integrated circuit and solid state memory was used along with the magnetic core, a third generation of computer technology began. This generation is further distinguished from the previous

one by the development of operating systems, complex multi-programming, many more high-level languages, and by the reduction of size, cost, and electrical usage.

With advances in computer technology and electronic components progressing rapidly, the fourth generation of computers began emerging in the late 1960's and early 1970's. In the late 1960's, the development of large scale integration (LSI) allowed a number of circuits with separate functions to be integrated on individual chips which were soldered on circuit boards. With the advent of the circuit boards, another great breakthrough in computer technology was experienced. Now, whole computers could be assembled easily and inexpensively. However, the LSI had one drawback, since each circuit served only a particular purpose, the LSI lacked flexibility. Then, in the early 1970's, the one-chip microprocessor was designed - a chip on which control instructions could be stored. This microprocessor contained 2,250 transistors in an area barely a sixth of an inch long and an eighth of an inch wide, and could now be 'taught' to do any number of operations. The introduction of this one-chip CPU (Central Processor Unit) has made possible the production of small, inexpensive calculators and minicomputers.

Computer technology is developing rapidly. Four generations of computer evolution have been experienced in less than 40 years, and by the 1980's further advancements with bubble memories and Josephson junctions will be taking place. By the year 2000, the capabilities of the computer will have increased dramatically, and it will be employed in ways that we are only beginning to imagine.

The Videopac Computer

Your Videopac Computer is a fourth generation microprocessor. Its versatility and sophistication make it one of the finest of its kind on the market. The Videopac Computer not only offers you hours of entertainment with games such as Baseball, American Football, Basketball/Bowling, Race/Spin-out, Space Rendezvous, Golf, Blackjack, Cosmic Conflict and Air-Sea War/Battle, but also educational and instructional games such as Cryptogram, Pairs/Logic Mathematician/Echo, and Computer Programming.

This Computer Programming Videopac offers you an opportunity to introduce computer technology to your children and also learn about it yourself. Computers are fast becoming a part of our every day world, and the Videopac Computer can help prepare you and your children for the future.

Since this introductory Computer Programming Videopac is just that, a beginning, this booklet will introduce you to the basic information and factual background which you will need to be able to write and implement your own programs. We begin with the general organization of a computer; a detailed explanation of its fundamental operation with definitions of a register and accumulator, binary, hexadecimal, and assembler language. We will then give a description of your Videopac Computer and the steps to follow to create your own program.

When studying Computers, be certain you never go past a word you do not fully understand. If the material becomes confusing, there will be a word just earlier that you have not understood. Don't go any further - go back, find the misunderstood word, and get it defined. A computer dictionary will be invaluable.

You will soon be entering data into your Videopac Computer keyboard and enjoying the thrill of seeing your own program appear on the screen.

The Fundamentals of Microprocessors

A microprocessor consists of a small number of components which execute specific operations in a sequential manner. There are seven basic components in all microprocessors:

1. Input/Output Devices (I/O)
2. Arithmetic Logic Unit (ALU)
3. Accumulator
4. Memory
5. Location Devices
6. Control Logic
7. Bus Lines

The INPUT/OUTPUT devices are known as the I/O ports. The Input device is usually a keyboard, similar to your Videopac Computer keyboard shown in Figure 1, while the Output device is usually a video screen of some type (as in the case of your Videopac Computer, the television screen), or tape. It is through the Input/Output devices that you may enter data and view the results.

Within the ARITHMETIC LOGIC UNIT (ALU), all simple reasoning and arithmetic operations are performed. The ALU accepts data from two sources (the Accumulator and Memory) and this data is called an operand. The ALU accepts one or both of these operands, performs arithmetic calculations or logical operations based on the operand(s), then outputs one result. The ALU is also known as the number cruncher, since it is here that the various inputs of data are synthesized and a solution is reached. (See Figure 2)

The ACCUMULATOR, a working register (a register is a circuit where data is stored), is a small memory device that provides temporary data and/or instruction storage for the ALU and may store the result of the ALU's operation or may be used as an operand source for the ALU. (See Figures 3 and 4)

Besides temporary storage, the microprocessor needs bulk storage such as is provided by the MEMORY component. We know that our microprocessor is able to carry out certain tasks in a sequential manner. This sequence of instructions is called a program and is stored in the Memory component of the microprocessor. The program requires constants with which to process data and these also must be stored in Memory. Thus, Memory becomes a library of information consisting of program instructions, constants, and other data. (See Figure 5)

The Memory unit of the microprocessor is composed of two electronic components - ROM and RAM. The ROM component or Read Only Memory is like a book - it is printed at the factory and cannot be changed. It is referred to as a factory programmable ROM; its contents cannot be changed. It is in the ROM that program instructions and constants (repetitive numbers required for mathematical computation by the ALU) are stored. (See Figure 6)

Figure 1

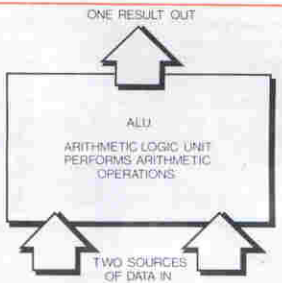


Figure 2

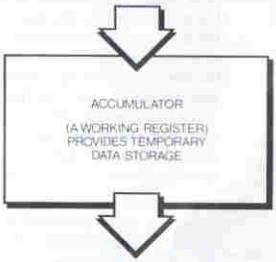
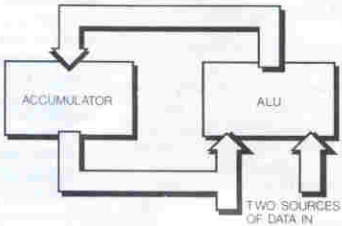


Figure 3



THE ACCUMULATOR MAY STORE RESULTS OF AN OPERATION OR BE USED AS AN OPERAND (SOURCE OF DATA) FOR AN OPERATION

Figure 4

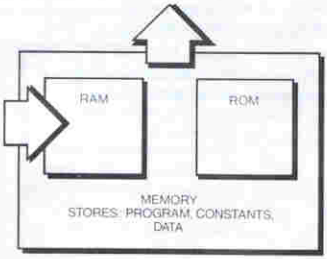


Figure 5

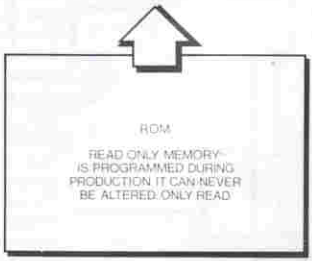


Figure 6

The second electronic component of the Memory unit is RAM or Random Access Memory. This component is like a blackboard. Data (programs, instruction sets, constants) can be entered and erased when desired and new data entered. The RAM may be programmed and reprogrammed many times. It is this component which allows you to write your own programs. The data you enter, unlike the program instructions and constants which are stored in ROM, may be changed, and therefore is stored in RAM. (See Figure 7)

The terms ROM and RAM describe the electronic components of the Memory unit. The Memory unit can also be viewed in terms of its functions, of which there are two - Program Memory and Data Memory. The PROGRAM MEMORY contains the addresses of the instruction sets and can send this information to the Instruction Register for decoding or to find an address (the location of data) in Data Memory. It is activiated by the Program Counter. The DATA MEMORY contains the addresses of data stored in Memory and sends this information to the Accumulator or, depending on the program, to other registers. It is activated by the Data Counter.

The PROGRAM and DATA COUNTERS are working registers which locate data inquired for by the ALU. Depending upon the nature of the data, the Program and Data Counters may find the necessary data in either the ROM or RAM components of the Memory unit. The microprocessor executes a program by finding a series of instructions in its Memory. Once the instructions are located, they are arranged and the program is executed in the proper sequence, which is essential to the accurate operation of the microprocessor. The Program Counter locates and identifies each instruction set and advances one step at a time, keeping the instructions in the proper sequence (i.e., Program Step = Ø0, Ø1, Ø2, Ø3, Ø4, etc.). When an instruction requires that some data be processed, the Data Counter locates and identifies that data in Memory and points to the address where the data is located. (See Figure 8)

Because data can be travelling in various directions within the microprocessor, it is necessary to have a CONTROL LOGIC component.
The Control Logic directs the functioning of all the other components and controls the data flow between them. The heart of the Control Logic component is the INSTRUCTION REGISTER. Here, the binary bits (more on these later) that compose the instruction sets are decoded and the necessary signals to implement the instruction are generated. (See Figure 9)

The Control Logic is connected to all other parts of the microprocessor by way of data control buses or BUS LINES, of which there are three. The DATA BUS transmits data between the ALU and Memory; the ADDRESS BUS transmits addresses of data being accessed by the ALU to Memory; and the CONTROL BUS which is a group of channels used for special control purposes (i.e., clearing the registers for resetting of microprocessor, stopping the microprocessor after the instructions have been terminated, etc.).

Several of the components discussed above compose the Central Processing Unit, which is the heart of the microprocessor. These components are the ALU, Accumulator, Program and Data Counters, Control Logic unit, and Instruction Register.
Figure 10 shows the complete microprocessor system, as we have discussed it. You can trace the route which data may take by following the arrows.
We have mentioned registers, bits, and computer language during our explanation of the basic components of the microprocessor. In the next chapter, we will discuss these important aspects of computer technology in depth.

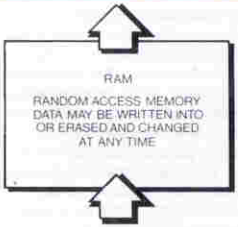


Figure 7

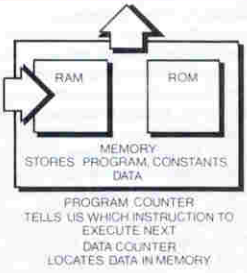


Figure 8

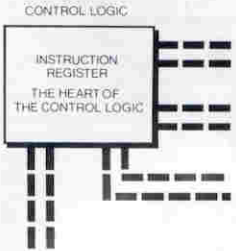


Figure 9

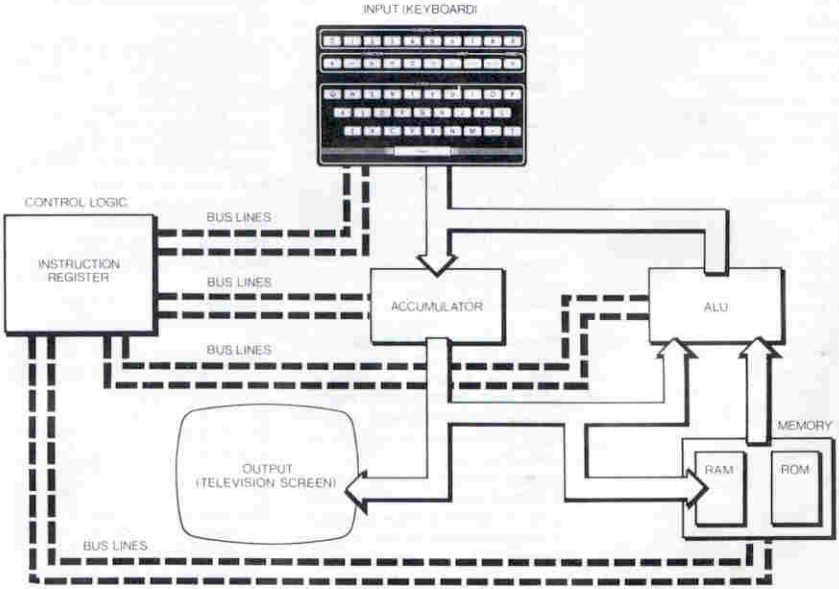


Figure 10

CHAPTER 2

The Binary Number System and Computer Languages

We mentioned binary bits in Chapter 1. Now we will explain what they are in detail and why they are so important in relation to microprocessors. In addition to this, we will explain what is meant by computer language and discuss the different types - binary or machine language, hexadecimal, and assembler.

Microprocessors perform functions accurately and at high speed by manipulating symbols according to a set of instructions, which are stored in the Program Memory. The operation of the microprocessor consists of executing the instructions and data in sequence. Thus, these instructions must be in a form that the microprocessor can understand, so both instructions and data are written in binary number codes, or machine language. In order to understand the operation of the microprocessor and to be able to write your own programs, you need to be able to write in machine language and to understand what is being represented in the microprocessor.

Binary or Machine Language

Let's first begin with some definitions. A number system is a set of symbols (digits) that may be operated upon by arithmetic rules. Each symbol or digit has its own name (i.e., in the decimal system, we have 0, 1, 2, 3, 4, 5, 6, 7, 8, 9). A number system also has a set of rules that define how to arrange the digits to form numbers.

Positional notation allows numbers to be written that express all quantities, no matter how large or small. The value of a digit depends on its position in the number. For example, the digits of the number 5555 are identical, yet each has a different value. To write 5555 is a short way of writing five thousand + five hundred + fifty + five or, if you wished to express it in the powers of 10, you could write, $5 \times 10^3 + 5 \times 10^2 + 5 \times 10^1 + 5 \times 10^0$. Ten is the base or radix of the decimal number system. In the binary system, the radix is 2; and in the hexadecimal system, the radix is 16.

Because the voltage levels in your microprocessor can only recognize 2 levels - on or off - the binary number system is used to encode data within the microprocessor. The binary number system is written with 1's and 0's and, as we mentioned above, has a radix of 2. For example, the binary number 11011 can be written 11011_2. Expressed in an equation, it looks like this:

$11011_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

We now perform those mathematical operations expressed in the above equation:

$2^4 = 2 \times 2 \times 2 \times 2 = 16$
$2^3 = 2 \times 2 \times 2 = 8$
$2^2 = 2 \times 2 = 4$
$2^1 = 2 = 2$
$2^0 = 1 = 1$

(You may ask, 'Why does $2^0 = 1$?' There is a rule in the binary number system which states: Any number not zero raised to

the zero power is 1. Since $2^0 = 2^1 - 2^1$ when written in powers and if two numbers of the same base are divided, the value of the power of the number in the denominator (the number below the fraction line) may be subtracted from the value of the power of the numerator (the number above the fraction line). Thus, $2^1 - 2^1 = 2^{1-1} = 2^0 = 1$.)

Now we must place the values of the numbers taken to the power of 2 and place them in our equation: $11011_2 = 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 27$

Below is a table listing eleven binary numbers and their decimal equivalents. Note the positional notation of the binary numbers: $2^4\ 2^3\ 2^2\ 2^1\ 2^0$ and their base 10 equivalents: 16, 8, 4, 2, 1. As a shortcut to what we have just done above, you could write:

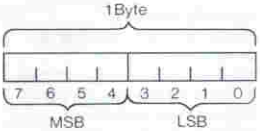
16 8 4 2 1
1 1 0 1 1

and add the values where there is a 1

in other words: 16 + 8 + 2 + 1 = 27.

Binary					Decimal
2^4	2^3	2^2	2^1	2^0	
16	8	4	2	1	
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10

The microprocessor is instructed what to do by programming it with a series of instructions (instruction sets to which we have previously referred, will be explained in detail later). It may be programmed by writing a sequence of instructions in machine language or binary code (ones and zeros). Computer terminology refers to digits in the binary system as bits, a contraction of **bi**nary and digi**ts**. Thus, in the binary number 11011, there are five bits. A bit is the basic unit of information used in a microprocessor. Bits are handled by the microprocessor in groups of eight and this group is called a byte; it is the smallest piece of data with which the microprocessor can work. Space for data is allocated in the registers within the microprocessor in 8 bit bytes. In other words, each register has 8 spaces in which data can be stored. The bits in a byte are identified by numbering them according to their position, (power of 2) from 0 to 7, from least significant bit (LSB) to most significant bit (MSB); thus:



By breaking the group of 8 bits or 1 byte into two halves, we have what is called in computer terminology 2 nibbles, each containing 4 bits.

As we stated above, a microprocessor is instructed what to do by programming a sequence of instructions in the binary code which the machine can directly interpret. In the next few examples we shall describe do not be confused by the fact that you do not understand how the binary code is equal to what is given in the explanation column. All that will become clear to you when we study the instruction sets in a later chapter. For now, we are only concerned with binary code, how it is written, and the other, simpler languages of hex (hexadecimal) and assembler. As for the Program Steps, these too will be further explained in a later chapter, but for now all you need remember is that each 8 bits of data require one program step. For example, if we were to write a program in binary, the first few steps would look like this:

Program Step Number	Machine Code	Explanation
ØØ	0110 0000	Load a value into register Ø, the value is 38.
Ø1	0011 1000	
Ø2	0110 0001	Load a value into register 1, the value is ØC.
Ø3	0000 1100	

Hexadecimal Language

As you can see, this type of programming is laborious and, with all those ones and zeros, subject to error if a large number of steps are written. So the hexadecimal language was developed. It is a more efficient way to represent any group of 4 bits of machine language code in a shorthand format. Hexadecimal is not a code, merely a means of replacing 4 consecutive bits by a single character. As we mentioned before, the radix of hexadecimal is 16. In hex notation, the first 10 values are represented by the digits 0 - 9 and the last six values by the letters A - F. Each number or letter represents a 4-bit binary number. The table below shows the corresponding values of the decimal, hex, and binary number systems.

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

Our program, written in hex, now looks like this:

Program Step Number	Hex Code	Explanation
ØØ	6Ø	Load a value into register Ø, the value is 38
Ø1	38	
Ø2	61	Load a value into register 1, the value is ØC
Ø3	ØC	

(The fact that the instruction 'Load a value' is equal to the number 6 will be explained later.)

Though you are entering your program in hex, the machine still reads only binary numbers; thus hex is the programmer's shorthand not the microprocessor's. However, one limitation which you may have noticed in binary and hex language, is that it is not self-documenting. In other words it does not give any indication of the operation being performed. We have had to supply an explanation column in order to know what is happening.

Assembly Language

Hence, assembly language is used to overcome the disadvantages of hex and binary language by allowing the use of alphanumeric symbols to represent machine operation codes, branch addresses, and other operands. (You will remember that an operand is the data which is entered into the microprocessor and which is affected, manipulated, and operated upon). In assembly language, our original program looks as follows (we will continue, at this point, to supply an explanation column, since you are not yet familiar with the assembly language):

Program Step Number	Assembly Code	Explanation
ØØ	LDV Ø,38	LoaD a Value in register Ø and the value is 38
Ø2	LDV 1,ØC	LoaD a Value in register 1 and the value is ØC

Notice that the step number goes from ØØ to Ø2. Where are Ø1 and Ø3? Remember, each step contains only 8 bits or 1 byte. Assembly language uses 2 bytes. That is, LDV Ø, represents one byte (6Ø in hex and 0110 0000 in binary) and 38 represents the second byte (38 in hex and 0011 1000 in binary). Though only steps ØØ and Ø2 are shown when writing your program, the machine (since it reads only in binary) advances one step each time a byte is completed. However it is important to remember, if you write your program in assembly language, to allow enough steps for each instruction. How to do this will be explained later.

Binary Arithmetic

Now that we have discussed binary and are familiar with the simpler methods of programming (hex and assembler languages), we will discuss the various arithmetic operations which can be performed in a microprocessor and how to write them in binary, since this is the

language which the machine understands. In the next chapter we will discuss entering this type of data into the microprocessor and study what goes on within the microprocessor. But for now, we will discuss binary arithmetic.

Converting Decimal Numbers to Binary and Vice Versa

To begin, we must know how to convert a decimal number to a binary number. You will remember that the radix of binary is 2, thus we repeatedly divide the decimal number by 2; each answer we get, we again divide it by 2. The remainder at any step of the division can only be 0 or 1. These remainders become the bits of the binary equivalent. For example, let's change the decimal number 25 to binary:

$25 \div 2 = 12$ remainder $1 = 2^0$
$12 \div 2 = 6$ no remainder $0 = 2^1$
$6 \div 2 = 3$ no remainder $0 = 2^2$
$3 \div 2 = 1$ remainder $1 = 2^3$
$1 \div 2 = 0$ remainder $1 = 2^4$

Note that we began with the least significant digit; therefore, to write our binary equivalent of 25, we must begin with the most significant digit (ie the left-most digit, remember our diagram of the MSB and LSB). Our binary equivalent becomes 11001. If you take the decimal equivalents 16 8 4 2 1 and under them place your binary number for 25 - 1 1 0 0 1 - and add the decimal equivalents where you have binary ones, your result is 16 + 8 + 1 = 25. The decimal equivalent of 11001 is thus 25.

