CHAPTER THREE

THE COMMAND SET

THE MARK III CONTROLLER COMMANDS

The command set of the XR Robot - Mark III Controller system has been designed to allow the complete control of the Mark III controller. Through the use of only 14 basic commands, the user can control the position of all eight motors, read any of the 16 input bits, set any of the 8 output bits and control the AUX ports. All of Rhino Robot's software uses these kernel commands to create the higher level languages, such as RoboTalk and Rhino-VAL.

The MARK III controller has the following commands:

Command	Description
<return>	Carriage return (initiate a move)
?	Return distance remaining
A-H	Set motor movement value
I	Inquiry command (read limit switches C-H)
J	Inquiry command (read limit switches A-B and inputs)
K	Inquiry command (read inputs)
L	Turn Aux #1 port ON
M	Turn Aux #1 port OFF
N	Turn Aux #2 port ON
O	Turn Aux #2 port OFF
P	Set output bits high
Q	Controller reset
R	Set output bits low
X	Stop motor command

The MARK III controller accepts commands as ASCII characters; each character is acted upon immediately upon receipt. Unlike most computer peripheral equipment, the controller does not wait for the receipt of a carriage return to signify a command completion; in fact the carriage return is considered a command itself.

When the motor move command letters **A-H** are received, the motor specifier is stored in the motor buffer over writing its previous contents. The receipt of a motor specifier also sets the direction buffer to the plus direction and clears (sets to zero) the move count buffer. Any sign character (+ or -) is stored in the direction buffer over writing its previous contents. When a number digit is received, the contents of the move count buffer is multiplied by ten and the new digit is added to the product. In this way the move count buffer correctly accumulates a multi-digit number.

When a carriage return character is received, the controller adds or subtracts the amount in the move count buffer to the value in the error register pointed to by the motor buffer. If the direction buffer has been set to a plus the amount is added; if the direction buffer has been set to a minus the amount is subtracted.

The following illustration shows the relationships of the input buffers and the motor error registers. In this example, a **C** command was received followed by a **-50**. When a carriage return is received, the C motor error register will be decremented by 50.

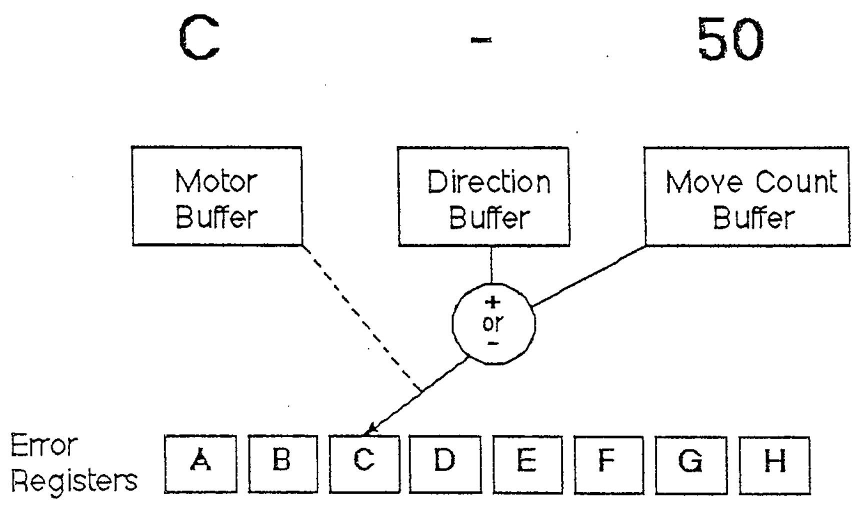

Figure 3.1
Processing of a Motor Move command

The receipt of a carriage return can have no effect on the controller if the move count buffer is zero, as it is after a motor specifier (A-H) has been received. You can take advantage of this fact if you want to terminate all commands sent to the MARK III with a carriage return which would be the case if you were using standard PRINT statements in BASIC to control the robot. Preceding all commands with a motor specifier allows you to use the carriage return.

In the following command descriptions, the format and the examples of the commands will illustrate the use of the leading motor specifier. This will lead to a more intuitive understanding of the command set.

<return> Initiate a motor move

Whenever a carriage return is received from the host computer, the controller takes the move count in the motor move count buffer and adds it to (or subtracts it from depending on the sign of the direction buffer) the value in the error register of the motor that is addressed by the motor buffer.

Thus, if the command sequence **C-40 <return>** is sent to the controller, a **C** will be stored in the motor buffer, a **minus** will be stored in the direction buffer and **40** will be stored in the move count buffer. Upon receipt of the <carriage return>, this data is transferred to the **C** error register and the controller will start the **C** motor in the negative direction with the intention of moving it 40 additional encoder steps in that direction. If the controller now receives a carriage return only, it will add another **-40** to the error register for the **C** motor. With the above sequence of commands, the motor will eventually make a move of **-80** total encoder counts.

? Question command

Requests steps remaining to move on motors A thru H.

When making long moves it is necessary to determine how far a motor has to move before more move information can be sent to the controller. The command used to determine how far a motor has yet to move is the Question command.

The format of the command is

```
[<motor ID>]<?> <return>
```

Examples of the command as issued from a BASIC program:

```
PRINT "D?" <return>
PRINT "A?" <return>
PRINT "C?" <return>
```

where the first letter identifies the motor error register to be interrogated and the question mark indicates that the remaining error count for that motor is to returned to the host computer. As always, the controller adds 32 to the error value before returning it to the host computer. Adding 32 to the count prevents the controller from sending ASCII command codes to the host computer. The controller always sends the absolute value of the error signal; it does not send the direction of the error signal. This means that you can determine how far a motor still has to move but cannot determine whether the move is to be in the positive or the negative direction.

NOTE: Using the "?" command does not re-issue the move in the move buffer because the motor ID letter preceding the "?" always clears the move buffer.

A to H Start motor commands

Starts motors A to H and moves them a number of encoder steps

The start command is used to instruct the controller to start a given motor and to move it in a given direction by a given number of encoder steps. The value is added to the move in progress.

The format of the command is

<motor ID>[<sign>]<encoder counts><return>

where <motor ID> is an uppercase letter A-H, <sign> is an optional + or - character (if no character is present a + is assumed) and <encoder counts> is a number from 0 to 127.

