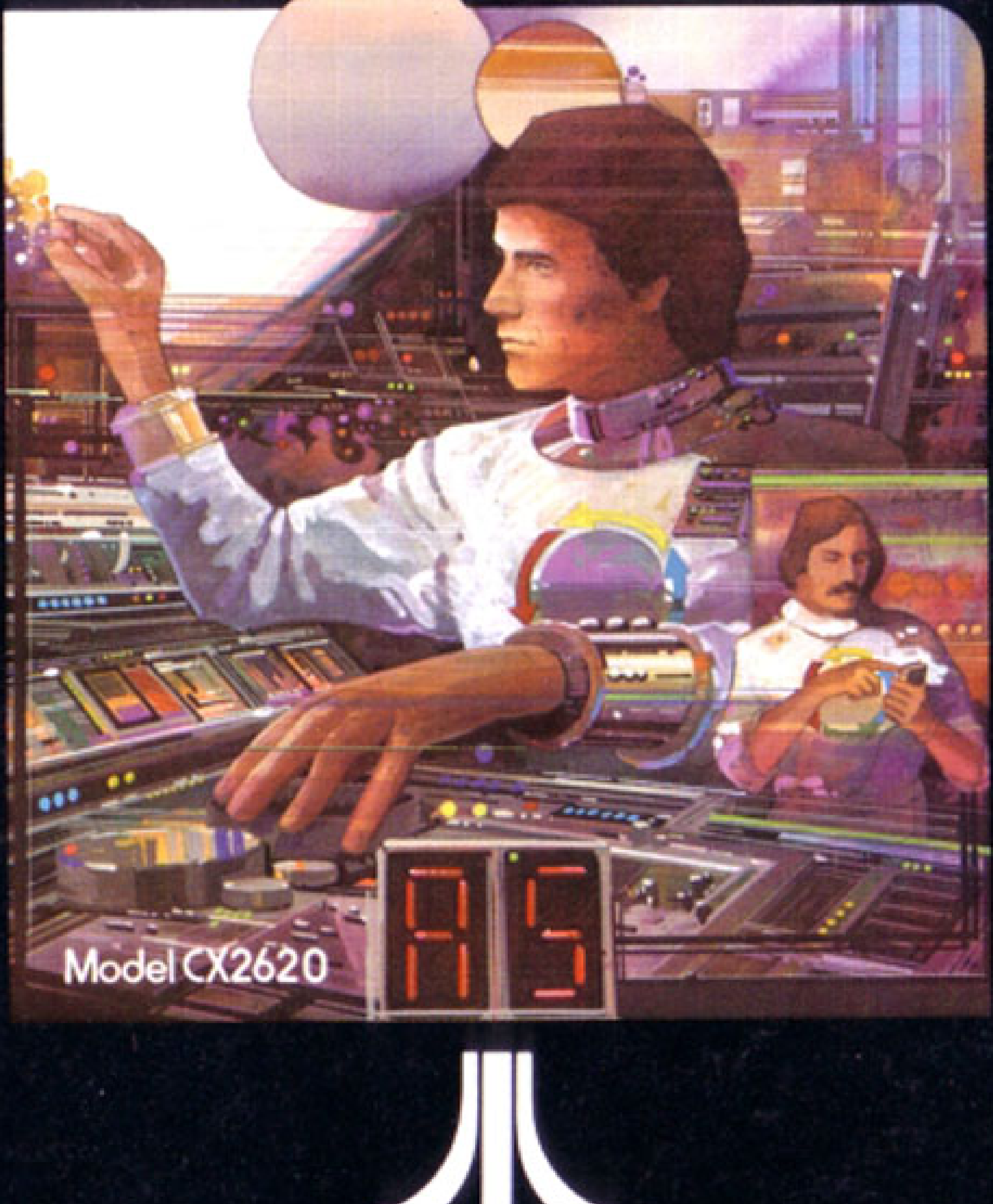
BASIC PROGRAMMING

GAME PROGRAM™ INSTRUCTIONS



ATARI®

A Warner Communications Company

ATARI, INC., Consumer Division
1195 Borregas Ave., Sunnyvale, CA 94086

BASIC PROGRAMMING

INTRODUCTION

BASIC PROGRAMMING is an instructional tool designed to teach you the fundamental steps of computer programming. **BASIC** is an acronym for **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. It was originated so that people could easily learn to "write" computer programs.

Computer programs are simply a series of instructions. The programs control the flow of information within the computer. **BASIC PROGRAMMING** allows you to give the Video Computer System™ the instructions it needs to carry out some simple tasks.

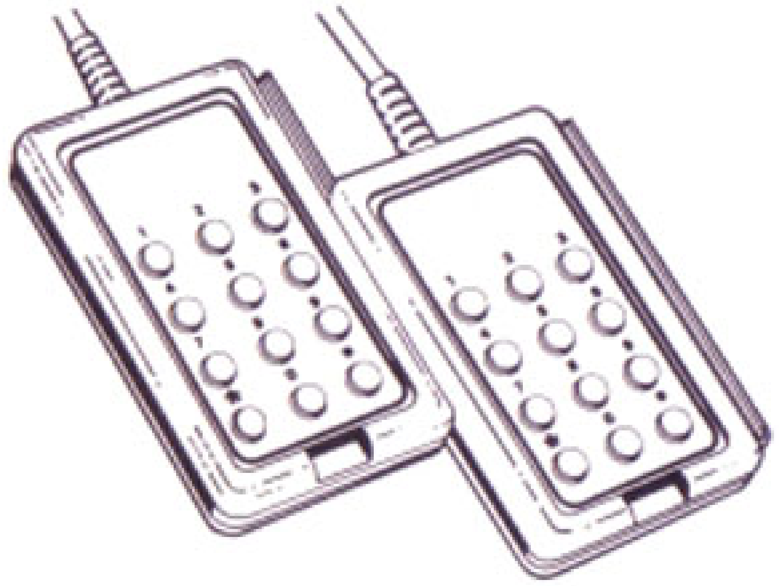
Keep in mind that **BASIC PROGRAMMING** has a limited amount of memory in comparison to more sophisticated computer systems. It is, however, an excellent instructional device for learning the essentials of computer programming.

NOTE: Turn the console **off** when inserting or removing a Game Program™. This will protect the electronic components and prolong the life of your Atari® Video Computer System.

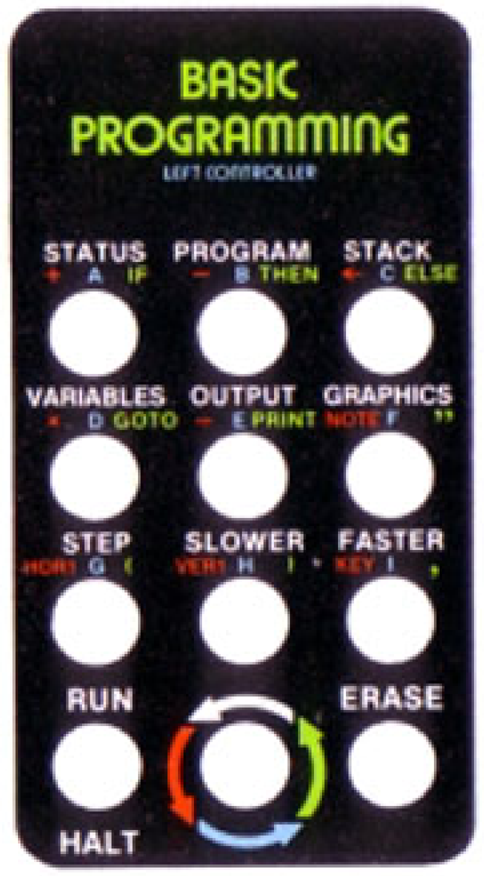
This Game Program may cause some television screens to "roll" slightly. If this occurs, adjustment of the VERTICAL HOLD may be necessary.

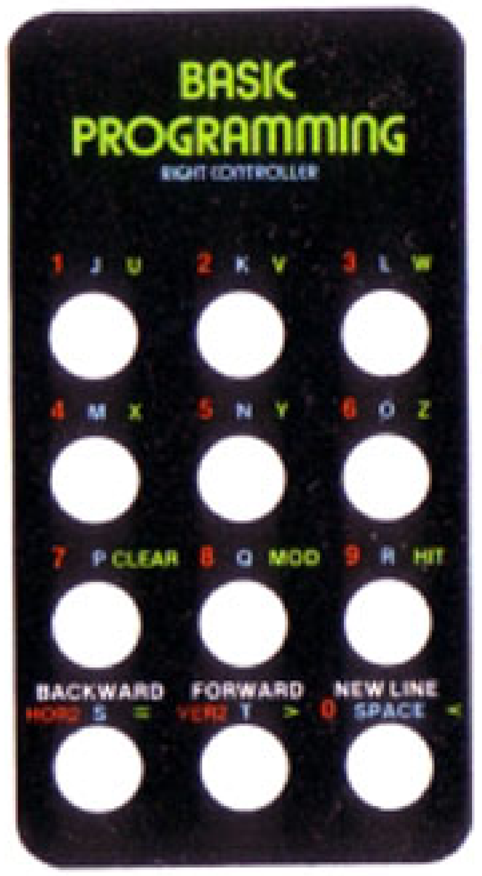
KEYBOARD CONTROLLERS

Use your Keyboard Controllers with this ATARI® Game Program.™ Be sure the Controllers are firmly plugged into the LEFT and RIGHT CONTROLLER jacks at the rear of your ATARI Video Computer System.™



To connect the two controllers, slide the tongue of the left controller into the groove on the right controller. The two Keyboard Controllers, locked together, form a 24-key Keyboard used to enter your program and to control the display on your television screen.