The obvious method for converting binary numbers to decimal would be to do as we have done above, or, if you don't remember the decimal equivalents, to select the one-bits in the binary number and convert each to a decimal and then add the results together. For example:

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$
$= 1\ 1\ 0\ 0\ 1$
$= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 16 + 8 + 0 + 0 + 1$
$= 25_{10}$

Binary Addition

Now that we know how to change decimal numbers to binary and vice versa, let's look at the addition of binary numbers. Binary arithmetic is easier to perform than decimal, but you have to learn some new rules since there are only two digits with which to work in binary - - ones and zeros. When adding binary numbers, remember the following combinations:

0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	0
			with a carry of 1

A carry-1 bit is produced from the addition of 1 + 1. Binary carries are treated in the same way as decimal carries; they are carried over to the left. For example:

Decimal	Binary
15	1111
+ 7	+ 0111
22	10110

Note that a condition of 1 + 1 = 0 gives a carry of 1. The next condition is 1 + 1 + 1 which can only equal 1 with a carry of 1.

Binary Subtraction

Since the microprocessor uses the twos complement method of subtraction, we shall discuss and explain it to you. However, before we begin, let's review the decimal system's complement method, so that the twos complement of the binary system might be easier to understand.

The nines (9s) complement of a decimal number is that value which is obtained by finding the difference between each of its digits and 9. Decimal subtraction using complements is performed by following two rules:
1. Add the complement of the subtrahend to the minuend.
2. Add the carry to the least significant digit. This carry is known as the end-around carry.
For example:

Normal Decimal Subtraction

231	minuend
- 056	subtrahend
175	difference

Nines Complement Decimal Subtraction

231 minuend
056 subtrahend

Subtract the subtrahend from the 9s complement:

```
  999
- 056
  943
```

Apply Rule 1:
Add the complement of the subtrahend to the minuend.

```
  231
+ 943
 1174
```

Apply Rule 2:
Add the carry to the least significant digit.
Sum is 1 174
end -around carry

```
  174
+   1 (end around carry)
  175
```

The subtrahend must always be the smaller of the two numbers. If it is not, invert the problem and change the sign of the result.

As you can see, in the decimal number system, it is awkward to use the complement method. However, microprocessors make complementing simple, since it uses 8 bit registers (remember, each register has 8 spaces for storing data) to contain many operands, and each register is filled with zeros until a bit of data is entered. Thus, all numbers are expressed in 8 bits regardless of the value (i.e., the binary number 1010 would reside in the register as 0000 1010).

By using this 8 bit number format, it is easy to obtain the twos (2s) complement. Just remember the following rules:
1. All ones in the subtrahend are changed to zeros and all zeros are changed to ones.
2. A one is then added to the least significant bit of the new subtrahend.
3. Add the twos complement of the subtrahend to the minuend and ignore the carry bit, if there is one.

Twos Complement Binary Subtraction

```
  0000 1010  minuend
- 0000 0101  subtrahend
```

Apply Rule 1:
All ones in the subtrahend are changed to zeros and all zeros to ones.
0000 0101 becomes 1111 1010

Apply Rule 2:
Add a one to the least significant bit of the new subtrahend.

```
  1111 1010
+         1
  1111 1011
```

Apply Rule 3:
Add the twos complement of the subtrahend to the minuend and ignore the carry bit, if there is one.

```
  0000 1010
+ 1111 1011
    becomes
  ØØØØ 1Ø1Ø
+ 1111 1011
  0000 0101
```

(If you need help in adding binary numbers, refer to the chart in the section BINARY ADDITION. Remember, when you add 1 + 1 in binary, it equals 0 with a carry of 1. Also, the last operation was to add 1 + 1; we write the 0, but the carry is lost. We don't need it, so we ignore it, since there is no extra space in the register in which to put it.)

Binary Multiplication

Multiplication in binary is a simple process. There are only two rules to remember:
1. The product of 1 × 1 = 1.
2. All other products = 0.

```
  0     1     0     1
× 0   × 0   × 1   × 1
  0     0     0     1
```

For example:

Decimal	Binary	
7	111	multiplicand
× 5	× 101	multiplier
35	111	partial product
	000	partial product
	111	partial product
	100011	product

(Don't forget, when adding your partial products, that 1 + 1 = 0 with a carry of 1. Refer to the chart in the section BINARY ADDITION.

As in decimal multiplication, binary multiplication involves a series of shifts and addition of the partial products. The partial products are equal to the multiplicand or to 0. Every 1 - bit in the multiplier gives a partial product equal to the multiplicand shifted left the corresponding number of places. Every 0 in the multiplier products a partial product of 0. Each partial product is shifted left one position from the preceding partial product, the same as in decimal arithmetic. In some microprocessors, this shifting to the left has been programmed on the ROM and the

microprocessor shifts automatically. However, in the Videopac Computer, multiplication is done by a series of additions.

For example, 7 × 3 to the Videopac Computer really means 7 + 7 + 7.

```
   7          0000 0111 =  7
 × 3        + 0000 0111 =  7
  21          0000 1110 = 14
            + 0000 0111 =  7
              0001 0101 = 21
```

To multiply 7 × 3, the Videopac Computer adds 7 three times. This may seem cumbersome to you, but the computer operates so quickly - it 'crunches' numbers in its ALU so fast that it takes no time at all to reach a solution. However, when writing a multiplication problem, you will need to program these addition steps into the microprocessor. How to do that will be covered under the instruction sets in a later chapter.

Binary Division

Binary division is similar to decimal long division. Yet it is simpler for there are only two rules to remember:

0 − 1 = 0 1 − 1 = 1

Division by 0 (1 − 0 or 0 − 0) is meaningless.

Before we give you an example, remember the following: Like decimal divison, binary division must first be approached by deciding if the divisor is larger or smaller than the first bit in the dividend furthest to the left. If it is larger, then you must consider the next bit also, and so on until you find a bit in the dividend which is larger than the divisor.

After you have found the bits in the dividend into which the divisor will go, you proceed just as in decimal division, multiplying and subtracting (using the twos complement) with right shifts.

Let's look at the following example in detail:

Decimal	Binary
3	
3 ⌈9	− 0011 ⌈1001 −
− 9	divisor dividend
0	

First, we must take the divisor, 0011 (3), and the first bit of the dividend, 1. Since this first bit of the dividend is regarded as 0001, its value is 1 and 3 cannot be divided into 1. So we take the first two bits of the dividend, 10. Again, regarded as the first two bits of data (0010), their value is only 2 and 3 cannot be divided into 2. So we take the first three bits of data, 100. Their value is 0100 (4), and we can divide 0011 (3) into 0100 (4), one time. Thus, a 1 is placed above the second 0 in the dividend.

```
          1  − quotient
divisor − 0011 ⌈1001 − dividend
```

We then multiply as in decimal devision (refer to your multiplication chart on page 12).

```
          1   − quotient
divisor − 0011 ⌈1001 − dividend
        − 011   − partial
                  dividend
```

Now we must subtract the partial dividend from the dividend using the twos complement. So let's apply Rule 1 for the twos complement: All ones in the subtrahend (partial dividend) are changed to zeros and all the zeros to ones.

```
        1
0011 ⌈1001
  − 011  becomes 100
```

Apply Rule 2 for the twos complement: Add a one to the least significant bit of the new subtrahend (partial dividend):

```
        1
0011 ⌈1001
  − 011
    becomes 100
          +   1                  1
        101   0011 ⌈1001
              becomes + 101
```

Apply Rule 3: Add the twos complement of the subtrahend (partial dividend) to the minuend (dividend) and ignore the carry bit, if there is one. (Refer to your addition table in the section 'Binary Addition'.)

```
              1
        0011 ⌈1001
           + 101
carry is dropped - - (1) 001
```

Just as in decimal division, you now bring the next bit of data down from the dividend to the partial dividend.

```
      1                      1
0011 ⌈1001  becomes 0011 ⌈1001
   + 101                  + 101
     001                    0011
```

You can see that we now have a divisor of 3 (0011) and a partial dividend of 3 (0011). Thus, 3 − 3 = 1, or 0011 − 0011 = 1. We now have in our example:

```
       11
0011 ⌈1001
   + 101
     0011
     0011
     0000
```

And our answer is 0011 or 3.

As in multiplication, the shifting operation (to the right in division) is often programmed in the ROM of the microprocessor. However, in the Videopac Computer, division is completed by a series of subtractions. For example:

```
   2
4 ⌈8
 − 8
   0
```

To the Videopac Computer, this problem really means 'How many times can 4 be subtracted from 8?'

```
08 = 0000 1000 - minuend
04 = 0000 0100 - subtrahend
```

Remember, in subtraction, you find the twos complement of the subtrahend:
0000 0100 becomes 1111 1011
and add one to the least significant bit:

```
1111 1011
          1
1111 1100
```

Then you add the twos complement of the subtrahend to the minuend:

```
 08 = 0000 1000
−04 = 1111 1100

    carry
    is dropped - (1) 0000 0100
```

Your answer is still 4, so we must again use the twos complement of −04, for example:

```
 0000 1000
+ 1111 1100
(1) 0000 0100
+ 1111 1100
(1) 0000 0000
```

We performed the subtraction operation twice with no remainder, thus 8 ÷ 4 = 2.

Again, as in multiplication, when we write a division problem we must program these subtraction steps into the microprocessor. How to do this will be covered under the instruction sets in a later chapter.

It may be helpful to remember that shift operations are used to multiply or divide binary numbers by powers of 2 (not multiples of 2). A left shift of one position multiplies by 2; a left shift of two bit positions multiples by 4; a left shift of three bit positions multiplies by 8; and so on. Similarly, in binary division, a right shift of one position divides by 2 (i.e., multiplies by 1/2, or .5); a right shift of two bit positions divides by 4 (i.e., multiplies by 1/4, or .25); a right shift of three bit positions divides by 8 (i.e., multiplies by 1/8, or .125); and so on. For your reference, a table containing the powers of two is located in the Appendix.

By now you may be frustrated and a bit discouraged by all these details, and you are probably wondering if you will ever get to punch the keys and write your own program. There is, however, a lot to learn and the more background information you acquire, the easier and more quickly you will understand the instruction sets and how they are used.

In the next chapter, we will review the basic components of the microprocessor and how they relate to the Videopac Computer, talk about the keycodes, and review the registers - - their operation, the data they contain, and the number of registers in the Videopac Computer.

This page is intentionaly blank

The Videopac Computer

In Chapter 1, we covered the seven basic components of a microprocessor. Before we begin the study of the instruction sets, let's review these components in relation to the Videopac Computer.

The Input/Output devices for the Videopac Computer are the keyboard (as shown in Figure 11) and your television screen respectively. The keyboard has 48 keys which have been previously encoded, each with an 8-bit code. The keys, with their hexcode (the code you use to enter the value) and their decimal equivalents, are shown in Figure 12. Each key has a different representation (library value) to the microprocessor and this representation (either decimal or key value) is determined by the instruction sets programmed (i.e., depending on the instruction set programmed before and after, a '1D' entered may be an 'H' or the decimal number '29'; more on this later).

You will remember that the Arithmetic Logic Unit (ALU), also known as the 'number cruncher', performs all arithmetic and reasoning operations. It is in the ALU that operands from different registers are manipulated to obtain a result. The Accumulator is the small memory device which stores data and/or instructions for the ALU. It can receive data from the keyboard or from a register within the microprocessor.

We know that bulk storage is needed within the microprocessor for storing instruction sets, constants, and

Keycodes, Hex Codes, and Decimal Equivalents

Key	Hexcode	Decimal	O	17	23
Ø	00	00	P	0F	16
1	01	01	Q	18	24
2	02	02	R	13	19
3	03	03	S	19	25
4	04	04	T	14	20
5	05	05	U	15	21
6	06	06	V	24	36
7	07	07	W	11	17
8	08	08	X	22	34
9	09	09	Y	2C	44
A	20	32	Z	21	32
B	25	37	Blank	0C	12
C	23	35	:	0A	10
D	1A	26	$	OB	11
E	12	18	Clear	2E	46
F	1B	27	?	0D	13
G	1C	28	.	27	39
H	1D	29	+	10	16
I	16	22	-	28	40
J	1E	30	×	29	41
K	1F	31	÷	2A	42
L	0E	14	=	2B	43
M	26	38	Enter	2F	47
N	2D	45			

Figure 12

Figure 11

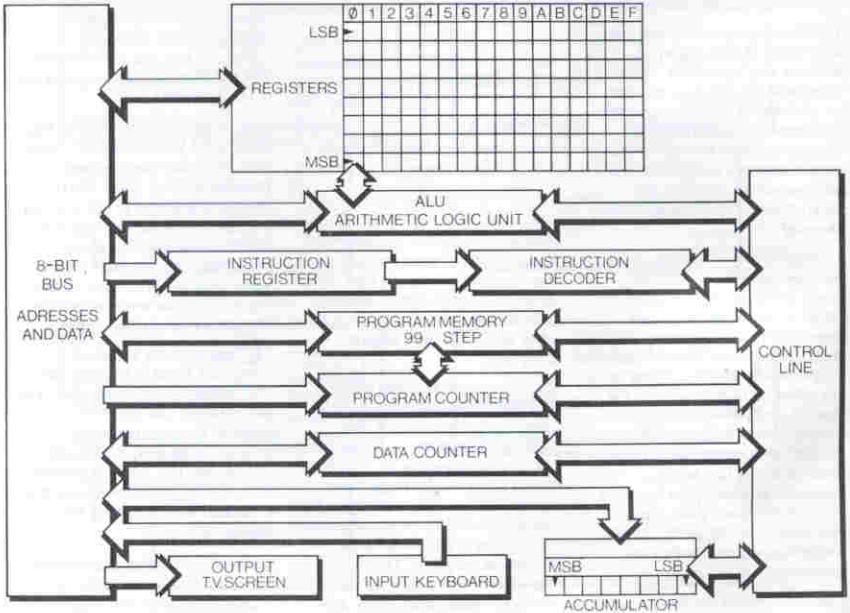


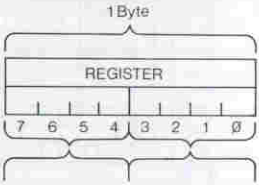
Figure 13

other data. You will remember that the Program Memory contains the addresses (location in register) of the instruction sets and sends this information to the Instruction Register for decoding or to find an address in Data Memory. Program Memory is activated by the Program Counter which locates and identifies each instruction set which has been entered. It begins at Program Step ØØ when the Reset button is pushed and advances one step at a time, as the program is used, so that the instructions can be executed in the proper sequence.

The Data Memory contains the addresses of data stored in Memory and sends this information to the Accumulator, or, depending upon the program, to another register. The Data Counter locates and identifies data in Memory and points to the address where that data is located, either in Program or Data Memory.

To keep order within the microprocessor, the Control Logic unit is connected to all other units via the Bus Lines (remember, Address Bus, Data Bus, Control Bus). The Control Logic directs the functioning of the other units and controls the flow of data between them. The centre of the Control Logic is the Instruction Register, for it is here that the binary bits which compose an instruction set, sent from Memory, are decoded and the necessary signals to implement the instruction are generated.

Figure 13 shows the basic components of the Videopac Computer, along with its registers. You will note that the Videopac Computer has 16 registers available for data input - Ø to 9 and A to F. Remember that each register can contain 8 bits of data or 1 byte. The data stored in the register can be an address (location) or it can be numeric (symbolic) data. Also, remember that the 8 bits of data are divided in half within the register. The first 4 bits are the Most Significant Bits and the last four are the Least Significant Bits.



Also, note from Figure 13, that the Videopac Computer contains 99 program steps, which means that we can write a program with up to 99 steps. That is a lot of steps to program, as you will see when we write and implement several programs.

We have also discussed binary, hex, and assembler languages. Like all microprocessors, the Videopac Computer operates by following a sequence of instruction sets. Although the Videopac Computer reads these instruction sets in binary, you may enter them in either hex or assembler language. These instruction sets, which tell the microprocessor what to do, and how to enter them will be fully explained in the next chapter.

Before we begin studying the instruction sets, let's follow the execution of a program as it passes through the components of the computer which we have just reviewed. First, remember that before a program can be executed, it must be entered and stored in Memory. Let's assume that this has been done with the program that will add two numbers; let's choose seven and ten.

Fig. 14 shows a diagram which represents the important registers of the Videopac Computer and you will note that the program to add seven and ten is stored in Memory.

First, note that each register has room for 8 bits of data, or 1 byte. Also note that the Program Counter is set at 0000 0000, since the program is just starting execution. Remember that it is the Program Counter which increments by one each time an instruction is performed, so that the instructions are executed in the proper sequence.

The first step the Videopac Computer takes is to fetch the first instruction from Memory. You will note that the Program Counter contains the exact address in Memory of the first instruction. This address passes from the Program Counter to the Address Register. (Refer to Figure 15)

Once the address is transfered from the Program Counter to the Address Register, the Program

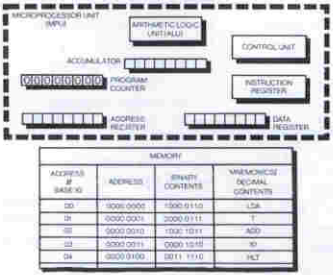


Figure 14

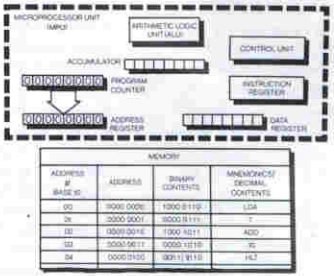


Figure 15

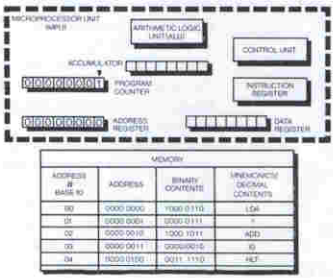


Figure 16

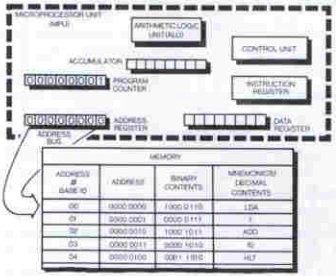


Figure 17

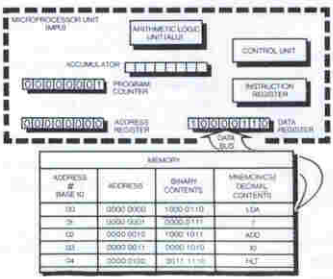


Figure 18

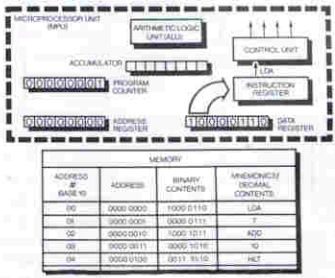


Figure 19

Counter is incremented by one, so that it is ready for the next instruction (the address in the Address Register remains 0000 0000). (See Figure 16)

The contents of the Address Register are sent to the identical address in Memory via the Address Bus. (See Figure 17)

The contents of Memory at address 0000 0000, which are 'LDA' or Load Accumulator, are now sent to the Data Register via the Data Bus. The contents are transfered in binary, i.e. 1000 0110, which you will remember is the only language the Videopac Computes can read. (See Figure 18)

Now, the contents of the Data Register must be decoded, so that the Videopac Computer can perform the instruction requested, which is Load the Accumulator. Therefore, the contents of the Data Register are sent to the Instruction Register, decoded, and the proper signals are sent to the Control Unit, which produces the necessary pulses to carry out the instruction. (See Figure 19)

Now that the first instruction has been fetched and decoded, the next step for the Videopac Computer is to load the accumulator with the next byte of information contained in Memory. We return to the Program Counter; its contents are 0000 0001. This address is transfered to the Address Register. (See Figure 20)

In Figure 21 you will note that the Program Counter has incremented by one and the address in the Address Register is being bussed to the identical address in Memory via the Address Bus.