Example: for programming in BASIC:

```
PRINT "C-93"
```

The above command will move the "C" motor in the negative direction by an additional 93 encoder positions. The positive sign is not needed for moves in the positive direction and may be omitted. The carriage return is used to execute the command. Only one motor can be addressed at one time. Other samples of the command are:

```
B-6 <return>
A+21 <return>
C33 <return>
D-125 <return>
```

H <return>	(clears the move count buffer, therefore no move is made)
<return>	(repeats the previous move)

Sending only a carriage return without a motor move, after a motor move command, repeats the last motor move command. Even carriage returns that are part of another command (discussed later) will re-issue the move command. In order to cancel a move command that would be carried out by the receipt of a carriage return, all commands should be preceded by a motor ID character (A thru H). Any motor ID letter sets the move buffer to zero if there are no numbers attached to it. Once the move buffer is cleared, other commands with carriage returns will not activate the move instruction. For example the commands similar to the following will clear the move buffer.

```
A <return>
CI <return>
EJ <return>
BL <return>
```

Once the move buffer is cleared, other commands may be issued without the motor ID characters. The move buffer is active after each non-zero motor move command. It should be cleared before using commands when necessary.

I I-Inquiry command

Returns status of microswitches on motor ports C, D, E, F, G, and H

The INQUIRY command allows the user to interrogate the status of the 6 microswitches on the C, D, E, F, G and H motors. The format of the command is as follows:

[<motor ID>]<I><return>

Where the motor ID is included if the move buffer is to be cleared.

Sample uses of the command as issued in a BASIC program:

```
PRINT "I" <return>
PRINT "AI" <return>          (Clears the move buffer first)
```

The command returns the status of the 6 microswitches in one byte. The returned byte is interpreted as follows **after subtracting 32:**

Bit	Motor
0 LSB	C
1	D
2	E
3	F
4	G
5	H
6	---
7 MSB	---

The controller adds decimal 32 to the byte before transmitting it to the host computer so that no control codes will be transmitted to the host computer. Upon receipt of the returned byte, it is the users responsibility to subtract 32 from the byte before using it. A closed microswitch is seen as a 1 (one). An open microswitch is seen as a 0 (zero).

Bits 6 and 7, the most significant bits, are not used.

The host computer must be ready to receive the inquiry byte before sending the controller another command. See detailed examples under the sections on running the robot with the Apple IIe and the IBM-PC.

J J-Inquiry command

Returns status of microswitches on motors A and B and input lines 1, 2, 3 and 4

The **J-INQUIRY** command tells the controller to send back the status of the microswitches on motors **A** and **B** and the status of input lines **1, 2, 3** and **4** of the 8 line input port. The returned data byte is of the form 00BA4321+32, where **A** and **B** are the levels of the **A** and **B** motor limit switches and **4, 3, 2, and 1** are the levels of the input lines **4** through **1**. As with all other information returned to the host computer, 32 is added to the returned value to ensure that no ASCII control character is returned to the computer. You use the **J-INQUIRY** command just like the **I-INQUIRY** command.

The format of the command is

[<motor ID>]<J> <return>

where the <motor ID> has to be included if you want to clear the move buffer.

Examples of the command as issued from a BASIC program:

```
PRINT "J" <return>
PRINT "CJ" <return>          (Clears the move buffer first)
```

When interpreting the bits returned by the controller, a value of 0 means that the input is low (or that a microswitch is closed). A value of 1 means that the input is high (or that a microswitch is open).

Bit	Meaning
0 LSB	Input 1
1	Input 2
2	Input 3
3	Input 4
4	Motor "A" Limit Switch
5	Motor "B" Limit Switch
6	---
7 MSB	---

Bits 6 and 7 are not used

The controller adds decimal 32 to the byte before transmitting it to the host computer so that no ASCII control codes will be transmitted to the host computer.

The host computer must be ready to receive the inquiry byte before sending the controller another command. See detailed examples under the sections on running the robot with the Apple IIe and the IBM-PC.

K   K-Inquiry Command

Returns the status of input lines 5, 6, 7 and 8.

The **K-INQUIRY** command tells the controller to send back the status of inputs 5 through 8 of the 8 line input port. The returned data is of the form 00008765+32, where 8, 7, 6, and 5 are the levels of input lines 8 through 5. The format and usage of the K command is similar to the I and J commands.

The format of the command is

```
[<motor ID>]<K> <return>
```

where the <motor ID> has to be included if you want to clear the move buffer.

Examples of command as issued form a BASIC program:

```
PRINT "K" <return>
PRINT "EK" <return>          (Clears the move buffer first)
```

The command returns values that may be interpreted as follows after subtracting 32 from the byte received.

Bit	Meaning
0 LSB	Input 5
1	Input 6
2	Input 7
3	Input 8
4	---
5	---
6	---
7 MSB	---

Bits 4, 5, 6 and 7 are not used

The controller adds decimal 32 to the byte before transmitting it to the host computer so that no ASCII control codes will be transmitted to the host computer.

The host computer must be ready to receive the inquiry byte before sending the controller another command. See detailed examples under the sections on running the robot with the Apple IIe and the IBM-PC.

L Turn Aux. Port #1 ON

Turns Aux Port #1 ON

The **L** command turns auxiliary port 1 ON. Auxiliary port 1 provides 1 amp at -20 volts DC. The forward/reverse switch above the aux. connector determines the polarity of the pins and can be used to reverse a PM DC motor connected to the port.

The format of the command is

[<motor ID>]<L> <return>

where the <motor ID> has to be included if you want to clear the move buffer.

Examples of command use as issued in a BASIC program:

```
PRINT "L" <return>
PRINT "CL" <return>          (Clears the move buffer first)
```

M — Turns Aux. Port #1 OFF

Turns Aux Port #1 OFF

The **M** command turns auxiliary port 1 OFF. You use it just as you would the **L** command. See description of **L** command.

N — Turns Aux. Port #2 ON

Turns Aux Port #2 ON

The **N** command turns auxiliary port #2 ON. Auxiliary port #2 provides 1 amp at +20 volts DC. The forward reverse switch above the aux. connector determines the polarity of the pins and can be used to reverse a PM DC motor connected to the port.

The format of the command is

[<motor ID>]<N> <return>

where the <motor ID> has to be included if you want to clear the move buffer.

Examples of command use as issued by a BASIC program:

```
PRINT "N" <return>
PRINT "CN" <return>     (Clears the move buffer first)
```

O — Turns Aux. Port #2 OFF

Turns Aux Port #2 OFF

The **O** command turns auxiliary port 2 OFF. You use it just as you would the **N** command. See description of **N** command.