Remove the Keyboard Controller labels from the envelope. Place the label marked **LEFT** over the Keyboard plugged into the **LEFT CONTROLLER** jack at the rear of your computer console. Place the label marked **RIGHT** over the Keyboard plugged into the **RIGHT CONTROLLER** jack at the rear of your computer console.

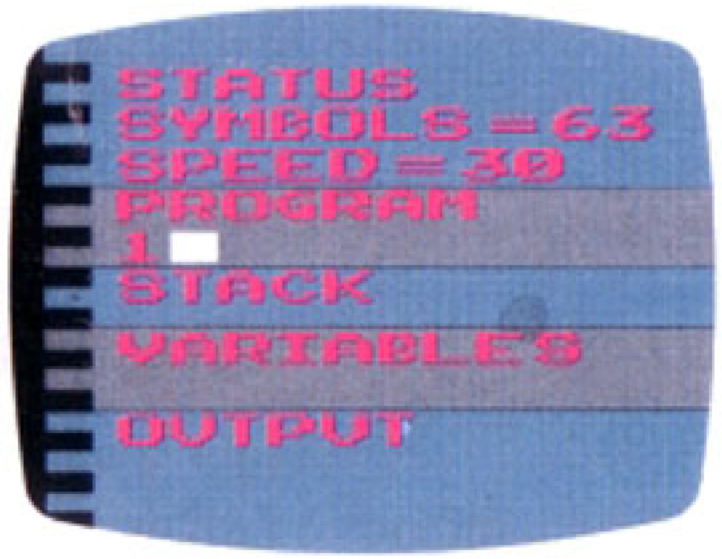
DISPLAY REGIONS

The display is divided into six regions:

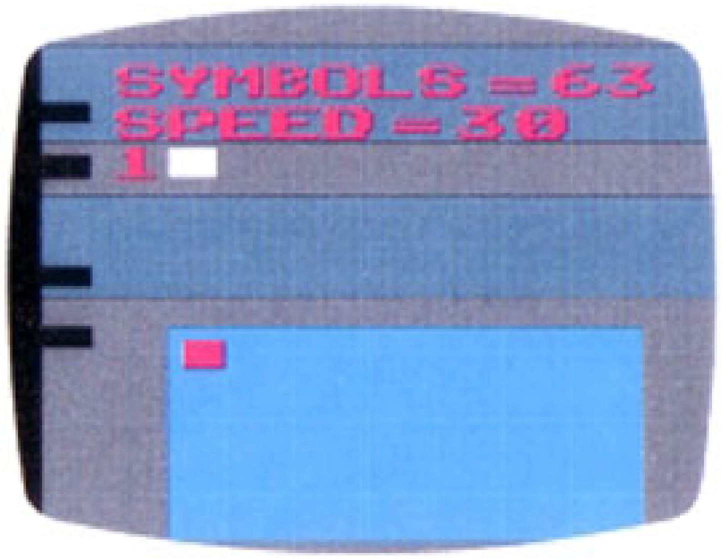
1. The **PROGRAM** region is used to give the computer your instructions.
2. The **STACK** region will give you the temporary results of your program as the computer is running it.
3. The **VARIABLES** region will give you the value of each variable item in your program as it is being run.
4. The **OUTPUT** region will show you the output being produced as your program is being run.
5. The **STATUS** region shows you the amount of memory available at any time for your program. This region will also give you the speed of execution of your program.
6. The **GRAPHICS** region has two colored squares which can be moved around under your program control.

Before beginning your program, place the **left difficulty** switch in the **b** position. This will show you where each region is on your display. When the **left difficulty** switch is placed in the **a** position, the titles of each region will disappear from the display and the GRAPHICS region will come into view.

The **right difficulty** switch has no function in this Game Program.



```
STATUS
SYMBOLS=63
SPEED=30
PROGRAM
1
STACK
VARIABLES
OUTPUT
```



```
SYMBOLS=63
SPEED=30
1
```

THE CURSOR

Place the **left difficulty** switch into the **b** position and turn your console **off** and then **on** again. In the **PROGRAM** region there is a white rectangle. This is the cursor. Locate the *shift control key* at the center of the bottom row on the left controller. Push this key four times. The color of the cursor will change from white to red, from red to blue, from blue to green, and from green to white again.

The cursor is used to input your program. Each of the four colors on the shift key corresponds to the colored commands or inputs on your Keyboard. The white mode is used to give your computer commands, the other modes are used to insert symbols into your program.

Moving the Cursor from Region to Region

Make sure the **left difficulty** switch is in the **b** position and turn your console unit **off** and then **on** again. Push the **FORWARD** key and the cursor will move from **PROGRAM** to **STACK**. Push the **FORWARD** key again and the cursor will move from **STACK** to **VARIABLES**. Push the key again and the cursor will move from **VARIABLES** to **OUTPUT**.

By pushing the **BACKWARD** key you can move the cursor back to the **PROGRAM** region. The **FORWARD** and **BACKWARD** keys can be pushed once for each region or held down.

Now place the **left difficulty** switch into the **a** position. The titles of each region will disappear and a portion of the **GRAPHICS** region will appear. Step the cursor through each of the regions again. Notice that the cursor will not move into the **GRAPHICS** region.

Removing the Regions from the Display

Place the **left difficulty** switch into the **b** position again and turn your console unit **off** and then **on**. On the left side of the Keyboard are a series of commands (white mode) that correspond with each of the regions: **STATUS, PROGRAM, STACK, VARIABLES, OUTPUT,** and **GRAPHICS.** Let's start with the **STATUS** region. Push the **STATUS** key once and the **STATUS** region will disappear from the screen and the **PROGRAM** region will move to the top of the display.

Push the **PROGRAM** key and the **PROGRAM** region will disappear and the **STACK** region will move to the top of the display. Push the **STACK** key and the **STACK** region will disappear and the **VARIABLES** region will move to the top of the display. Push the **VARIABLES** key and the **VARIABLES** region will disappear causing the **OUTPUT** region to move to the top of the display.

Remove the **OUTPUT** region by pushing the **OUTPUT** key. This will also cause the **GRAPHICS** region to be fully visible on the display. Push the **GRAPHICS** key to remove the **GRAPHICS** region. Your display should show no regions.

Now push the **STACK** key. The **STACK** region will reappear on your display. Remove the **STACK** region, and bring up the **OUTPUT** and **VARIABLES** regions. You may display or remove any region with the **left difficulty** switch in either the **a** or **b** position.

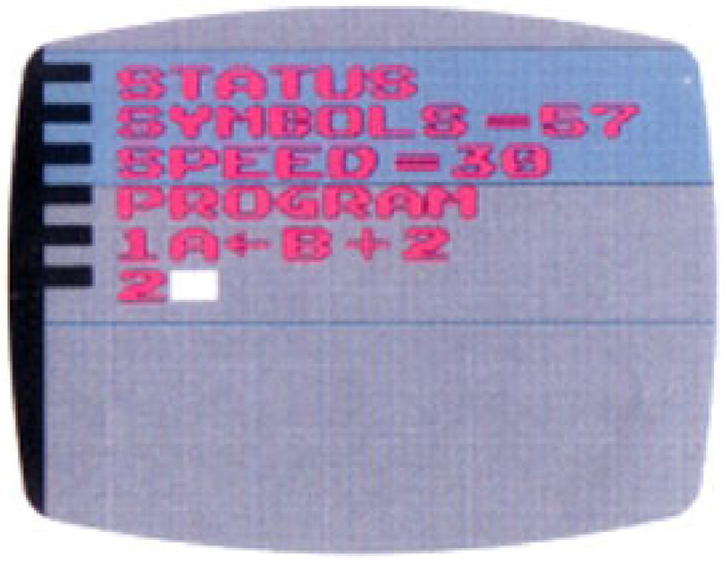
It is important to note that whether or not the various regions are displayed has no effect on the execution of a program.

RUNNING A PROGRAM

Let's do a simple program. Make sure the **left difficulty** switch is in the **b** position and turn your console unit **off** and then **on** again. (Another way to erase a program and reset all values is to depress the **game select** switch. Pushing the **game reset** switch erases all values and returns a program to its beginning without erasing the program.)

Remove the **STACK, VARIABLES, OUTPUT,** and **GRAPHICS** regions from the display. The cursor will be in the white mode and to the right of the number **1** in the **PROGRAM** region. Change the cursor to the blue mode and push the **A** key. The letter **A** will appear on the display next to the **1**. (Each line is numbered, making it possible to see where one line ends and the next one begins.) Now change the cursor to the red mode and push the ← key. A small arrow will appear next to the **A**. Now go back to blue and input **B**. Changing the cursor to red, input **+** and the number **2**. Now go back to the white mode and input **New Line**. You must always be in the white mode in order to start a new line in your program.

Your display should look like this.