In Memory the address (0000 0001) is located and its contents ('7' or in binary 0000 0111) are sent to the Data Register via the Data Bus. Since the instruction called to 'Load the Accumulator', the contents of the Data Register ('7' or 0000 0111) are immediately loaded into the Accumulator. (See Figure 22)
Next, the Videopac Computer must again fetch another instruction, which happens to be 'Add'. The Videopac Computer travels through the same steps we have outlined so far: (Refer to Figure 23)
1. The contents of the Program Counter (0000 0010) are transfered to the Address Register.
2. The Program Counter increments by one (0000 0011). (See Figure 24)
3. The address travels to Memory via the Address Bus.
4. The binary contents (1000 1011 or 'Add') at Memory address 0000 0010 are transfered to the Data Register.
5. The contents of the Data Register are decoded by the Instruction Register which tells the Control Unit what operation is to be implemented.

The execution of the Add instruction now takes place. (Refer to Figure 24)
1. The contents of the Program Counter (0000 0011) are transfered to the Address Register.
2. The Program Counter increments by one (0000 0100). (See Figure 25)
3. The address travels to Memory along the Address Bus.
4. The binary contents (0000 1010 or '10') at Memory address 0000 0011 are transfered to the Data Register via the Data Bus.
5. The binary contents of the Data Register are immediately sent to the ALU (Arithmetic Logic Unit), and the binary contents of the Accumulator (which were '7' or 0000 0111) are transfered to the other input of the ALU.
6. The ALU adds the two operands and the sum, '17' or 0001 0001, is loaded into the Accumulator.
Since our program is now at an end, we must instruct the Videopac Computer to stop execution. Therefore, we fetch the 'Halt' instruction from Memory. The same procedure is performed as before. (Refer to Figure 25). The sum is in the Accumulator, the 'Halt' instruction is being read by the Control Unit and all execution will be halted. Instead of 'Halt', we could have programmed an output instruction which would have allowed the sum, 17, to be displayed on the screen. All steps would have been the same with the Control Unit instructing the Videopac Computer to display the contents of the Accumulator on the screen. You will see how this can be done in the next chapter.

Well, now is the time for which you have been waiting. We are ready to study the instruction sets and begin to punch the keys of our Videopac Computer.

www.videopac.org
the worlds best videopac website

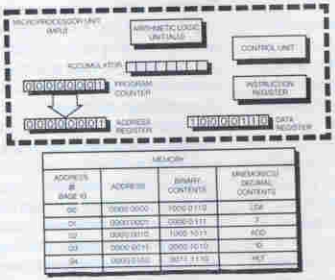


Figure 20

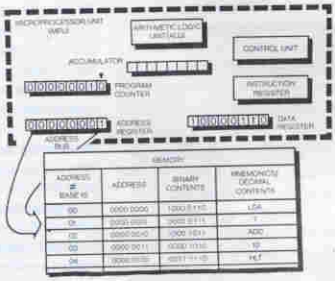


Figure 21

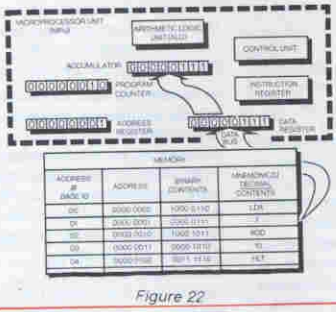


Figure 22

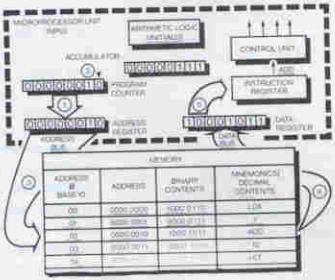


Figure 23

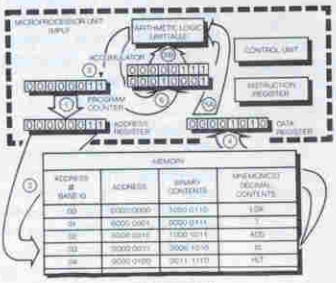


Figure 24

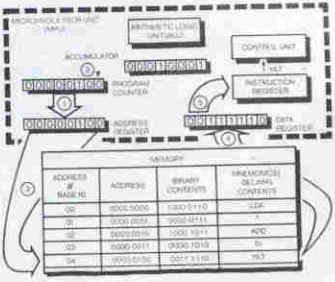


Figure 25

CHAPTER 4

The Videopac Computer Instruction Sets

The architecture of the Videopac Computer was described in the previous chapter. You should now be familiar with the computer and its basic operation. In order to be able to write your own program, you must now study the instruction sets. You will remember that these instruction sets are the codes which tell the Videopac Computer what to do (i.e., load a value into a register, output a value from a register, input to the accumulator, etc.). In other words, the instruction sets move data from one register to another and perform operations on this data. Something which you should always keep in mind and which will become obvious as you begin to write your own programs is the fact that the Computer can do nothing but what you tell it to do. Although it has been preprogrammed with certain data (constants, stored symbols, etc.) which are stored in ROM, the Videopac Computer cannot execute a program until you have entered that program step by step, using the instruction sets. Every step must be written by you and entered into the Videopac Computer before it can function. Thus, the instruction sets - what they are, when to use them, and how to enter them - will be explained in this chapter.

Basically, the instruction set does two things. First, it tells you the operation that is to be performed, and secondly, it tells you the destination and the value to be placed in that destination. For example, the instruction LDV Ø,18 means 'Load a value (operation) into register Ø' (destination) and the value is 18'. We have used LDV Ø,18 previously and we know it as assembler language. This language is also called mnemonic (pronounced 'new monic') and is valuable to the programmer since he can tell at a glance the type of operation being performed and the register and value being used. ' Zero is always written Ø to distinguish it from the letter 'O'.

As we discuss each instruction set, we will give the mnemonic symbol for each set. Also included will be the operational code, which is the hexadecimal code for the instruction set. This is the code you will use most often, since it is easier and less lengthy than assembler (mnemonic) to write. Each description of an instruction set will also include an operation symbol which explicitly describes the operation being completed by the instruction set. For example:

Instruction Set	Input to Register
Mnemonic Code	INP,R
Operational Code	7R
Operation Symbol	IP → R

You will remember from Chapter 2 when we discussed bits and bytes, that each program step can only hold one byte or 8 bits of data. Each instruction set varies, some are 1-byte and some are 2-byte instructions. For example, the instruction 'Input to Register' is a one-byte instruction. Its Op Code is 7R (R stands for register of your choice). However, 'Load a Value' is a two-byte instruction. Its Op Code is 6RNN (R stands for the register of your choice and NN stands for the value you wish to place in the register). We will note the number of bytes in each instruction set.

At the end of this chapter we have listed all the instruction sets, their Mnemonic and Op Codes, the Operation Symbol, number of bytes, and a remark column. After stydying the instruction sets in detail, this sheet will be a good reference for you when writing a program.

Also, before we begin, you should know that the Videopac Computer has been programmed with a group of symbols stored in ROM (Read Only Memory). These symbols are shown in Fig. 32 along with their hex values. Also preprogrammed in the Videopac Computer is the special use of Register B.

Register B has been programmed to position symbols or characters on the television screen. It has been given eleven positions, from Ø0 (furthest left on the screen) to ØA (furthest position to the right). The eleven positions are shown in Fig. 26.

The hex numbers, Ø0, Ø1, Ø2, etc., do not appear on the screen. They only show the relative position some symbol of your choosing would have on the screen. Note that when register B outputs to the screen, it automatically increments by one. In other words, if we output a symbol in position Ø2, the symbol will appear in the Ø2 position and the register B will then advance to Ø3. If the register outputs at ØA, it automatically resets to Ø0 on the next step.

Before we describe each instruction set, let's discuss the following three instruction sets in detail.

1. Load a Value into a Register
2. Output from a Register
3. Input to Accumulator

After we have studied these, we will step through a program using them and stored symbols. Please note that Operational Code is abbreviated Op Code.

The first instruction is 'Load a Value into a Register'.
The codes are:

Mnemonic	LDV.R.NN
Op Code	6RNN
Operation	R = NN

Remember that the videopac Computer has two languages in which it can be programmed – assembler and hex. However, when you program in assembler, the Videopac Computer automatically changes the data to hex. (i.e., if you were to enter in assembler language LDV.B Ø5, when you reviewed it on the screen, it would be in hex, 6BØ5, which is a two-byte instruction). The Mnemonic Code would be used to program in assembler language, and the Op Code would be used to program in hex. Let's look at each in detail:

The mnemonic LDV.R.NN means:

LDV = Load a value – This tells the microprocessor which action to perform.
R = Register – The Videopac Computer has 16 registers – Ø-9 and A-F. You may use any of them; however, remember that register B is used for positioning a symbol on the screen.
NN = Some value, like Ø0, Ø9, 13, 75, etc.

For example
LDV.R.NN (Assembler or Mnemonic Code)

LDV.Ø.18 – Means load a value into register Ø and the value is 18.
LDV.B Ø5 – Means load a value into register B and the value is Ø5.

Let's look now at the Op Code 6RNN

6 = Load a value – Tells microprocessor what action to perform.

R = A register you choose. Ø-9 or A-F

NN = Some value.

For example:

6RNN (Hex Code)

6Ø18

- Load a value
- Into Register Ø
- The value is 18

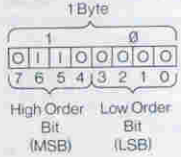
6BØ5

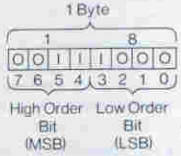
- Load a value
- Into Register B
- The value is Ø5

This instruction set is used when you wish to program into a specific register a specific value.

You will recall that we mentioned instruction sets may be 1 or 2 bytes in length. The instruction LDV Ø 18 is a 2-byte instruction. Since LDV Ø 18, once entered into the machine, becomes 6Ø18, let's look at that instruction in relation to bits and bytes.

Remember, a byte is made of 8 bits. 4 are the the most significant bits (or high order bits) and 4 are the least significant bits (or low order bits). Six (6) would be the high order bit and Ø would be the low order bit, together they would be one byte. The same is true of 18, thus making 6Ø18 (LDV Ø 18) a two-byte instruction. We must then allow two program steps for this instruction.





Since the Videopac Computer has only 99 progam steps, when writing a program, we must keep a record of how many steps have been used. Thus, we must know many steps (bytes) are used for each instruction set.

The next instruction to be studied is 'Output from a Register'. In the first instruction set, we loaded a value into a register, now we must output that value to the screen. Remember, the Videopac Computer does only what it is told, so we must instruct it each step of the way.

The codes for the instruction 'Output from a Register' are as follows:

Mnemonic	Out.R
Op Code	CR
Operation	R → Out

Referring back to our example, LDV Ø 18 and LDV B Ø5 (in Op code, 6Ø18 and 6B05, respectively), we now wish to output from register Ø our value 18 in position Ø5 (remember register B is our position register) on the television screen. After every input instruction (unless you are using it as a pause), you must write an output instruction, if you wish to display the contents on the screen.

Using our example, LDV Ø 18, our output instruction becomes Out Ø in mnemonic or CØ in operational code; for LDV B Ø5 our output instruction becomes Out B or CB. In other words, the symbol designated by hex 18 (the letter 'Q', refer to Figure 12) would be displayed on the television screen in position Ø5.

For your information, the registers in the Videopac Computer would look as in Fig. 27.

Since this first progam will be a short, simple one, we want to place a pause operation as the last step, so that the microprocessor does not need to run through all 99 steps to execute the program. In order to create this pause, we can use either instruction set 'Input to Accumulator' or 'Input to Register'. Either instruction set will allow the microprocessor to pause at the last step of our program. When this occurs, a question mark appears in the upper left corner of the screen and program execution is halted until a key is depressed. When a key is depressed, its library value is stored in the accumulator and the question mark disappears.

Let's choose the instruction 'Input to Accumulator' as our pause operation. The Mnemonic is INA and the Op Code is Ø4.

Now let's write a short program using the three instruction sets we have just studied:
1. Load a Value into a Register
2. Output from a Register
3. Input to Accumulator
and a symbol from the symbol sheet of Fig. 32. Let's select symbol 3A and we will step through the program. The first step is to insert your Videopac in the machine, and press 'RESET'. This brings the word 'COMMAND' to the screen and you are ready to begin. Since we are going to be programming, we press 'P' for Program and 'M' for Hex input, since we are going to enter our program in hex language. We then press 'I' for Input and the program step number ØØ appears.

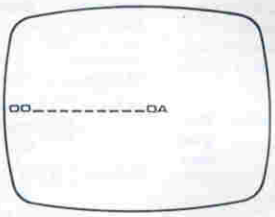


Figure 26

REGISTERS									
MSB							LSB		
0	0	0	1	1	0	0	0	0	=HEX 18=LETTER Q
								1	
								2	
								3	
								4	
								5	
								6	
								7	
								8	
								9	
								A	
0	0	0	0	0	1	0	1	B	=HEX 05
								C	
								D	
								E	
								F	

Figure 27

You are now ready to enter the program. Referring to the program below, you are now at Instruction Step number 4, please proceed from there

Instruction Step	Explanation	Program Step
1 Press RESET A	Command	
2 Press P	Program	
3 Press M	Hex Input	
4 Press 1	Input Program Step 00 appears on screen	ØØ
5 Press 6	Load a Value into...	
6 Press O	Register Ø	
7 Press ENTER	Enter	
8 Press 3	Value loaded into Register Ø is...	Ø1
9 Press A	3A	
10 Press ENTER	Enter	
11 Press 6	Load a value into...	Ø2
12 Press B	Register B	
13 Press ENTER	Enter	
14 Press O	Value loaded into Register B is...	Ø3
15 Press O	ØØ (furthest eft position on screen)	
16 Press ENTER	Enter	
17 Press C	Output Register...	Ø4
18 Press O	Ø	
19 Press ENTER	Enter	
20 Press O	Input to...	Ø5
21 Press 4	Accumulato	
22 Press ENTER	Enter	
23 Press RESET A	Command	
24 Press E	Execute	

If every instruction step was entered correctly, your television screen should look like Figure 28

Let's now move the figure of the man to a different position on the television screen. Let's pick the furthest right position, ØA. Remember, that it is Register B contents we must change. For the moment, step through the program again and change step 15 to A. Your screen should now look like Figure 29.

Do you understand what has happened? Remember the instruction, 'Load a Value into a Register' (LDV,R:NN, or in Op Code, 6RNN)? Well, look at your program. You will note that at instruction step number 11, we pressed 6 and, then at instruction step number 12, we pressed B, then pressed Enter. At that point, we had instructed the microproccessor to load a value into register B (6B). In instruction steps 14 and 15, initially, we had loaded the value ØØ into the microprocessor. It then knew that whatever was displayed on the screen (i.e., the man figure which we loaded into register Ø in instruction steps 5 to 9) would be displayed at the position ØØ, which is the furthest left position. We then changed the value of register B from ØØ to ØA, and the position of the man changed to the furthest right position.

Now, since you understand what has been done above and how to change the contents of a register from one value to anther, we will show you a simpler way to do it without having to re-enter the whole program. At this point, we have our program entered and our man displayed at position ØA on the screen. Follow the steps below and then we will discuss what you have done.

Instruction Step	Explanation
1 Press RESET A	Command
2 Press P	Program
3 Press M	Hex Input
4 Press R	Roll
5 Press U	Program counter will go from step ØØ to Ø1 and display on the screen 3A
6 Press U	Program counter at 02 with the value 6B on the screen.
7 Press U	Program counter at 03 with value ØA on screen. This is the program step we wish to change from ØA to Ø5
8 Press CLEAR	Program
9 Press M	Hex Input
10 Press I	Input – That is, input new data.
11 Press O	Zero (Ø)
12 Press 5	Five (5)
13 Press ENTER	Enter
14 Press RESET A	Command
15 Press E	Execute

Your screen should now look like Figure 30.

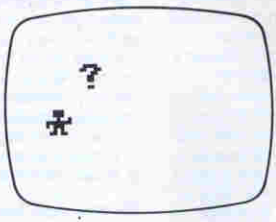


Figure 28

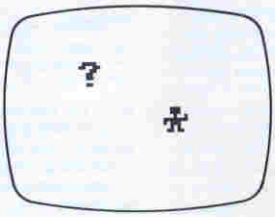


Figure 29

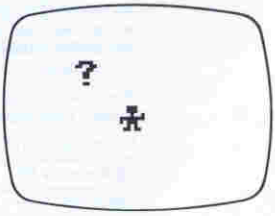


Figure 30

The microprocessor is equipped with a Roll operation, so that once a program is entered, if the programmer decides to change a program step, he may do so without having to re-enter the complete program. We entered the Roll mode from the Hex input mode, then pressed 'U' in order to roll up (we could have pressed D to roll in reverse). We rolled to the step we wished to change (Ø3), then pressed 'CLEAR' and all data at that program step was erased. We then had to return to the Hex Input mode, so we pressed 'M' then 'I' for input and then entered Ø5, our new data. Note that the program step did not change from Ø3 until we had entered Ø5. It then rolled to program step Ø4.

Let's discuss three more instruction sets and write a program using them. Our program will be to place symbols on the television screen in all eleven positions, using the following instruction sets:
1. Input to Accumulator
2. Output from Accumulator
3. Branch Unconditionally

We have used 'Input to Accumulator' previously as a pause operation. We now wish to use it to input data from the keyboard. The codes for 'Input to Accumulator' are:

Mnemonic	INA
Op Code	Ø4
Operation	IP → A

When this instruction has been programmed, a question mark appears in the upper left portion of the screen. Program execution is halted until a key has been depressed on the keyboard. When a key is depressed, its library value is stored in the accumulator and the question mark disappears. In order to display the contents of the accumulator on the television screen, we must program the instruction 'Output from Accumulator'.

The codes for the instruction 'Output from Accumulator' are:

Mnemonic	OTA
Op Code	ØB
Operation	A → OUT

Once programmed, this instruction tells the microprocessor to output to the screen the contents of the accumulator. This output of data will be displayed in whatever position register B is currently set. Remember that register B will automatically advance to the next position right when data is displayed on the screen, and that if Register B is in position ØA (furthest right position), it will display the data and roll back to position ØØ (furthest left position).

Since we wish to place a symbol in each of the eleven positions on the television screen, we need to instruct the microprocessor to repeat the instructions 'Input to Accumulator' and 'Output from Accumulator', over and over again. Thus, we use the instruction set known as 'Branch Unconditionally'. This instruction causes the microprocessor to return to a designated program step and continue from there. The codes for this instruction are

Mnemonic	GTO.NN
Op Code	12NN
Operation	GTO → PC = NN

For example, if we wished to branch to a certain program step, we would write: GTO.ØØ or 12ØØ, which means branch to step ØØ.

Using these three instruction sets, let's enter the program below and see them in action:

Instruction Step	Explanation	Program Step
1 Press RESET A	Command	
2 Press P	Program	
3 Press M	Hex Input	
4 Press I	Input	ØØ
5 Press O	Op code for Input to Accumulator	
6 Press 4	Read keyboard	
7 Press ENTER	Enter	
8 Press O	Op code for Output from Accumulator	Ø1
9 Press B	Output key to screen	
10 Press ENTER	Enter	
11 Press 1	Op code for GTO	Ø2
12 Press 2		
13 Press ENTER	Enter	
14 Press O	Indicates program step to branch	Ø3
15 Press O	Program step is ØØ	
16 Press ENTER	Enter	
17 Press RESET A	Sets program counter to ØØ	
18 Press E	Executes program - A question mark appears on screen; the computer is awaiting input. Program is now ready.	
19 Press O	The zero symbol appears in any one of the spaces shown in Figure 14. This is because we did not initialize register B to ØØ. Keep pressing Ø until you are in the furthest left position on the screen.	
20 Press 1	The 1 symbol appears to the right of the zero.	
21 Press 2	The 2 symbol appears to the right of the 1.	
22 Press 3	Etc.	
23 Press 4		
24 Press 5		
25 Press 6		
26 Press 7		
27 Press 8		
28 Press 9		
29 Press A		

Your screen should now look like Figure 31.

You should have noticed two things as you executed your program: 1) There are only eleven positions on the screen; 2) Any symbol entered after the ØA (furthest right position) starts in the furthest left position and rewrites over the old data. Go ahead and play with the keyboard for a time. You can type your own messages!