P — Set Output Line High

Sets an Output line HIGH

The **P** command tells the controller that the next digit it receives identifies the output line to be set **high.** The 8 output lines of the output port are numbered from 1 to 8. The output lines are set high during startup and after a reset. The MARK III controller output lines provide TTL level signals.

The format of the command is

[<motor ID>]<P><output line number> <return>

where the <motor ID> has to be included if you want to clear the move buffer.

Examples of command use as issued by a BASIC program:

```
PRINT "P3" <return>
PRINT "AP6" <return>          (Clears the move buffer first)
```

Q — Reset

Resets the entire Controller

The **Q** command tells the controller to reset itself. The controller will clear all of its internal registers, turn off all motors, turn off all the auxiliary ports, set all output lines high and reset its communication port according to the BAUD switch in the controller.

The **Q** command is a convenient way of resetting the controller (in software) without having to press the reset button.

The format of the command is

<Q> <return>

Examples of command use as issued in a BASIC program:

```
PRINT "Q" <return>
```

Complicated software programs often start with the "Q" command to ensure that the controller is at a known (reset) state when the program starts.

R — Set Output Line Low

Sets an Output line LOW

The **R** command is similar to the **P** command but the next digit received after the **R** identifies the output line to be set **low** by the controller. The 8 output lines are numbered from 1 to 8. The output lines are set high during startup and after a reset. The MARK III controller output lines provide TTL level signals.

The format of the command is

[<motor ID>]<R><output line number> <return>

Examples of command use as issued by a BASIC program:

```
PRINT "R5" <return>
PRINT "CR2" <return>          (Clears the move buffer first)
```

X Stop motor command

Stops motors A thru H

It is often necessary to turn off a motor that is stalled. One way to do this is to determine how far the motor is from completing it's move and then sending a move command that will reverse the motor far enough to cancel the remaining portion of the move. A faster way is to send the stop command.

The format of the stop command is

```
<motor ID><X> <return>
```

Examples of the command as issued from a BASIC program:

```
PRINT "BX" <return>
PRINT "DX" <return>
PRINT "HX" <return>
PRINT "AX" <return>
```

where the first character identifies the motor to be stopped and the "X" is the stop command. The "X" command is followed by a carriage return and does not re-issue the preceding move command because the motor ID letter clears the buffer. When the "X" command is received, the remaining portion of the motor move, (the portion that was still to be moved,) is lost and cannot be recovered. If the information is important, the user should first determine how far the motor still has to go with the "?" command, store the information in the host computer and then send the "X" command to stop the motor.

USING THE COMMANDS

Detailed instruction and examples on how to use the commands are given in the sections on using the Apple IIe and the IBM-PC as the host computer. The examples given send the command in the context of a subroutine, although ordinary Print, Peek and Poke commands and machine code (assembly language) routines can also be used to send and receive information. The commands used will depend on the language that you are using. See the manual for your particular computer and the language that you are using to see how these commands can be used.

The fastest way to send commands and receive information is with the use of assembly language (machine code) programming. Sophisticated users interested in developing complicated command structures for the Rhino XR system will use machine code, however, the scope of this manual can not cover machine code programming.

Two fairly complicated **sample programs** are provided on disk with the XR system to show beginning users how to program complicated programs in machine code and/or in BASIC. Interested experimenters will do well to study these programs at length. One of the programs (the teach pendant emulator) actually emulates the teach pendant on the keyboard of the computer. The other (XYZ) is an XYZ program that controls the robot in XYZ coordinates. Of course your entire program does not have to be in machine code. You can take useful machine code routines out of the sample programs and incorporate them into your BASIC programs.

RoboTalk™

Those users wishing to use a **higher level language** to control the XR system can use **RoboTalk™** to do all their programming. RoboTalk is provided with each system. See your RoboTalk Manual.

Rhino-VAL

Those users wishing to use an **industrial robotic language** to control the XR system can use Rhino's emulation of **VAL™**, the Unimation language, to do their programming. Rhino-VAL is provided on disk and versions are available for the Apple IIe and the IBM-PC.

CHAPTER FOUR

RUNNING THE XR WITH ANY HOST COMPUTER

If you have one of the computers listed below, we suggest that you skip directly to one of the two following chapters that covers your computer.

COMPUTER	CHAPTER
Apple IIe	Five
IBM-PC	Six

Each chapter provides step-by-step instructions on how to connect the controller to the computer, a tutorial on the fundamental controller commands, and important programming techniques specific to the XR. Each section is specific for the computer covered.

If you own either of the computers listed above, you also received a software package designed for your machine. If your immediate concern is running the XR, and not programming, we recommend that you refer to the manual that came with the software. Then, if you want to learn more about programming, return to the section that covers your computer.

If you don't own one of the computers covered individually, go to the following pages for the general information you need to interface your computer to the RHINO system. We suggest that you read these pages and then skip to the section which covers a computer that most closely resembles your own. For example, if you have an IBM compatible machine, skip to the section on the IBM PC. Then adapt the sample programs to your own computer.

GENERAL INTERFACING INSTRUCTIONS

Electrical connections:

No matter what computer you use as a host, it must have an RS-232c serial interface, capable of both sending and receiving data. The Mark III Controller uses only three of the 25 communication lines on the DB 25 connector:

Line 2 carries data transmitted by the controller, received by the host computer

Line 3 carries data received by the controller, sent by the host computer

Line 7 is the common data ground line.

Therefore, your computer's RS-232c connections must be configured as follows:

Line 2 should be configured to receive data
Line 3 should be configured to transmit data
Line 7 should be configured to be the data ground

Handshaking:

The Mark III Controller does not use a handshake protocol. Therefore, the DB25 connectors must be modified. Jumper wires must be soldered to connector pins so that, essentially, your computer shakes hands with itself.

Usually, jumpers should be soldered between pins 4 and 5 and between pins 6, 8 and 20. To know exactly what to do, you will have to read the manual for the RS-232c port for your particular computer. You may be able to use the data cable supplied with the Rhino robot system.

Consult your computer manual to determine what signals it requires, you may have to make further modifications. Remember to treat the controller as a serial (ASCII) I/O device.

DATA FORMAT

Configure your computer's RS232c port for the following data format:

9600 Baud
7 data bits
even parity
2 stop bits

Although it depends on your computer, you'll probably have to set up the data format with software instructions at the start of each of your programs. You can find illustrations of how to do this in the sections covering the individual computers.

OPEN THE PORT

You also have to configure your computer so that commands are routed to the proper port. Using your computer manual, set up as if your computer were transmitting to an ASCII serial I/O device. This may require some switch setting, but more likely it will require some software instruction. Again, consult your computer manual.

