USER'S GUIDE TO

VAL™

398H2A

A Robot Programming and Control System

Unimation Robotics

Unimation Inc.
A Condec Company
Shelter Rock Lane
Danbury, CT 06810
(203) 744-1800

Version 12 **June 1980**

FOREWORD

This programming manual is intended for use in the field. The material contained herein constitutes Volume II - User's Guide to VAL™.

The furnishing of this document does not constitute or imply a grant of any license under any patents, patent applications, trademarks, copyrights or other rights of Seller or of any third party.

Please note that specifications contained herein are subject to change by UNIMATION INC. without prior notice. This manual is periodically reviewed and revised to accurately reflect and incorporate improvements and engineering changes made on equipment since the previous publication of this manual. UNIMATION INC. is not responsible for errors or omissions which may appear in this manual.

PUMA™ Robot Training School

Courses are conducted at our facility in Danbury, Connecticut, and can also be conducted at your facility. For pricing, scheduling, and additional information, contact:

Technical Training/Publications Department
Unimation Inc.
Shelter Rock Lane
Danbury, Connecticut 06810
(203)796-1003

CONTENTS

CONTENTS (Continued)

CHAPTER 1

INTRODUCTION

VAL is a computer-based control system and language designed specifically for use with Unimation Inc. industrial robots.

A computer-based system provides the ability to easily define the task a robot is to perform, since tasks are completely defined by user-written programs. Other benefits of using a computer-based system for programming and controlling industrial robots include: (1) the ability to respond to information from sensory systems such as computer vision, (2) improved performance in terms of trajectory generation, and (3) the ability to work in unpredictable situations or moving frames of reference.

The VAL robot language is permanently stored as a part of the VAL system. This includes the programming language used to direct the system for individual applications. The VAL language is easily learned. Its instructions are clear, concise, and generally self-explanatory. All commands and communications with the robot use easily understood word and number sequences. Control programs are written on the same computer that controls the robot.

As a real-time system, VAL's continuous trajectory computation permits complex motions to be executed quickly, with efficient use of system memory and reduction in overall system complexity. The VAL system continuously generates robot control commands, and can simultaneously interact with a human operator, permitting on-line program generation and modification.

A convenient feature of VAL is the ability to use libraries of manipulation routines. Thus, complex operations may be easily and quickly programmed by combining predefined subtasks. For example, a typical palletizing task can be simply programmed by combining two or three "standard" routines.

1.1 OVERVIEW

This manual is intended to be used after a robot system has been properly unpacked and installed, as explained by the equipment manual for the system. The equipment manual should also be referenced if the system should malfunction.

The VAL control and programming system are described in this manual. Various robot servo control modes are discussed; and the VAL commands and instructions are presented with examples. In addition, examples of robot control programs are given to be used as guides for generating programs for individual applications.

The VAL language consists of monitor commands and program instructions. Monitor commands are used to prepare the system for execution of user-written programs. Program instructions provide the repertoire necessary to create VAL programs for controlling robot actions.

The appendices include descriptions of compound transformations, tool transformations, continuous-path servo mode, and diagnostic commands; a list of recent changes to VAL; a complete list of messages output by VAL; and a summary of the VAL commands and instructions.

1.2 TERMINOLOGY

The following terms may be new to the reader, or their use in this manual may differ from earlier experience. These terms are explained more fully in Chapter 3.

MONITOR — An administrative computer program that oversees operation of a system. The VAL monitor accepts user input and initiates the appropriate response; follows instructions from user-written programs to direct the robot; and performs the computations necessary to control the robot.

EDITOR — An aid for entering information into a computer system, and modifying existing text. The VAL editor is used to enter and modify robot control programs.

PROGRAM — A list of instructions telling a computer how to do something. VAL programs are written by system users to describe tasks the robot is to perform.

LOCATION — A position of an object in space, and the orientation of the object. Locations are used to define the positions and orientations the robot tool is to assume during program execution.

1.3 NOTATIONS AND CONVENTIONS

A variety of shortened notations are used in this manual. Several conventions apply to numerical values to be supplied to VAL commands and instructions. These notations and conventions are described in the following paragraphs.

Preceding each monitor-command description are two symbols indicating when the command can be typed by the user. A dot (.) signifies the command can be performed when VAL is in its top-level monitor mode and no user program is being executed (that is, when the system prompt is a dot). An asterisk (*) indicates the command can be performed at the same time VAL is executing a user program (that is, when the system prompt is an asterisk). If both symbols are present the command can be executed in either case.

When entering any monitor command or program instruction, the function name can be abbreviated to as many characters as are necessary to make the name unique (see Appendix G). No space can be left between the system prompt and a command name. At least one space is required between a function name and its arguments (additional information required to specify the function). One or more spaces are required between function elements not separated by commas or other separators (extra spaces are ignored). Monitor commands and program instructions are always completed by typing a carriage return; that is, by pressing the RETURN key.

For commands and instructions, angle brackets, < >, are used to enclose an item which describes the actual argument to appear. Thus the programmer can supply the appropriate item in that position when entering the command or instruction. Note that these brackets are used in this manual only for clarification, and are never to be included as part of a command or instruction.

Many VAL commands and instructions have optional arguments. In the descriptions in this manual, optional arguments are enclosed in square brackets, [ ]. If there is a comma following such an argument the comma must be retained if the argument is omitted, unless nothing follows. For example, the monitor BASE command has the form

BASE [<dx>],[<dy>],[<dz>],[<rotation>]

To specify only a 300-millimeter change in the Z direction, the command could be entered in any of the following ways:

BASE 0,0,300,0 BASE ,,300, BASE ,,300

Note that the commas preceding the number 300 must be present to correctly relate the number with a Z-direction change. Like angle brackets, square brackets are never entered as part of a command or instruction.

Several types of numerical arguments can appear in commands and instructions. For each type there are restrictions on the values that are accepted by VAL. The following rules should be observed:

1. Distances are entered to define locations to which the robot is to move. The unit of measure for distances is the millimeter, although units are never explicitly entered for any value. Values entered for distances can be positive or negative, with their magnitudes limited by a number representative of the maximum reach of the robot (for example, 1024 mm and 700 mm for the PUMA 500 and PUMA 250 robots, respectively). Within the resultant range, distance values can be specified in increments of 0.01 mm. Note, however, that some values cannot be represented internally, and are stored as the nearest representable value.

2. Angles in degrees are entered to define and modify orientations the robot is to assume at named locations, and to describe angular positions of robot joints. Angle values can be positive or negative, with their magnitudes limited by 180° or 360° depending on the usage. Within the

resultant range, angle values can be specified in increments of 0.01°. Some values cannot be represented internally, however, and are stored as the nearest representable value.

3. Arguments indicated as being integer variables can be satisfied with an integer value except where noted. Such a value can range from -32768 to 32767.

4. Joint numbers are to be integers from one up to the number of joints in the robot, including the hand if a servo-controlled hand is operational. Joint numbering starts with the rotation about the base.

5. Channel numbers used to identify the external signal lines can have integer values from one up to the maximum number of lines available to the system. A negative channel number indicates an electrical "low" state.

6. When a command or instruction requires a numerical value for some other kind of argument, the magnitude of the value can range from 0 to 327.67. This applies, for example, to values supplied for speeds and for times.

CHAPTER 2

GETTING STARTED WITH VAL

This chapter is intended to help you start using VAL. You will learn how to use the most important features of your robot system, using examples representing common tasks. To get maximum benefit from this chapter, you should use a VAL system to do each example. Before you start, however, refer to your system equipment manual to make sure your system is properly set up.

2.1 USE OF THE TERMINAL

Once your robot system is set up, you are ready to turn it on and put it to work. You should first turn on the terminal. A printing terminal allows you to save the output for future reference. CRT or "display" terminals are often convenient because they are much faster than printing terminals.

If you are using a CRT terminal, you should wait for it to warm up before proceeding. When the terminal is ready for use its cursor is displayed in the upper left corner of the display. The function of the cursor is to indicate where information will next appear on the display. That is, either information being input to the system from the keyboard, or being output from the system to the terminal.

To enter information into the system, you type the appropriate characters and then press the key labeled "RETURN" at the end of each line. (Some terminals have the RETURN key labeled differently.) Note that the system does not respond to what you type until you complete the line with a RETURN. Also, if you make a mistake while typing, you can use the BACKSPACE (or RUBOUT) key to back up the cursor and erase the error (on a printing terminal the appearance is different, but the result is the same).

In the examples, characters that you are to type are underlined, and the symbol "®" is used to represent the RETURN key.

You should be careful to type the examples exactly as shown so that you see the intended effect. You may get frustrated by unexpected results, including error messages, if you do not type things as shown. If you happen to type something which leads to an error message, you can refer to Appendix F for assistance.

2.2 TURNING THE SYSTEM ON AND OFF

After the terminal is ready, you can turn on the robot system. First turn on the AC POWER switch on the Unimation Computer/Controller. A message similar to the following should be displayed on your terminal:

```
VAL-500 V12H (CP, E200)
INITIALIZE (Y/N)?
```

This has identified the characteristics of the system you are using. In this case the system has a five-axis PUMA robot (500), version 12H (V12H) of VAL, the Continuous Path feature (CP), and the robot has a 200-line encoder on joint 2 (E200). Note that the version number indicated should agree with the version of the "User's Guide to VAL" being used.

At this point VAL is asking if you want the system initialized. During initialization the contents of all the computer RAM memory are erased, which is used to store programs and robot location data. Since you have just turned on power, you MUST respond with a "Y" and press RETURN, after which the display shows:

```
VAL-500 V12G (CP,E200)
INITIALIZE (Y/N)? Y®
ARE YOU SURE (Y/N)?
```

You must now type another "Y®". If you type anything other than a "Y" to either of these questions, the system is not properly initialized, and it will not operate correctly.