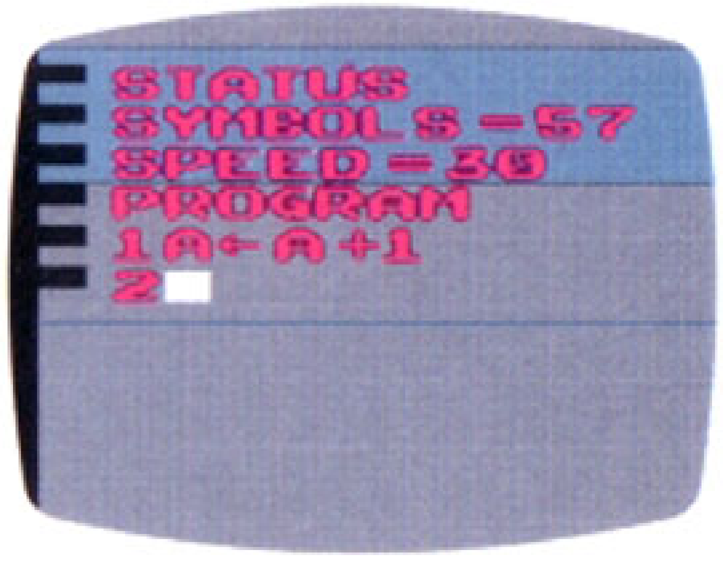


```
STATUS
SYMBOLS=57
SPEED=30
PROGRAM
1A←B+2
2
```

If you make an error in entering your program, it can be erased using the **ERASE** key. Notice that the **ERASE** key is **not** color coded. It can be used when the shift key is in any color mode.

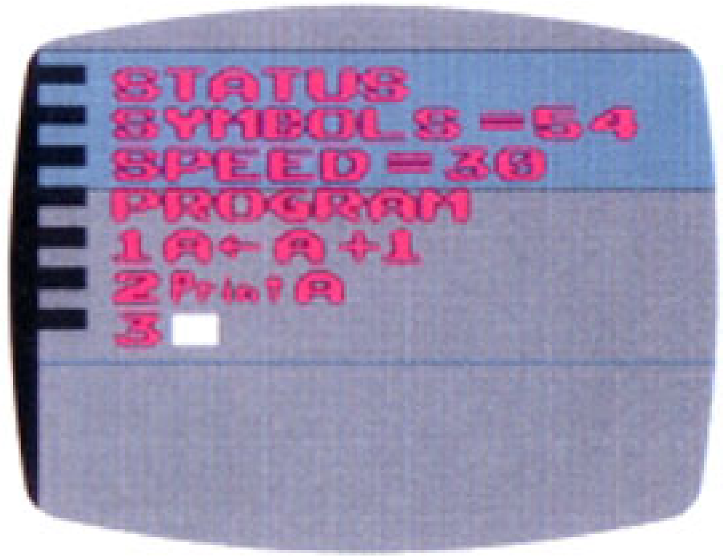
Using the line that you have just entered into your program, let's do an exercise. Push the **ERASE** key once. The cursor will move from line **2** directly to the right of the **2** on line **1**. Push the **ERASE** key again and the **2** will be erased from the program. Now enter **1**, using the red mode. Change the cursor to the white mode and using the **BACKWARD** key, move the cursor until it is directly to the right of the **B** in the program. Push the **ERASE** key and the letter will be removed from the program. Now replace it with **A** (blue mode). In the white mode, use the **FORWARD** key to move to the end of line **1** and input **New Line**.

Remember that the cursor should not be on the symbol to be erased but directly to the right of it. Your screen display will now look like this:



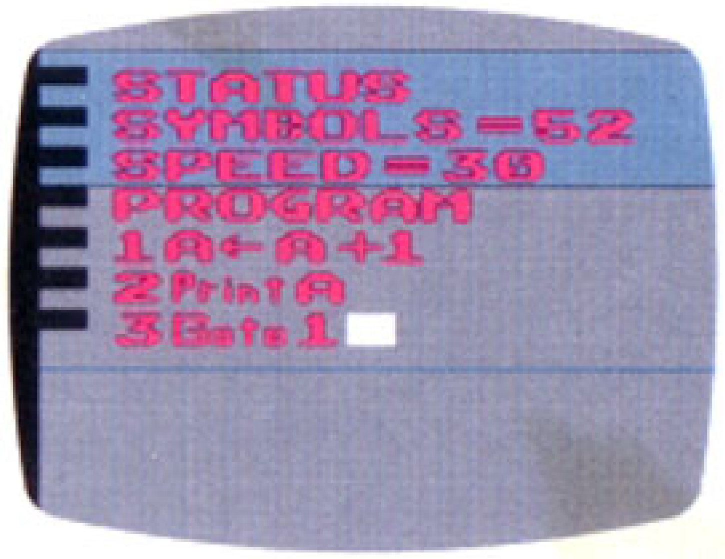
```
STATUS
SYMBOLS=57
SPEED=30
PROGRAM
1A←A+1
2
```

The cursor is directly to the right of the **2** in line **2**. Now input **PRINT** using the green mode. Then, with the blue mode, input **A** and go to a New Line. Your display will look like this:



```
STATUS
SYMBOLS=54
SPEED=30
PROGRAM
1A←A+1
2Print A
3
```

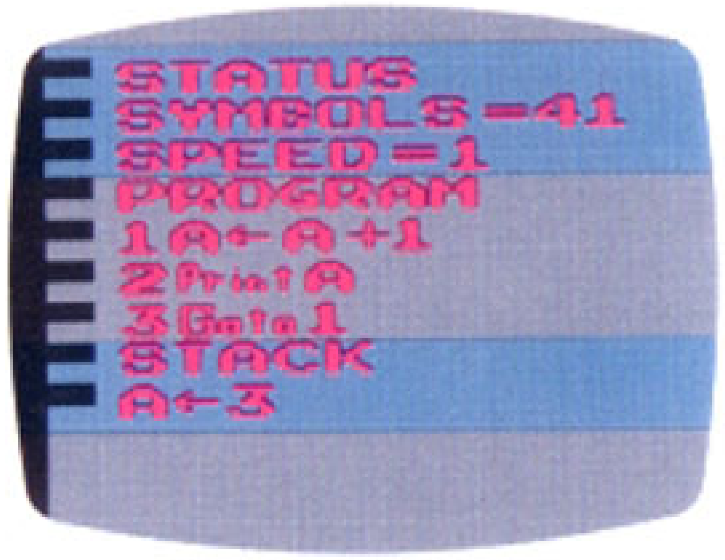
Now, with the cursor in the green mode, input **GOTO** and then, in the red mode, input **1**. Before we do anything else, let's look at the display again.



```
STATUS
SYMBOLS=52
SPEED=30
PROGRAM
1A←A+1
2Print A
3Goto 1
```

Notice that the **STATUS** region shows we have 52 "bytes" or symbols of memory remaining. The **SPEED** is set at 30 (**SPEED = 30**). Put the cursor into the white mode and depress the **SLOWER** key. Notice that every time you depress this key the speed decreases. Depress the **FASTER** key and the **SPEED** will increase.

Before we start the program, input **SPEED = 1**. Now depress the **STACK** key. Press the **RUN/HALT** key twice and the program will begin. You can watch the computer work each part of the program.



```
STATUS
SYMBOLS=41
SPEED=1
PROGRAM
1A←A+1
2Print A
3Goto 1
STACK
A←3
```

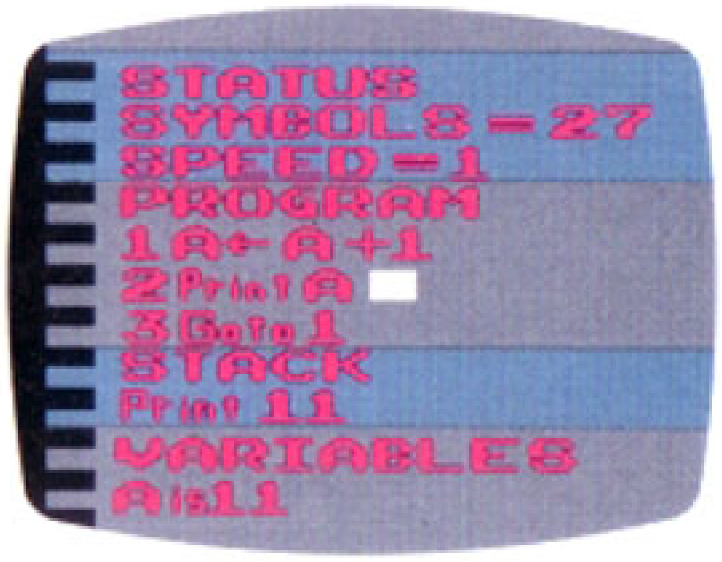
To stop the program press **RUN/HALT**. As noted before, pressing the **game reset** switch erases all values in your program (without erasing the program itself), and returns the program to its beginning.

Let's analyze the program step by step as the computer runs through it. Change **SPEED** to **60**. With the cursor in the white mode depress the **STEP** key. This will take each part of the program one step at a time, each time it is pressed.

In line **1** the program reads **A** ← **A + 1**. The computer reads this as **A** "becomes" **A + 1**. Watch the **STACK** region as you step through line **1**. In line **2**, you have told the computer to **Print A**. This means you want the computer to print the value of A (as determined by line 1) in the **OUTPUT** region.