You may be able to use the information provided in this manual for the IBM-PC and the Apple IIe as a guide to help you.

CHAPTER SIX

RUNNING THE XR WITH AN IBM-PC

Interfacing with the IBM-PC

You need an IBM Asynchronous Serial Communications Card and an RS-232C cable with an IBM Adapter Plug for proper connection between your IBM-PC and the Mark III controller. RHINO provides the cable and the adapter plug with every Mark III Controller. The customer provides the serial card which can be installed in any of the expansion slots of the IBM PC; configure the card for COM1. Install the data cable as follows:

1. Be sure that the main power and motor power switches on the controller are **off**, and that the IBM-PC power is **off**;
2. Plug the female connector of the IBM adapter plug into the RS-232C port of your IBM-PC. Plug the data cable into the other end of the IBM adapter.
3. Connect the data cable into the port on the front of the controller labeled **"Computer"**.

Setting Up Communications

The following steps need to be taken to set up communications between your IBM-PC and the Mark III Controller

1. Insert your IBM system DOS master disk into the left drive and turn on the PC.
2. Place the Mode Switch on the front panel of the MARK III controller to the COMPUTER (down) position and turn on the **main power** on the controller.
3. Turn on the **motor power** on the controller.
4. Press the **RESET** button on the Mark III controller.
5. Load **BASICA** after the DOS disk is booted.

If you have to stop the arm because it is going to strike an obstacle, or for any emergency, press the reset button on the Mark III Controller. With this in mind, let's go on.

Whenever you program with your IBM-PC, you must start by setting up communications between your computer and the controller. Start by typing in the following line.

```
10 OPEN "COM1:9600,E,7,2,CS,DS,CD" AS #1
```

This line opens the communications port number 1 as file number 1 and sets the BAUD rate and data format. It also disables time out errors.

Move The Motors With The START Command

Now, to make your XR move. Type in:

```
20 PRINT #1,"F+50"
30 END
```

Watch your XR as you run the program. Your XR will move 50 encoder holes on it's waist axis and stop.

You used one of the four fundamental commands you can send to the controller, the START command.

The "F" designated the motor.

The "+50" designated the direction (+) and the distance (50 encoder holes).

Now change the sign in line 20 so it looks like this:

```
20 PRINT #1,"F-50"
```

By placing a - sign before the 50, you send the motor in the opposite direction. We included the + sign in the first example as an illustration, but absence of a sign is interpreted as a +. Run the program again and notice that the XR now moves in the opposite direction before stopping.

Now change line 20 so it looks like this:

```
20 PRINT #1,"E+50"
```

Run the program and observe the E motor (shoulder) movement.

Change line 20, one motor at a time, to move the D, C, and B motors. Don't use the A motor for now because of its limited travel. Just change the letter to that corresponding to the motor you want to move.

For now, do not use any number other than 50 for the number of steps.

More About The START Command--The Error Registers

Now that you have used the basic START command, you are ready to learn more about it.

Consider our first START command "F+50" as an example. When we sent that command (using the PRINT statement), the motor "F" moved. But much more than that happened.

The MARK III controller maintains an 8 bit error register for each of the 8 motors. When the error register for a specific motor is zero, the controller removes all power to the. motor and the motor remains stationary. When the error register is non-zero, the controller connects either a positive voltage or a negative voltage to the motor. The polarity of the power is a function of what is in the error register. If the error value is positive, the motor moves in one direction and if it is negative, the motor moves in the other direction. Before we sent a command, the error registers for all the motors were zero.

A START command adds the motor movement value to the error register for the motor specified. In our example, that motor was the F motor. The value added was 50.

As soon as a value is added to the error register for a motor, the controller starts that motor in the direction that will take the error register to zero. As the motor moves, the encoders are read and the count in the error register is decremented. When the error register reaches zero, the power is turned off. So, when our START command added 50 to the F motor register, the controller moved that motor 50 encoder steps in the right direction to return the error register to zero.

The command "F-50" adds a value of -50 to the F motor error register. When the controller receives a "-" sign, it moves the motor in the opposite direction from the + direction to reach the zero error position.

Remember:

1. A START command adds a value to the designated motor's error register.

2. The value added to the error register represents the number of steps the designated motor encoder must move to return the register to zero.

3. The controller software is designed so that motors seek the zero error condition.

4. The sign in the START command designates direction of motion.

With this in mind, move on to the next command, the QUESTION command.

THE QUESTION COMMAND

Type in the program listed below.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"F+50"
30  PRINT #1,"F?";: GOSUB 90
40  IF W<>0 THEN 30
50  PRINT #1,"F-50"
60  PRINT #1,"F?";: GOSUB 90
70  IF W<>0 THEN 60
80  END
90  IF LOC(1)=0 THEN 90 ELSE W$=INPUT$(LOC(1),#1)
100 W=ASC(W$)-32
110 RETURN
```

Run the program and you will see that it simply turns the waist in one direction and then In the other. You might be wondering why we did not simply follow line 20