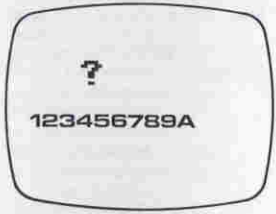


Figure 31

Figure 32

The Complete Instruction Sets

Add Accumulator to Register

Mnemonic	ADD.R
Op Code	ER
Operation	(A = R + A)

Use:
To add the contents of a specified register (R) to the contents of the accumulator and to store the result in the accumulator. If the result is larger than two digits, only the lowest two digits will be kept.
Example:
Accum. = Ø9, Reg. 7 = Ø5 → ADD.7 → Accum. = 14, Reg. 7 = Ø5
Accum. = 90, Reg. 7 = 15 → ADD.7 → Accum. = Ø5, Reg. 7 = 15

Branch on Decimal Borrow

Mnemonic	BDB.NN
Op Code	10NN
Operation	(A_H = 9) → PC = NN

Use:
To instruct the microprocessor to branch to the specified program step (NN) if the high order digit of the accumulator is a '9'.
Example:
Accum. = 95 → BDB.99 → Branch to step 99
Accum. = Ø5 → BDB.99 → No branch is taken

Branch on Decimal Carry

Mnemonic	BDC.NN
Op Code	11NN
Operation	($A_H \neq 0$) → PC = NN

Use:
To instruct the microprocessor to branch to the specified program step (NN) if the high order digit of the accumulator is not a 'Ø'.
Example:
Accum. = 15 → BDC.99 → Branch to step 99
Accum. = Ø5 → BDC.99 → No branch taken

Branch if Register equals Accumulator

Mnemonic	BEQ.R.NN
Op Code	3RNN
Operation	(R = A) → PC = NN

Use:
To instruct microprocessor to branch to a specified program step (NN) if the contents of the accumulator are equal to the contents of the specified register (R). (See sample programs 'One Digit Multiplication' and 'Six Letter Guess' for examples.)
Example:
Accum. = Ø5, Reg. B = Ø5 → BEQ.B.99 → Branch to step 99
Accum. = Ø9, Reg. B = Ø5 → BEQ.B.99 → No branch taken

Branch if Register is greater than Accumulator

Mnemonic	BGT.R.NN
Op Code	4RNN
Operation	(R > A) → PC = NN

Use:
To instruct the microprocessor to branch to a specified program step (NN) if the specified register (R) is greater than the accumulator.
Example:
Accum. = Ø4, Reg. A = Ø5 → BGT.A.99 → Branch to step 99
Accum. = Ø5, Reg. A = Ø4 → BGT.A.99 → No branch taken
Accum. = Ø4, Reg. A = Ø4 → BGT.A.99 → No branch taken

Branch if Register is less than Accumulator

Mnemonic	BLS.R.NN
Op Code	5RNN
Operation	(R < A) → PC = NN

Use:
To instruct the microprocessor to branch to a specified program step (NN) if the specified register (R) is less than accumulator. (See sample program 'One Digit Division' for example.)
Example:
Accum. = Ø5, Reg. 8 = Ø4 → BLS.8.99 → Branch to step 99
Accum. = Ø4, Reg. 8 = Ø5 → BLS.8.99 → No branch taken
Accum. = Ø5, Reg. 8 = Ø5 → BLS.8.99 → No branch taken

Branch if Register not equal to Accumulator

Mnemonic	BNE.R.NN
Op Code	2RNN
Operation	(R ≠ A) → PC = NN

Use:
To instruct microprocessor to branch to a specified program step (NN) if accumulator is not equal to a specified register (R). (For example, see sample program 'Message'.)
Example:
Accum. = Ø9, Reg. D = Ø5 → BNE.D.99 → Branch to step 99
Accum. = Ø5, Reg. D = Ø5 → BNE.D.99 → No branch taken

Branch if Accumulator equals zero

Mnemonic	BRZ NN
Op Code	13NN
Operation	(A = Ø) → PC = NN

Use:
To instruct microprocessor to move to another program step if conditions are satisfied. Most often used in arithmetic problems. (See sample program 'One Digit Division' for example.)
* Post Program:
If the accumulator is zero, a branch to the specified program step (NN) is taken. If the accumulator is not zero, no branch takes place and the program moves to the next step.
Example:
Accum. = ØØ → BRZ 99 → Branch to step 99
Accum. = 65 → BRZ 99 → No branch takes place.
* **Note:** Post program means 'after the instruction set has been programmed' and explains what is happening on the screen and/or in the Videopac Computer.

Set Accumulator to zero

Mnemonic	CLR
Op Code	Ø1
Operation	(A = Ø)

Use:
To clear accumulator and set its contents to Ø.
Example: CLR (Ø1 in Op Code) is programmed and the accumulator = Ø.

Subtract one from Accumulator

Mnemonic	DEC
Op Code	Ø2
Operation	(A = A-1)

Use:
To decrement the contents of the accumulator by one.
Example:
Accum. = 10 → program DEC → Accum. = Ø9

Branch unconditionally

Mnemonic	GTO NN
Op Code	12NN
Operation	GTO → PC = NN

Use:
To instruct the microprocessor to branch to a specified program step (NN). (See sample program 'Message' for example.)
Example:
GTO 99 → Branch to step 99
GTO 34 → Branch to step 34

*** Go to subroutine**

Mnemonic	GTS NN
Op Code	14NN
Operation	GTS → PC = NN

Use:
To instruct microprocessor to branch to a specified program step (NN) which contains an operation which you may wish to use several times in one program. This instruction set allows you to use the same operation several times without having to rewrite it. The next sequential step number is saved for returning from the subroutine. (See sample program 'Area Problems Using Subroutine and Return' for example.)
Example:
Step # 40 = GTS 90 → Branch to step 90
Step # 90 = RET → Branch to step 42
* **Note:** You must have a 'Return from Subroutine' when you have a 'Go to Subroutine'.

Halt Program execution

Mnemonic	HLT
Op Code	FF
Operation	HLT = FF

Use:
To halt execution of program in order to enter a different operational mode to check registers. Used for troubleshooting. The halt instruction is entered after your program is entered. In other words, you would enter your complete program, then, using the Roll mode, you would enter a halt instruction (FF) in place of an instruction already programmed. After entering the Display mode and checking the registers for errors, you would return to the program step containing FF, clear it, and re-enter the program step you had removed.

Input to Accumulator

Mnemonic	INA
Op Code	Ø4
Operation	Ip → A

Use:
To input data from the keyboard (symbol or numeral) into the accumulator.
Post Program:
A question mark appears in the upper left portion of the screen and program execution is halted until a key is depressed. When

depressed, the library value of the key is stored in the accumulator and the question mark disappears. Only one keyboard depression is required. Ip → A means 'input from the keyboard to the accumulator'.
Example:
INA (Ø4 in Op Code) has been programmed. A question mark appears on the screen. You depress key '7', an Ø7 is stored in the accumulator.
INA → depress 7 → Accumulator = Ø7

Add one to Accumulator

Mnemonic	INC
Op Code	Ø3
Operation	(A = A + 1)

Use:
To increment the contents of the accumulator by one.
Example:
Accum. = Ø9 → program INC → Accum. = 10

Input to Register

Mnemonic	INP.R
Op Code	7R
Operation	Ip → R

Use:
To store a value from the keyboard (symbol or numeral) in a specified register.
Post program:
A question mark appears in the upper left portion of the screen and program execution is halted until a key has been depressed on the keybord. When depressed, the library value of the key is stored in the register you have chosen and the question mark disappears. Only one keyboard depression is required. Ip → R means 'input from the keyboard to the register'.
Example:
INP.3 (73 in Op Code) has been programmed. A question mark appears on the screen. You depress key 'X', register 3 stores the hex code (22) for the key 'X'.
INP.B → depress key '4' → Register B stores Ø4 and register B (the positioning register) is positioned at Ø4.

Load Accumulator from Register

Mnemonic	LDA R
Op Code	9R
Operation	R → A

Use:
To load the accumulator with the contents of a specified register.
Example:
Accum. = Ø9, Reg. B = Ø5 → LDA.B → Accum. = Ø5, Reg. B = Ø5
Accum. = 34, Reg. 3 = 12 → LDA.3 → Accum. = 12, Reg. 3 = 12

Load a value into a Register

Mnemonic	LDV R,NN
Op Code	6RNN
Operation	R = NN

Use:
To load a value (NN) into a specified register (R).
Example:
LDV.B.Ø4 (6BØ4 in Op Code) - Register B has been set at Ø4 position and any initial output will be displayed at that position. B = Ø4.
LDV.7.3A (673A in Op Code) - The symbol represented by 3A has been loaded into register 7. Register 7 = 3A.

Load Accumulator from Program step

Mnemonic	MOV
Op Code	Ø9
Operation	R_C → PC → A

Use:
To load accumulator with the contents (two digit value) contained in the program step specified by register C. Rc → PC → A means 'load the contents of register C into the program counter, then load the data contained at that program step into the accumulator'.
Exampler:
Step # Ø6 = FF, Reg. C = Ø6 → MOV → Accum. = FF, Reg. C = Ø6
Note: When using the MOV instruction in a program, register C must remain empty. In other words, you should not program any value in register C.

No operation

Mnemonic	NOP
Op Code	ØØ
Operation	NO = ØØ

Use:
To implement a delay in execution of the program. Can be used when writing a program to utilize several program steps, so that when checking the program, if an extra instruction step is needed, several will be vacant.
Post Program:
A delay is caused within the program execution.

Output from Accumulator

Mnemonic	OTA
Op Code	ØB
Operation	A → OUT

Use:
To display data stored in the accumulator on the television screen.
Post program:
The contents of the accumulator are displayed on the screen in whatever position register B is set. A → OUT means 'output data from accumulator to screen'.
Example:
OTA (ØB in Op Code) is programmed. Referring to our previous example (input to accumulator), the accumulator contains Ø7, thus Ø7 is displayed on the screen.
OTA (ØB in Op Code) - Ø7 is displayed in position set by register B.

Output from Register

Mnemonic	OUT R
Op Code	CR
Operation	R → OUT

Use:
To display the contents of a specified register on the television screen.
Post Program:
The library value stored in a specified register is displayed on the television screen in whatever position register B is set. R → OUT means 'the contents of a specified register are being displayed on the screen'.

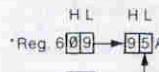
Example:
OUT.3 (C3 in Op Code) has been programmed. Referring to our example above, register 3 contains the hex code for the letter 'X' (22), thus an 'X' is displayed on the screen.
OUT.B (CB) → Referring to our example above, a symbol ('X') is displayed in position Ø4 on the screen.
Note: If you have a series of output instructions, one right after another, you must place a 'No Operation' instruction after every third output instuction. For example:
OUT.1
OUT.2
OUT.3
NOP
OUT.4

Combine two digits

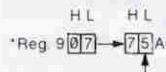
Mnemonic	PAK.R
Op Code	8R
Operation	R_L - - - A_H R_L + 1 - - - A_L

Use:
To combine two digits from two specified registers in the accumulator. This instruction is used when working with numbers. Since the microprocessor reads the numbers in hex, we must instruct it to combine two digits in order to produce and display a base 10 number.
Post Program:
Th low order bit of a specified register (R) is loaded into the high order bit of the accumulator and the low order bit of the next register (R + 1) is loaded into the low order bit of the accumulator.

Example:
Reg. 6 = Ø9, Reg. 7 = 25 →
PAK.6 → Accum. = 95



Reg. 9 = Ø7, Reg. A = 35 →
PAK.9 → Accum. = 75



* **Note:** The high order bit of the first register must always be a zero (Ø).

*** Return from subroutine**

Mnemonic	RET
Op Code	Ø7
Operation	RET → PC = NN

Use:
To instruct the microprocessor to return to the specified program step (NN). This would be the step immediately following the instruction set 'Go to Subroutine'. (See sample program 'Addition Flash Cards (Guess Answer)' for example.)
Example:
Step # 40 = GTS 90 → Branch to step 90
Step # 90 = RET → Branch to step 42
* **Note:** You must have a 'Go to Subroutine' in order to have a 'Return from Subroutine'.

Load Accumulator with random number

Mnemonic	RND
Op Code	Ø8
Operation	RND → A

Use:
To load accumulator with a random number.
Post Program:
The accumulator selects a random number from ØØ to 99.
Example:
Accum. = 10 → program RND → Accum. = any number from ØØ to 99

One second buzz

Mnemonic	SIG
Op Code	Ø5
Operation	BUZ = 1

Use:
To implement a one second buzz.
Example:
Program SIG (Ø5 in Op Code) → a buzz is heard for one second.

Store Accumulator in Register

Mnemonic	STO R
Op Code	AR
Operation	A → R

Use:
To store contents of the accumulator in a specified register.
Example:
Accum. = 66 → STO 3 → Reg. 3 = 66
Accum. = 15 → STO Ø → Reg. Ø = 15

Subtract Accumulator from Register

Mnemonic	SUB R
Op Code	DR
Operation	(A = R-A)

Use:
To subtract the contents of the accumulator from a specified register and store the results in the accumulator.

Post Program:
The contents of the accumulator are subtracted from the contents of a specified register (R) and the result is stored in the accumulator. If the accumulator's value is greater than the contents of the specified register, the register is assumed to be its contents plus 100.

Example:
If the register is larger than the accumulator:
Accum. = Ø5 Reg. 7 = Ø9 → SUB 7 → Accum. = Ø4, Reg. 7 = Ø9
Accum. = 15, Reg.7 = 90 → SUB 7 → Accum. = 75, Reg. 7 = 90
Example:
If the register is smaller than the accumulator;
Accum. = Ø1, Reg.7 = ØØ → SUB 7 → Accum. = 99, Reg. 7 = ØØ

Note: 0-1 = -1, however, according to the above statement, we have 100 -1 = 99.

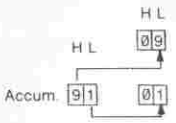
Separate two digits

Mnemonic	UNP R
Op Code	BR
Operation	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$

Use:
To seperate two digits in the accumulator and store them in two specified registers.
Post Program:
The high order bit of the accumulator is loaded into the low order bit position of the specified register (R). The low order bit of the accumulator is loaded into the low order bit position of the next register (R + 1)

Example:
Accum. = 23 → UNP 4 → Reg. 4 = Ø2, Reg. 5 = Ø3

Accum. = 91 → UNP D → Reg. D = Ø9, Reg. E = Ø1



Description	Mnemonic	Op Code	Operation	No. of Bytes	Remarks
Input					
Input to Register	INP.R	7R	Ip → R	1	1 key depression only
Input to Accumulator	INA	Ø4	Ip → A	1	1 key depression only
Output					
Output from Register	OUT.R	CR	R → OUT	1	Reg. B sets position
Output from Accumulator	OTA	ØB	A → OUT	1	of output to screen
One second Buzz	SIG	Ø5	BUZ = 1	1	
Change Accumulator contents mathematics					
Set to Ø	CLR	Ø1	(A = Ø)	1	Accum. = Hex ØØ
Subtract 1	DEC	Ø2	(A = A-1)	1	Decrement by 1
Add 1	INC	Ø3	(A = A + 1)	1	Increment by 1
Load with Random No.	RND	Ø8	RND → A	1	
Load from Storage	MOV	Ø9	R_C → PC → A	1	Reg. C points to step # where data is stored. That data will then be moved to accumulator.
Combine 2 digits	PAK.R	8R	R_L → A_H R_L + 1 → A_L	1	R_L = Reg. low order bit A_H = Accum. high order bit
Separate 2 digits	UNP.R	BR	A_H → R_L A_L → R_L + 1	1	**Note:** If R_L = Reg. 4 then R_L + 1 = Reg. 5
Load from Register	LDA.R	9R	R → A	1	
Subtract from Reg.	SUB.R	DR	(A = R − A)	1	
Add Register	ADD.R	ER	(A = R + A)	1	
Change Register Contents					
Store Accumulator	STO.R	AR	A → R	1	
Load a Value	LDV.R.NN	6RNN	R = NN	2	Load R with value NN

Description	Mnemonic	Op Code	Operation	No. of Bytes	Remarks
Control execution order					
No Operation	NOP	ØØ	NO = ØØ	1	
Halt	HLT	FF	HLT = FF	1	
Go to Subroutine	GTS.NN	14NN	GTS → PC = NN	2	
Return from Subrout.	RET	Ø7	RET → PC = NN	1	
Branching decision					
Branch on Decimal Borrow	BDB.NN	10NN	(A_H = 9) → PC = NN	2	NN = ØØ through 99
Branch on Decimal Carry	BDC.NN	11NN	(A_H ≠ Ø) → PC = NN	2	R = Ø-9, A-F
Branch Unconditionally	GTO.NN	12NN	GTO → PC = NN	2	PC = Program Counter
Branch if Accumulator is Ø	BRZ.NN	13NN	(A = Ø) → PC = NN	2	The program counter points to the step number.
Branch if Reg. ≠ Accumulator	BNE.R.NN	2RNN	(R ≠ A)→ PC = NN	2	
Branch if Reg. = Accumulator	BEQ.R.NN	3RNN	(R = A) → PC = NN	2	
Branch if Reg. > Accumulator	BGT.R.NN	4RNN	(R > A) → PC = NN	2	
Branch if Reg. < Accumulator	BLS.R.NN	5RNN	(R < A) → PC = NN	2	

Videopac Computer Operating Modes

We have learned how to program the Videopac Computer and we have used several different operating modes. You will recall using the COMMAND MODE, the EXECUTION MODE, and the ROLL MODE.

There are eight operating modes in the Videopac Computer. They are:
COMMAND
EXECUTION
DISPLAY
PROGRAM
ASSEMBLER
HEX INPUT
INPUT
ROLL

These operating modes allow you to perform specific functions. Let's look at each in detail and refer to the block diagram in Figure 33 as we explain each mode.

Command Mode

To enter the COMMAND MODE, you may press 'Reset' or 'Clear' if you are in any of the following modes:
ASSEMBLER
DISPLAY
HEX INPUT
If you are in the EXECUTION MODE, to enter the COMMAND MODE, press 'Reset'.
Once in the COMMAND MODE, you may enter the following modes:
EXECUTION
DISPLAY
PROGRAM
by pressing:
E – To enter the EXECUTION MODE. Program execution will begin with step ØØ. You are ready to play your game, write your message, solve your problem, etc. if you have already entered your program.
or by pressing
C – To enter the CONTINUE MODE. This mode is used to locate a problem within a register.

For example, let's assume we have a 40 step program that is not working correctly. A branch decision was made at some lower step number and we would like to see if the correct branch was taken to the step number we had indicated, say step 14. At step 14 we would replace the op code at that step number with a halt statement (op code FF). To examine the contents of the program counter (which would contain the program step 14, thus informing us of the correct branch), we would press 'D' to enter the DISPLAY MODE. We would then press 'P' to display the contents of the program counter. If the correct result is displayed, we would press 'Clear' which returns us to the COMMAND MODE. We now press 'P' to enter the PROGRAM MODE, then press 'M' to enter the HEX INPUT MODE, and then press 'R' to enter the ROLL MODE. We now must roll up ('U') from step ØØ to step 14 where the FF statement is located and replace it with the original op code. We may now place an FF statement at some other step to check another part of the program. Please note that only one FF statement at a time can be present in the program.
or by pressing
D – To enter the DISPLAY MODE (to be explained in detail later).
or by pressing
P – To enter the PROGRAM MODE (to be explained in detail later).

To leave the COMMAND MODE, you may turn the Videopac

Computer off* or enter another operating mode (i.e., C, D, E, or P).

* **Note:** All programming is erased when the power is turned off or the Videopac is removed from the machine.

Display Mode

To enter the DISPLAY MODE, you must press 'D' from the COMMAND MODE. In this mode, you may display on the screen any register you wish to review. This mode is often used to troubleshoot problems, since you can check the contents of each register.

To check the registers, you press:

Ø-To display the contents of register Ø
1-To display the contents of register 1
2-To display the contents of register 2
3-To display the contents of register 3
4-To display the contents of register 4
5-To display the contents of register 5
6-To display the contents of register 6
7-To display the contents of register 7
8-To display the contents of register 8
9-To display the contents of register 9
A-To display the contents of register A
B-To display the contents of register B
C-To display the contents of register C
D-To display the contents of register D
E-To display the contents of register E
F-To display the contents of register F
P-To display the contents of the Program Counter
S-To display the contents of the Subroutine Counter
X-To display the contents of the Accumulator

To leave the DISPLAY MODE, press 'Clear' or 'Reset' to enter the COMMAND MODE.