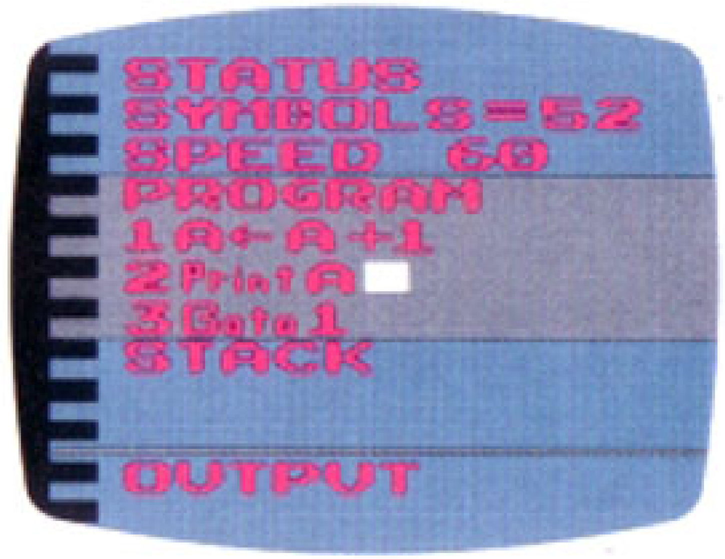
We'll see how that works later. Line 3 tells the computer to **Goto 1**. The computer is to return to line 1 and find a new value for **A**. Each time the program is run through it will find a new value for **A**, print that value, and then return to line **1**. The computer will continue to run through the program in this manner until all the "memory" is used. The amount of remaining memory is shown in the **SYMBOLS** area of the **STATUS** region.

Bring up the **VARIABLES** region on the display. Change the **SPEED** to **1** and clear the values of the program by pressing the **game reset** switch. Push the **RUN/HALT** key and watch as the computer runs through the program. The computer will show you the current value of **A** at each step of your program. Your screen display will look like this:



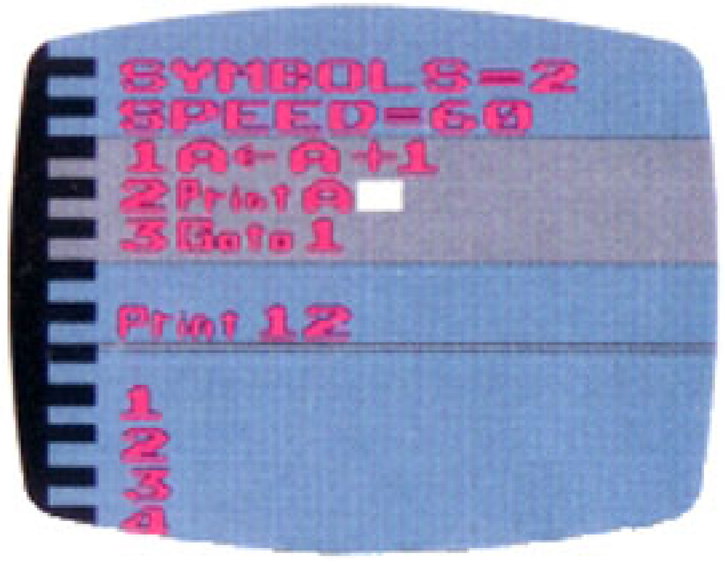
```
STATUS
SYMBOLS=27
SPEED=1
PROGRAM
1A←A+1
2Print A
3Goto 1
STACK
Print 11
VARIABLES
A is 11
```

Stop the program and clear the values (**game reset**). Remove the **VARIABLES** region and bring up the **OUTPUT** region on the display. Change **SPEED** to 60. The display will now look like this:



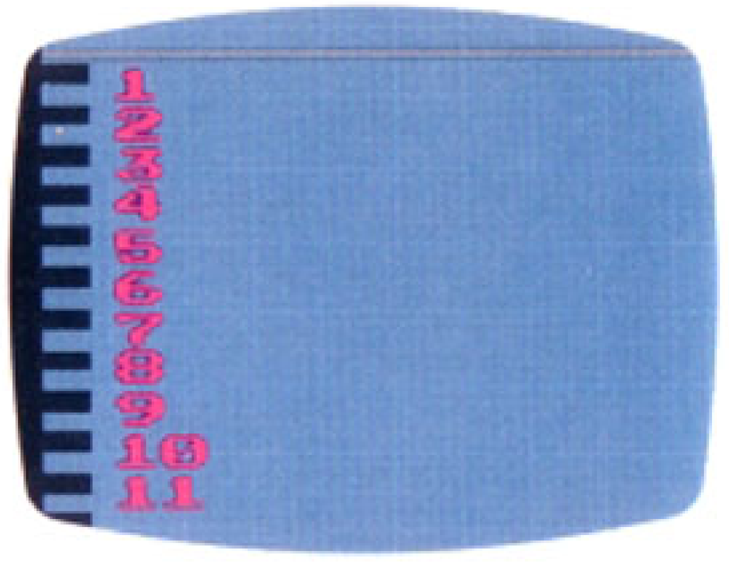
```
STATUS
SYMBOLS=52
SPEED 60
PROGRAM
1A←A+1
2Print A
3Goto 1
STACK

OUTPUT
```

Press the **RUN/HALT** key and start the program. In line **2**, the computer is instructed to **PRINT A**. When it reaches this command, it will print the current value of **A** in the **OUTPUT** region. Watch the **SYMBOLS** area of the **STATUS** region. Even though 1 shows in the **OUTPUT** region on your display, the computer is still printing the changing value of **A**. Put the **left difficulty** switch in the **a** position. Your display will now look something like this:



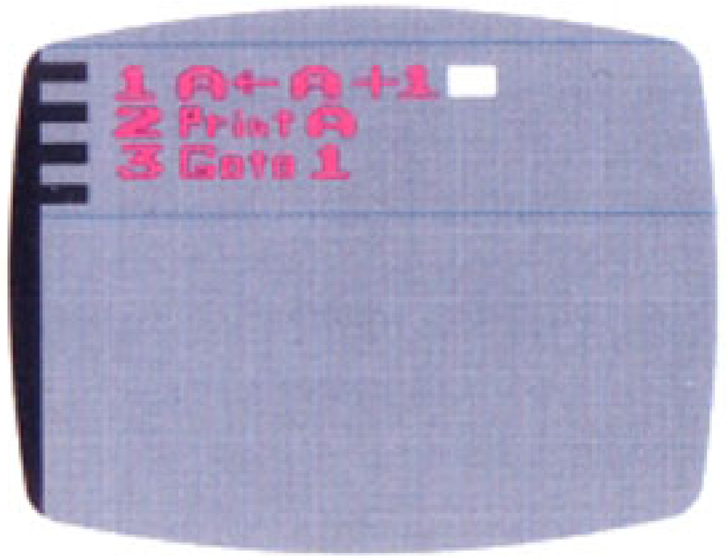
```
SYMBOLS=2
SPEED=60
1A←A+1
2Print A
3Goto 1

Print 12

1
2
3
4
```

How can we show more of the **OUTPUT** region on the display? Remove the **STATUS, PROGRAM,** and **STACK** regions from the display. Your display will look like this:



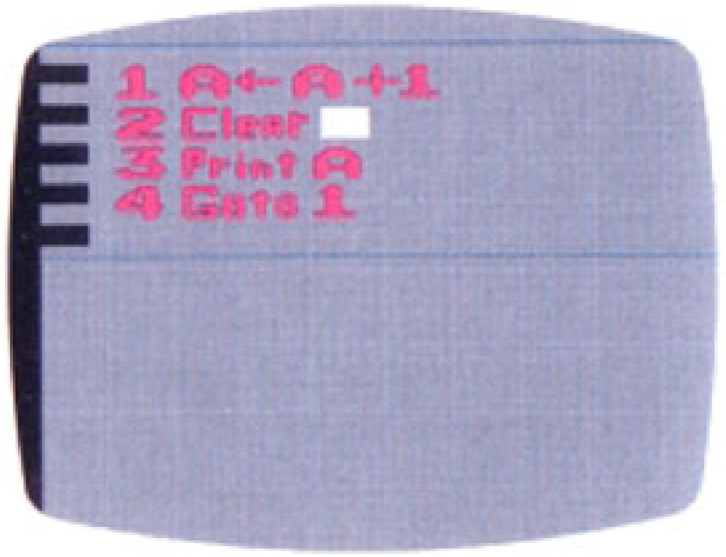
```
1
2
3
4
5
6
7
8
9
10
11
```

Suppose we don't want to overload the **OUTPUT** region of the program? Stop the program and erase the values with the **game reset** switch. Remove the **OUTPUT** region from the display and bring up the **PROGRAM** region. Using the **FORWARD** key, move the cursor to the end of line **1**, so that your display looks like this:



```
1 A←A+1
2 Print A
3 Goto 1
```

Now depress the **NEW LINE** key. We will insert a new command here. Input **CLEAR** (green mode). Your display will look like this:



```
1 A←A+1
2 Clear
3 Print A
4 Goto 1
```

Bring up **STACK, VARI-ABLES** and **OUTPUT**. Leave the **left difficulty** switch in the **a** position and start the program. As the program passes line **2 CLEAR** it erases the value of **A** in the **OUTPUT** region and then prints the next value of **A** in line **3**.

BASIC PROGRAMMING can only work with two-digit numbers, so as it reaches 99 it will "wraparound" and begin again showing A's value to be 0.

USING THE NOTE FUNCTION

Each time the program stores a number into **NOTE** (red mode), a note from the musical scale will be sounded by your television speaker.

Let's do some programs to demonstrate. If you have any programs in your Video Computer System, depress the **game select** switch on the console unit.