```
20 PRINT #1,"F+50"       with
50 PRINT #1,"F-50"
```

Instead, we send a QUESTION command on line 30. Why?

Remember that a START command adds a value to a motor's error register. If we sent an "F-50" immediately after the "F+50", the -50 count in the second command would combine with the remains of the +50 command and sent the F motor in the opposite direction before it completed the full "+50" steps in the first START command. The QUESTION command allows us to check the error register and make sure the first 50 steps have been executed before the next series of (-50) steps are sent.

How The QUESTION Command Works

The QUESTION command instructs the MARK III controller to return the current value of the error register for a specific motor. Specifically, it asks how much further the designated motor must travel until the register is again zero. In our sample program, after we sent the first START command, we immediately sent a QUESTION to check the status of the F error register.

We do so on line 30. Again "F" designates the motor. The QUESTION command is accomplished with the "?". Line 30 also calls a subroutine to allow your computer to receive the answer to the QUESTION. That subroutine, which begins on line 90, is written so you can use it any time you use a QUESTION or any inquiry command in your programs.

Line 40 loops through the QUESTION until the answer (W) is zero. When it is zero, we know that the error register is zero and all the original steps have been executed. Only then does the program move on to send the START command in the opposite direction.

More About The Question Command
Making A Longer Move

Thus far, our sample programs have limited START commands to 50 steps. The reason has to do with the error registers and communications. Two factors create problems.

1. The largest signed decimal number we can store in an 8 bit error register is 127 (1 bit for the sign of the number, and seven bits to represent up to 127). Numbers larger than 127 will overflow the register. When this happens, the register overflows into the sign bit. This reverses the sign in the error register and thus reverses the motor. In other words, the motor can start to move in the opposite direction to that specified in the START command if the register overflows.

2. When the controller receives a QUESTION command, it automatically adds

32 (hexadecimal 20) to the answer before it transmits it to the host computer. It does so because some computers treat values below decimal 32 (hex 20) as commands instead of data. Note that in subroutine 90, line 100 subtracts 32 from the answer received in response to the QUESTION command. This compensates for the 32 added by the controller. Since the communication line is set for 7 data bits, the largest value that can be returned is 127 which means that the largest value the controller can send is 127-32 or 95.

A motor's error register can be zero, but it can never be lower (because we use the sign bit for direction of travel of the motor). In other words, you can find how far a motor is from the zero position but you cannot find out what direction the zero position is in. The error count is always positive. By adding 32 to every value it transmits to the host computer, the controller ensures that the computer never receives an answer less than 32 (hex 20).

EXAMPLE - Suppose you send a START command of 120 steps. The motor starts to seek zero immediately. If the controller receives a QUESTION when the motor has traveled 20 steps, the answer sent back (the number of steps until zero) will be 100. But the controller will add 32 to the 100 before it sends the answer back to the host. 132 is more than you can represent with 7 bits so the communication line will truncate the 132 to 7 bits and an error will occur.

All this may seem puzzling to you at first, however you will begin to understand the system as you work with it. If you are confused, just keep this rule in mind:

DON'T LET THE ERROR REGISTER EXCEED 95

"But," you may be asking, "how do I get the XR to execute a move longer than 95 steps?"

The answer is to first add a value to a motor's error register with the START command. Monitor the status of the error register with the QUESTION command. Then, add to the register with another START before the register reaches zero, but do not add so much as to exceed the 95 limit.

The following program illustrates one way to make a move of 500 steps.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  FOR I = 1 TO 10
30      PRINT #1,"F+50"
40      PRINT #1,"F?";: GOSUB 90
50      IF W>45 THEN 40
```

```
60  NEXT
70  END
80  IF LOC(1)=0 THEN 80 ELSE W$=INPUT$(LOC(1),#1)
90  W=ASC(W$)-32
100  RETURN
```

This program uses two key techniques to accomplish the 500 step move.

1. The FOR/NEXT loop that begins on line 20 divides and conquers by adding 500 steps to the F motor error register, 50 steps at a time. It loops the computer through line 60 ten times, then exits the loop.

2. The IF / THEN statement on line 50 loops the computer through the QUESTION command on line 40 until the answer is 45 or less. When the answer is 45 or less it is safe to continue through the FOR loop and add the next 50 steps without exceeding the 95 limit.

Remember that in our simple program to rotate the F motor back and forth, the program looped until all of the first 50 steps were executed. This was necessary because the second move was in the opposite direction, and would have reversed the motor before it completed the original 50 steps.

But for the long move above, we do not have to wait for the error register to reach zero before adding to it. We want to avoid letting the error register reach zero; because if it does, the motor will stop. And we want to keep adding to the register without exceeding 95.

You might think of an error register as a gas tank. You never want it to get to empty, but you can't let it overflow, either.

The program above showed one way to do this. It added 50 steps to the register, and as soon as there was room for another 50, it added still another 50.

The program worked as an illustration, but it has drawbacks. The most important drawback is that it waits for the register to come all the way down to 45 before it adds more steps. This was not a problem here, but as you program more complex moves, and connect accessory motors to the controller, the computer will be busy, and timing will become more critical. You will have less time to check and fill the registers.

A better way is to always fill the register as full as possible. Instead of deciding beforehand when and how many steps to add, let the computer calculate how many steps it can add without exceeding 95, and automatically add that many steps. The following program does just that.

```
10 OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20 N=500
30 H=95
40 IF N≤H THEN 80
50 PRINT #1,"F+"; STR$(H): N=N-H
60 PRINT #1,"F?";:  GOSUB 90: H=95-W
70 GOTO 40
80 PRINT #1,"F+";STR$(N): END
90 IF LOC(1)=0 THEN 90 ELSE W$=INPUT$(LOC(1),#1)
100 W=ASC(W$)-32
110 RETURN
```

In line 20, N represents the total number of steps we want the motor to move. We used 500, but you could change N to any number of steps.

We use H as the variable for the number of steps we add to the error register. In line 30, we assign H an initial value of 95; the maximum number we can safely add to the error register.

In line 50, we send a START command that is slightly different than the ones we have used up to now. Instead of specifying the number of steps, we use the variable H. Line 50 adds H number of steps to the **F** motor error register. Since we have assigned H an initial value of 95, the computer will add 95 the first time it executes line 50. Line 50 also subtracts H from N, it decrements our original total steps by the number added to the error register.

Line 60 sends a QUESTION command and calls the subroutine to send the command and receive the answer (W).

Line 60 also assigns H a new value, 95-W. Since W represents the number of steps the F motor must travel before the error register is zero, 95-W is the number of steps we can add to the error register without exceeding 95.

Therefore, H now becomes whatever number required to bring the error register to 95.

Line 70 loops back to line 40. Line 40 checks whether N is less than or equal to H. If N is not less than or equal to H, the computer goes to line 50 to add H more steps to the error register to keep the motor moving.

The program will repeat this loop until N≤H. Remember, line 50 subtracts H from N each time H has been added to the error register. So N, our original number of steps, gets smaller each time through the loop.

When N≤H, the computer goes to line 80, another START command. This START command, however, adds N steps to the error register instead of H steps. At this point, sending N steps completes the original N=500 steps.

As you can see, this method keeps the error register as full as possible. When you write programs to run several motors, this will be critical. Since the program will have to check more than one error register (by means of the QUESTION command), it is important to fill the error register completely each time the QUESTION command is sent. Otherwise, the error register may run to zero (the motor stops) by the time the computer gets back around to check.

The STOP And INQUIRY Commands

In this section we introduce you to the STOP and INQUIRY commands. We'll talk about each separately and then provide a sample program that uses both.

The STOP Command

When you use a START command, a motor will travel the assigned number of steps and stop. But as you develop programs of your own, you'll need to stop motors under specified conditions.

For example, many applications present the possibility of running the robot arm into an object. If the robot crashes into an obstacle, you need a way of recovering from the accident without losing your program and positional information. A servo motor will try to keep moving as long as it has power (ie., as long as its error register is non-zero). The STOP command enables us to bring a specific error register to zero on command.