NOTE

It is important to remember that you must initialize the system every time power is turned on. (Unless your system has non-volatile memory which has already been initialized.)

You should now see a dot (".") displayed, which indicates that the system has started up successfully. (If a message is displayed, refer to Appendix F.) The dot displayed on your terminal is called a "prompt" because it indicates the system is waiting for input.

You will see three different prompts while using a VAL system. They not only indicate the system is waiting for input, but also give an indication of what portion of the system is waiting. The dot prompt comes from the VAL monitor, which is the administrative portion of the system. Any of the VAL monitor commands (presented in Chapter 4) can be input in response to the dot prompt.

While using the VAL editor, which is used to create and modify programs, the prompt is a number followed by a question mark. The proper responses to this prompt are any VAL program instruction, or any of the various editor commands.

The third prompt you will see is an asterisk ("*"). This prompt comes from the monitor, but it is different from the dot prompt to indicate that a user-written VAL program is active. When an asterisk prompt is displayed, most of the VAL monitor commands can be entered. In Chapter 4 you will see which commands can be input after an asterisk prompt.

The next step in turning on the system is to turn on power to the robot. Currently the servos in the robot are inactive, and the joints are being held in position by brakes or friction. To turn on the servo system, you should (1) hold one hand over the button on the Computer/Controller labeled "ARM POWER OFF", and (2) press the button labeled "ARM POWER". You should immediately press the ARM POWER OFF button if the robot should start to move when power is applied. Any such motion indicates that something is wrong with the system.

The last step in turning on the system involves calibrating the robot position-feedback system. Look at the manual control unit, and note the six "mode" buttons along the left side of the unit. Unless you have already pressed one of these, the light next to the OFF mode button should be on. This indicates that none of the "active" modes has been selected. During the calibration operation the robot is driven by VAL. For this to be possible, the system must be in COMPUTER mode, which is selected by pressing the button labeled "COMP".

The requirements for calibrating the robot depend on the model of robot system you are using. For the PUMA 500 or 600 robot systems, the robot can be at any location within its workspace when the operation is begun. To calibrate the PUMA 250 robot system, however, the robot must initially be located in its "nest."

If you are working with a PUMA 250 robot system, and the robot is not already in the nest, you must perform the following steps. Press the button labeled "COMP" on the manual control unit; then type "LIMP®".

```
.LIMP®
ARE YOU SURE (Y/N)?
```

When you are ready to have all the robot joints become limp, type "Y®" in response to the query.

```
.LIMP®
ARE YOU SURE (Y/N)? Y®
HIT RETURN WHEN ARM IN NEST
```

Then you should manually move the robot to put the wrist flange in the nest. (Note the pins which prevent the flange from being put into the nest with the wrist in the wrong orientation.) Press RETURN to reenable the servos. VAL responds with "OK" on the terminal to confirm the operation.

From here the procedure is the same for all VAL systems. To calibrate the system, type "CALIBRATE®" on the terminal.

NOTE

Most monitor commands and program instructions can be abbreviated. The minimum abbreviations are indicated in Appendix G. Commands and instructions are shown in their full form in this chapter.

```
.CALIBRATE®
ARE YOU SURE (Y/N)? Y®
OK
.
```

You have seen that VAL sometimes responds with a query about whether you are sure you want to do what you requested. This is done to give you a chance to change your mind when you request something that could have a significant effect on the status of your system. In this case you are sure, and should respond with "Y®" for "yes."

All the robot joints should immediately move a small amount. This motion allows VAL to initialize the servo control system. Note that the CALIB light on the manual control has gone out, indicating that the system is calibrated.

This completes the procedure for turning the system on. Turning it off is much simpler. All you need to do is (1) press the ARM POWER OFF button on the Computer/Controller, and (2) turn off power to the controller and the terminal.

2.3 MANUAL CONTROL OF THE ROBOT

Once the system is initialized and calibrated, it is ready for use. You should control it with the manual control unit first, so you can get an idea of how the robot moves.

There are manual control modes which drive the robot in various ways. The SPEED knob can be used with these modes to control the speed of motion, which can be varied from very slow (counterclockwise settings) to fast, even while the robot is being moved. (For now you should not use the fully counterclockwise setting of the SPEED knob.) Note that there is no relationship between the motion speeds obtained under manual control and those produced by computer control. For safety, the maximum speed at which the robot moves under manual control is less than the maximum possible under computer control.

You can use the manual control unit to drive each robot joint by selecting JOINT mode, turning the SPEED knob to a small setting, and pressing the appropriate joint-select switch to one side. You can vary the speed of the motion by turning the SPEED knob. Also, you can move more than one joint at a time. If you move a joint to the extreme of its range, motion stops and a message appears in the display of the manual control to indicate the cause.

After you have experimented with JOINT mode, select FREE mode. (Be careful while using FREE mode with joints 2 and 3 to make sure you are prepared to support the robot when either of these joints is released.) In FREE mode, when you move a joint-select switch toward "+" the servo system releases control of that joint, and you can move it to a new position by pushing on the members of the robot. Pushing the joint switch toward "-" (or selecting another mode) returns control to the servo, and holds the joint in the new position. As with JOINT mode, more than one joint can be selected at a time.

WORLD mode provides another way to drive the robot. Refer to the labels to the left of the switches while using this mode. These indicate that the robot tool can be made to move in X, Y, Z coordinate directions, and rotate about those axes. In the case of WORLD mode, the coordinate directions are fixed to the base of the robot. Try moving the robot in WORLD mode, using the SPEED control to vary the speed of motion. In this mode all the joints are moved in unison as required to make the robot move as requested.

The final manual control mode is TOOL mode. This mode is very similer to WORLD mode because the robot tool moves relative to a set of X-Y-Z axes. But in TOOL mode the axes are fixed to the tool, rather than to the base of the robot. Experiment with TOOL mode until you understand the difference between it and WORLD mode.

You have now used the three modes in which the robot can be driven under manual control. The robot moves as long as you hold one or more of the toggle switches off center. There is a special setting of the SPEED knob which is different. If you turn the knob fully counterclockwise to the INC ("increment") setting, each time a toggle switch is moved off center, regardless of how long it is held, the robot moves the minimum amount the servo system can resolve. As before, the motion is determined by the mode selected and toggle switch pressed. The INC setting permits very precise positioning of the robot.

2.4 CREATING A VAL PROGRAM

You will now create a program to have the robot perform a simple task. A program is a list of the things you want the robot to do for you. When a program is "executed," the system looks at each line of the program, and has the robot do whatever is instructed by that line. Thus, the lines of a program are said to contain "instructions."

There are two aspects of every application which must be described to the VAL system. These are (1) the motions and actions the robot is to perform, and (2) the places in the workspace to which the robot is to move. Either of these can be considered first.

The task you will program is to pick up a part from one location and place it at another location. This is a very common type of task, and is often referred to as "Pick and Place." Imagine an indexing conveyor bringing

boxes past the robot. The job of the robot is to pick parts from a chute and place them in successive boxes.

Begin by identifying all the motions and actions required of the robot to complete the task. One possible sequence of robot activity is as follows:

1. Move to a location above the part in the chute.
2. Move to the part.
3. Close the hand.
4. Remove the part from the chute.
5. Carry the part to a location above the box.
6. Put the part into the box.
7. Open the hand.
8. Withdraw from the box.

Assuming that this sequence is begun with the hand already open, you can see that the sequence could be repeated indefinitely with the desired result.

To tell the VAL system how to do this task, you need to create a VAL program. To do this you use the VAL editor, which provides the means to enter programs into the computer, and later make any needed changes. Type "EDIT DEMO.1®" to tell VAL you want to edit a program named DEMO.1 . VAL can tell this program does not currently exist in the computer memory, and creates a space for it. As a result you see the following lines on your terminal:

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
     1.?
```

VAL is waiting for you to enter the first step of your program. You should type "APPRO PART,50®".

NOTE

It is important that you distinguish between the letter "O" and the number "0" whenever you enter information to the system.

The following lines should be on your terminal:

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
   1.?APPRO PART,50®
   2.?
```

Now VAL is asking for the next step. Enter the remainder of the program as given below. Recall that if you make a mistake on a line, you can correct it if you have not pressed the RETURN key. If you do not notice a mistake before pressing RETURN, VAL outputs an error message if it cannot understand the line. In that case VAL prompts again for the same line. If the line is

acceptable to VAL, but is not what you want, you will learn later how you can delete, replace, or correct the line.

The following lines should be displayed after the complete program is entered:

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
   1.?APPRO PART,50®
   2.?MOVES PART®
   3.?CLOSEI®
   4.?DEPARTS 150®
   5.?APPROS BOX,200®
   6.?MOVE BOX®
   7.?OPENI®
   8.?DEPART 75®
   9.?
```

This completes the program. When the program is executed, it causes the robot to perform the steps which described the task. The exact meaning of each line is:

1. Move to a location 50 millimeters above the part in the chute.
2. Move along a straight line to the part.
3. Close the hand.
4. Withdraw the part 150 mm from the chute along a straight-line path.
5. Move along a straight line to a location 200 mm above the box.
6. Put the part into the box.
7. Open the hand.
8. Withdraw 75 mm from the box.

To make sure the instructions are correctly entered, review them using some more features of the editor. Type "S®" to display the first step of the program. Notice that VAL has rearranged the information on the line slightly, but the meaning is unaffected. If you press RETURN without typing anything, the following step is displayed. You can review the whole program this way.

```
9.?S®
1.        APPRO PART, 50.00
1.?®
2.        MOVES PART
2.?
```

If you find a line that contains an error, you can simply retype the line after it is displayed, and the new version replaces the old line. To delete a line from the program, type "D®" when the cursor is at the step number of the line to be deleted.

You know how to display the next line of the program by pressing RETURN. To print the next five lines type "P 5®". To see the preceding line, type "L®" (for "last"). Try these editor commands.

Once you are done editing the program, you have to exit the editor and return to the VAL monitor to execute the program. Type "E®" to do that.