Input the following:

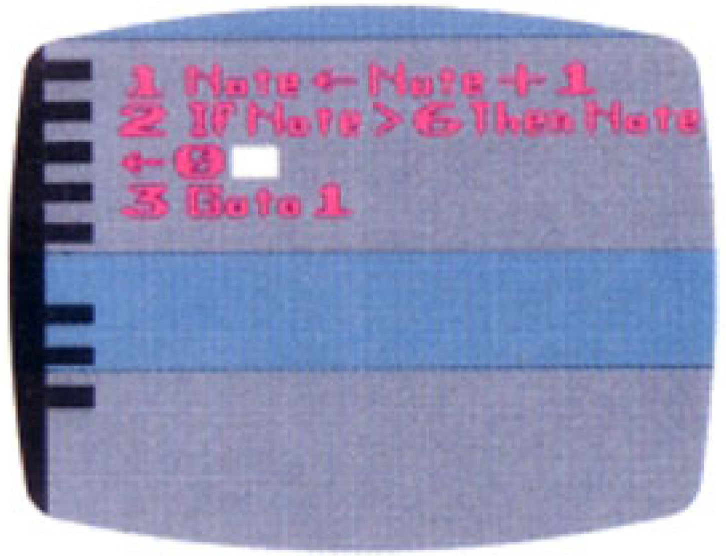
```
1   Note ← Note + 1
2   Goto 1
```

Now start the program. Slow the program down and watch it being worked in the **STACK** region. Also notice that the program will print the current value of **NOTE** in the **VARIABLES** region.

Stop the program and insert a new line **2** (in the white mode, move the cursor to the end of line 1 and press **NEW LINE**. The previous line **2** will become line **3**).

```
2   If Note  > 6 Then Note ← 0
```

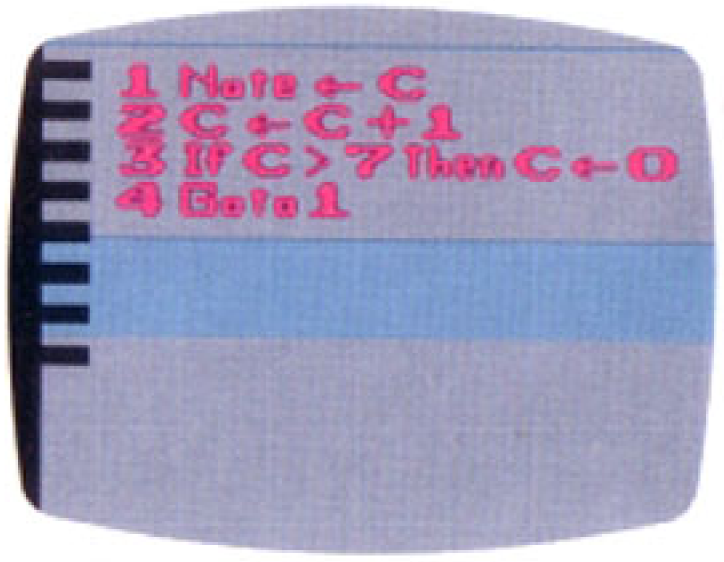
Your display will look like this:
(If **STACK** and **VARIABLES** regions are removed.)



```
1 Note←Note+1
2 If Note>6 Then Note
←0
3 Goto 1
```

With the **IF** and **THEN** commands, we are instructing the program that IF something happens THEN it is to do something else. In this case, **IF** Note is more than 6, **THEN** Note is to be changed to 0. Start the program and watch it being worked in the **STACK** region. Notice that as the value of Note reaches 7, it becomes more than (>) 6 and the program changes the value to 0.

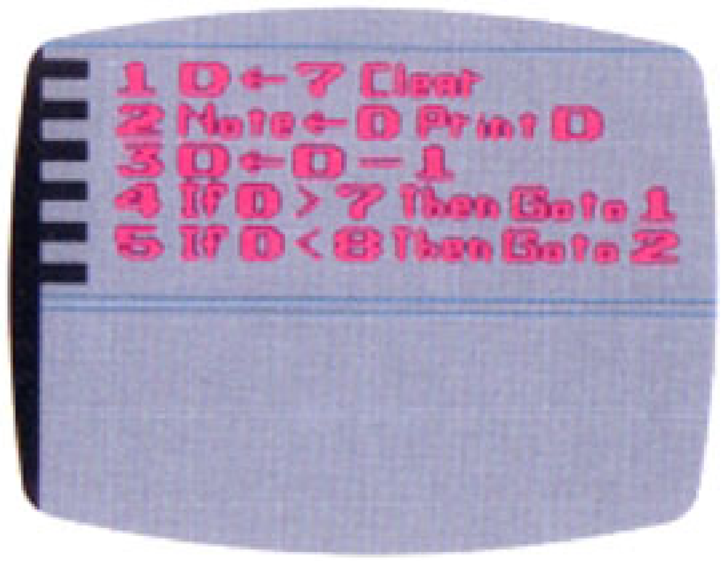
Let's do another program to achieve the same result in a different way. Input the following:



```
1 Note ← C
2 C ← C + 1
3 If C > 7 Then C ← 0
4 Goto 1
```

Run the program. Notice that the same results are achieved, except that the notes are spaced evenly. If you want the program to give you the value of **C** in the **OUTPUT** region, input **Print C** and **Clear** as instructions anywhere in the program. To cut down on the flicker in the **OUTPUT** region as the program is giving you the value, insert a comma, (green mode) after the value in the Print line, **Print C**.

You may use any letter of the alphabet for the variable in your programs. The following program has all the key functions we have learned so far and uses nearly all the available "memory" in the computer. After entering this program, remove the program from the display, bring up the **STATUS, STACK,** and **OUTPUT** regions, and run the program.



```
1 D ← 7 Clear
2 Note ← D Print D
3 D ← D − 1
4 If D > 7 Then Goto 1
5 If D < 8 Then Goto 2
```

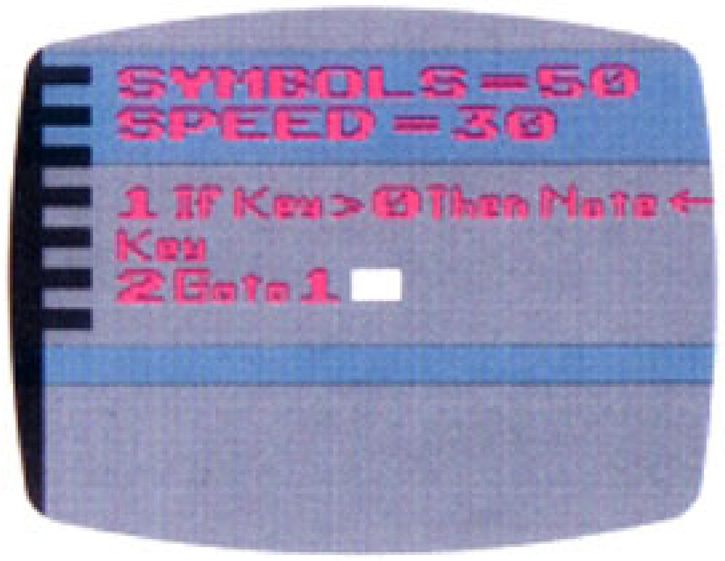
In this program we have given two **IF/THEN** commands. If the conditions set up in the first command are not met (line 4) the computer will move to the next line (line 5). Given enough memory, there can be several commands between any two **IF/THEN** commands.

In some cases we can put two commands on one line, as in line **1** and line **2** in the above program. In this way we can save some of the memory for later in the program.

USING THE KEY AND PRINT FUNCTIONS

The **KEY** function (red mode) is used to input a variable while the program is running. The program will evaluate **KEY** and replace it with a number that you input from the right side of the Keyboard. If no number is inputted, then the program will read **KEY** as **0**.

Here's a simple program using the **KEY** function:

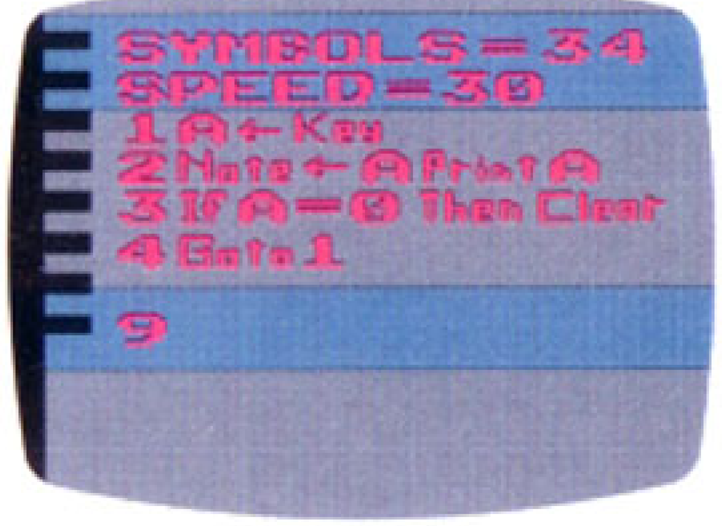


```
SYMBOLS = 50
SPEED = 30
1 If Key > 0 Then Note ←
Key
2 Goto 1
```

Start the program and push some of the keys on the right side of the Keyboard. With practice you may be able to play a tune.

The **KEY** function can also be used for programming in the **GRAPHICS** region as we will see later.