Program Mode

To enter the PROGRAM MODE, press 'P' from the COMMAND MODE, or press 'Clear' if you are in the ROLL MODE.

The PROGRAM MODE sets the Videopac Computer to accept a program. From this mode, you may press 'A' to enter Assembler language (mnemonic), or you may press 'M' to enter hex language (Op Code).

To leave the PROGRAM MODE, press 'Reset' and you will enter the COMMAND MODE, or you may leave the PROGRAM MODE by pressing 'A' to enter the ASSEMBLER MODE or by pressing 'M' to enter the HEX INPUT MODE.

Assembler Mode

To enter the ASSEMBLER MODE, press 'A', if you are in the PROGRAM MODE, or press 'Clear' if you are already in the INPUT MODE for assembler language.

Once you have pressed 'A', you are in the ASSEMBLER MODE and you may now press 'I' to enter the INPUT MODE for assembler language, or you may press 'R' to enter the ROLL MODE.

To leave the ASSEMBLER MODE, press 'Clear' or 'Reset' to enter the COMMAND MODE, or press any valid ASSEMBLER MODE command (i.e. 'I' or 'R'). (Be sure to refer to Figure 33).

Hex Input Mode

To enter the HEX INPUT MODE, press 'M' if you are in the PROGRAM MODE or press 'Clear' if you are in the INPUT MODE for machine language.

Once in the HEX INPUT MODE, you may press 'I' to enter the INPUT MODE for machine language (Op Code), or press 'R' to enter the ROLL MODE.

To leave HEX INPUT MODE, press 'Clear' or 'Reset' to enter COMMAND MODE or press any valid HEX INPUT MODE command (i.e., 'I' or 'R').

Input Mode

To enter the INPUT MODE, press 'I', if you are in either ASSEMBLER or HEX INPUT MODE.

Once you are in the INPUT MODE, you may enter any assembler language instruction (Mnemonic) if you have entered from the ASSEMBLER MODE, or you may enter any machine language instruction (Op Code) if you have entered from the HEX INPUT MODE. This is the mode in which you will enter your program.

To leave the INPUT MODE, you may press 'Reset' to enter the COMMAND MODE, or press 'Clear' to enter the ASSEMBLER or HEX INPUT MODE.

Roll Mode

To enter the ROLL MODE, press 'R' if you are in either ASSEMBLER or HEX INPUT MODE.

Once you are in the ROLL MODE, you may press 'U' to display the program steps from ØØ to 99, or you may press 'D' to display the program steps from 99 to ØØ. This mode is often used to check a program step to be sure it contains the correct data.

To leave the ROLL MODE, press 'Clear' to enter ASSEMBLER or HEX INPUT MODE.

Now that we have studied the various operation modes, let's enter a program using the various modes. Review flow diagram 1 before entering the program below.

Instruction Step	Explanation	Program Step	Illustration
1 Press RESET Δ	'Command' appears on the screen.		
2 Press P	'Program' appears on the screen.		Operational Modes (Command, Program, Hex Input)
3 Press M	'Hex Input' appears on the screen.		
4 Press I	Step ØØ appears on the left side of the screen.	ØØ	
5 Press O 6 Press 4 7 Press ENTER	Ø4 is the Op Code for 'Input to Accumulator'. Enter. Step Ø1 appears on screen.	Ø1	Input to Accumulator
8 Press 7 9 Press 3 10 Press ENTER	73 is the Op Code for 'Input to Register' with 3 meaning register 3. Enter. Step Ø2 appears on screen.	Ø2	Input to Register 3
11 Press 5 12 Press 3 13. Press ENTER	53 is Op Code for 'Branch if Register is less than Accumulator'; 3 = Register. Enter. Step Ø3 appears on screen.	Ø3	Branch if R 3 < Accum No Yes
14 Press O 15 Press 9	If contents of register 3 are less then the contents of accumulator, go to step Ø9.		Go to Program step Ø9

Instruction Step	Explanation	Program Step	Illustration
16 Press ENTER	Enter. Step Ø4 appears on screen.	Ø4	
17 Press F 18 Press F	FF is Op Code for 'Halt'. If register 3 contents are greater than Accumulator, program counter will exit at step Ø4.		FF Statement - Exit at Step Ø4 if R3 > Accumulator
19 Press ENTER	Enter. Step Ø5 appears on screen.	Ø5	
			Command Mode
20 Press O 21 Press O	ØØ is Op Code for 'No Operation'. Used to allow computer to move through program steps without performing any operations.		
22 Press ENTER	Enter. Step Ø6 appears.	Ø6	
23 Press O 24 Press O	ØØ		No Operation
25 Press ENTER	Enter. Step Ø7 appears.	Ø7	
26 Press O 27 Press O	ØØ		
28 Press ENTER	Enter. Step Ø8 appears.	Ø8	

Instruction Step	Explanation	Program Step	Illustration
29 Press O 30 Press O	Ø0		
31 Press ENTER	Enter. Step Ø9 appears.	Ø9	
32 Press F 33 Press F	FF is Op Code for 'Halt'. If reg 3 contents are less than accum., program counter will exit at step Ø9.		FF Statement Exit at step Ø9
			Command Mode
34 Press ENTER	Enter. Step 1Ø appears.	1Ø	
35 Press O 36 Press 4	Ø4 is Op Code for 'Input to Accumulator'.		Input to Accumulator
37 Press ENTER	Enter. Step 11 appears.	11	
38 Press O 39 Press B	ØB is Op Code for 'Output from Accumulator'. In other words, whatever was in Accumulator will be displayed on screen.		Output from Accumulator
40 Press ENTER	Enter. Step 12 appears.	12	
41 Press O 42 Press 4	Ø4 is Op Code for 'Input to Accumulator'. This will be used simply to allow the output from step 11 to be displayed on screen (used as a pause operation).		Input to Accumulator (used as pause)

Instruction Step	Explanation	Program Step	Illustration
43 Press ENTER	Enter. Step 13 appears.	13	
44 Press RESET ▲	Reset. Program is stored and you are back in the Command mode.		

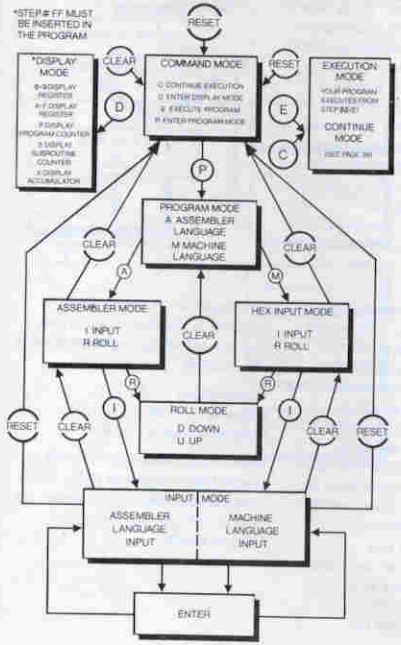
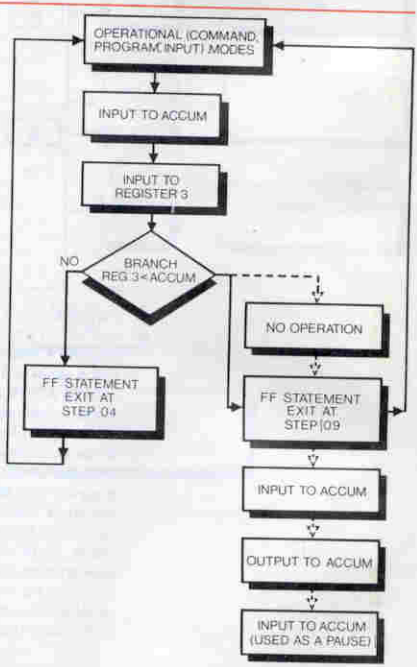


Figure 33



Flow diagram 1

Review flow diagram 2, then inplement the following.

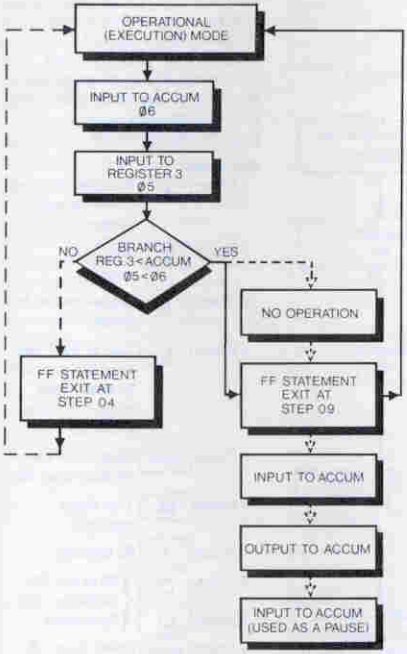
Execution Mode

Instruction Step	Explanation	Program Step	Illustration
1 Press [E]	Execution mode is entered and a question mark appears on the screen.		Operational Mode (Execution)
2 Press [6]	Hex 'Ø6' is now in the accumulator and the question mark is still on the screen.		Input to Accum. Ø6
3 Press [5]	Hex 'Ø5' is now in register 3 and the Computer returns to the Command mode. ('Command' appears on the screen.)		Input to Reg. 3 Ø5 → Command

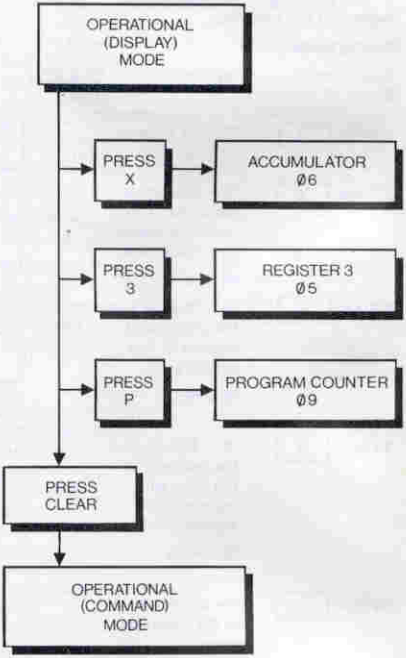
Remember, our program had an instruction which stated 'Branch if register is less than accumulator'. Since register 3 now equals Ø5 and the accumulator equals Ø6, those conditions are satisfied. Our program further states that if those conditions are satisfied, then the Computer should exit at program step Ø9. Program step Ø9 has an FF instruction (halt) which returns us to the Command mode. (Note, nothing has appeared on the screen because the program has halted at a program step prior to the instruction 'Output from the Accumulator'. Refer to flow diagram 2). However, we know that the following registers should still contain:

Accumulator = Ø6
Register 3 = Ø5
Program Counter = Ø9

Now, the question is, 'Can we check these registers to be sure they contain the correct data?' Of course we can, by using the Display mode.



Flow diagram 2



Flow diagram 3

Review flow diagram 3, then implement the following.

Display Mode

Instruction Step	Explanation	Program Step	Illustration
1 Press D	'Display' appears on screen.		Operational Mode (Display)
2 Press X	'X' - This displays the contents of the Accumulator. You should see: Ø6 X REG		Accumulator Ø6
3 Press 3	'3' - This displays the contents of register 3. You should see: Ø5 3 REG		Register Ø5
4 Press P	'P' - This displays the contents of the program counter. You should see: Ø9 P REG		Program Counter Ø9
5 Press CLEAR	We are now back in the Command mode.		Operational Mode (Command)

We know now that all registers contain the appropriate data. We now wish to return to our program and replace the halt instruction (FF) at program step Ø9 with a no operation instruction (ØØ). This must be done by entering the Roll mode.

Roll mode

1 Press P — We have entered the Program mode.

2 Press M — 'Hex Input' appears.

3 Press R — We are in the Roll mode.

4 Press U — The 'U' (up) key must be pressed 9 times. Your screen will then show: Ø9 FF

5 Press CLEAR — Clear. This clears FF from step Ø9.

6 Press M — 'Hex Input' appears.

7 Press I — Ø9 appears.

8 Press O, 9 Press O — ØØ is the Op Code for 'No Operation'.

10 Press ENTER — Enter. Step 10 appears.

11 Press RESET Δ — Reset. 'Command' appears. We are back in the Command mode.

We shall now enter the Execution mode and implement the same program. We will enter Ø6 into the accumulator and Ø5 into register 3. However, after we have entered these values, note that the computer will continue to display a question mark rather than returning to the Command mode. We will know that the computer has branched correctly at program step Ø9. Since that step now contains a no operation instruction (ØØ), it has allowed the Computer to step through the program. (Refer to flow diagram 4). You may now press any key on the keyboard and it will appear on the screen.

Execution Mode

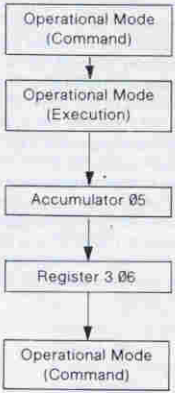
Instruction Step	Explanation	Program Step	Illustration
1 Press E	We have entered the Execution mode and may now implement our program.		Operational Mode (Execution)
2 Press 6	Hex 'Ø6' is in the Accumulator.		Accumulator Ø6
3 Press 5	Hex 'Ø5' is in register 3.		Register 3 Ø5
4 Press W	A 'W' appears on the screen. You could press any key of your choice.		Input to Accum.
			Output Accum.

Notice, in our original program in this chapter, that program step 12 is another 'Input to Accumulator' instruction. We programmed that step simply to allow the output from step 11 (the W) to be displayed on the screen. Remember that program execution is halted after the instruction 'Input to Accumulator', until a key is pressed and additional information is entered into the accumulator. This instruction is a good one to use for troubleshooting, since execution is completely halted.

Branching When Register 3 is Larger than the Accumulator

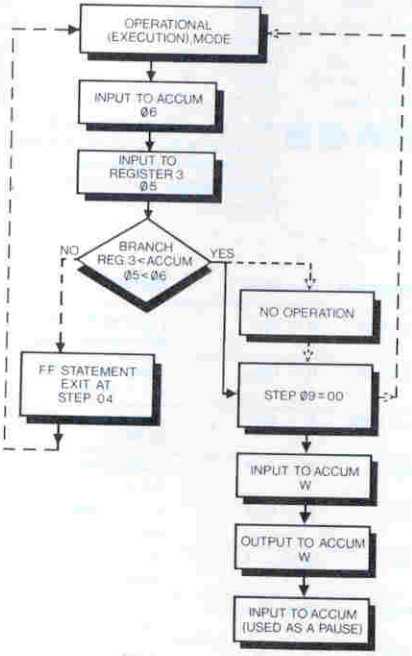
Let's now return to the Command mode, which will erase the values currently in the accumulator and register 3. Then we will enter the Execution mode and enter new values so that register 3 will be larger than the accumulator and branching will take place at program step Ø4. (Refer to flow diagram 5).

Instruction Step	Explanation	Program Step
1 Press RESET [A]	Reset. This returns us to the Command Mode.	
2 Press [E]	Execution mode. A question mark appears on the screen.	
3 Press [5]	Hex '05' is now in the Accumulator and the question mark remains on the screen.	
4 Press [6]	Hex '06' is now in register 3 and we return to the Command mode.	

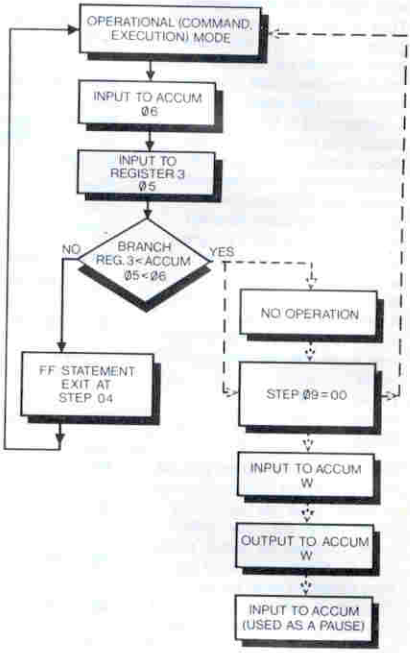


Now, since the Accumulator equals Ø5 and register 3 equals Ø6, the program will be halted at program step Ø4. The register should contain the following:
Accumulator = Ø5
Register 3 = Ø6
Program Counter = Ø4

It is left as an exercise for you to display these registers on the screen to be sure they contain the correct data. If you are not sure, go back to the Display mode instructions for flow diagram 4.



Flow diagram 4



Flow diagram 5

Assembler Mode Programming

Thus far, we have done all our programming in the Hex Input mode, though we have discussed the Assembler mode. You will remember that when programming in the Hex Input mode, you have used the Op Code symbols and that when programming in the Assembler mode, you must use the Mnemonic symbols. As a last exercise, let's enter a program called 'Creepy Crawler' (as written at the end of this chapter), first in Hex (Op Code) and then in Assembler (Mnemonic).

First, the Hex Input. Remember how to get the Hex Input mode?

Press RESET
Press 'P'
Press 'M'
Press 'I'

You are now at program step ØØ. Referring to the Creepy Crawler program, enter the hex values shown under Op Code. Don't forget to press the 'Enter' button after each byte of data. (If you have forgotten what a byte consists of, refer to chapter 4). If you have entered the program correctly, your screen should look similar to Figure 34.

You are now in the Assembler mode at program step ØØ and ready to enter the program in Mnemonics. Since we have not used this language before, we will step through the first program step with you, using the Mnemonic column of the program.

Press L
Press D
Press V
Press .
Press Ø
Press .
Press 2
Press F
Press Enter

We are now at program step Ø2. Continue in the same way and you should see on your screen a picture similiar to the one in Figure 34.

Creepy Crawler

Step	Opcode	Mnemonics	Operation	Remarks
ØØ	6Ø 2F	LDV Ø,2F	R = NN	Load reg.Ø = 2F; 2F = block figure
Ø2	61 ØC	LDV 1,ØC	R = NN	Load reg.1 = ØC; ØC = blank
Ø4	6B ØØ	LDV B,ØØ	R = NN	Sets output position to ØØ
Ø6	CØ	OUT Ø	R → OUT	Outputs block to screen
Ø7	C1	OUT 1	R → OUT	Outputs blank to screen
Ø8	Ø5	SIG	BUZ = 1	Buzz for one second
Ø9	Ø8	RND	RND → A	Accum. selects random number
1Ø	BB	UNP B	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack reg.B used for output position.
11	12 Ø6	GTO Ø6	GTO → PC = NN	Go to step Ø6 and repeat.

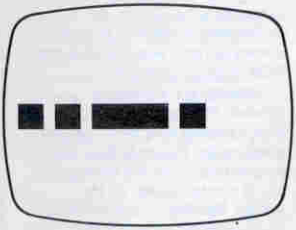


Figure 34

Sample Programs/Conclusion

Before explaining the sample programs which follow, an explanation of the four programs already entered and stored in the ROM of your Computer Programming Cartridge is needed. You may have noticed, after inserting your cartridge and pressing the Power button, that by pressing '1' on the keyboard a set of blocks appear on the screen, flashing in random order and accompanied by a buzz sound. This is 'Creepy Crawler'. It is just a sample of the programs you can enter in your Videopac Computer. The program for 'Creepy Crawler' appears at the and of the last chapter and you may wish to enter this program in your Videopac Computer, substituting another symbol (chosen from the symbols in Figure 32) for the block symbol.

If you press '2' after inserting your cartridge and pressing the Power button, an addition problem appears on the screen without a solution. This is 'Flash Card'. You are to enter the solution. For example, '4 + 2 =' appears on the screen; you press 'Ø8' (remember, you must press 2 digits). Since this is not the correct answer, a 'NO' appears. You then press 'Ø6' and the 'Ø6' appears on the screen. Another problem does not appear on the screen until you press a key.

If you press '3', the 'High-Low Guess' game is implemented. At this point, a question mark appears on the screen and the computer chooses a number from ØØ to 99, but does not display it on the screen. You then press any number between ØØ and 99. Whatever number you enter will be displayed on the screen with an 'L' or an 'H', which tells you that your guess is either lower or higher than the number the computer has chosen. You may keep guessing until you choose the correct answer, which is identified by an 'X' beside the number you have chosen.