```
9.?S®
1.        APPRO PART, 50.00
1.?®
2.        MOVES PART
2.?E®
```

2.5 DEFINING LOCATIONS

The next step in programming the task is to define the locations which are to correspond to the names in the program. Note that by "location" we mean the point in space to which the tool is to move, and the orientation the tool is to assume at that point. Recall that this step could have been done first; sometimes that is preferable.

There are several ways to define the robot locations corresponding to location names. Only one of them is used here. "Teach Mode," another method for defining locations, is explained in a later section. Other ways to define locations are described later in this manual.

Begin by defining the location of the part in the chute. Using the manual control, move the robot to a location where you might imagine a part to be located. That is, move the hand to the desired point in space, with the desired orientation.

Once you have the robot where you want it, enter the following command:

```
.HERE PART®
   X/JT1    Y/JT2    Z/JT3    O/JT4    A/JT5    T/JT6
   34.75   530.47   -23.06 -167.459   57.437  114.247
CHANGE?
®
.
```

This has told VAL to assign the name "PART" to the current location of the robot. The numbers displayed are the X-Y-Z coordinates of the hand, and three angles (O, A, T) which define the orientation of the hand.

Similarly, move the robot to another location a short distance away, and type "HERE BOX®®". You should always try to minimize the distances between locations to minimize the travel time between them. Thus, in setting up a task, try to locate things as close together as possible.

Now that you have told the system--with your program--what robot actions are to be performed, and have defined the locations involved, you are ready to have the system execute your program.

2.6 EXECUTING A VAL PROGRAM

You should get used to using the STATUS command. Try it now. You should see the following information displayed:

```
.STATUS®
MONITOR CONTROLLED SPEED:  100.00
TOOL TRANSFORM:
"EXEC" LOOPS COMPLETED = 0.
"EXEC" LOOPS REMAINING = 0.
 RTN   NEXT STEP
.
```

The first line indicates that the "monitor" speed is set to 100.00, which means that subsequent motions are to be performed at "full speed." The monitor speed is set by the monitor SPEED command. Do that now so robot motions will be slow enough to allow manual intervention before something drastic happens.

To change the monitor speed to 20, and verify the change, type

```
.SPEED 20®
.STATUS®
MONITOR CONTROLLED SPEED:  20.00
TOOL TRANSFORM:
"EXEC" LOOPS COMPLETED = 0.
"EXEC" LOOPS REMAINING = 0.
 RTN   NEXT STEP
.
```

This means 20% of "full speed," which is a slow, safe speed. The STATUS command indicates the speed has changed as requested.

The other lines displayed by the STATUS command will become meaningful as you learn more of the system features. For now, note there is no program executing, since no information is displayed below "RTN" as shown above.

Now that the speed is set to a safe value, you are ready to execute your program. Remember the program was written with the assumption that the robot hand would be open initially. Make sure it is open, or the program will not do exactly what you want. To open the hand, select any of the manual control modes and press the TOOL switch toward "O".

CAUTION

If the robot does something undesirable, you can stop it by pressing the OFF button ("panic button"). The ARM POWER OFF button can also be pressed to stop the arm in an emergency.

Since you want the computer to control the robot, you must select COMP mode. To execute the program, keep a finger near the OFF button on the manual control, and type

```
.EXECUTE DEMO.1®
*
```

The robot should go through all the steps defined by the program, and the following lines should be displayed on your terminal:

```
.EXECUTE DEMO.1®
*PROGRAM COMPLETED:  STOPPED AT STEP 9.
.
```

This indicates the robot stopped because the program execution requested by the EXECUTE command was completed. If you want to change the locations, return to the previous section and repeat the steps for defining locations.

Of course it is not too useful to have the robot stop after moving one part. Thus, you would like to tell VAL to execute the program a number of times, or even indefinitely. To execute the program five times in succession, type "EXEC DEMO.1,5®". The robot stops after the fifth complete cycle, and the above message is displayed.

Several "shortcut" features have been incorporated in VAL for convenience. For example, after a program completes execution, it can be executed again without having to retype the name of the program. Type "EXECUTE®" to have the program executed one time, or "EXECUTE ,3®" to have it executed three times. Note that a comma was required in this last case to indicate properly that the "3" was a repeat specification. If you type "EXECUTE 3®", VAL gives an error message because it interprets this as a request to execute a program named "3", which is not a valid program name.

Type "EXECUTE ,-1®" to execute your program an indefinite number of times. The negative repeat specification is a signal to VAL that you want the program to be executed until you stop it.

A program can be stopped three ways. Each time you stop execution, VAL indicates the program step at which the program stopped. The number displayed is for the step that would have been executed next, not the number of the step just completed or interrupted.

If you type "ABORT®" on the terminal, the program stops after completing the step currently executing, and the following is displayed:

```
.EXECUTE ,-1®
*ABORT®
*
ABORTED
STOPPED AT STEP 8.
.
```

Pressing the OFF button on the manual control causes the program to stop immediately, and the following display. Note that COMP mode is no longer selected on the manual control, and must be selected before the program can be executed again.

```
.EXECUTE ,-1®
**PANIC BUTTON HIT*
STOPPED AT STEP 6.
.
```

The program stops immediately if you press the ARM POWER OFF button; and the message below is displayed. When the button is pressed, power to the robot is turned off, and OFF mode is selected on the manual control.

```
.EXECUTE ,-1®
**HARDWARE SERVO DISABLED*
STOPPED AT STEP 3.
.
```

NOTE

Using this method to stop program execution puts high stress on the drive components of the robot. Thus, the ARM POWER OFF button should only be used in emergencies.

Start the program executing again so you can try the ability of VAL to simultaneously interact with an operator. The first thing to note is that while a program is executing, VAL displays an asterisk ("*") as its prompt instead of a dot (".") . This is done to indicate a program is active. Try the STATUS command a few times to see what happens. Each time VAL displays lines similar to the following.

```
.EXECUTE ,-1®
*STATUS®
MONITOR CONTROLLED SPEED:  20.00
TOOL TRANSFORM:
"EXEC" LOOPS COMPLETED = 4.
"EXEC" LOOPS REMAINING = INFINITE
 RTN   NEXT STEP
 DEMO.1    6.
*
```

You can see how many times the program has been executed (four in the above example). Also note the name of the program is indicated, and step #6 was about to be executed.

Change the speed to 40 by typing

```
*SPEED 40®
*
```

Note the change does not become effective immediately. After trying a few other speed settings, set the speed to 60.

Type "SWITCH" to see the status of the VAL switches.

```
*SWITCH®
CP ENABLED
CRT DISABLED
HHT ENABLED
MESSAGES ENABLED
SRV.ERR DISABLED
*
```

These are described in Chapter 4, but for now consider the "CP" switch. When this switch is enabled the continuous-path feature of VAL is active, and the robot moves smoothly from one motion to another, as it should be doing now. Disable the CP switch by typing "DISABLE CP®", and note the difference in the motions. The robot stops briefly between each motion. Type "ENABLE CP®" to restore continuous path. The dialogue for these commands should look like this:

```
*DISABLE CP®
OK
*ENABLE CP®
OK
*
```

You have seen a few of the commands that can be issued to VAL while a program is executing. As you learn more of the commands available, you will learn that most of them can be processed in this mode, as well as when no program is active. It is significant to note that you can use the editor while a program is executing. Thus, you can work on a program while the system is in use performing a task. You can even edit the program that is executing! That must be done with caution, however, because any change has its effect the next time the step is executed, and this may cause the robot to behave in an unexpected way.

Stop the program by typing "ABORT®", or by pressing the OFF button.

2.7 USING TEACH MODE

You learned earlier how to create a program and define the locations used in that program. For some applications it is useful to combine both of those steps into one by using teach mode. Teach mode is just another way to enter program instructions and locations into the system.

Suppose you want to add a sequence of motions to the program DEMO.1. Imagine you want the robot to move to three new locations after it removes the part from the chute, and before it places the part into a box. Teach mode provides a useful way to make the necessary changes to the program.

First use the editor to access the program. Then move to the line after the point at which you want to insert instructions. For the example above, that means your terminal should display these lines:

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
   1.        APPRO PART, 50.00
   1.?®
   2.        MOVES PART
   2.?®
   3.        CLOSEI 0.00
   3.?®
   4.        DEPARTS 150.00
   4.?®
   5.        APPROS BOX, 200.00
   5.?
```

You have a choice of two types of teach mode. Each will insert motion instructions into your program; they differ in the type of motion that is produced. "T" teach mode produces joint-interpolated motion instructions, which are usually preferred. "TS" teach mode produces straight-line motion instructions. To invoke "T" mode, for example, type "T NEW1®". Note the display on the manual control indicates "TEACH MODE".

Now, each time you press the RECORD button on the manual control, a MOVET instruction is automatically be inserted into the program. Move the robot to three locations near those you have already taught, and change the hand opening at each location. Press the RECORD button when the robot is at each new location. When you press the button, the display on the manual control blinks to indicate the button has been pressed, and a new line appears on the terminal. Note the name you specified for the new locations ("NEW1") has its suffix number incremented for each location, and the number after the name is zero when the hand is closed, and a positive number when the hand is open.

To end teach mode, press the RETURN key on the terminal. While using teach mode, the following lines should have been added on your terminal:

```
   5.?T NEW1®
   5.        MOVET NEW1, 0.03
   6.        MOVET NEW2, 0.00
   7.        MOVET NEW3, 0.03
®  8.        APPROS BOX, 200.00
   8.?
```

There are several things to note about teach mode. As mentioned above, the suffix number on the location name is increased for each location. Three MOVET instructions were entered into the program, and the APPROS instruction that used to be step #5 is now step #8. That is because teach mode inserts the MOVET instructions ahead of the step at which teach mode is started.