Try this program using the **KEY, NOTE,** and **PRINT** functions:



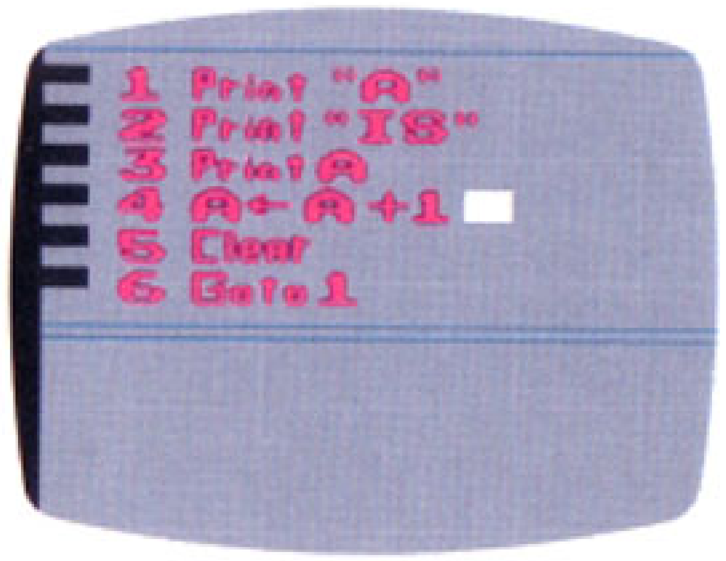
```
SYMBOLS = 34
SPEED = 30
1 A ← Key
2 Note ← A Print A
3 If A = 0 Then Clear
4 Goto 1

9
```

Watch this program in the **OUTPUT** region. Notice that as you input a number from the right side of the Keyboard the program will play that tone and display that number in the **OUTPUT** region.

Now let's make a minor modification in the program. In line **2**, insert a comma (,) after **Print A** and start the program. By inserting the comma here you have told the program that you want as many of the variables printed on one line as possible.

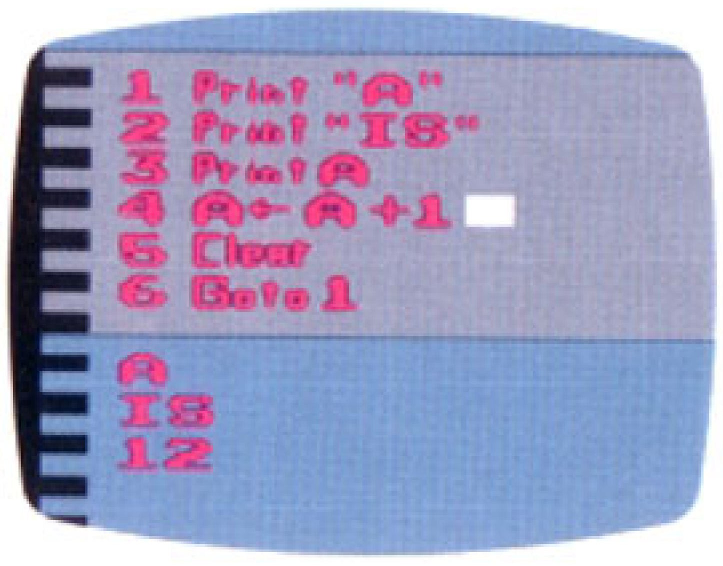
USING THE PRINT FUNCTION

As we showed in earlier programs, you can use the **PRINT** function to instruct the program to print the value of a variable in the **OUTPUT** region. The **PRINT** function can also be used to instruct the program to print words on the display. Words to be displayed must be enclosed in quotation marks ("). Set the **SPEED** at **8** and input the following program:



```
1 Print "A"
2 Print "IS"
3 Print A
4 A←A+1
5 Clear
6 Goto 1
```

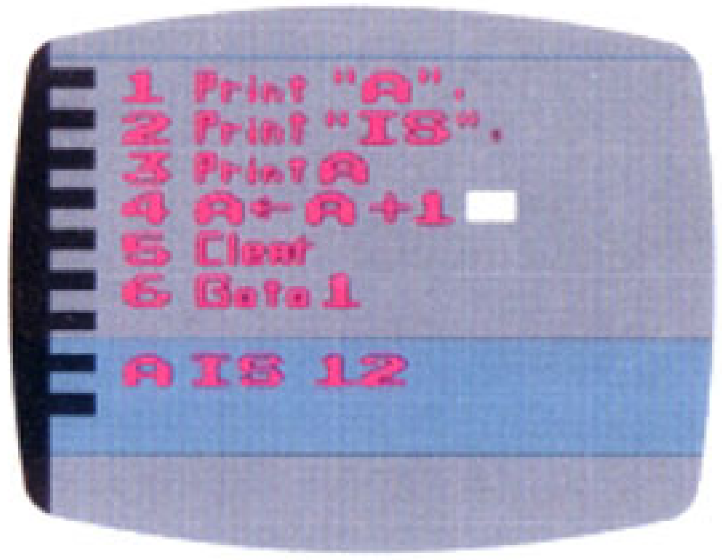
Now run the program and watch the **OUTPUT** region. The program prints the words you have inputted, but they are "stacked."



```
1 Print "A"
2 Print "IS"
3 Print A
4 A←A+1
5 Clear
6 Goto 1
```

```
A
IS
12
```

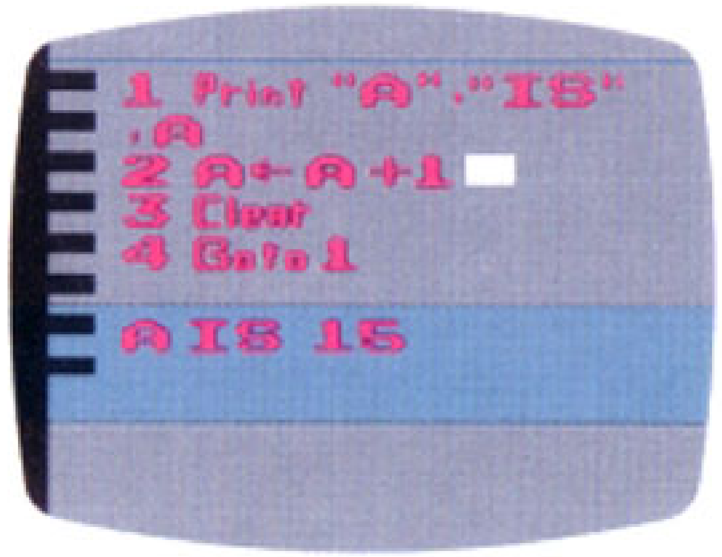
Let's change the program slightly by inputting a comma (,) at the end of line **1** and line **2**. The program now looks like this:



```
1 Print "A",
2 Print "IS",
3 Print A
4 A←A+1
5 Clear
6 Goto 1
```

```
A IS 12
```

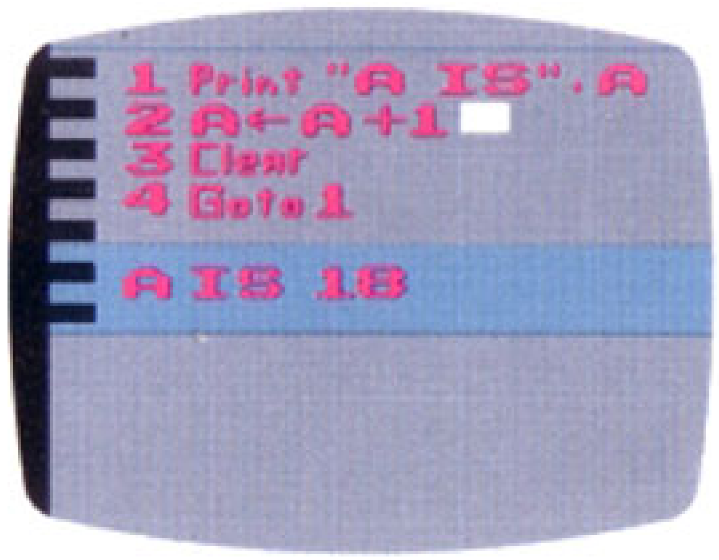
The comma (,) in line **1** has instructed the program that whatever is on line **2** is to be displayed in the **OUTPUT** region on the same line as the instruction on line **1**. The comma in line **2** has instructed the program that the line **3** instruction is to follow line **2**. These instructions can be shortened by changing the program to read:



```
1 Print "A","IS"
,A
2 A←A+1
3 Clear
4 Goto 1
```

```
A IS 15
```

The program can be further shortened:



```
1 Print "A IS",A
2 A←A+1
3 Clear
4 Goto 1
```

```
A IS 18
```

Writing the program in this fashion will also save some memory.

USING THE GRAPHICS REGION

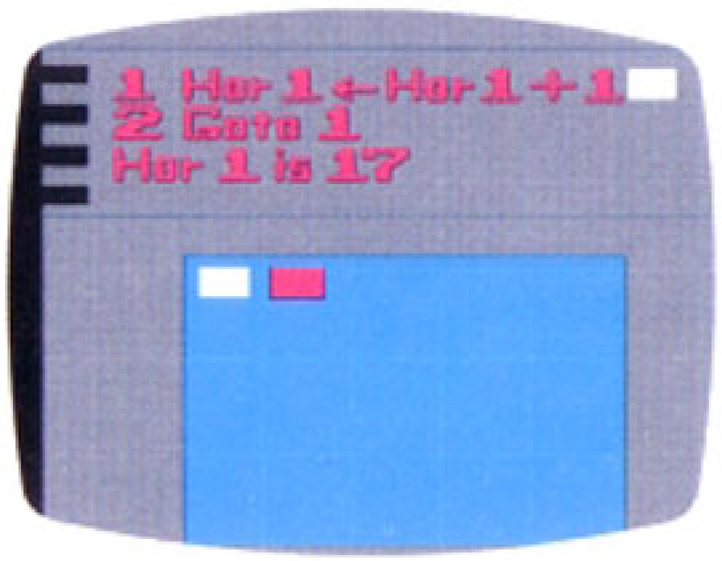
The **GRAPHICS** region is the blue rectangular field on your television screen. At first glance it appears to contain one red square in the upper left corner. The red square actually covers a white square and both may be moved independently on the field under your program control.