A STOP command looks like this for the IBM PC:

```
PRINT #1,"FX";
```

"F" designates the motor

"X" is the STOP command

With the STOP command, your programs can incorporate protections against stalls. You can use the QUESTION command to check the status of an error register. The program would send the controller a STOP command in the event that the (a non-zero) error register status did not change during a number of successive checks.

Another use of the STOP command has to do with the microswitches mounted on the XR. The next section, which covers the INQUIRY command, will illustrate the use of the STOP command with the INQUIRY command.

The INQUIRY Command. Reading The Microswitches

The XR is equipped with six microswitches, one for each of the six motors on the arm. Each axis closes its switch at a defined point. Therefore, we have an identifiable and repeatable orientation point for each axis on the XR; the point at which the microswitch is closed.

The INQUIRY commands (**I** and **J**) gives us the means to determine whether any of the switches on the robot arm are closed at a given time. The MARK III controller can read switches on all its ports. So, as your XR is connected now, the controller reads the switches as follows:

Motor	Port	Joint	Can controller read switch?	Command Used
A	A	Fingers	Yes	J
B	B	Wrist	Yes	J
C	C	Wrist flex	Yes	I
D	D	Elbow	Yes	I
E	E	Shoulder	Yes	I
F	F	Waist	Yes	I
G	G	Accessory	Yes	I
H	H	Accessory	Yes	I

In the standard configuration, with motors **A-F** connected to ports **A-F** on the controller, the I-INQUIRY command reads motors **C-F** on the XR-3 and **G** and **H** on the accessories.

In order to read the **A** and **B** port switches, it is necessary to use the **J**-INQUIRY command.

Using the I and J-INQUIRY Commands

The INQUIRY commands work much like the QUESTION command. Each requires the same subroutine to send the command and to receive the answer. The controller adds 32 to the answer before it transmits it to your computer to avoid ASCII command codes, and the receiving subroutine subtracts 32 to yield an accurate answer.

But instead of designating individual motors when you send the INQUIRY, you send the command and interpret the answer according to the port you wish to read. The command looks like this:

```
PRINT #1,"I";        (or "J")
GOSUB XXX
```

where XXX is the number of the same subroutine you've used for the QUESTION command. The subroutine enables the computer to receive the answer to an INQUIRY.

Interpreting The Answers

We will discuss the I-INQUIRY command only. The J-INQUIRY is similar but gives different information.

The key to using the INQUIRY command is interpreting the answer sent back from the controller. Remember that the controller communicates using one byte only, and that the single byte answer uses one bit in the byte for each switch.

The bits are assigned to the ports as follows:

Port	Bit		
C	0		
D	1		
E	2		
F	3		
G	4		
H	5		
-	6	not used	
-	7	not used	MSB

A microswitch can either be open or be closed. When a switch is open, the corresponding bit contains a one. When all switches are open, all six bits contain a one, which has a decimal value of 63. (In fact, the controller sends a 95. Remember that the controller adds 32 to anything it sends to your computer to avoid command codes. Our subroutine to receive the answer automatically subtracts 32 to yield the true value.)

If a switch is closed, the corresponding bit contains a 0, and the answer to an I-INQUIRY will be 63 minus a value unique to the open switch and bit. The table below illustrates what happens when one switch is closed and all others are open.

IF THE SWITCH AT PORT X IS CLOSED, AND ALL OTHERS ARE OPEN	THE ANSWER RETURNED IN RESPONSE TO AN INQUIRY BECOMES. (Ignoring the 32 count offset)	
If X=C	62	(63-1)
If X=D	61	(63-2)
If X=E	59	(63-4)
If X=F	55	(63-8)
If X=G	47	(63-16)
If X=H	31	(63-32)

As you can see, when a switch is closed, a value unique to that switch is subtracted from 63. If we could be sure that only one switch is closed at any one time, things would be simple. We could just send the I-INQUIRY command and check the answer to determine if the switch we were checking for was closed.

The problem is, at any one time, more that one switch may be closed. The answer received will be 63 minus the sum of the values for all closed switches. For example, if switches **C**, **D**, and **E** were closed when we sent the INQUIRY, the answer returned would be 56 ((63-(1+2+4)).

To check for any one switch, we must mask the answer for the other switches that might be closed.

The following subroutine does just that. We would call this subroutine after sending an INQUIRY and receiving the answer. The variable W represents that answer, and we are checking whether the **D** microswitch is closed.

```
100 C=0: D=0: E=0: F=0: G=0: H=0
110 IF W>31 THEN H=1: W=W-32
120 IF W>15 THEN G=1: W=W-16
130 IF W>7 THEN F=1: W=W-8
140 IF W>3 THEN E=1: W=W-4
150 IF W>1 THEN D=1: W=W-2
160 IF W>0 THEN C=1
```

```
170 IF D=1 THEN XXX     (Go to desired point in the program).
```

Let's run through the subroutine with an actual value to see how it works. Assume that the **C** and **D** switches are closed, but that we only care about the D motor. Remember that if all switches are open, the answer returned in response to an I-INQUIRY command is 63.

Remember also that if a switch is closed, the answer returned will be 63 minus a unique value for that switch:

SWITCH CLOSED	CONTROLLER SUBTRACTS
C	1
D	2
E	4
F	8
G	16
H	32

Given this, we know that if switches **C** and **D** are closed, the answer returned in response to the INQUIRY command will be 60.

Let's see how our subroutine determines whether the **D** switch is closed.

Line 100 assigns a variable to each switch. It simply uses the letters assigned to each port. It assigns the value zero to each variable to initialize it to a known quantity. As discussed above, zero corresponds to an open switch.

Line 110 checks to see whether the **H** switch is closed. We know that when W>31 the **H** switch is open. We know this because even if the **H** switch were closed and all others open, W would be at least 31. Therefore, if W>31, the H switch must be open. Since W is sixty, the **H** switch is open, and we can now set H=1 to indicate that it is open.

If **H** is open; we know that 32 is part of W. Before checking the **G** switch, we must mask the 32. That's why before moving to line 120, we set W=W-32. So now W=28.

Line 120 checks the **G** switch. We know that if W>15, the **G** switch must be open because even if all other switches were open (remember we've masked the H switch),W would be only 15. Now **G**=1 and we again mask because we know it is open. Now W = 28-16 = 12.

Understand how the routine checks for each switch and masks for its value before reading further. Run through line 130 with 12, then take the new W and run through 140.