Exit from the editor, set the speed to 20, and reexecute the program. In the middle of the old sequence of motions the robot should move to each of the new locations, and the hand should open and close as you requested.

```
   8.?E®
.SPEED 20®
.EXECUTE®
*PROGRAM COMPLETED:  STOPPED AT STEP 12.
.
```

Teach mode can be used to create a whole program. Note, however, with teach mode you cannot automatically generate anything but MOVET and MOVEST instructions. To have other instructions in a program you always have to type them in as you did when you originally created DEMO.1 . Try using TS teach mode to define a new program with four or five steps. For example, to use TS teach mode to create a program named "TEACH", type

```
.EDIT TEACH®
.PROGRAM TEACH
   1.?TS NAME1®
   1.       MOVEST NAME1, 0.00
   2.       MOVEST NAME2, 0.00
   3.       MOVEST NAME3, 0.00
   4.       MOVEST NAME4, 0.00
®  5.?E®
.EXECUTE TEACH®
*PROGRAM COMPLETED:  STOPPED AT STEP 5.
.
```

When you are done teaching locations, type "®" to end teach mode, and "E®" to exit the editor. Select COMP mode, and execute the program to see its effect.

2.8 EDITING A PROGRAM

You have already used some of the features of the VAL editor to create and modify programs. Other editor features are used in this section.

Returning to the program named DEMO.1, assume you want to make the following changes:

1. Delete the three motions inserted with teach mode.
2. Have the program "loop" indefinitely, independently of the EXECUTE command used to initiate execution.
3. Add a "counter" to display the number of parts moved.

To make these changes you will have a chance to try all the editing commands. First, access the program for editing. If you are not already at step #1, move there by typing "S®". Now display the whole program on your terminal by typing "P 22®".

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
   1.        APPRO PART, 50.00
   1.?P 22®
   2.        MOVES PART
   3.        CLOSEI 0.00
   4.        DEPARTS 150.00
   5.        MOVET NEW1, 0.03
   6.        MOVET NEW2, 0.00
   7.        MOVET NEW3, 0.03
   8.        APPROS BOX, 200.00
   9.        MOVE BOX
  10.        OPENI 0.00
  11.        DEPART 75.00
  12.?
```

To delete the three motion instructions, first move to step #5. The command "S 5®" does that. Now if you type "D 3®", that step, and the two that follow it, are deleted, and all the following steps are moved up three steps. To confirm that, type "S®" and then "P 22®". After deleting these three lines your program should be back to its original form. You can EXECUTE it to see that it performs the original task.

```
12.?S 5®
   5.        MOVET NEW1, 0.03
   5.?D 3®
   5.        APPROS BOX, 200.00
   5.?S®
   1.        APPRO PART, 50.00
   1.?P 22®
   2.        MOVES PART
   3.        CLOSEI 0.00
   4.        DEPARTS 150.00
   5.        APPROS BOX, 200.00
   6.        MOVE BOX
   7.        OPENI 0.00
   8.        DEPART 75.00
   9.?
```

A program "loop" means that one or more steps are repetitively executed. This is done by having control "branch" from one step in a program back to a previous step. Branches can be done every time, or they can be done only when certain conditions are met. These are called unconditional and conditional branches, respectively.

Whenever a branch is done, there must be a way to identify the step to which the branch is made. You have seen that the step numbers displayed by the editor can be used to identify steps in the program. But you have also seen that the step number associated with an instruction can change if insertions or deletions are done. Thus, there needs to be a way to attach labels to instructions, independent of the step numbers. Any positive integer less than 32767 can be used as such a label in VAL programs, and there can be as

many labels in a program as you like. You will now see how to add a label and a branch instruction to your program.

To make the example program execute indefinitely, there needs to be an unconditional branch from the last step to the first. Thus, a label must be added to the first instruction. You can do that two ways. Replace the whole line, adding the label, by typing "S®" to skip to the first line, and then type a new line as shown.

```
9.?S®
1.        APPRO PART, 50.00
1.?10 APPRO PART,50®
2.        MOVES PART
2.?
```

The number "10" is the label that has been assigned to the APPRO instruction. Type "L®" to display the last (previous, not final) line to confirm the change. To practice the other way to add a label to a line, replace the line again with "APPRO PART,50".

```
2.?L®
1.     10 APPRO PART, 50.00
1.?APPRO PART,50®
2.        MOVES PART
2.?L®
1.        APPRO PART, 50.00
1.?
```

You can replace a portion of a line using the editor R command. Skip to the line to be changed, and type spaces until the cursor is beneath the first character to be replaced. Then type an "R" and one space; and the new character(s) to go into the line. In the case of adding the label "10", this involves simply typing "R 10" when the cursor is positioned after "1.?". Do this and note the effect. The replace command is useful for making minor changes to lines.

```
1.        APPRO PART, 50.00
1.?R 10®
1.     10 APPRO PART, 50.00
1.?
```

To make the program branch to the step with the label "10", add a step at the end of the program containing "GOTO 10". That is, after "9.?" type "GOTO 10®". Exit from the editor and type "LISTP DEMO.1®". Your program should be displayed as follows:

```
   1.?S 9®
   9.?GOTO 10®
  10.?E®
.LISTP DEMO.1®
.PROGRAM DEMO.1
   1.     10 APPRO PART, 50.00
   2.        MOVES PART
   3.        CLOSEI 0.00
   4.        DEPARTS 150.00
   5.        APPROS BOX, 200.00
   6.        MOVE BOX
   7.        OPENI 0.00
   8.        DEPART 75.00
   9.        GOTO 10
.END
```

You can see that every time step #9 is executed, control branches to step #1, which is what was desired. If you now EXECUTE the program, the only way to have it stop is to use one of the aborting methods described earlier.

The final change to the program involves adding a counter, which is just a number stored in the system memory used to keep track of the number of times something has happened. For your program, you want to count the number of parts that have been put into boxes.

First you need to insert a step to "initialize" the count to zero. Skip to the first line of the program with "S®". Typing "I®" causes the editor to enter "insert" mode. Now type a new first line containing "SETI COUNT=0". This instruction tells VAL to SET the value of the Integer variable named COUNT to zero. Any name could have been chosen for the variable, COUNT is convenient because it conveys the meaning the number has to you.

```
.EDIT DEMO.1®
.PROGRAM DEMO.1
   9.        GOTO 10
   9.S®
   1.     10 APPRO PART, 50.00
   1.?I®
   1.?SETI COUNT=0®
   2.?®
   2.     10 APPRO PART, 50.00
   2.?®
   3.        MOVES PART
   3.?
```

To end insert mode, press RETURN when the cursor is at the beginning of an empty line. Note that the old first line is now step #2, and all the following steps have been renumbered also.

Skip to step #9 and insert the instructions shown below. Once again, type "I®", enter the two lines, and press RETURN to end insert mode.

```
 3.?S 9®
 9.        DEPART 75.00
 9.?I®
 9.?SETI COUNT=COUNT+1®
10.?TYPEI COUNT®
11.?®
11.        DEPART 75.00
11.?
```

Your first reaction may be that the arithmetic indicated in the first of these steps is wrong. In computer programming, an equal sign does not mean "is equal to," but rather is used to mean "is replaced by." Thus, the first step causes the value stored in the variable COUNT to be increased by one each time the step is executed. The second step displays the value of COUNT on the terminal.

EXECUTE the program and note that after each cycle VAL displays a line giving the current value of COUNT. It is important that the step which sets COUNT equal to zero was placed outside of the loop. If that had not been done, COUNT would never go beyond one. Do you see why?

```
  11.?E®
.EXECUTE DEMO.1®
*  COUNT     =        1.
   COUNT     =        2.
   COUNT     =        3.
   COUNT     =        4.
*PANIC BUTTON HIT*
STOPPED AT STEP 8.
.
```

2.9 USING THE FLOPPY DISK

When you finish editing a program, you usually want to store it for future use. Recall that unless your system has non-volatile memory, all the information in memory is lost when power is turned off.

The floppy disk provides a means to save programs and location information outside of the VAL system. This is useful for maintaining a library of manipulation routines, as well as for short-term storage of programs under development.

Jump ahead to Section 4.4, which describes the VAL commands for using the floppy disk, and read the instructions regarding handling floppy diskettes. Connect a floppy drive to the Computer/Controller and insert a diskette you can use freely. Always remember to handle diskettes carefully.

Before a new diskette can be used, it must be formatted. Type "FORMAT®". This establishes a data structure on the diskette so information can be stored. VAL asks if you are sure you want to format because this operation erases the entire diskette.

```
.FORMAT®
ARE YOU SURE (Y/N)? Y®
OK
.
```

Information on the diskette is stored in files. There are two types of files--one for programs, and one for location data. Type "LISTF®" to see a list of the files on a diskette. The directory of the diskette is used to keep track of where files are located on the diskette. You do not see that information, but you do see the names and sizes of all the files on the diskette. There are no files listed if you have a blank diskette.

```
.LISTF®
   DISK DIRECTORY
264 OF 264 BLOCKS UNUSED
.
```

The easiest way to save the contents of the system memory is to use a STORE command like this:

```
.STORE DEMO®
     .PROGRAM DEMO.1
     .PROGRAM TEACH
     .LOCATIONS
OK
.
```

This creates a program file containing all the programs in memory, and another file containing the named robot locations used by those programs. If you type "LISTF®", you see that VAL created two files, DEMO.PG and DEMO.LC. These contain programs and locations, respectively.

```
.LISTF®
   DISK DIRECTORY
     DEMO.PG     002
     DEMO.LC     004
258 OF 264 BLOCKS UNUSED
.
```

The names of both of your programs were displayed as they were stored. If you want to select which programs are to be stored, use a command like this:

```
.STORE DEMO2=DEMO.1®
     .PROGRAM DEMO.1
     .LOCATIONS
OK
.
```

The programs to be stored are listed to the right of the equal sign, separated by commas. This time the program named TEACH was not stored. ("DEMO.1" can not be used as a file name because periods are not permitted in file names.)