When the '4' is pressed, the 'Is Number Between Limits?' game is implemented. Three numbers appear across the screen. For example: Ø3 Ø7 ØØ. The Ø3 represents a low number; the Ø7 represents a high number; and the ØØ represents your score. Again, the computer chooses a number, but does not display it. You must guess if the number is between 3 and 7. If you think it is, you press 'YES'. If you do not think the number is between 3 and 7, then you press 'NO'. If you are correct, the number appears on the screen between the 3 and 7 and your score increments by 1. If you are wrong, the computer will sound a 'beep', your score remaining the same. After each Yes or No guess, press any key and a new problem will appear.

These programs are preprogrammed for your enjoyment to use until you are able to write your own programs.

Following are sample programs which you may enter into the computer for practice. We have attempted to use each instruction set in some way in order to give you a working example of how each can be used. However, it is only from practice and trial and error that you will come to know

how and when to use the instruction sets.

In some instances, a page of explanation is presented before the program. This was done in order to give a more detailed explanation of the use of particular instruction sets and/or the reasons why a particular instruction set was necessary in a certain situation.

Also, included in the Appendix, are blank program pages for use when writing your own programs. Computer programming is not an easy subject to learn; however, it is extremely interesting and a subject which is constantly changing as new advances in computer technology are made. This manual is only a beginning introduction to the basics of computer programming and we hope it has been interesting to you. Through your use of the Videopac Computer and its Computer Programming Videopac, you may develop an interest and desire to research and study the more advanced computer languages and systems.

Good luck!

Sample programs

Add Two 1-Digit Numbers and Display the Sum

The first three programs deal with the same problem, adding two 1-digit numbers and displaying the sum; however the complexity of the program increases each time (A, B, C, respectively); as you will note the program steps also increase.

In Problem A, after the program has been entered and the execution implemented, a question mark appears on the screen. The computer is asking for some data. When the first number is entered, it is not displayed. When the second number is entered, the answer appears on the screen immediately. The problem is not erased until two more digits are entered.

In Problem B, the question mark again appears. The first number is entered and nothing appears on the screen. When the second number is entered, the complete problem appears on the screen, for example: 2 + 4 = 06. You will note, if you look at the program that in this case the '+' and '=' signs had been programmed into the computer so that they would appear on the screen. Again, the problem does not erase from the screen until you enter two new digits.

Problem C begins with a question mark on the screen. The first number entered appears on the screen along with a '+' sign. The second number entered appears on the screen along with an '=' and the answer. Note that when the first digit of the second problem is entered, the first problem disappears from the screen. This occurs because of the following:

1. Program step 16 is a pause operation which allows the problem to remain on the screen until another number is entered.
2. Program step 17 and 18, load register C with ØB which stands for the decimal number 11, which is also the number of positions available on the screen.
3. Program step 19 and 20, register 7 is loaded with blank spaces.
4. Program step 21 loads the accumulator with the decimal number 11 from register C.
5. Program step 22 and 23, load register 4 with ØØ.
6. Program step 24, the accumulator is decremented by 1.
7. Program step 25, a blank space from register 7 is displayed on the screen.
8. Program step 26 and 27, instructs the computer to return to program step 24 if the accumulator is not equal to register 4. In other words, the computer will keep returning to step 24, decrementing by 1, and displaying blanks on the screen, until the accumulator is equal to register 4, which is zero (ØØ).
10. Program step 28 and 29, once the accumulator is equal to register 4, the computer moves on to step 28 and 29 which set register B at ØØ, the furthest left position on the screen.
11. Program step 30, instructs the computer to return to program step Ø3 and repeat the program, thus you are able to enter problem after problem.

Add Two 1-Digit Numbers and Display Sum (Problem A)

Step	Opcode	Mnemonics	Operation	Remarks
ØØ	7Ø	INP.Ø	Ip → R	Input Reg. Ø with 1s number
Ø1	Ø4	INA	Ip → A	Input Accum. with 2nd number
Ø2	EØ	ADD.Ø	(A = R + A)	Add Reg. Ø to Accumulator
Ø3	B1	UNP.1	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack Reg. 1 and Reg. 2 (separate digits)
Ø4	6B ØØ	LDV.B.ØØ	R = NN	Set output position to ØØ
Ø6	C1	OUT.1	R → OUT	Output Reg. 1, 1st digit sum
Ø7	C2	OUT.2	R → OUT	Output Reg. 2, 2nd digit sum
Ø8	12 ØØ	GTO.ØØ	GTO → PC = NN	Go to step ØØ & repeat.

Add Two 1-Digit Numbers and Display Sum (Problem B)

ØØ	7Ø	INP.Ø	Ip → R	Input Reg. Ø with 1st number
Ø1	Ø4	INA	Ip → A	Input Accum with 2nd number
Ø2	6B ØØ	LDV.B.ØØ	R = NN	Set output position
Ø4	CØ	OUT.Ø	R → OUT	Output 1st number from reg. Ø
Ø5	63 1Ø	LDV. 3. 1Ø	R = NN	Load Reg. 3 = 1Ø,. 1Ø = (+) sign
Ø7	C3	OUT.3	R → OUT	Output Reg. 3, + on screen
Ø8	ØB	OTA	A → OUT	Output 2nd number

	Step	Opcode	Mnemonics	Operation	Remarks
	Ø9	63 2B	LDV.3.2B	R = NN	Load Reg. 3 with (=) sign
	11	C3	OUT.3	R → OUT	Output Reg. 3, = on screen
	12	EØ	ADD.Ø	(A = R + A)	Add Reg. Ø to Accumulator
	13	B1	UNP.1	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack Reg. 1 and Reg. 2
	14	C1	OUT.1	R → OUT	Output Reg. 1, 1st digit sum
	15	C2	OUT.2	R → OUT	Output Reg. 2, 2nd digit sum
	16	12 ØØ	GTO.ØØ	GTO → PC = NN	Go to step ØØ and repeat
Add Two 1-Digit Numbers and Display Sum (Problem C)	ØØ	6B ØØ	LDV.B.ØØ	R = NN	Set output position to ØØ
	Ø2	7Ø	INP.Ø	Ip → R	Input 1st number to Reg. Ø
	Ø3	CØ	OUT.Ø	R → OUT	Output Reg. Ø
	Ø4	63 1Ø	LDV.3.1Ø	R = NN	Load Reg. 3 = 10; 10 = (+) sign
	Ø6	C3	OUT.3	R → OUT	Output + sign from Reg. 3
	Ø7	Ø4	INA	Ip → A	Input to Accumulator
	Ø8	ØB	OTA	A → OUT	Output from Accumulator
	Ø9	63 2B	LDV.3.2B	R = NN	Load Reg. 3 with (=) sign
	11	C3	OUT.3	R → OUT	Output = sign from Reg. 3
	12	EØ	ADD.Ø	(A = R + A)	Add Reg. Ø to Accum.
	13	B1	UNP.1	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack Accum. to Reg. 1 and Reg. 2

Step	Opcode	Mnemonics	Operation	Remarks
14	C1	OUT 1	R → OUT	Output Reg. 1
15	C2	OUT 2	R → OUT	Output Reg. 2
16	7Ø	INP.Ø	Ip → R	This is used as a pause operation
17	6C ØB	LDV.C.ØB	R = NN	Load Reg. C with hex ØB (#11)
19	67 ØC	LDV.7.ØC	R = NN	Load Reg. 7 with blank spaces.
21	9C	LDA.C	R → A	Load Accum. from Reg. C
22	64 ØØ	LDV.4.ØØ	R = NN	Load Reg. 4 with ØØ
24	Ø2	DEC	(A = A-1)	Subtract 1 from Accum.
25	C7	OUT 7	R → OUT	Output Reg. 7 (blank spaces)
26	24 24	BNE.4.24	(R ≠ A) → PC = NN	Branch if Reg. 4 ≠ Accum.
28	6B ØØ	LDV.B.ØØ	R = NN	Set output position to ØØ
30	12 Ø3	GTO.Ø3	GTO → PC = NN	Go to step Ø3 and repeat

One Digit Multiplication

Refer to Chapter 2 to refresh your memory as to the method which the Videopac Computer uses to multiply. Then enter the program. Remember, it multiplies by a series of additions which we must program into the computer. Let's look at the steps in detail.

First, after the program is entered, a question mark appears on the screen and the computer is waiting for input. The first digit entered is the multiplicand and appears on the screen with an 'X' sign. The second digit is entered (the multiplier) and the complete problem appears with answer, for example 7 × 3 = 21. You will note that program step ØØ (which positions register B at the furthest left position) through program step 11 are the instructions which allow the problem to be displayed on the screen. It is at program step 12 that the mathematical operations, which must be performed in order to solve the problem, begin. Let's look at these instructions in detail:

1. Program step 12, loads the accumulator from register Ø with the 7 which is the multiplicand.
2. Program step 13, adds the accumulator (Ø7) to register Ø (note register Ø still contains an

Ø7). Remember that the Videopac Computer multiplies by a series of additions. This is the first addition.

3. Program step 14, the sum of the accumulator and register Ø is stored in register 2.

4. Program step 15, the accumulator is loaded from register 1 which holds the multiplier, 3, and this is the number of times we must add 7 in order to arrive at an answer.

5. Program step 16, decrement the accumulator by 1. The accumulator contains the 3 (the number of times we must add 7). We have added 7 twice (Ø7 + Ø7), so we must now decrement the multiplier (3) by 1 so that we can keep track of the number of times we have added.

6. Program step 17, store the difference in register 1. The difference is now 2 and we return that value to register 1. (Note: The multiplicand (Ø7) has been added twice at this point; however, since we can only decrement the accumulator by 1, we have to program the computer to stop adding Ø7 after the third time, so we wish to stop adding when the multiplier equals Ø1 rather than Ø0.)

7. Program step 18 and 19, load register 3 with Ø1. This is done so that when the accumulator contains the contents of register 1 (the multiplier) and they are compared to the contents of register 3 (Ø1, which is when we want the adding operation to stop), the computer will either stop adding (if the accumulator is equal to register 3) or it will loop back to the specified program step (if they are not equal) and continue to add Ø7 again.

8. Program step 20 and 21, instructs the computer to branch to step 25 if the contents of the accumulator and register 3 are equal. At which time the answer will be unpacked and displayed on the screen.

9. Program step 22, load accumulator from register 2, which contains the sum of the addition of Ø7 + Ø7 which is 14.

10. Program step 23 and 24, instruct the computer to return to step 13 and repeat addition.

11. Program step 25, when the accumulator and register 3 are equal (both contain Ø1), the computer steps to program step 25, and the accumulator is loaded from register 2 which contains the sum of the addition operations (21).

12. Program steps 26 through 29, unpack the answer and display it on the screen.

One-Digit Multiplication

Step	Opcode	Mnemonics	Operation	Remarks
Ø0	6B Ø0	LDV B,Ø0	R = NN	Set output position
Ø2	70	INP Ø	Ip → R	Multiplicand to Reg. Ø
Ø3	CØ	OUT Ø	R → OUT	Output Multiplicand
Ø4	66 29	LDV 6,29	R = NN	Reg. 6 = 29; 29 = (X)
Ø6	C6	OUT 6	R → OUT	Output Reg. 6; Reg. 6 = (X)
Ø7	71	INP 1	Ip → R	Multiplier to Reg. 1
Ø8	C1	OUT 1	R → OUT	Output Multiplier
Ø9	67 2B	LDV 7,2B	R = NN	Reg. 7 = 2B; 2B = (=)
11	C7	OUT 7	R → OUT	Output Reg. 7

Step	Opcode	Mnemonics	Operation	Remarks
12	9Ø	LDA.Ø	R → A	Load Accum. from Reg. Ø
13	EØ	ADD.Ø	(A = R + A)	Add Reg. Ø to Accum.
14	A2	STO.2	A → R	Store sum in Reg. 2
15	91	LDA.1	R → A	Load Accum. from Reg. 1
16	Ø2	DEC	(A = A − 1)	Decrement Accum. by 1
17	A1	STO.1	A → R	Store difference in Reg. 1
18	63 Ø1	LDV.3.Ø1	R = NN	Load Reg. 3 with Ø1
20	33 25	BEQ.3.25	(R = A) → PC = NN	(R = A) Go to step 25
22	92	LDA.2	R → A	Load Accum. from Reg. 2
23	12 13	GTO.13	GTO → PC = NN	Go to step 13
25	92	LDA.2	R → A	Load Accum from Reg. 2
26	B4	UNP.4	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Reg. 2 Unpack two digits
27	C4	OUT.4	R → OUT	Output Reg. 4
28	C5	OUT.5	R → OUT	Output Reg. 5
29	12 ØØ	GTO.ØØ	GTO → PC = NN	Go to step ØØ and repeat

One Digit Division

Refer to Chapter 2 to refresh your memory regarding binary division.

Remember, the Videopac Computer divides by a series of subtractions, and the quotient is the number of times the divisor can be subtracted from the dividend. Now, enter the program on the following pages. A question mark appears on the screen after execution is implemented. When the first number (dividend) is entered, it appears on the screen with a '÷' sign. The second number (divisor) is entered and appears along with an '=' sign and the answer. Let's look at the program in detail:

1. Program step ØØ and Ø1, initialization of register 3 to contain ØØ. This register will contain the sum of the subtraction operations the computer performs in order to

One-Digit Division

Step	Opcode	Mnemonics	Operation	Remarks
ØØ	63 ØØ	LDV.3.ØØ	R = NN	Reg. 3 = ØØ (initialization)
Ø2	6B ØØ	LDV.B.ØØ	R = NN	Reg. B = ØØ (positioning)
Ø4	7Ø	INP.Ø	Ip → R	Dividend to Reg. Ø
Ø5	CØ	OUT.Ø	R → OUT	Output Reg. Ø
Ø6	69 2A	LDV.9.2A	R = NN	Reg. 9 = 2A. 2A = (←)
Ø8	C9	OUT.9	B → OUT	Output Reg. 9
Ø9	71	INP.1	Ip → R	Divisor to Reg. 1
10	C1	OUT.1	R → OUT	Output Reg. 1
11	6A 2B	LDV.A.2B	R = NN	Reg. A = 2B; 2B = (=)
13	CA	OUT.A	R → OUT	Output Reg. A
14	91	LDA.1	R → A	Load Accum. from Reg. 1
15	DØ	SUB.Ø	(A = R-A)	Sub. Accum. from Reg. Ø
16	AØ	STO.Ø	A → R	Store difference in Reg. Ø
17	93	LDA.3	R → A	Load Accum. from Reg. 3
18	Ø3	INC	(A = A + 1)	Add 1 to the Accum.
19	A3	STO.3	A → R	Store sum in Reg. 3
20	9Ø	LDA.Ø	R → A	Load Accum. from Reg. Ø
21	13 4Ø	BRZ.4Ø	(A = Ø) → PC = NN	Branch to step 40 if A = Ø
23	91	LDA.1	R → A	Load Accum. from Reg. 1
24	5Ø 28	BLS.Ø.28	(R < A) → PC = NN	Branch to step 28 if R < A
26	12 15	GTO.15	GTO → PC = NN	Go to step 15
28	93	LDA.3	R → A	Load Accum. from Reg. 3

Step	Opcode	Mnemonics	Operation	Remarks
29	B4	UNP.4	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack two digits
30	C4	OUT.4	R → OUT	Output 1st digit, Reg. 4
31	C5	OUT.5	R → OUT	Output 2nd digit, Reg. 5
32	66 1Ø	LDV.6.1Ø	R = NN	Reg. 6 = 1Ø; 1Ø = (+)
34	67 13	LDV.7.13	R = NN	Reg. 7 = 13; 13 = (R)
36	C6	OUT.6	R → OUT	Output + sign
37	C7	OUT.7	R → OUT	Output R
38	12 ØØ	GTO.ØØ	GTO → PC = NN	Go to step ØØ
40	93	LDA.3	R → A	Load Accum. from Reg. 3
41	B4	UNP.4	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Unpack two digits
42	C4	OUT.4	R → OUT	Output Reg. 4, 1st digit
43	C5	OUT.5	R → OUT	Output Reg. 5, 2nd digit
44	66 ØC	LDV.6.ØC	R = NN	Reg. 6 = ØC; ØC = blank
46	67 ØC	LDV.7.ØC	R = NN	Reg. 7 = ØC; ØC = blank
48	C6	OUT.6	R → OUT	Output blank
49	C7	OUT.7	R → OUT	Output blank
50	12 ØØ	GTO ØØ	GTO → PC = NN	Branch to step ØØ

Area Problems Using 'Go To Subroutine' and 'Return'

This problem was included to give you an example of how and when to use the instructions 'Go to Subroutine' and 'Return'. For a detailed explanation of these instructions, refer to Chapter 4. Now enter the program as it is written on the following pages. You now enter a number which will be the base and then a number which will be the height. Then press '1' in order to find the area of a rectangle, or press '2' in order to find the area of a triangle. The answer immediately appears on the screen. We will discuss each of these problems (rectangle and triangle) separately. First note that program step ØØ (which positions register B at the furthest left position on the screen) through program step Ø8 will be the same for both problems, with step Ø8 being your selection of '1' or '2'. The values Ø1 and Ø2 have been loaded into registers 1 and 2 respectively at program steps Ø2 and Ø4. Let's look at the rectangle problem first.

1. Program step Ø8, the accumulator is loaded with our input, either '1' or '2'. At this time, we will select '1' and it is loaded into the accumulator.
2. Program step Ø9, instructs the computer to branch to program step 13 if the accumulator (which contains Ø1) equals the contents of register 1 (which is Ø1).
3. Program step 13 and 14, instruct the computer to go to step 66, which contains the multiply routine. In order to find

the area of a rectangle, we multiply the base times the height.
4. Program step 66, load accumulator from register 3 which contains the base (for example, 8).
5. Program step 67, add register 3 (which contains the value 8) to the accumulator (which contains the value 8).
6. Program step 68, store accumulator (which contains sum of 8 + 8) in register 5.
7. Program step 69, load accumulator from register 4, which contains the height (for example, 3).
8. Program step 70, decrement accumulator by 1. (Remember, in multiplying, we must decrement the multiplier by 1 each time we perform an addition operation, until the multiplier equals Ø1).
9. Program step 71, store accumulator (which is 2) in register 4.
10. Program step 72 and 73, instructs the computer to branch to program step 77 if the accumulator is equal to register 1 (register 1 = Ø1).
11. Program step 74, load accumulator from register 5; register 5 = 16.
12. Program step 75, go to step 67. Addition operations are repeated until accumulator is equal to register 1 (at program step 72 and 73), at which time the computer branches to step 77.
13. Program step 77, load accumulator from register 5; register 5 = 24 (the answer).
14. Program step 78, instructs the computer to return to program step immediately following subroutine instruction. That takes us to:
15. Program step 15, instructs the computer to unpack register 5.
16. Program step 16 and 17, output the answer to the screen.
17. Program step 18, instructs the computer to go to blanking routine at program step 58.
18. Program step 58 through 63, instructs the computer to output blanks in order to erase old problem and answer.
19. Program step 64, instructs the computer to return to program step ØØ in preparation for a new problem.

Now let's look at a triangle problem, for example, base = 6, height = 2. Remember, for a triangle you multiply the base times the height, then divide the answer by 2. At program step Ø8, we choose '2', and 2 is loaded into the accumulator. Program step Ø9 and 10 do not apply, so the computer steps to program step 11.
1. Program step 11, instructs the computer to branch to program step 20 for the triangle routine.
2. Program step 20, instructs the computer to branch to step 66 for the multiply routine.
3. Program step 66 through 75, perform the addition operations and continue to loop until the accumulator equals register 1 (Ø1). Then the computer branches to step 77.
4. Program step 77, load accumulator with register 5, which holds the answer for B × H or 6 × 2 = 12. We must now divide this answer by 2 to find the area of a triangle.
5. Program step 78, instructs the computer to return to the program step immediately following the subroutine from which it branched originally.
6. Program step 22, a pause is implemented.
7. Program step 23, store accumulator (which contains 12) in register 3. This now becomes the dividend.
8. Program step 24 and 25, load register 4 with Ø2; this becomes the divisor.
9. Program step 26 and 27, load register 7 with ØØ. This is the initialization operation, since this register will hold the sum of the subtraction operations.
10. Program step 28, load accumulator from register 4 (which contains the divisor, 2).
11. Program step 29, subtract accumulator from register 3 (which contains dividend, 12).
12. Program step 30, store the difference in register 3; register 3 = 10.
13. Program step 31, load accumulator from register 7; register 7 = ØØ.
14. Program step 32, add one to the accumulator. Remember, this is done to keep track of the number of times we subtract the divisor from the dividend.
15. Program step 33, store sum in register 7; register 7 = Ø1.
16. Program step 34, load accumulator from register 3; register 3 = 10, dividend.
17. Program step 35 and 36, branch to step 54 if accumulator equals ØØ.
18. Program step 37, load accumulator from register 4; register 4 = 2, divisor.