After line 130 and line 140, W should equal 0. The next line, 150, checks for the **D** switch. Let's see what happens when a switch is closed.

```
150 IF W>1 THEN D=1:W=W-2
```

Since W now is 0, and 1 is not greater than 1, D remains zero. W remains 0, as there is no value to mask.

Line 160 checks for the only remaining switch, the **C** switch.

```
160 IF W>0 THEN C=1
```

Since W is still zero, and zero is not greater than zero, **C** remains zero. Since all switches have been checked, the subroutine ends here.

With the subroutine complete the variables are now as follows:

C=0	(C switch is closed)
D=0	(D switch is closed)
E=1	(E switch is open)
F=1	(F switch is open)
G=1	(G switch is open)
H=1	(H switch is open)

Since we are checking for the **D** switch, we have to use an IF... THEN ... statement to make the results of the subroutine useful. For example:

```
IF D=0 THEN XXX
```

where XXX represents the line we want to execute if the D switch is closed.

The STOP and INQUIRY Commands. A Sample Program

The program below supposes that an accessory, the carousel, is connected to the **G** port on the controller. It runs the carousel and uses the INQUIRY command to detect when the cam on the carousel closes the microswitch. When the microswitch is closed, the carousel (**G** motor) is sent a STOP command, bringing the **G** error register to zero to stop the motor.

```
10 OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20 PRINT #1,"G+50"
30 PRINT #1,"I";: GOSUB 110
40 GOSUB 140
50 IF G=0 THEN 90
60 PRINT #1,"G?";: GOSUB 110
70 IF W>45 THEN 30
80 GOTO 20
90 PRINT #1,"GX";
100  END
110 IF LOC(1)=0 THEN 110 ELSE W$=INPUT$(LOC(1),#1)
120 W=ASC(W$)-32
130 RETURN
140 C=0: D=0: E=0: F=0: G=0: H=0
150 IF W>31 THEN H=1: W=W-32
160 IF W>15 THEN G=1: W=W-16
170 IF W>7 THEN F=1: W=W-8
180 IF W>3 THEN E=1: W=W-4
190 IF W>1 THEN D=1: W=W-2
200 IF W>0 THEN C=1
210 RETURN
```

Line 20 lists a START command, adding 50 steps to the G motor error register to get it started.

Line 30 sends the INQUIRY command to immediately begin checking the switches. (Line 30 also calls the subroutine to transmit the command and receive the answer.)

Line 40 calls the subroutine to interpret the answer to the INQUIRY.

Line 50 checks whether the G switch is closed (G=0). If it is closed, it sends the computer to line 90, a STOP command.

Line 60 checks the status of the G motor error register and calls the subroutine to send the command and receive the answer.

Line 70 sends the computer back to the INQUIRY command if there is not enough room in the G motor error register for another 50 steps.

Line 80 loops back to line 20 to add 50 more steps to the error register and keep the G motor moving.

Line 90 is the STOP command executed when the G switch is found. It brings the G motor error register to zero, stopping the motor.

The rest of the program lists the subroutines that we have already discussed.

Note that we use 50-step increments in our START command. But as discussed earlier, in more complicated programs it is better to start with 95 steps and fill the error register as full as possible each time, just as illustrated earlier.

A More Efficient Subroutine To Interpret Answers To An Inquiry Command

The routine we have used to interpret answers to an INQUIRY worked as an illustration. But it is long and awkward. Now that you know how to interpret an answer, here is a shortcut.

It uses the AND operator, which is part of IBM BASICA. It allows you to check specifically for the bit corresponding to the switch you are checking.

To show you just how much easier using the AND operator is, the line below would replace the entire subroutine--lines 140 through 200 in the sample program just given.

```
IF W AND 16 THEN XXX ELSE XXX
```

The AND operator is based on principles that require a lengthy explanation. You can find more information in your IBM BASIC manual.

But to use the AND operator, you really only need to know the following:

1. The integer values that correspond to each switch when it is open are:

SWITCH	INTEGER
C	1
D	2
E	4
F	8
G	16
H	32

2. Any time you use the AND operator to check the status of a single microswitch, just substitute the integer value for that switch for the 16. For example, if you want to check the H switch, the line should look like this:

```
IF W AND 32 THEN H=1 ELSE H=0
```

When the microswitch checked is closed, the computer executes the command listed after ELSE (ie. H=0) and when the microswitch checked is open, the computer executes the program line listed after THEN (ie. H=1).

Hard Home

Now you have seen how the INQUIRY command works to read the microswitches. You may want to use the INQUIRY for any number of reasons, but an important reason will be to find a repeatable starting robot position.

You may want to write a program to move the robot from point A to point B. That program won't work more than once unless the robot starts from the same point each time you run it. The way to establish such a starting point is to send each axis to the position where it actuates the microswitch and stop it there.

We call this switch defined robot position the **hardware home** position, or **hard home** for short. We've written a hard home program for your IBM-PC. This program finds the microswitches for motors D, E, and F and stops each motor in this position. In fact, it finds the center of each microswitch, that is, it stops at the midpoint between the point at which the switch closes and the point at which it reopens. This precision is vital if you wish to repeat programs accurately.

You'll find the program in the Appendices.

In practice, you should call the hard home routine once before each of your programming sessions begins.

Our hard home routine sets only three switches. You may wish to modify it to set the other three. If you do, remember that ports A and B of the controller are read with the J-INQUIRY command. Address them accordingly in your commands.

Note: The fingers and wrist flex, there are readily identifiable visual references. The fingers can be opened as far as they will go. The hand should be perpendicular to the work surface, fingers down and parallel with the body.

SUMMARY OF COMMANDS AND SUBROUTINES

The OPEN COMMUNICATIONS line should begin each program. It opens a communications file at 9600 BAUD, 7 data bits, even parity, 2 stop bits, and disables time out errors.

```
OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
```

The START command moves a specified motor a specified distance and direction by adding a value to the motor's error register.

Example:

```
PRINT #1,"F+50"
```

PRINT #1 sends the command through communications file #1

"F" designates the motor

"+50" designates the number of encoder holes (distance) to be moved and direction (+) to be moved. A - sign before the 50 would reverse the motor 50 steps.

The QUESTION command asks how much further a motor must travel until its error register is zero.

Example:

```
PRINT #1,"F?"
```

PRINT #1 sends the command through communications file #1

"F" designates the motor being checked

"?" is the QUESTION command

The QUESTION must always be followed by the input subroutine that allows the computer to receive the answer.