```
.LISTF®
   DISK DIRECTORY
     DEMO.PG     002
     DEMO.LC     004
     DEMO2.PG    002
     DEMO2.LC    001
255 OF 264 BLOCKS UNUSED
.
```

Once a file is no longer needed (for example, because it is to be replaced by a new version) use the DELETE command to erase the file. To delete the location file created above, type

```
.DELETE DEMO2.LC®
ARE YOU SURE (Y/N)? Y®
OK
.
```

You might want to do this because you have redefined the locations to be used by your program. Then you would want to store the new location data. The STOREL command is like the STORE command, except that it only creates a location file. Use a STOREL command to create a new location file named DEMO3. Then use LISTF to see the effect.

```
.STOREL DEMO3=DEMO.1®
     .LOCATIONS
OK
.LISTF®
   DISK DIRECTORY
     DEMO.PG     002
     DEMO.LC     004
     DEMO2.PG    002
     DEMO3.LC    001
255 OF 264 BLOCKS UNUSED
.
```

After a program and its location data have been stored in files on a diskette, the way to load them into the system is to use a LOAD command. For example, to load the files DEMO.PG and DEMO.LC you would use the command

```
.LOAD DEMO®
     .PROGRAM DEMO.1
     .PROGRAM TEACH
     .LOCATIONS
OK
.
```

CAUTION

When information is loaded from the floppy disk, it writes over any information already in the system memory which has the same name. For example, if there is a program named TEST in memory, and a file is loaded which contains a program with that name, the version in memory is lost during the LOAD operation.

2.10 CONCLUSION

You should now have a good idea of how the basic features of VAL are used. You have seen how to turn the system on and off, and how to issue commands to the VAL monitor, including how to execute programs. You have learned how to use the manual control, to move the robot around and to teach locations. You have also seen another way to define locations--use of the HERE command. Use of the editor to create and modify programs is very important to understand. Finally, you have seen how to use the floppy disk to save programs and location data, and to enter them back into the system.

The next chapter provides a through description of the VAL system. The details of the commands and instructions you have used, and all the other commands and instructions available, are described in the following chapters.

CHAPTER 3

THE VAL SYSTEM

The function of VAL is to regulate and control a robot system by following user commands or instructions. In addition to being a compact stand-alone system, VAL has been designed to be highly interactive to minimize programming time, and to provide as many programming aids as possible. In this chapter, each of the capabilities available in a VAL system are described briefly.

3.1 EXTERNAL COMMUNICATION

Terminal. The standard VAL system uses an operator's console terminal and a manual control box to input commands and data from the user. The operator's console serves as the primary communication device and can be either a display terminal (CRT) or a printing terminal (TTY). To facilitate typing, commands and instructions can be abbreviated, as shown in Appendix G.

Manual Control. The manual control box is used for moving the robot under direct operator control and for teaching location data. Its switches can be used to reposition individual joints of the robot, to change the position and orientation of the tool relative to either World or Tool frames of reference, and to selectively turn off servo control of individual joints, allowing the operator to position the robot by hand.

Floppy Disk. The VAL system supports an optional minifloppy disk drive. This is not required for operation of the system, but is useful for offline storage of programs and location data. Commands are provided for reading and writing files containing user programs and their associated location data. In addition, file maintenance commands for formatting and initializing diskettes, and commands for listing diskette directories and deleting files, are provided.

The disk drive can also be used to read special diagnostic programs into the system. These programs, written by Unimation Inc. and supplied on diskettes, are used for isolating hardware problems.

Signal Lines. To synchronize the robot with other devices, input and output signal lines are provided. These lines can be sensed and set by user-programs. They can be used for starting and stopping other equipment or for controlling the robot using simple sensors of various types. These external interlocks and sensor inputs are described in Chapter 5.

3.2 THE VAL OPERATING SYSTEM

The computer in the controller has two levels of operation. The top level is called the VAL operating system, or monitor, because it administers operation of the system, including interaction with the user. The second level is used for diagnostic work on the controller hardware.

3.2.1 The Monitor

The system monitor is a computer program stored in programmable read-only memory (PROM) in the Computer/Controller. PROM memory retains its contents indefinitely, and thus VAL is immediately available when the controller is turned on. The monitor is responsible for control of the robot, and its commands come from the manual control unit, the system terminal, or from user programs.

To increase its versatility and flexibility, the VAL monitor can perform most of its commands even while a user program is being executed. Commands that can be processed in this way include those for controlling the status of the system, defining robot locations, storing and retrieving information on the floppy disk, and creating and editing robot control programs.

Some robot motions require the monitor to perform substantial computations. When such motions are requested by user programs or from the manual control, there is little or no computer time available to interact with the terminal. This makes the monitor unresponsive, or even appear inactive. When no program is executing and the robot is not being controlled manually, the monitor requires little overhead time and is very responsive to input from the terminal.

Since some monitor commands cannot be processed while a program is executing, it is important to know the status of the monitor. This information is provided by the prompt displayed when the system is accepting input. A dot (".") is displayed as a prompt when the monitor is not executing a user program. An asterisk ("*") prompt is displayed when a program is being executed. All the monitor commands can be processed when the dot prompt is given, but certain commands are not accepted after an asterisk prompt as indicated in Chapter 4.

Being able to continue to process commands while a program is executing can be useful in many situations. User programs can be created, edited, and stored. The speed of the robot can be modified. The system execution status can be queried. And location data can be modified or "touched up," a most useful feature for many applications.

3.2.2 Octal Debugging Tool

The second level of computer operation is not often used. The Octal Debugging Tool (ODT) is used to communicate directly with the computer registers and memory locations, which is very useful for diagnosing hardware problems.

ODT is a standard feature of the Digital Equipment Corporation LSI-11 computer processor module, and thus is always available for use. Some instructions for using ODT are presented in this manual where applicable. For complete details on the capabilities of ODT and the commands it accepts, refer to the Digital Equipment Corporation "Microcomputer Processor Handbook."

3.3 USER-WRITTEN PROGRAMS

Just as a spoken language is based on a set of words that are grouped to convey a message, the VAL programming language is based on a set of instructions that can be combined to describe complete robot tasks. This manual is dedicated to explaining the meaning (that is, the effect) of the VAL program instructions.

In a normal application the motions and actions of the robot are controlled by a program stored in the computer random-access memory (RAM). Such programs are created by the user for particular applications, and contain instructions to the monitor telling it how the task is to be accomplished. Such robot control programs are called "user programs" or "VAL programs" (because they are written in the VAL program language).

Each line, or step, of a user program consists of a single VAL instruction having an easily understood effect on the robot system. A user-written VAL program is nothing more than that--an orderly collection of individual instructions that direct the motions and actions of the robot.

For convenience, the creation of a user program for a particular application can be considered in two steps. One deals with the places to which the robot is to move, referred to as locations. The other step deals with defining the robot motions between locations, and the actions at or between locations. This step involves writing a program using VAL instructions, and entering it into the system using the program editor.

3.3.1 Program Editor

VAL contains an easy-to-use program editor that allows the user to create or modify robot control programs. When using the editor, the user types the desired information on the system terminal just as if on a typewriter. The editor stores the lines in the system. Once a line has been stored, the editor allows the user to return to that line to review it, change it, or delete it entirely. The ability to modify user programs is an important feature of the VAL editor.

The program editor is a useful aid in program debugging since modifications to programs can be made and tested very quickly. In fact, a program being executed can be simultaneously edited. Thus, a program that follows a repetitive cycle can be modified while executing, and the change takes effect the next time the altered instructions are executed. This feature might be used, for example, to fine tune the speed of execution of a program on a step-by-step basis.

In addition to accepting textual material from the system terminal, the editor has a simple mode of operation whereby a program step and location definition are automatically generated each time the RECORD button on the manual control unit is depressed. This is called teach mode. Each program step produced this way consists of a motion instruction having as its destination the location of the robot tool at the time the button was pressed. In this way, programs or segments of programs consisting of sequential motion commands can be quickly entered. Subsequently, programs produced in this manner can be modified using the standard editor commands.

3.3.2 Subroutines

To simplify the combination of existing task segments to define new tasks, VAL programs can include subroutines. A subroutine is a separate program containing a group of VAL instructions defining a task or portion of a task. The instructions in the subroutine do not appear in the referencing program. There is just an instruction causing program execution to branch to the subroutine. When execution of the subroutine is completed, VAL returns to the original program and continues. Subroutines can call other subroutines, with up to ten such levels possible.

There are two main advantages to the use of subroutines. If a sequence of instructions appears several times in a program, they can be stored once as a subroutine and logically included in the program any number of times with subroutine calls. This can save considerable memory space. The second advantage to the use of subroutines is the modularity they encourage. Subroutines can be written for frequently encountered tasks, and combined for individual applications.

3.4 LOCATION VARIABLES

Data representing the position and orientation of the robot tool are referred to as robot locations. There are two ways robot locations can be represented in the system.

3.4.1 Precision Points

A location can be expressed in terms of the positions of the individual robot joints, and is then called a precision point. This yields maximum precision when the robot is directed to move to the location. Precision points have the disadvantages of being robot dependent, and their components cannot be manipulated during program execution.

3.4.2 Transformations

Alternatively, locations can be expressed in terms of the Cartesian coordinates (X, Y, Z) and orientation angles of the robot tool relative to a reference frame fixed in the base of the robot. (This reference frame is called the world coordinate system.) Such location variables are called transformations, and provide a more intuitive representation of locations than do precision points, since the components are easily related to the workspace. Furthermore, since transformations only refer to the position and orientation of the tool, they are independent of the geometry of the robot.