19. Program step 38 and 39, branch to step 42 if register 3 is less than the accumulator.
20. Program step 40, if the computer has not branched at this point to another step number, this instruction loops the computer back to program step 29, so that additional subtraction operations can be performed.

At program step 35, the computer, after completing the subtraction operations so that the accumulator and register 3 (the dividend) equal ØØ, branches to step 54*. At program step 54, the accumulator is loaded from register 7 (which contains the number of times we subtracted, the answer). Program step 55 then unpacks this answer and program steps 56 and 57 output the answer to the screen. Blanks are output, since in this example there is no remainder, and at step 64 the computer is instructed to return to program step ØØ in preparation for a new problem.

***Note:** If there had been a remainder, the computer would have branched at program step 38 to step 42 and when the answer unpacked and displayed on the screen, a '+ R' would also have been displayed.

Area Problems Using 'Go to Subroutine' and 'Return'

Step	Opcode	Mnemonics	Operation	Remarks
ØØ	6B ØØ	LDV.B.ØØ	R = NN	Reg. B = ØØ (Positioning)
Ø2	61 Ø1	LDV.1.Ø1	R = NN	Area of rectangle - Select '1'
Ø4	62 Ø2	LDV.2.Ø2	R = NN	Area of triangle - Select '2'
Ø6	73	INP.3	Ip → R	Input 'B' value (base)
Ø7	74	INP.4	Ip → R	Input 'H' value (height)
Ø8	Ø4	INA	Ip → A	Select 1 or 2
Ø9	31 13	BEQ.1.13	(R = A) → PC = NN	Go to rectangle routine
11	32 20	BEQ.2.2Ø	(R = A) → PC = NN	Go to triangle routine
13	14 66	GTS.66	GTS → PC = NN	Go to multiply subroutine
15	B5	UNP.5	$A_H \rightarrow R_L$	Reg. 5 = A_H
			$A_L \rightarrow R_L + 1$	Reg. 6 = A_L
16	C5	OUT.5	R → OUT	Output 1st digit
17	C6	OUT.6	R → OUT	Output 2nd digit
18	12 58	GTO.58	GTO → PC = NN	Go to blanking routine

Step	Opcode	Mnemonics	Operation	Remarks
20	14 66	GTS 66	GTS → PC = NN	Go to multiply subroutine
22	ØØ	NOP	NO = ØØ	No operation (pause)
23	A3	STO 3	A → R	Store Accum. in Reg. 3
24	64 02	LDV 4 Ø2	R = NN	Load Reg. 4 with Ø2
26	67 ØØ	LDV 7 ØØ	R = NN	Load Reg. 7 with ØØ
28	94	LDA 4	R → A	Load Accum. from Reg. 4
29	D3	SUB 3	(A = R − A)	Subtract Accum. from Reg. 3
30	A3	STO 3	A → R	Store difference in Reg. 3
31	97	LDA 7	R → A	Load Accum. from Reg. 7
32	Ø3	INC	(A = A + 1)	Add one to Accumulator
33	A7	STO 7	A → R	Store sum in Reg. 7
34	93	LDA 3	R → A	Load Accum. from Reg. 3
35	13 54	BRZ 54	(A = Ø) → PC = NN	Branch to step 54 if A = Ø
37	94	LDA 4	R → A	Load Accum. from Reg. 4
38	53 42	BLS 3 42	(R < A) → PC = NN	Branch to step 42 if R < A
40	12 29	GTO 29	GTO → PC = NN	Go to step 29
42	97	LDA 7	R → A	Load Accum. from Reg. 7
43	B8	UNP 8	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Reg. 8 = A_H Reg. 9 = A_L
44	C8	OUT 8	R → OUT	Output 1st digit
45	C9	OUT 9	R → OUT	Output 2nd digit
46	6E 1Ø	LDV E 1Ø	R = NN	Load Reg. E with 1Ø, 1Ø = (+)

Step	Opcode	Mnemonics	Operation	Remarks
48	6F 13	LDV.F.13	R = NN	Load Reg. F with 13; 13 = (R)
50	CE	OUT.E	R → OUT	Output +
51	CF	OUT.F	R → OUT	Output R
52	12 58	GTO.58	GTO → PC = NN	Go to step 58
54	97	LDA.7	R → A	Load Accum. from Reg. 7
55	B8	UNP.8	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Reg. 8 = A_H Reg. 9 = A_L
56	C8	OUT.8	R → OUT	Output 1st digit
57	C9	OUT.9	R → OUT	Output 2nd digit
58	6E ØC	LDV.E.ØC	R = NN	Load Reg. E with ØC; ØC = blank
60	6F ØC	LDV.F.ØC	R = NN	Load Reg. F with ØC; ØC = blank
62	CE	OUT.E	R → OUT	Output Blank
63	CF	OUT.F	R → OUT	Output blank
64	12 ØØ	GTO.ØØ	GTO → PC = NN	Go to step ØØ
66	93	LDA.3	R → A	Load Accum. from Reg. 3
67	E3	ADD.3	(A = R + A)	Add Reg. 3 to Accumulator
68	A5	STO.5	A → R	Store Accum. in Reg. 5
69	94	LDA.4	R → A	Load Accum. from Reg. 4
70	Ø2	DEC	(A = A − 1)	Decrement Accum. by 1
71	A4	STO.4	A → R	Store Accum. in Reg. 4
72	31 77	BEQ.1.77	(R = A) → PC = NN	If Reg. 1 = Accum., branch to 77
74	95	LDA.5	R → A	Load Accum. from Reg. 5

Step	Opcode	Mnemonics	Operation	Remarks
75	12 67	GTO 67	GTO → PC = NN	Go to step 67
77	95	LDA 5	R → A	Load Accum. from Reg. 5
78	Ø7	RET	Ret → PC = NN	Return to Subroutine

One-Digit Addition Flash Card

This program differs from the Flash Card program already programmed in the Videopac in several ways. First, in this program, the old problem is erased automatically and a new problem is displayed on the screen. Secondly, and perhaps most important, the reward for guessing the correct answer is greater with this program. Just wait and see!

There are several new uses of instructions included in this program which we should discuss in detail.

1. Program step ØØ and Ø1, load a blank into register A. This will become the eraser which will cause the old problem to disappear from the screen.
2. Program step Ø2 through Ø9, you should be familiar with these instructions.
3. Program step 10, the accumulator is loaded with a random number.
4. Program step 11, the random number is unpacked.
5. Program step 12 and 13, sets the output position of register B at ØØ.
6. Program step 14 through 18, you should be familiar with these instructions. Note that a 'No Operation' instruction was needed at program step 17, since we had programmed three output instructions in a row.
7. Program step 19, the accumulator is loaded from register Ø with one digit of the unpacked number.
8. Program step 20, the accumulator is added to register 1 which contains the second digit of the unpacked number.
9. Program step 21, the sum is stored in register 2.
10. Program step 22 through 26, you should be familiar with these operations.
11. Program step 27, outputs blank to screen. This blank appears between the answer on the screen and the word 'NO' if the answer happens to be wrong.
12. Program step 28 and 29, instructs the computer to branch to step 45 if the correct answer is given.
13. Program step 45 through 61, instructs the computer to buzz a melody (reward). At program step 46, the computer is instructed to go to subroutine at program step 70.
14. Program step 62 and 63, reset

register B to ØØ.
15. Program step 64, instructs the computer to output a blank.
16. Program step 65, load accumulator from register B which now equals Ø1, since a blank has been output. (Remember, register B automatically increments by 1 each time there is an output instruction).
17. Program step 66 and 67, if accumulator = Ø, the computer branches to program step 10 and repeats the program by selecting a new random number.
18. Program step 68 and 69, if the accumulator is not equal to ØØ (remember, it contains the contents of register B), the computer returns to program step 64 and blanks are output and register B increments until it reaches ØA (the furthest right position). At this point, register B rolls back to ØØ. The accumulator now equals ØØ and the computer will branch to program step 10.

At program step 46, the computer was instructed to go to subroutine at program step 70. The subroutine is as follows.
1. Program step 70 and 71, register 7 is loaded with ØØ (this is used as a reference point).
2. Program step 72 and 73, register E is loaded with 75 (this number was randomly chosen to cause a slight pause between the buzz sounds).
3. Program step 74, load accumulator from register E.
4. Program step 75, no operation (used as a slight pause).
5. Program step 76, decrement accumulator by 1 (remember it contains 75).
6. Program step 77 and 78, if accumulator is not equal to register 7 (ØØ), the computer is instructed to branch to step 75 and repeat decrementing, until the accumulator equals ØØ. It is this repetition which causes the pauses between the buzz sounds.
7. Program step 79, instructs the computer to return to the next 'go to' instruction (GTS) until it steps to program step 62 and erases the problem. At step 66, the computer returns to step 10 and the program is repeated with a new problem.

If the wrong answer is given, the program progresses as follows:
1. All steps are the same until program step 28 is reached.
2. If at program step 28 the wrong answer is given, the computer steps to program step 30 and 31, and 'NO' is displayed on the screen next the wrong answer.
3. Program step 32 and 33, register B is loaded with Ø4 (positioned at Ø4).
4. Program step 34, input first number of second guess.
5. Program step 35 through 40, outputs blanks, erasing the old answer and the word 'NO'.
6. Program step 41 and 42, sets register B at output position Ø5.
7. Program step 43, instructs the computer to branch to step 24.
8. Program step 24, input second number of second guess.

As an exercise for you, review the remaining program steps.

One-Digit Addition Flash Card (Guess Answers)

Step	Opcode	Mnemonics	Operation	Remarks
ØØ	6A ØC	LDV A ØC	R = NN	Load a blank into Reg. A
Ø2	68 1Ø	LDV 8 1Ø	R = NN	Load a + sign into Reg. 8
Ø4	69 2B	LDV 9 2B	R = NN	Load an = sign into Reg. 9
Ø6	6C 2D	LDV C 2D	R = NN	Load 'N' into Reg. C
Ø8	6D 17	LDV D 17	R = NN	Load 'O' into Reg. D
1Ø	Ø8	RND	RND → A	Load Accum. with random number
11	BØ	UNP Ø	$A_H \rightarrow R_L$ $A_L \rightarrow R_L + 1$	Separate digits

Step	Opcode	Mnemonics	Operation	Remarks
12	6B ØØ	LDV,B,ØØ	R = NN	Set output position
14	CØ	OUT Ø	R → OUT	Output first digit
15	C8	OUT 8	R → OUT	Output + sign
16	C1	OUT 1	R → OUT	Output second digit
17	ØØ	NOP	No = ØØ	No operation
18	C9	OUT 9	R → OUT	Output = sign
19	9Ø	LDA Ø	R → A	Load first digit
20	E1	ADD 1	(A = R + A)	Add to second digit
21	A2	STO 2	A → R	Store sum in Reg. 2
22	73	INP 3	INP → R	Input first digit guess
23	C3	OUT 3	R → OUT	Output first digit guess
24	74	INP 4	INP → R	Input second digit guess
25	C4	OUT 4	R → OUT	Output second digit guess
26	83	PAK 3	$R_L \rightarrow A_H$ $R_L + 1 \rightarrow A_L$	Combine digits
27	CA	OUT A	R → OUT	Output blank
28	32 45	BEQ 2 45	(R = A) → PC = NN	If correct guess - Buzz
30	CC	OUT C	R → OUT	Output 'N'
31	CD	OUT D	R → OUT	Output 'O'
32	6B Ø4	LDV B Ø4	R = NN	Set output position to Ø4
34	73	INP 3	INP → R	Input first number of second guess
35	C3	OUT 3	R → OUT	Output first number
36	CA	OUT A	R → OUT	Output blank
37	CA	OUT A	R → OUT	Output blank
38	ØØ	NOP	No = ØØ	No operation

Step	Opcode	Mnemonics	Operation	Remarks
39	CA	OUT.A	R → OUT	Output blank
40	CA	OUT.A	R → OUT	Output blank
41	6B Ø5	LDV.B.Ø5	R = NN	Set output position to Ø5
43	12 24	GTO.24	GTO → PC = NN	Go to step 24
45	Ø5	SIG	Buz = 1	Buzz
46	14 7Ø	GTS.7Ø	GTS → PC = NN	No sound
48	Ø5	SIG	Buz = 1	Buzz
49	Ø5	SIG	Buz = 1	Buzz
50	Ø5	SIG	Buz = 1	Buzz
51	14 7Ø	GTS.7Ø	GTS → PC = NN	No sound
53	Ø5	SIG	Buz = 1	Buzz
54	14 7Ø	GTS.7Ø	GTS → PC = NN	No sound
56	14 7Ø	GTS.7Ø	GTS → PC = NN	No sound
58	Ø5	SIG	Buz = 1	Buzz
59	14 7Ø	GTS.7Ø	GTS → PC = NN	No sound
61	Ø5	SIG	Buz = 1	Buzz
62	6B ØØ	LDV.B.ØØ	R = NN	Set position to ØØ
64	CA	OUT.A	R → OUT	Output blank
65	9B	LDA.B	R → A	Load Accum. from Reg. B
66	13 1Ø	BRZ.1Ø	(A = Ø) → PC = NN	Branch on Accum. = Ø to step 10
68	12 64	GTO.64	GTO → PC = NN	Go to step 64
70	67 ØØ	LDV.7.ØØ	R = NN	Load Reg. 7 with ØØ
72	6E 75	LDV.E.75	R = NN	Load Reg. E with 75
74	9E	LDA.E	R → A	Load Accum. from Reg. E

Step	Opcode	Mnemonics	Operation	Remarks
75	Ø0	NOP	No = Ø0	No operation
76	Ø2	DEC	(A = A – 1)	Subtract 1 from Accum.
77	27 75	BNE 7.75	(R ≠ A) → PC = NN	Branch if Accum. ≠ to zero
79	Ø7	RET	RET → PC = NN	Return from subroutine

Please note that the following program sheets are different from the program sheets used previously. Up to this point, the sample programs have been written step by step and explained for you. The following programs are written on slightly different program sheets which contain several very useful tools.

First, note the addition of the label column. This is an important and helpful tool, when writing your own programs. You will recall in some of the previous sample programs we encountered 'Go to' and 'Branch to' instructions. These instructions generally referred to a program step later in the program. If, when first writing a program, you must use one of these instructions, you will have to complete the program before inserting the correct program step number. This is where the Label column helps. It allows you to indicate a branching or looping instruction when you write it, so that you can later insert the correct program step. The sample program 'Message' is a good example of how the Labelling column is used.

The Comment column is used in the same way as the Remarks column in the earlier programming sheets.

The second difference in the two types of programming sheets is the addition of the Byte column. This column keeps record of the number of bytes in each instruction, so that when initially writing a program, you will know that the program step must increment by 1 or 2 depending upon the number of bytes in each instruction. Remember, that each program step can only contain 8 bits of data or 1 byte, but that an instruction may be 1 byte or 2 bytes long.

The third and last difference in the two types of programming sheets is the addition of the column marked Register Use. This column will also prove very useful when initially writing your program. The contents of each register should be indicated in this column as you load it with a value. This prevents using the same register twice and also is helpful when reviewing an arithmetic program, since at a glance you can tell which register contains the divisor, dividend, multiplier, or multiplicand.

Let's now take a look at the last three sample programs.

Three Ways to Enter and Output a Letter

These three sample programs are presented to show you the three different instructions which can be used to input and output a letter on the screen.

For the first example, we have chosen to input and display the letter 'H' or 1D in hex code. With this type of program, whatever is loaded into the register and is output to the screen will remain on the screen. You cannot change it. With this program, you could enter a complete message and have it remain on the screen.

The second example uses the instructions, 'Input to a Register' and 'Output from a Register', but does not designate any particular value. Thus, once the program is entered, any value can be entered and it will be displayed on the screen.

The third example is similar to the second in that any value may be entered, but it is input to the Accumulator rather than to a Register.

You will note, in all three examples the last instruction was 'Input to a Register' which was used as a pause since no output instruction was indicated, thus only one keyboard depression could be made. As an exercise for you, using example two or three, program the appropriate instruction sets in order to create a loop so that all 11 positions on the screen may be used.

Three Ways to Enter and Output a Letter (For this example, use 'H')

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code		Reg. use
1. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	0-
	LDV Ø,1D	Load Reg. Ø = 1D; 1D = H	2	Ø2	6Ø	1D	1-
	OUT,Ø	Output Reg. Ø = H	1	Ø4	CØ		2-
End	INP,1	Ip → R (used as pause)	1	Ø5	71		3-
							4-
							5-
							6-
							7-
							8-
							9-
2. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	A-
	INP,Ø	Ip → R	1	Ø2	7Ø		B-
	OUT,Ø	R → OUT	1	Ø3	CØ		C-
End	INP,1	Ip → R (pause)	1	Ø4	71		D-
							E-
							F-
3. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	
	INA	Ip → A	1	Ø2	Ø4		
	OTA	A → OUT	1	Ø3	ØB		
End	INP,1	Ip → R (pause)	1	Ø4	71		

You will note, in all three examples the last instruction was 'Input to a Register' which was used as a pause since no output instruction was indicated, thus only one keyboard depression could be made. As an exercise for you, using example two or three, program the appropriate instruction sets in order to create a loop so that all 11 positions on the screen may be used.

Three Ways to Enter and Output a Letter (For this example, use 'H')

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code		Reg. use
1. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	0-
	LDV Ø,1D	Load Reg. Ø = 1D; 1D = H	2	Ø2	6Ø	1D	1-
	OUT Ø	Output Reg. Ø = H	1	Ø4	CØ		2-
End	INP 1	Ip → R (used as pause)	1	Ø5	71		3-
							4-
							5-
							6-
							7-
							8-
							9-
2. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	A-
	INP Ø	Ip → R	1	Ø2	7Ø		B-
	OUT Ø	R → OUT	1	Ø3	CØ		C-
End	INP 1	Ip → R (pause)	1	Ø4	71		D-
							E-
							F-
3. Start	LDV B,ØØ	Positioning	2	ØØ	6B	ØØ	
	INA	Ip → A	1	Ø2	Ø4		
	OTA	A → OUT	1	Ø3	ØB		
End	INP 1	Ip → R (pause)	1	Ø4	71		

Six Letter Guess

After being entered, this program allows you to enter a six letter word into the computer. Six dots appear on the screen and your opponent enters a letter. If it is used in the word, it appears on the screen in the correct position. If the letter does not appear in the word, nothing happens.

Let's look at some of the program steps in detail.

1. Program step Ø0, used as a flag or reference position. Ø1 is loaded into register 7. Ø1 was chosen rather than Ø0, because it can only mean the decimal number 1 and nothing else.
2. Program step Ø4, Ø5, and Ø6, input 1st letter into register 9, load a dot into register 1, output register 1 to screen. This is an initialization process and steps Ø7 through 27 are the same. This is done so that the six does appear on the screen when the word is first input. Note the Register Use column.
3. Program step 28 through 37, positions the computer to Ø0 each time a guess is taken and outputs to the screen either the correct letter guessed or a dot.
4. Program step 38 and 39, instruct the computer to return to Ø0 if accumulator = register 7 in preparation for a new word if the previous word has been guessed correctly. (Note: this is a Flag or reference point.)
5. Program step 40, inputs a guess to accumulator; it is compared to register in program steps 41 through 52.
6. Program step 53 and 54, instructs the computer to go to program step 71 if a letter in the word is missing.
7. Program step 71 and 72, loads register 8 with a dot.
8. Program step 73, loads the accumulator from register 8.
9. Program step 74 through 85, instructs the computer to branch to program step 28 if the register is equal to the accumulator (in other words, if the register still remains a dot).
10. Program step 86 and 87, loads register 7 with a 2B (=). This is a Flag.*
11. Program step 88, loads the accumulator from register 7.
12. Program step 89 and 90, sound the buzz which indicates the word has been displayed correctly.
13. Program step 91 and 92, instruct the computer to go to step 28 for positioning.
14. Program step 28 through 37, displays word on screen.
15. Program step 38 and 39, instruct the computer to return to Ø0 if accumulator = register 7.
16. Program step Ø0, loads register 7 with Ø1 and game continues.