To move the squares you must change their coordinates. The red square is object number 1. Its horizontal coordinate is represented on the Keyboard by **Hor 1** and its vertical coordinate by **Ver 1. Hor 2** and **Ver 2** are the coordinates of object number 2, the white square.

Both objects or squares start out in the upper left corner of the blue field, with only the red square showing. The upper left corner is the zero (0) position or origin. When the variables **Hor 1, Ver 1, Hor 2,** and **Ver 2** are not defined (given values) they have values of 0 (therefore causing the squares to remain in the upper left corner position).

When a coordinate is assigned a value other than 0, (e.g. **Hor 1** ← **10**), the object involved will jump to the corresponding position on the blue field when the program is executed.

Input the following program:

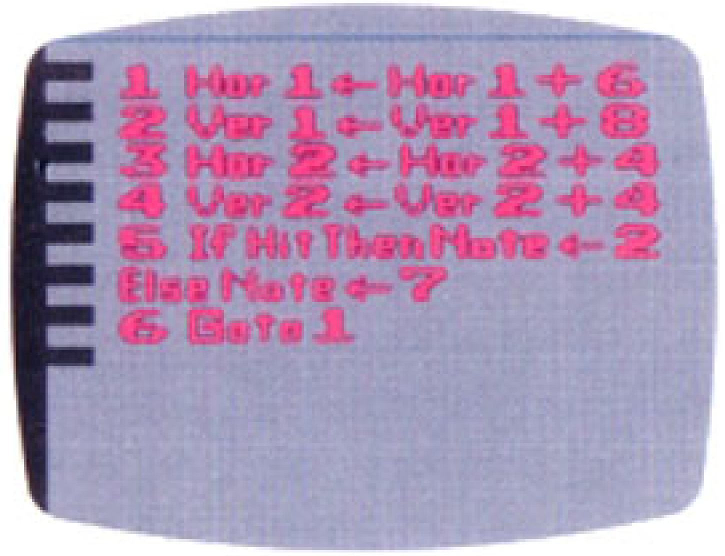


```
1 Hor 1 ← Hor 1 + 1
2 Goto 1
Hor 1 is 17
```

This program will move the red square to the right, one horizontal space at a time. Run this program and you will see the red square moving slowly to the right. Check the **VARIABLES** region and you will see the value of **Hor 1** increasing. It will increase until it reaches **99**, the largest number allowed, and then drop back to **0**. When **Hor 1** reaches **99**, the red square will have reached the far right side of the blue field. At this point, the red square "wraps around" and starts again from the far left side of the field, with **Hor 1** at **0**.

The horizontal coordinates (**Hor 1** and **Hor 2**) then, range from 0 to 99. The vertical coordinates (**Ver 1** and **Ver 2**) also range from 0 to 99, with 0 being the position at the top of the blue field, and 99 being the position at the bottom.

Input the following program:



```
1 Hor 1 ← Hor 1 + 6
2 Ver 1 ← Ver 1 + 8
3 Hor 2 ← Hor 2 + 4
4 Ver 2 ← Ver 2 + 4
5 If Hit Then Note ← 2
Else Note ← 7
6 Goto 1
```

Remove the program from the screen and make sure the **GRAPHICS** region is fully visible. Set the speed at **60** and run the program. This program shows you one way to move the squares on the field. It also reveals how the **HIT** and **ELSE** functions may be put to use.

In line **5** the program is instructing the computer to sound the **2** note **IF** the squares **HIT**, which they do periodically. To **HIT** means the squares must occupy the same coordinates. Otherwise (**ELSE**) the computer is instructed to play the **7** note, which it will do until the squares hit.

(Remove **ELSE NOTE** ← **7** from line **5** and note **2** will be played when the squares hit and no other note will be sounded.)

USING THE MOD FUNCTION

Mod is an arithmetic operator, much like the division operator (÷). **BASIC PROGRAMMING** employs integer division, which means it works with whole numbers only, and leaves no remainder. For example, what is 14 ÷ 5? You might say 2 and 4/5, or 2, remainder 4. In **BASIC PROGRAMMING** 14 ÷ 5 is 2. Although 5 divides into 14 twice (making 10), with 4 left over, **BASIC** uses whole numbers only, so the answer it gives you is 2.

The computer will give you the same answer for 12 ÷ 4 as it will for 13 ÷ 4. The answer in both instances is 3, since 4 will divide into both 12 and 13, 3 whole times. The computer will not recognize the remainder of 1 in the case of 13 ÷ 4.

Now for the **Mod** function. **Mod** gets the remainder left over after dividing the first number into the second number. So, **14 Mod 5** is **4**. Five divides into 14 twice (making 10) with 4 left over. **Mod** gets the remainder left over, therefore **Mod** is **4**.

What is **13 Mod 4**? The answer is **1** since **1** is left over after dividing **4** into **13** (which makes 12). How about **12 Mod 4**? In this case, **Mod** is **0** since 4 divides into 12 evenly with nothing or **0** remaining.

Normally, dividing by 0 is undefined, mathematically speaking. In **BASIC PROGRAMMING**, dividing by 0 just gives a result of 0, so that 5 ÷ 0 is 0.

Operator Priorities

In an equation, the arithmetic operators (+, −, ×, ÷, Mod, etc.) are worked out in order of an established priority. **BASIC PROGRAMMING** uses the following operator priority guide:

1. × ÷ (Highest)
2. + −
3. Mod
4. =
5. ← (Lowest)

Using Parenthesis

Using parenthesis (green mode, left side of Keyboard), gives priority to whichever numbers the parenthesis surround when working out an equation.

For example, when working out the following equation:

A 5 + 3 × 2 Mod 7

the first step is: 3 × 2 = 6
the second step is: 5 + 6 = 11
the third step is: 11 Mod 7 = 4
A = 4

However, the same equation with parenthesis inserted, is worked out differently and gives a different answer for A.

A (5 + 3) × 2 Mod 7

first step: 5 + 3 = 8
second step: 8 × 2 = 16
third step: 16 Mod 7 = 2
A = 2

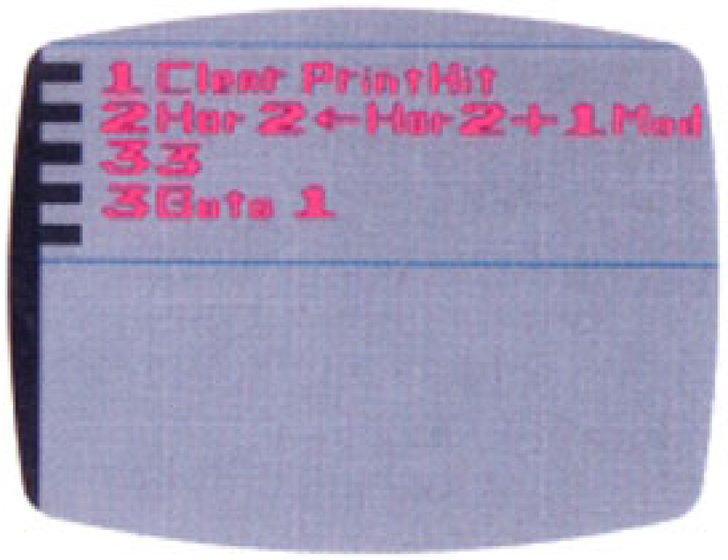
SAMPLE PROGRAMS

Here are some sample programs. They will help you see the wide range of possibilities you have to work with. Remember to pay close attention to how a particular command (**IF, THEN, HIT, ELSE, PRINT, KEY,** etc.) affects a program. Remember too that a **KEY** command may depend on your input from the right side of the Keyboard for successful operation of the program.

Whenever a particular program confuses you, stop the program, reset it to the beginning, and **STEP** through it. By watching the program in the **STACK** region and seeing how the computer works out each step, you should be able to understand what is happening and why it is happening.

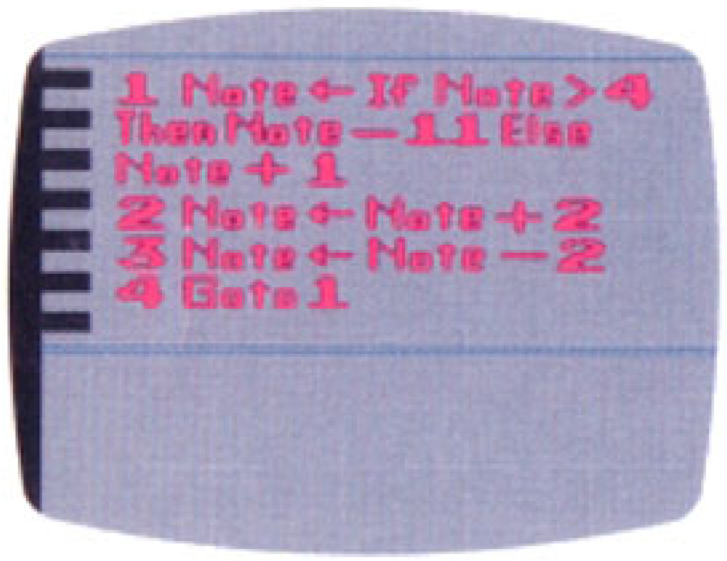
After running the programs and becoming familiar with the fundamentals of computer programming in general, you'll be well on your way toward writing some programs of your own.