The use of transformations is especially advantageous where relative translations and rotations are to be performed. This is because the VAL system provides means to easily alter components of transformations--with commands entered at the terminal, and with instructions that can be included in user programs. It is also possible to use transformations to define locations relative to other locations.

However, the use of transformations to define locations can at times be somewhat inaccurate due to the complexity of the computations necessary to convert between Cartesian world coordinates and joint variables. Thus, if maximum precision is required, precision points should be used.

3.4.2.1 Compound Transformations - For applications requiring the definition of locations relative to other locations (or reference frames), compound transformations are available. These transformations are written as strings of transformation names separated by colons. The transformations comprising a compound transformation are called relative transformations because they define locations relative to other locations. Compound transformations are discussed in Appendix A.

3.4.3 Defining Location Variables

Locations are stored in the system as named variables. That is, locations are referred to by user-selected names, and the location corresponding to any name can be changed if required. The names for precision points and transformations can have an arbitrary number of characters. Each name must

start with a letter and can contain only letters, numbers, and periods. Precision-point names must always be preceded by the symbol "#" to distinguish them from transformation names. Thus, the same name can be used for a precision point and a transformation (for example, #PART and PART). The number of precision points and transformations that can be stored in the system is limited only by the size of the system memory.

Location data can be defined and modified using any of a number of methods provided in VAL. The robot tool location can be recorded using the manual control RECORD button as mentioned above. Monitor commands provide means for defining and modifying locations, and they can be defined while program statements are being typed to the program editor. Finally, locations can be defined and modified dynamically during execution of user programs. The commands and program instructions for location definition and modification are decribed in Chapters 4 and 5.

When the term "location" is used in this manual, it is meant to refer to any of the above location-variable types unless there is a statement to the contrary. That is, when a location is called for, one can usually use a precision point, a transformation, or a compound transformation.

3.5 PROGRAM EXECUTION

Once a user program has been entered into the VAL system, it is possible to execute the program once, a number of times, or indefinitely. Manual controls can be used to halt programs immediately, or after completion of the current step. Programs can be halted by command from the console, and can be executed a single step at a time for debugging purposes.

After a program has halted as a result of a command, push button, or an unexpected runtime error condition, it is normally possible to continue execution without restarting the program from the beginning. That is, after any problems have been corrected, program execution can be continued either at the step interrupted or at the following program step.

As aids to program development, VAL contains a command for slowing the speed of execution of an entire user program. Also, a monitor command can be used to execute a single VAL instruction. For example, it can be used to move the robot to named locations without having to create separate one-step programs.

3.5.1 System Interrogation

A monitor command can be used to determine the status of the user program currently executing. This command displays data including the number of times the program has executed, and information on the program step currently being executed. If the main program has called one or more levels of subroutines, information regarding the status of each of the subroutines is

given. The overall speed setting and the current tool transformation assignment are also displayed.

Commands are also provided for displaying user programs on the system terminal, displaying and altering the state of system operational options, and displaying current memory space usage.

In addition to these means of interrogating the system, instructions are available for displaying information from user programs. Thus, programs can generate messages to the operator.

3.6 TRAJECTORY CONTROL

VAL uses two different methods to control the path a robot follows from one location to another. These "trajectory generation" methods differ with respect to computational accuracy and speed, and can be classified according to the type of path the tool tip moves along. The methods either (1) interpolate between the initial and final position of each joint, producing a complicated tool-tip curve in space, or (2) move the tool tip along a straight-line path.

Neither method is ideal in the sense it produces an optimal path for all applications. With VAL, the user can choose what type of motion is used for each robot motion. Also, the continuous-path feature can be used to have the robot perform any consecutive motions as a smoothly connected sequence.

The trajectory generation methods are described below. Note that the interpretation of motion speed is different for the two types of motion.

3.6.1 Joint-Interpolated Motion

Joint-interpolated motions are produced by interpolating each joint control variable (joint angle, for example) from its initial value to its desired final value. All the joint variables are interpolated to make the joints complete their motions simultaneously, and the total motion time is set to that of the joint requiring the longest time to complete its motion. Joint-interpolated motions move each joint the minimum amount required to achieve the desired final position, and thus move the robot tool along controlled, predictable paths. Also, joint-interpolated motions provide the fastest response of the robot.

For joint-interpolated motions the monitor and program speed settings are combined to determine the fraction of maximum acceleration and speed at which the joints are driven. Assuming its motion is large enough, each joint is accelerated to a constant speed, moved at that speed, and then decelerated to its final position.

While the path is predictable, major drawbacks of joint-interpolated motion for robots with rotating joints are that the robot tool moves along a complex curve in space, and the speed at which the tool moves is not constant during the motion.

3.6.2 Straight-Line Motion

Straight-line motions are produced by applying an interpolating function to the world-coordinate location of the robot tool and rapidly transforming the interpolated tool location to joint commands. This type of motion is useful when the robot tool must move relative to an external object, or under command from a sensor. An example of the first situation is a robot performing a precision insertion, where the relative motion required between the robot and the workpiece is along a straight line.

An example of moving under sensor control is visually-controlled arc welding, where the robot tool path is continuously corrected using information from a camera. Since feedback from the camera is typically an incremental correction signal in the X, Y, or Z position of the welding tip in the world coordinate system, it is very convenient to have this correction added to the command produced by the Cartesian interpolation function.

The motion speed of the robot tool tip can be accurately controlled when straight-line motion is used because the system interprets the commanded motion speed as a Cartesian speed. That is, the joints of the robot are driven in such a way that the tool tip accelerates to a constant speed, travels at the constant speed, and decelerates to its final position. During the motion, the tool is smoothly rotated from its initial orientation to its final orientation. Under limited circumstances a joint speed required to maintain a constant tool speed is greater than the joint maximum speed. If this occurs, the joint is driven at its maximum speed, and all the other joints are slowed to compensate so the tool follows a straight-line path, although at a speed slower than requested.

Straight-line motions have the disadvantage of being slower than corresponding joint-interpolated motions, and thus straight-line motions should only be used where needed for path control or tool speed control. At all other times, joint-interpolated motions should be employed. This is particularly significant when continuous-path motions are being performed.

3.6.3 Enhanced Trajectory Control

When moving a robot tool to a desired location, often no direct path exists and motions to several intermediate locations must be performed. For example, tools must normally be withdrawn from the location where they were last used. And it is often necessary to approach new work locations from a different direction than would be produced if a direct approach were taken. To reduce the number of locations that must be explicitly taught in such

situations (and the memory required to store extra locations), the VAL language includes instructions for producing motions to retract the tool, and to locate the tool above a final destination. These "departure" and "approach" motions are performed relative to the current tool location, or a specified location, respectively, and greatly facilitate the development of subroutines to perform common operations.

Since no activity takes place at departure and approach locations, or at any locations introduced to avoid obstacles, it is often undesirable for the robot to stop at these locations. Stopping would waste time and introduce unnecessary accelerations to an object being carried by the tool. In applications such as arc welding, stopping at each location is more than undesirable, it is disastrous because such applications require that the tool be kept in motion, often at a preprogrammed, controlled speed. For these situations VAL has a continuous-path feature that smoothly transitions between adjacent motions; subject to the maximum allowable accelerations of the robot. Such transitions can occur between any combination of straight-line and joint-interpolated motions.

CHAPTER 4

MONITOR COMMANDS

This chapter presents all the commands recognized by the VAL monitor. The following information is provided for each command: (1) the command syntax (refer to the Section 1.3, Notation, for meanings of angle and square brackets); (2) an indication of when the command can be issued; (3) a description of the function performed by the command; and, (4) if the command takes arguments, a representative example of how the command might look when typed, with an explanation of its effect.

When a command is typed, spaces are optional except that at least one space is required after the command name and between elements not otherwise separated (for example, with a comma or equals sign); indicated commas must be provided as necessary to correctly position the arguments (refer to Section 1.3, Notation).

4.1 DEFINING LOCATIONS

There are several commands that can be used for defining locations and determining the current location of the robot. In the following commands, all distances are to be in millimeters, and each "location" argument can be satisfied with either a precision point, a transformation, or a compound transformation.

The names for precision points and transformations can have an arbitrary number of characters. Each name must start with a letter and can contain only letters, numbers, and periods. Precision-point names must always be preceded by the symbol "#" to distinguished them from transformation names.

POINT <transformation> [= <transformation 2>] ... [:<transformation n>]
or POINT <precision point> [= <precision point 2>]

[.*] Sets the value of the location variable on the left equal to that of the variable on the right. If the second variable is not specified, the value of the first variable is either left unchanged, or set equal to a default value if it was previously undefined.

The value of the first variable is displayed and its components may be modified by typing new values after the query "CHANGE?". Component values must be separated by commas, and values not to be changed may be omitted. Changes are asked for continuously until no change is made. If a precision point is being defined, all its joint variables are