The STOP command brings a designated motor's error register to zero, stopping the motor.

Example:

```
PRINT #1,"FX"
```

PRINT #1 sends the command through communications file #1

"F" designates the motor

"X" is the actual STOP command

The INQUIRY command asks whether the microswitches are opened or closed.

Example:

```
PRINT #1,"I";
```

PRINT #1 sends the command through communications file #1

"I" is the command

An INQUIRY command must always be followed by the input subroutine that allow the computer to receive the answer. In addition, a subroutine to interpret the answer must follow the input subroutine. The semi-colon should follow the PRINT statement to suppress the sending of a carriage return which is interpreted by the MARK III controller as a motor move command.

Output/Input subroutine--use after every QUESTION and INQUIRY command.

```
nnn  IF LOC(1)=0 THEN nnn ELSE W$=INPUT$(LOC(1),#1)
     W=ASC(W$)-32
     RETURN
```

Subroutine to interpret answer to an INQUIRY command--call after command is sent and Input subroutine is called.

```
IF W AND N THEN XXX ELSE XXX
```

W is the answer received in response to an INQUIRY command

N is the integer value corresponding to the bit of the switch to be checked

THEN XXX will be executed if the switch is closed. XXX indicates the line to be executed.

ELSE YYY will be executed if the switch is OPEN. YYY indicates the line to be executed.

USING THE INPUT/OUTPUT COMMANDS

The MARK III controller has eight TTL inputs, eight TTL outputs and two unencoded motor power port called the AUX ports. Eight commands are available to completely control these features. The use of these commands follow closely the format already developed to control the motor movements; in fact they use the same subroutines.

The MARK III inputs and outputs are TTL compatable and should be treated as low current (less than 15ma at 5 VDC) devices. Do not apply an external voltages to these ports, use the 5 volt source provided on the I/O front panel. The inputs are internally pulled up by 10k ohm resistors and therefore appear to be ON when left unconnected. To control the inputs, a switch should be connected to the input port and the ground terminal. When the switch is closed the input will be OFF; with the switch open the input will be ON.

The outputs are active LOW which means that when the port is ON the output will be LOW and when the port is OFF the output will be HIGH.

The following pages will give examples on how to implement each of these commands.

J-Inquiry Command.   Read Inputs 1 through 4

The J command, like the I command, returns a byte in which the first four bits represent the state of inputs 1 through 4 as follows:

Bit	Input
0	1
1	2
2	3
3	4

The same type of routine that was used with the I command can be used to set an arry variable to a 1 or a 0 dependent on the state of the input. Here we will use the variable I to represent the inputs, with I(0)=Input 1, I(1)=Input 2 etc.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"G+50"
30  PRINT #1,"J";:  GOSUB 110
40  GOSUB 140
50  IF I(1)=1 THEN 90
60  PRINT #1,"G?";:  GOSUB 110
70  IF  W>45 THEN 30
80  GOTO 20
90  PRINT #1,"GX";
100  END
110  IF LOC(1)=0 THEN 110 ELSE W$=INPUT$(LOC(1),#1)
120  W=ASC(W$)-32
130  RETURN
140  FOR T=0 TO 3
150       IF W AND (2 ^ T) THEN I(T)=1 ELSE I(T)=0
160  NEXT T
170  RETURN
```

This program will move the G axis motor, a belt conveyor for instance, until INPUT #2 is turned ON (high TTL level). In an actual workcell, a sensor which goes HIGH when a part is detected would be connected to the input port; when the part arrives at the sensor position, the conveyor will stop.

Lines 140 through 170 implement the routine that determines which inputs are ON by checking each of the first four bits in the byte returned by the controller in response to the J command. Line 50 checks if the input is ON; if it is, the STOP command is executed and the G port is turned off.

K-Inquiry command. Read inputs 5 through 8.

The implementation of the **K** command is identical to the **J** command except that the returned byte contains the state of inputs 5 through 8 in the low four bits. The routine that determines the state of each input is identical to the example program above; instead of using a J in line 50 use a K.

Commands L, M, N, O. Sets the state of the AUX ports.

The two AUX ports on the MARK III controller provide + or - 20VDC motor power, the polarity of which is determined by the setting of the reversing switches near the AUX ports. The four commands (L, M, N, O) set the AUX ports on or off, they do not set the polarity of the output. These commands are simple one byte commands that do not invoke a response from the MARK III controller. The implementation of all four commands is identical, therefore the following example will show how to turn on AUX#2 and then turn it off.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"N";
30  FOR J=1 TO 1000: NEXT
40  PRINT #1,"O";
50  END
```

Line 20 issues the **N** command which will turn on AUX port #2.

Line 30 implements a time delay to allow the user to see the effect of the AUX commands.

Line 40 issues the **O** command which will turn off AUX port #2.

The MARK III controller has two indicator lights connected to the AUX ports to indicate if they are ON. Even of you have nothing connected to the AUX ports, this program can be run and verified since it will cause the light on AUX #2 to come on then go off.

P and R Commands. Set Output Lines High (OFF) and Low (ON)

Eight TTL level outputs are available on the MARK III controller, each of which can be set or reset upon command. The P command instructs the controller to turn OFF an output (make the output go high); the R command instructs the controller to turn ON an output (make the output go low). The format of both commands is the letter of the command followed by the port number that is to be changed.

Suppose it is desired to initialize output #3 by turning it off and then to turn on output #3 when the robot comes to rest to indicate that the material handling operation has been completed. The following program will initialize move the waist 240 counts and then turn ON the output.

```
10  OPEN "COM1: 9600, E, 7, 2, CS, DS, CD" AS #1
20  PRINT #1,"P3";
30  N=240
40  H=95
50  IF N<=H THEN 90
60  PRINT #1,"F+" + STR$(H):  N=N-H
70  PRINT #1,"F?": GOSUB 140:  H=95-W
80  GOTO 50
90  PRINT #1,"F+" + STR$(N)
100  PRINT #1,"F?": GOSUB 140
110  IF W>0 THEN 100
120  PRINT #1,"R3";
130  END
140  IF LOC(1)=0 THEN 140 ELSE W$=INPUT$(LOC(1),#1)
150  W=ASC(W$)-32
160  RETURN
```

This program is familiar to you by now since it implements a long move as discussed previously. However, line 20 directs the controller to turn off output #3 before starting the programmed motor move. Lines 30 through 90 implements the long move. The program waits until the robot comes to a complete stop in lines 100 through 110. Upon completion of the move, line 120 outputs the command to turn on port 3.