Hit

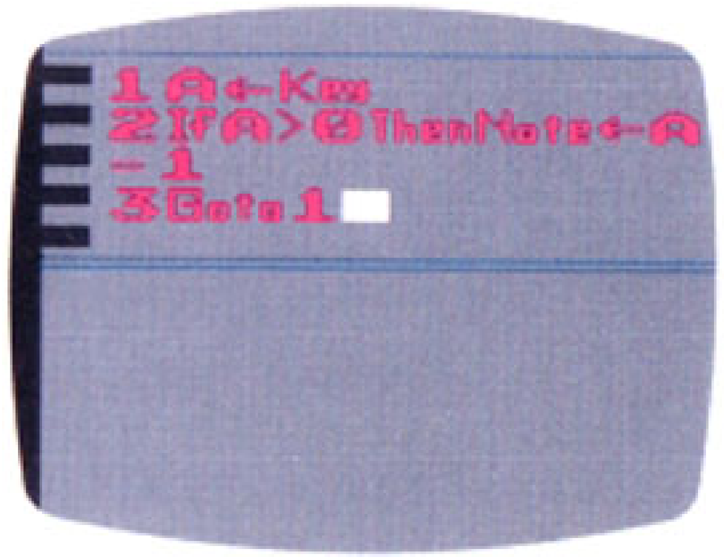


```
1 Clear PrintHit
2Hor 2←Hor 2+1Mod
33
3Goto 1
```

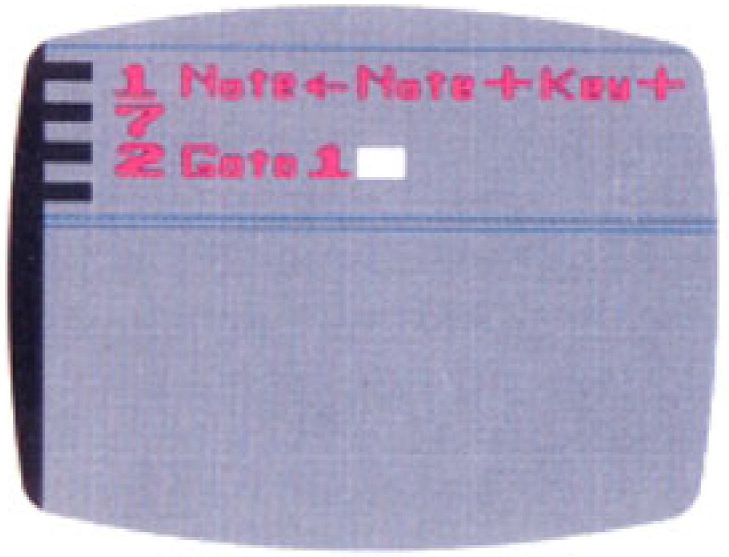
Music



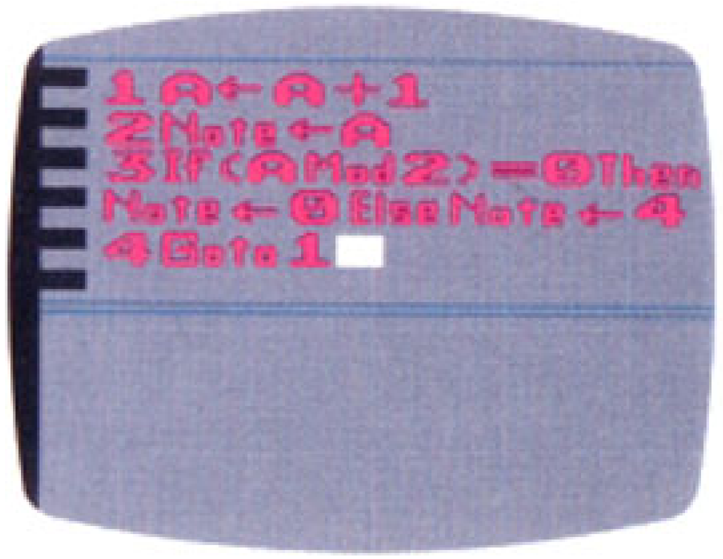
```
1 Note←If Note>4
Then Note−11 Else
Note+1
2 Note←Note+2
3 Note←Note−2
4 Goto 1
```



```
1 A←Key
2If A>0 Then Note←A
−1
3Goto 1
```



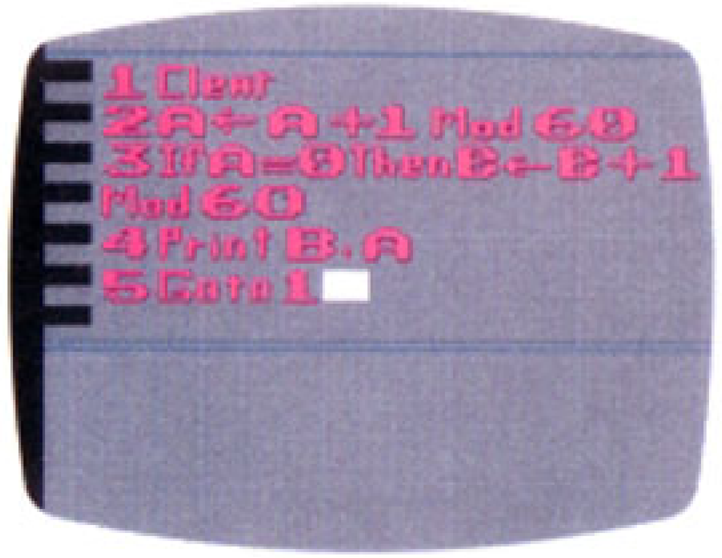
```
1 Note←Note+Key+
7
2 Goto 1
```



```
1A←A+1
2Note←A
3If (A Mod 2)=0 Then
Note←0 Else Note←4
4Goto 1
```

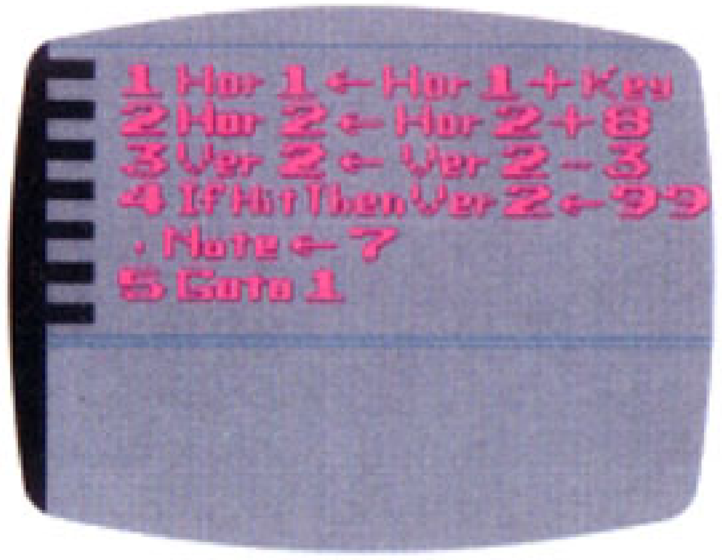
Clock-Like Program

When running the Clock program bring up the **OUTPUT** region only. The **SPEED** may be set at 30 or 60.



```
1 Clear
2A←A+1 Mod 60
3If A=0 Then B←B+1
Mod 60
4Print B.A
5Goto 1
```

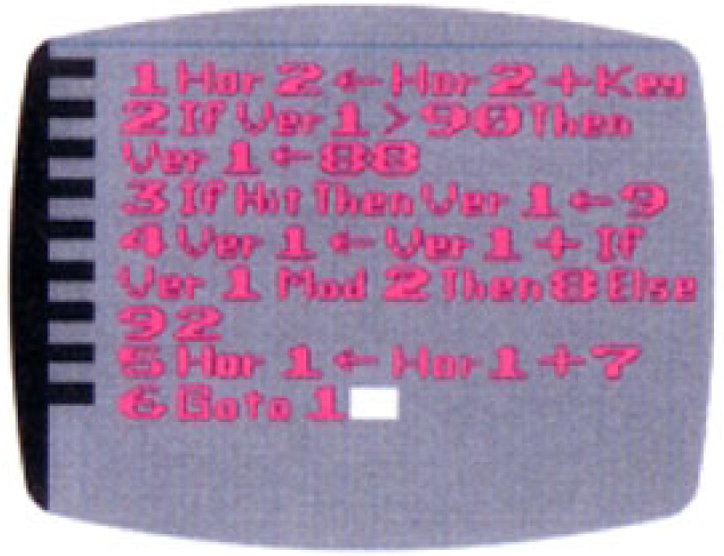
Pong® Game (Ball & Paddle)



```
1 Hor 1←Hor 1+Key
2Hor 2←Hor 2+8
3Ver 2←Ver 2−3
4 If Hit Then Ver 2←99
, Note←7
5 Goto 1
```

Pong® Game

without sound



```
1 Hor 2←Hor 2+Key
2 If Ver 1>90 Then
Ver 1←88
3 If Hit Then Ver 1←9
4 Ver 1←Ver 1+If
Ver 1 Mod 2 Then 8 Else
92
5 Hor 1←Hor 1+7
6 Goto 1
```

C011402-20 Rev. 2

Printed in USA