displayed. Otherwise a transformation is being defined, and its value is displayed as X, Y, Z, O, A, T, where X, Y, Z specify the position of a point centrally located between the finger tips in world coordinates, and O, A, T specify the orientation of the hand (Euler angles in degrees). With all three angles zero, the hand points in the negative Y direction with the fingers in a horizontal plane. An orientation vector which points from one finger tip to the other is parallel to the X axis. O specifies a rotation of the orientation vector about a vertical direction. Thus at O = 90°, the hand, still horizontal, would point along the positive X direction. A is a rotation about the orientation vector. With A = 90°, the hand always points straight down. T is a final rotation about the axis of the wrist and corresponds to a rotation of the final joint.

POINT PICK1 = PICK	Sets the value of transformation PICK1 equal to that of transformation PICK, and allow for changes.
POINT #PARK	Prepares for definition or modification of the precision point PARK.

DPOINT [<location 1>],...,[<location n>]

[.*] Deletes the values of an arbitrary number of location variables (precision points and/or transformations, but not compound transformations). This operation can be used to recover most of the memory storage required by location variables no longer needed. Deleted locations cannot be listed using any of the output routines; attempts to reference such variables within program instructions generate error messages at execution time until their values are redefined.

DPOINT PICK,#PLACE	Deletes the values of transformation PICK and precision point PLACE.

HERE <location>

[.*] Defines the value of a transformation or precision point to be equal to the current robot location. As with POINT, post modification of the location coordinates is permitted.

Only the right-most transformation of a compound transformation is defined. An error message results if any of the other transformations in a compound transformation are not already defined.

HERE PLACE	Defines the transformation PLACE to be equal to the current robot location.
HERE #PICK	Assigns the current location of the robot to the precision point PICK.

WHERE 1 (ONE) - DYNAMIC JOINT AND ABSOLUTE
WHERE 2 (TWO) - " JOINT ANGLES RELATIVE

WHERE

[.*] Displays the current location of the robot in Cartesian world coordinates and joint variables, together with the current hand opening if the robot is equipped with a position-servoed hand. (Note that some joint angle values displayed are shifted by 360°. That is, a value shown as 170° can also be interpretted as -190°.)

TEACH <location>

[.*] This command is used for recording a series of location values under the control of the RECORD button on the manual control unit. Each time the RECORD button is depressed, one location variable is defined and given the value corresponding to the location of the robot at the instant the RECORD button is pressed. Each successive location variable is automatically assigned a name which is the result of concatenating the base name (specified by the argument to this command) together with a number one larger than that for the previous location variable. (For example, if the command "TEACH P1" is typed, the first recorded location variable is P1, the next P2, followed by P3, and so forth.) If a compound transformation is specified, the right-most transformation is operated upon.

The display on the manual control box blinks when the RECORD button is pressed to signal the user that the location has been recorded. This mode is terminated by pressing the RETURN key on the terminal.

Note that there are other teaching commands that permit the creation of motion instructions at the same time as defining locations. These commands (T and TS) are described in Section 4.2, Program Editing.

TEACH WELD1 — Initiates monitor teach mode so that when the RECORD button on the manual control is pressed, transformations WELD1, WELD2, etc. are defined, consecutively, as the robot location at the instant the button is pressed.

BASE [<dX>],[<dY>],[<dZ>],[<Z rotation>]

[.] When the VAL system is initialized, the origin of the reference frame of the robot is assumed to be located at the intersection of the axes of joints 1 and 2, with the X, Y, and Z axes fixed in space. This command offsets and rotates the reference frame as specified. A default value of zero is assumed for any omitted argument. (Refer to the hardware manual for your robot for the directions of these axes.)

If, after locations have been defined by transformations relative to the robot reference frame, the robot is moved relative to those locations--to a point [<dX>,<dY>,<dZ>] in the reference frame, and rotated <Z rotation> degrees about the Z axis--the BASE command can be used to compensate so

that motions to the previously defined locations will still be as desired. Note that this command has no effect on locations defined as precision points.

A convenient use for this command is to realign the coordinate axes so that SHIFT and DRAW instructions cause displacements in desired, non-standard directions.

BASE 300,,-50,30 — Redefine the world reference frame because the robot has been shifted 300 mm in the positive X direction and 50 mm in the negative Z direction, and has been rotated 30 degrees about the Z axis, from its initial location and orientation.

TOOL [<transformation>]

[.] The internal matrix used to represent the location and orientation of the tool tip relative to the location and orientation of the tool mounting flange of the robot is set equal to the value of the specified transformation. If no transformation is specified, the tool transformation is set equal to the value of the "null tool." The null tool has its center at the surface of the tool mounting flange and its coordinate axes parallel to that of the last joint of the robot (it is represented by the transformation [0,0,0,90,-90,0]). The tool transformation is automatically set equal to that of the null tool on system power up, and after a ZERO command. See Appendix B for an examples of the use of tool transformations.

The relative tool transformation is automatically taken into consideration each time the position of the robot is requested, when a command is issued to move the robot to a location defined by a transformation, and when manually-controlled motions are performed in world or tool coordinates. Note that if the tool transformation is modified after the TOOL command is issued, the change does not affect motions of the robot until the TOOL command is reissued.

This function can be issued as a top-level monitor command or as a program instruction. However, the monitor TOOL command cannot be issued when the manual control is in either TOOL or WORLD mode.

TOOL HAND — Replaces the current tool transformation assignment with the value of the transformation HAND.

4.2 PROGRAM EDITING

As mentioned previously, user written programs are used to direct the motions and actions of the robot. The commands associated with the VAL editor are explained in this section. These editing commands can be used for creating and modifying user programs.

Programs are referred to by names consisting of an arbitrary number of characters. Like location variables, program names must begin with a letter and can be followed by any sequence of letters, numbers, and periods. The number of programs that can be stored is limited only by the amount of system memory available.

Each instruction within a program is referred to by a unique step number. Steps are numbered consecutively and the numbers are automatically adjusted by the editor whenever instructions are inserted or deleted. Because these step numbers are likely to be changed as a program evolves, they would not be useful for identifying steps for program-controlled branching. For this use program steps can contain step labels, which are integer numbers ranging from 0 to 32767.

The complete format of a program step is

<step number> [<label> <space>] <instruction name> [<space> <arguments>]

Note that when a line is entered into the system, extra spaces can be entered between any elements in the line; those indicated above are the minimum required. The VAL editor always adds or deletes spaces in program lines to make them conform with its standard spacing.

The first editor command below is used to start an editing session. Then the other commands are used to view the program and make changes.

EDIT [<program>],[<step>]

[.*] Permits the user to create or modify the program named <program>, starting at the indicated step. If no step number is specified, editing proceeds from the first step of the program. If no program is specified, the last program edited is reopened; if the step number is also omitted, editing proceeds from the step before where the last editing session was terminated.

EDIT ASSEMBLY — Accesses the program named ASSEMBLY for editing, at its first step if the program does not currently exist, or was not the last program edited; otherwise editing starts at the step preceding the one at which editing was last terminated.

<any program instruction>

[.*] Stores the program instruction at the current step. If an instruction is already present at the step, it is replaced by the instruction typed in. The next step is then displayed for editing.

<carriage return>

[.*] Typing an empty line (that is, pressing just the RETURN key) leaves the current step unaltered and displays the next instruction for editing. If the current step contains no instruction, the command is ignored.

C <program>,[<step>]

[.*] Changes editing from the current program to a new user program starting at the indicated step. <Step> can be omitted, and editing begins with the first instruction of the program. Note that this command has the effect of an E command followed by an EDIT command.

C DRUM Changes to the program named DRUM at its first step.

D [<nstep>]

[.*] Deletes the specified number of instructions starting with the current instruction. If nstep is not specified only the current step is deleted. All subsequent steps are automatically renumbered.

E

[.*] Exits EDIT mode and returns to MONITOR mode.

I

[.*] Moves instructions (starting at the current step) down one step, and inserts the program instruction typed in next. This process is automatically repeated until an empty line is entered (that is, just the RETURN key is pressed).

L

[.*] Leaves the current step unaltered and displays the previous step for editing.

P [<nstep>]

[.*] Displays the next <nstep> program steps and sets the current line number equal to that of the last line displayed. If omitted, nstep defaults to one.

R <character string>

[.*] Replaces characters in the currently displayed program instruction. Starting at the location under which the "R" is typed, the program instruction is replaced character for character. A space character must separate the name of this function (that is, "R") from the character string. The first character substituted is the character following the space character, and the substitution process continues until the RETURN key is pressed. As soon as RETURN is typed, an attempt is made to reinterpret the program instruction. Thus the resulting text must be a valid program instruction after the substitution is performed.

For example, the command "R 123" would replace the character under which the "R" was typed with a "1" and the following two characters with "2" and "3", respectively.

S [<step>]

[.*] Leaves the current step unaltered and displays the instruction at the indicated step in the user program. If no step is given, the first instruction of the program is displayed for editing.

T <location>

[.*] Initiates "joint-interpolated" program-teach mode. In this mode, each time the RECORD button on the manual control box is depressed, a MOVET instruction is inserted into the program being edited. The instruction generated contains as its arguments a location variable (with its name derived from that supplied to this command) and the current hand opening. The value of the location variable is set equal to the tool position and orientation at the time the RECORD button is pressed. Successive variables are assigned names in exactly the same manner as with the monitor TEACH command. That is, each name is a concatenation of the base variable name and a number. The suffix number is incremented each time a new instruction is generated. (See the TEACH command for an example of how names are produced.) By successively manually positioning the robot and pressing the RECORD button, a complete motion task can be taught. The display on the manual control box blinks when the RECORD button is pressed to signal the user that the location has been recorded. Teach mode is terminated by pressing RETURN on the terminal keyboard.

T #WELD1 — Initiates program teach mode so that when the RECORD button on the manual control is pressed, a joint-interpolated motion instruction is generated for motion to the robot location at that instant, with the locations to be defined by precision points WELD1, WELD2, etc., consecutively.

TS <location>

[.*] Initiates "straight-line" program-teach mode. This mode is like "joint" teach mode above, except that a MOVEST instruction is generated each time the RECORD button on the manual control box is depressed. Thus, when the program is executed, there is straight-line motion to the corresponding location rather than joint-interpolated motion. As with the other teach modes, the RETURN key on the terminal keyboard is pressed to terminate this mode.