***Note:** The 2B or = sign was used as a flag in this instance; however, any sign could have been used instead.

Six Letter Guess

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
Reset	LDV 7,Ø1	R = NN (flag)	2	Ø0	67 Ø1	0-
Start Pos.	LDV B,Ø0	Positioning	2	Ø2	6B Ø0	1- 1st ■
	INP 9	Input 1st letter	1	Ø4	79	2- 2nd ■
	LDV 1,27	Read 1st dot	2	Ø5	61 27	3- 3rd ■
	OUT 1	1st dot on screen	1	Ø7	C1	4- 4th ■
	INP A	Input 2nd letter	1	Ø8	7A	5- 5th ■
	LDV 2,27	Read 2nd dot	2	Ø9	62 27	6- 6th ■
	OUT 2	2nd dot on screen	1	11	C2	7- Ø1 (flag)
	INP C	Input 3rd letter	1	12	7C	8- 7th ■

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
	LDV.3.27	Read 3rd dot	2	13	63 27	9- 1st letter
	OUT.3	3rd dot on screen	1	15	C3	A- 2nd letter
	INP.D	Input 4th letter	1	16	7D	B- Positio- ning
	LDV.4.27	Read 4th dot	2	17	64 27	C- 3rd letter
	OUT.4	4th dot on screen	1	19	C4	D- 4th letter
	INP.E	Input 5th letter	1	20	7E	E- 5th letter
	LDV.5.27	Read 5th dot	2	21	65 27	F- 6th letter
	OUT.5	5th dot on screen	1	23	C5	
	INP.F	Input 6th letter	1	24	7F	
	LDV.6.27	Read 6th dot	1	25	66 27	
	OUT.6	6th dot on screen	1	27	C6	
Position	LDV.B.ØØ	Position on screen	2	28	6B ØØ	
	OUT.1	Put dots on screen	1	3Ø	C1	
	OUT.2	Put dots on screen	1	31	C2	
	OUT.3	Put dots on screen	1	32	C3	
	NOP	No operation	1	33	ØØ	
	OUT.4	Put dots on screen	1	34	C4	
	OUT.5	Put dots on screen	1	35	C5	
	OUT.6	Put dots on screen	1	36	C6	
	NOP	No operation	1	37	ØØ	
Flag	BEQ.7.ØØ	Reset	2	38	37 ØØ	
	INA	Ip → A, input guess	1	40	Ø4	
	BEQ.9.55	Letter in	2	41	39 55	
	BEQ.A.58	Word Missing	2	43	3A 58	
	BEQ.C.61		2	45	3C 61	

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
	BEQ.D 64		2	47	3D 64	
	BEQ.E 67		2	49	3E 67	
	BEQ.F 70		2	51	3F 70	
	GTO.71		2	53	12 71	
	STO.1	Got 1st letter	1	55	A1	
	GTO.43	Check next position	2	56	12 43	
	STO.2	Got 2nd letter	1	58	A2	0-
	GTO.45	Check next position	2	59	12 45	1- 1st ■
	STO.3	Got 3rd letter	1	61	A3	2- 2nd ■
	GTO.47	Check next position	2	62	12 47	3- 3rd ■
	STO.4	Got 4th letter	1	64	A4	4- 4th ■
	GTO.49	Check next position	2	65	12 49	5- 5th ■
	STO.5	Got 5th letter	1	67	A5	6- 6th ■
	GTO.51	Check next position	2	68	12 51	7- Ø1 (flag)
	STO.6	Got 6th letter	1	70	A6	8- 7th ■
Missing	LDV 8,27	Load Reg. 8 with dot	2	71	68 27	9- 1st letter
	LDA 8	R → A; A = dot	1	73	98	A- 2nd letter
	BEQ 1,28	Position (step 28)	2	74	31 28	B- Position
	BEQ 2,28	Position (step 28)	2	76	32 28	C- 3rd letter
	BEQ 3,28	Position (step 28)	2	78	33 28	D- 4th letter
	BEQ 4,28	Position (step 28)	2	80	34 28	E- 5th letter
	BEQ 5,28	Position (step 28)	2	82	35 28	F- 6th letter
	BEQ 6,28	Position (step 28)	2	84	36 28	
	LDV 7,2B	Set flag to =	2	86	67 2B	

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
	LDA 7	R → A	1	88	97	
	SIG		1	89	Ø5	
	SIG		1	9Ø	Ø5	
	GTO 28	Positioning	2	91	12 28	

Message

After being entered, this program allows you to press any number between 1 and 6 to call programmed message to the screen. In the program as it written, we have programmed six messages. After studying the program, you may wish to substitute your own messages.

This program provides a good example of the use of the Label column. You will note the first step, ØØ and Ø1, is load a value into register Ø and the value is 9Ø. You will note that program step 9Ø is the 'No Operation' instruction after the last message, and that program steps 91 through 96 are a relocation table. The hex code at each of these program steps is the first program step number of each of the messages. It is this first instruction, 'load a value into register Ø and the value is 9Ø' which allows you to select any number between 1 and 6 to call a message to the screen. Let's look at a few of the other instructions in the program.

1. Program step Ø2 and Ø3, load register 1 with ØC (blank). This blank will be used in messages which have more than one word.
2. Program step Ø4, input to the accumulator; you may select 1, 2, 3, 4, 5, 6, and whichever you choose will be input to the accumulator.
3. Program step Ø5, add register Ø to accumulator. In other words, if we had chosen number 2, the contents of register Ø (which are 9Ø) are added to the accumulator (which is 2), thus 92 is now in the accumulator.
4. Program step Ø6, store accumulator in register C; register C now equals 92.
3. Program step Ø7, register C moves the program counter to program step 92, and the contents at program step 92 (which are 36) are loaded into the accumulator. This is the 'Move' instruction or 'Load accumulator from a program step'. You will remember that register C is always used with this instruction. (Refer to Chapter 4 'Load accumulator from program step' for further information).
6. Program step Ø8, store accumulator (36) in register C; C now equals 36.
7. Program step Ø9 and 1Ø, load register B (positioning) with the value ØØ (the furthest left position).
8. Program step 11 and 12, load register 2 with the number 11 (the number of positions on the screen), which is hex code 2B.
9. Program step 13 and 14, load register 3 with ØØ to be used as a reference.
10. Program step 15, load the accumulator from register 1; register 1 equals a blank. This begins the loop which erases an old message from the screen in preparation for a new message. You will note program steps 15 through 21, load the accumulator with a blank, output the blank, load the accumulator from register 2 (2B or 11), decrement the accumulator by 1, store the result in register 2, and the computer branches to step 15 if the accumulator is not equal to register 3 (ØØ). Remember, when erasing, each of the 11 positions must be filled with a blank.
11. Program step 22 and 23, loads register B with ØØ (furthest left position). This is used to position register B in preparation for a new message.
12. Program step 24, takes the contents of register C (36), moves to that program step (36) and loads the contents at that program step (14) into the accumulator.
13. Program step 25 and 26, if the accumulator equals ØØ at this point, the computer would branch to program step Ø4, and prepare itself for a new message. If the accumulator contains a value (as in this example, it contains 14) then the computer steps to program step 27.
14. Program step 27, output the contents of the accumulator to the screen; a 'T' appears. (Refer to your chart of hex codes Figure 12).

15. Program step 28 and 29, instruct the computer to go to step 24 and loop through the previous instructions to display message*. When the message is completed (note at the end of each message there is a no operation instruction, ØØ), and the computer steps to program step 25, the accumulator will be equal to register 3 (ØØ), and the computer will branch to program step Ø4, in preparation for a new message.

This completes the sample programs we have prepared for you. In the appendix, you will find blank program sheets to use when writing your own program.

* **Note:** When repeating the loop at program step 24, the contents of register C remain the same (36); however, the program counter increments by one each time so that the appropriate program step is reached.

Message

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
Start	LDV Ø.9Ø	Location table	2	ØØ	6Ø 9Ø	0- 9Ø
	LDV 1.ØC		2	Ø2	61 ØC	1- ØC (blank)
Re-start	INA	Press 1,2,3,4,5, or 6	1	Ø4	Ø4	2- ØB (11)
	ADD.Ø	(A = R + A)	1	Ø5	EØ	3- ØØ
	STO.C	A → Reg. C	1	Ø6	AC	4-
	MOV	Rc → PC → A	1	Ø7	Ø9	5-
	STO.C	A → Reg. C	1	Ø8	AC	6-
Blanks	LDV.B.ØØ	R = NN (positioning)	2	Ø9	6B ØØ	7-
	LDV.2.ØB	R = NN	2	11	62 11	8-
	LDV.3.ØØ	R = NN	2	13	63 ØØ	9-
Loop 1	LDA.1	R → A	1	15	91	A-
	OTA	A → OUT	1	16	ØB	B-
	LDA.2	R → A	1	17	92	C-
	DEC.	(A = A − 1)	1	18	Ø2	D-
	STO.2	A → R	1	19	A2	E-
	BNE.3.15	Loop 1	2	2Ø	23 15	F-
Out-put	LDV.B.ØØ	R - NN (positioning)	2	22	6B ØØ	
Loop 2	MOV	Rc → PC → A	1	24	Ø9	

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
	BEQ,3 04	Restart	2	25	33 04	
	OTA	A → OUT	1	27	0B	
	GTO 24	Loop 2	2	28	12 24	
Mess. 1	*	Output 'H'	1	30	1D	
		Output 'E'	1	31	12	
		Output 'L'	1	32	0E	
		Output 'L'	1	33	0E	
		Output 'O'	1	34	17	
		End Mess. 1	1	35	00	
Mess. 2		Output 'T'	1	36	14	
		Output 'A'	1	37	20	
		Output 'K'	1	38	1F	
		Output 'E'	1	39	12	
		Blank	1	40	0C	
		Output 'A'	1	41	20	
		Blank	1	42	0C	
		Output 'L'	1	43	0E	
		Output 'O'	1	44	17	
		Output 'O'	1	45	17	
		Output 'K'	1	46	1F	
		End Mess. 2	1	47	00	
Mess. 3		Output 'R'	1	48	13	
		Output 'E'	1	49	12	
		Output 'M'	1	50	26	0- 90 1- 0C (blank)
		Output 'A'	1	51	20	2- 0B (11)
		Output 'R'	1	52	13	3- 00
		Output 'K'	1	53	1F	4-

*When entering single letters and numbers, the hex code only is used, since there is no mnenonic code for them.

Label	Mnemonic Code	Comment	Byte #	Step #	Hex Code	Reg. Use
		Output 'A'	1	54	2Ø	5-
		Output 'B'	1	55	25	6-
		Output 'L'	1	56	ØE	7-
		Output 'E'	1	57	12	8-
		End Mess. 3	1	58	ØØ	9-
Mess. 4		Output 'N'	1	59	2D	A-
		Output 'E'	1	6Ø	12	B-
		Output 'W'	1	61	11	C-
		Blank	1	62	ØC	D-
		Output 'F'	1	63	1B	E-
		Output 'O'	1	64	17	F-
		Output 'R'	1	65	13	
		Blank	1	66	ØC	
		Output '7'	1	67	Ø7	
		Output '8'	1	68	Ø8	
		End Mess. 4	1	69	ØØ	
Mess. 5		Output 'Q'	1	7Ø	18	
		Output 'U'	1	71	15	
		Output 'E'	1	72	12	
		Output 'S'	1	73	19	
		Output 'T'	1	74	14	
		Output 'I'	1	75	16	
		Output 'O'	1	76	17	
		Output 'N'	1	77	2D	
		Output 'S'	1	78	19	
		Output '?'	1	79	ØD	
		End Mess. 5	1	8Ø	ØØ	
Mess. 6		Output 'C'	1	81	23	

Table of Powers of Two

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 844	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5

Keycodes, Hex Codes, and Decimal Equivalents

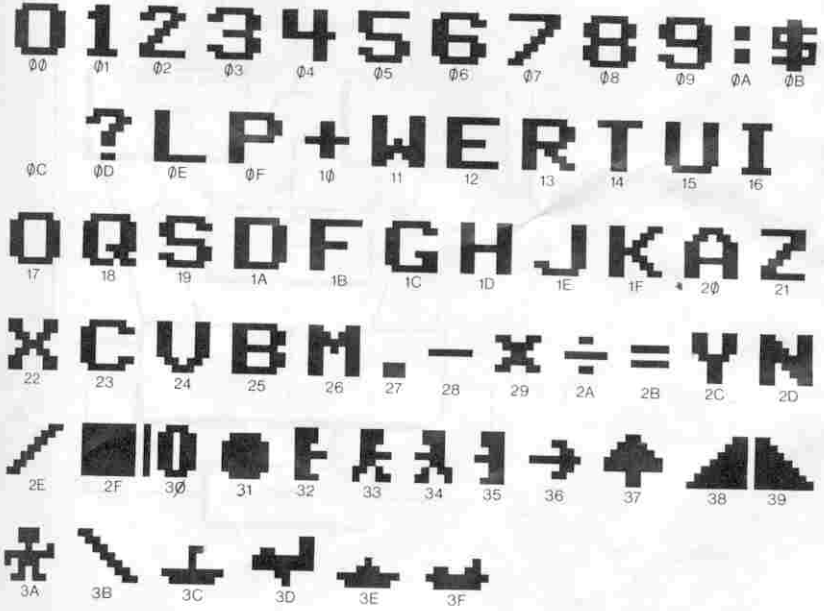
Key	Hexcode	Decimal	Key	Hexcode	Decimal
Ø	00	00	O	17	23
1	01	01	P	OF	16
2	02	02	Q	18	24
3	03	03	R	13	19
4	04	04	S	19	25
5	05	05	T	14	20
6	06	06	U	15	21
7	07	07	V	24	36
8	08	08	W	11	17
9	09	09	X	22	34
A	20	32	Y	2C	44
B	25	37	Z	21	32
C	23	35	Blank	OC	12
D	1A	26	:	OA	10
E	12	18	$	OB	11
F	1B	27	Clear	2E	46
G	1C	28	?	OD	13
H	1D	29	.	27	39
I	16	22	+	10	16
J	1E	30	−	28	40
K	1F	31	×	29	41
L	0E	14	÷	2A	42
M	26	38	=	2B	43
N	2D	45	Enter	2F	47

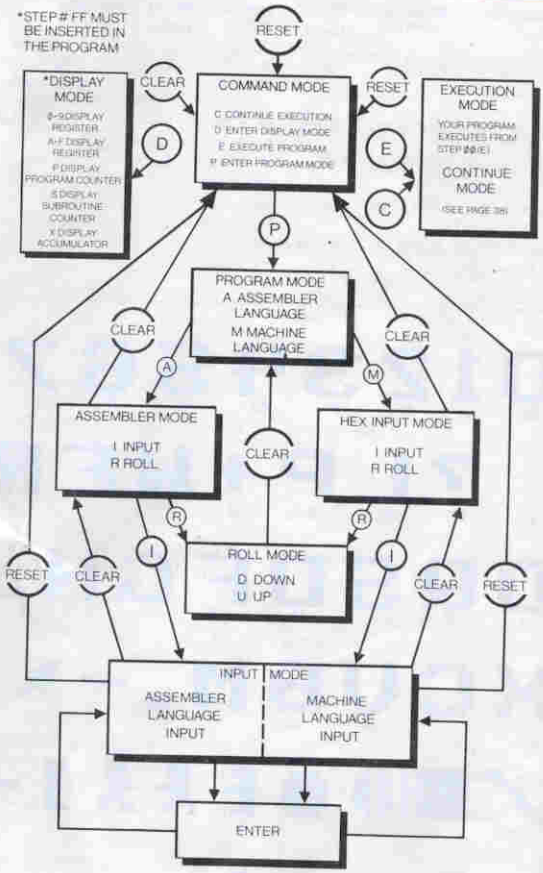
Instruction Sets

Description	Mnemonic	Op Code	Operation	No. of Bytes	Remarks
Input					
Input to Register	INP.R	7R	Ip → R	1	1 key depression only
Input to Accumulator	INA	Ø4	Ip → A	1	1 key depression only
Output					
Output from Register	OUT.R	CR	R → OUT	1	Reg. B sets position
Output from Accumulator	OTA	ØB	A → OUT	1	of output to screen
One second Buzz	SIG	Ø5	BUZ = 1	1	
Change Accumulator contents Mathematics					
Set to Ø	CLR	Ø1	(A = Ø)	1	Accum. = Hex ØØ
Subtract 1	DEC	Ø2	(A = A−1)	1	Decrement by 1

Description	Mnemonic	Op Code	Operation	No. of Bytes	Remarks
Add 1	INC	Ø3	(A = A+1)	1	Increment by 1
Load with Random No	RND	Ø8	RND → A	1	
Load from Storage	MOV	Ø9	R_C → PC → A	1	Reg. C points to step # where data is stored That data will then be moved to accumulator.
Combine 2 digits	PAK R	8R	R_L → A_H R_L + 1 → A_L	1	R_L = Reg. low order bit A_H = Accum high order bit
Separate 2 digits	UNP R	BR	A_H → R_L A_L → R_L + 1	1	Note: If R_L = Reg. 4 then R_L + 1 = Reg. 5
Load from Register	LDA R	9R	R → A	1	
Subtract from Reg.	SUB R	DR	(A = R − A)	1	
Add Register	ADD R	ER	(A = R + A)	1	
Change Register Contents					
Store Accumulator	STO R	AR	A → R	1	
Load a Value	LDV R NN	6RNN	R = NN	2	Load R with value NN
Control Execution order					
No Operation	NOP	ØØ	NO = ØØ	1	
Halt	HLT	FF	HLT = FF	1	
Go to Subroutine	GTS NN	14NN	GTS → PC = NN	2	
Return from Subrout	RET	Ø7	RET → PC = NN	1	
Branching Decision					
Branch on Decimal Borrow	BDB NN	10NN	(A_H = 9) → PC = NN	2	NN = ØØ through 99
Branch on Decimal Carry	BDC NN	11NN	(A_H ≠ Ø) → PC = NN	2	R = Ø-9, A-F
Branch Unconditionally	GTO NN	12NN	GTO → PC = NN	2	PC = Program Counter

Description	Mnemonic	Op Code	Operation	No. of Bytes	Remarks
Branch if Accumulator is Ø	BRZ NN	13NN	(A = Ø) → PC = NN	2	The program counter points to the step number
Branch if Reg. ≠ Accumulator	BNE R NN	2RNN	(R ≠ A) → PC = NN	2	
Branch if Reg. = Accumulator	BEQ R NN	3RNN	(R = A) → PC = NN	2	
Branch if Reg. > Accumulator	BGT R NN	4RNN	(R > A) → PC = NN	2	
Branch if Reg. < Accumulator	BLS R NN	5RNN	(R < A) → PC = NN	2	




*STEP # FF MUST BE INSERTED IN THE PROGRAM
RESET
*DISPLAY MODE
0-9 DISPLAY REGISTER
A-F DISPLAY REGISTER
P DISPLAY PROGRAM COUNTER
S DISPLAY SUBROUTINE COUNTER
X DISPLAY ACCUMULATOR
CLEAR
D
COMMAND MODE
C CONTINUE EXECUTION
D ENTER DISPLAY MODE
E EXECUTE PROGRAM
P ENTER PROGRAM MODE
RESET
E
C
EXECUTION MODE
YOUR PROGRAM EXECUTES FROM STEP 00 (E)
CONTINUE MODE
(SEE PAGE 38)
P
PROGRAM MODE
A ASSEMBLER LANGUAGE
M MACHINE LANGUAGE
CLEAR
CLEAR
A
M
ASSEMBLER MODE
I INPUT
R ROLL
CLEAR
HEX INPUT MODE
I INPUT
R ROLL
R
R
I
I
ROLL MODE
D DOWN
U UP
RESET
CLEAR
CLEAR
RESET
INPUT MODE
ASSEMBLER LANGUAGE INPUT
MACHINE LANGUAGE INPUT
ENTER