TS SPOT1 — Initiates program teach mode so that when the RECORD button on the manual control is pressed, a straight-line motion instruction is generated for motion to the robot location at that instant, with the locations to be defined by transformations SPOT1, SPOT2, etc., consecutively.

4.3 PROGRAM AND LOCATION-DATA LISTING

The following commands are used to view programs and the values of location variables. The execution of these commands can be aborted by pressing the RETURN key on the terminal keyboard.

DIRECTORY

[.*] Displays the names of all user programs in the system memory that have at least one program step defined.

LISTL [<location 1>],...,[<location n>]

[.*] Displays the values of the listed location variables. If no location-variable names are specified, the values of all defined location variables are displayed.

LISTL PICK,#PLACE — Displays the values of transformation PICK and precision point PLACE on the system terminal.

LISTP [<program 1>],...,[<program n>]

[.*] Displays all the steps of the listed user programs. If no program names are specified, all programs with at least one program step defined are listed.

LISTP ASSEMBLY,MOTOR Displays the programs named ASSEMBLY and MOTOR on the system terminal.

4.4 PROGRAM AND LOCATION-DATA STORAGE

The following commands are used to store and retrieve user programs and location data on minifloppy diskettes.

The diskette is logically divided into files, which are referred to by names with six or fewer characters. File names must begin with a letter followed by any combination of letters and numbers. Files containing user programs are distinguished from files containing location data: program file names have the extension ".PG" appended, while location-data file names end with the extension ".LC". These file-name extensions are automatically added to the six-character file name typed by the user and therefore do not have to be typed. (The one exception to this rule is the DELETE command.) File names must always be unique for any given diskette; however, it is possible to create one program file and one location-data file with the same name since each is assigned a different file extension.

In all cases where an operation may result in destructively writing over valid data contained on the diskette, verification is first requested before the operation is performed. The execution of these commands can be aborted by pressing the RETURN key at any time.

Certain precautions should be taken when using floppy diskettes to make sure they are not damaged and their data not lost. The diskettes are fragile and particularly susceptible to bending. Thus, care should be exercised when handling diskettes, especially when inserting them into the disk drive.

TO INSERT A DISKETTE

1. Make sure the hole in the center of the diskette is approximately centered in the opening of the envelop.

2. Hold the diskette with the arrow on the top and pointing toward the drive.

3. Open the door of the drive and gently slip the disk into the slot until you feel it detent into position.

4. Gently close the door.

The following precautions should also be observed:

1. Do not touch the recording surface of diskettes.
2. Do not expose diskettes to large magnetic fields.
3. Do not write on diskette identification labels with a ball-point pen or pencil. Use only a felt-tip pen.
4. Do not use an eraser on diskette identification labels.
5. Do not fold or bend diskettes.
6. Do not place paper clips or rubber bands on diskettes.
7. Do not place heavy objects on a diskette.
8. Do not leave diskettes out of protective envelopes when not in use.
9. Do not leave diskettes near CRT terminals.
10. Do not expose diskettes to excessive environmental conditions.
    Allowable conditions for operating and storage:
    10°C to 52°C (50°F to 125°F)
    20% to 80% RH with no condensation
    Allowable conditions for shipping:
    -40°C to 52°C (-40°F to 125°F)
    5% to 95% RH with no condensation
    Diskettes must be acclimitized to operating temperature for at least one hour before use.

The useful life of a diskette is limited since the read/write head of the disk drive contacts the diskette during data transfers. (The red light on the front of the disk drive indicates when the head is in contact with the diskette.) The rated life of diskettes is about 150 hours of head contact time. Under normal circumstances, this should correspond to several hundred read/write cycles.

CAUTION

When using the floppy-disk drive, never open its access door while the red light on its front panel is on. This light indicates that the drive is in an active state, and should not be disturbed.

FORMAT

[.*] This command must always be executed before a new diskette can be used for storing programs or data. It creates the file directory and has the effect of erasing the entire diskette by writing over any information that it may contain.

FILE NAME - 6 CHAR MAX.

LISTF

[.*] Displays the file directory of the diskette currently loaded in the minifloppy disk, and the amount of space still available for storage.

STOREP <file> [= <program 1>],...,[<program n>]

[.*] Stores the specified programs in the indicated disk file. If no program names are given, all the programs in memory are saved in the disk file.

STOREP F1=CYCLE — Stores the program named CYCLE into a floppy-disk file to be named F1.PG.

STOREL <file> [= <program 1>],...,[<program n>]

[.*] Stores the names and values of all the location variables referenced in the specified user programs. If no programs are specified, all location variables with defined values are stored in the disk file.

STOREL F2=MOTOR — Stores all the location variables referenced by the program named MOTOR into a floppy-disk file to be named F2.LC.

STORE <file> [= <program 1>],...,[<program n>]

[.*] Stores the specified programs, and the location variables they reference, in a program file and a location file, each with the specified name. If no program names are given, all the programs and location variables in memory are saved in the disk files. Note that this command is equivalent to a STOREP command followed by a STOREL command.

STORE F3=CYCLE,MOTOR — Stores the two programs into a file named F3.PG, and stores all the location variables they reference into a file named F3.LC.

LOADP <file>

[.*] Loads the program(s) contained in the indicated disk file into the system memory. If a program already exists in memory which has the same name as one contained in the disk file, it is deleted and replaced by the program on the diskette.

LOADP F1 — Loads all the programs in floppy-disk file F1.PG into the system memory.

LOADL <file>

[.*] Loads all the location variables contained in the specified disk file into the system memory. If a location variable is already defined in memory and a variable with the same name is contained in the disk file, the value in memory is replaced by the value read from the diskette.

LOADL F2 — Loads all the location variables in floppy-disk file F2.LC into the system memory.

LOAD <file>

[.*] Loads the programs and location variables contained in <file>.PG and <file>.LC into the system memory. If a program or location variable already exists in memory, which has the same name as one contained in a disk file, it is deleted and replaced by the information on the diskette. Note that this command is equivalent to a LOADP command followed by a LOADL command.

LOAD F3 — Loads all the programs contained in file F3.PG, and the location variables contained in file F3.LC, into the system memory.

DELETE <file><ext>

[.*] After requesting confirmation, deletes the specified file from the diskette. For this command, the file name and extension (".PG" for program files and ".LC" for location data files) must be given. For example, to delete a file which was produced by the command "STOREP A", type "DELETE A.PG".

DELETE F3.LC — If the user responds with a "Y" to the subsequent confirmation prompt, deletes the floppy-disk file named F3.LC.

COMPRESS

[.*] The floppy-disk system uses a file structure in which files are allocated space on the diskette sequentially. Thus, deleting a file creates an unavailable area on the diskette. The COMPRESS command repacks the files and recovers any space made unavailable by earlier file deletion.

ERASE

[.*] Erases the contents of the floppy diskette and initializes the diskette file-directory information.

4.5 PROGRAM CONTROL

The following commands are used for controlling execution of user programs and individual program instructions.

SPEED <value>

[.*] Specifies the speed of all subsequent robot motions that are under program control. The value for the speed can range from 0.01 (very slow) to 327.67 (very fast), where 100 is "normal" speed. (This monitor speed parameter is set to 100 when the system is initialized.)

The speed at which robot motion occurs is related to the product of the speed set by this monitor command and the speed set by a program SPEED instruction. For example, if the monitor speed value is 50 and the program instruction speed setting is 60, the robot moves at approximately 30% of its "normal" speed. Note that motion speed has different meanings for joint-interpolated motions and straight-line motions (refer to Section 3.6, Trajectory Control).

SPEED 30 Sets the monitor speed parameter to 30% of "normal."

EXECUTE [<program>],[<nloop>],[<step>]

[.] Executes the specified user program <nloop> times. A loop is terminated when either a STOP instruction or the last defined step of the program is encountered. The value of nloop can range from -32768 to 32767. Negative values for nloop put the program into a continuous loop. If nloop is omitted or zero, the program is executed one time. If no program is specified, execution of the last user program EXECUTE'd is reinitiated. If a step is specified, the motion program begins execution at that step for the first pass, otherwise, the program starts execution at step 1. Successive passes always begin at the start of the program.

EXECUTE ASSEMBLY,-1 Initiates execution of the program named ASSEMBLY, with execution to continue indefinitely (until the user aborts its execution, a HALT instruction is executed, or a run-time error occurs).

ABORT

[*] Terminates program execution after completion of the step currently being executed.

NEXT [<program>],[<nloop>],[<step>]

[.] Controls program single-step-execution mode. If any of the three optional arguments are specified, program execution is initiated in exactly the same manner as for the EXECUTE command. Unlike the EXECUTE command, however, program execution is halted immediately after the first program instruction is completed. If all three arguments are omitted, the next program instruction is executed and the program is again halted. As with the PROCEED and RETRY commands (described below), a NEXT command with no arguments can only be executed after a PAUSE instruction, a non-fatal run-time error, or after single-step execution of the preceding program instruction.

NEXT ASSEMBLY,,23 Executes only step number 23 of program ASSEMBLY.

DO [<program instruction>]

[.] Executes the single specified program instruction as though it were the next step in a program. This command can be used to move the robot (for example, DO MOVE T1) or to alter the sequence of program step execution (for example, DO GOTO 100). If no instruction is specified, the instruction specified by the preceding DO is performed once again.

DO MOVE SAFE.POSITION Causes the robot to move to the location defined by the transformation SAFE.POSITION by means of a joint-interpolated motion.

PROCEED

[.*] Proceeds execution of the user program at the step following the one where execution was halted due to a PAUSE instruction, an ABORT command, or a run-time error. If a program is currently running but is in a WAIT loop (that is, waiting for an external signal to be received) typing PROCEED has the effect of skipping over the WAIT instruction. This command has no effect if a program is executing but is not in a WAIT loop.

RETRY

[.] Restarts program execution after a run-time error similar to the PROCEED command. After a RETRY command, however, execution resumes at the previously executed instruction step. This allows the user to RETRY a step that has been aborted.

When a program terminates execution due to a PAUSE, STOP, or HALT instruction, or from any of a number of run-time errors, a message is displayed on the terminal. To indicate where in the program the termination occurred, the message is followed by "STOPPED AT STEP <step>" or, if termination occurred during execution of a program subroutine, "STOPPED AT <subroutine

name>-<step>." In both cases, <step> is the program step number of the step immediately following the instruction at which execution stopped.

4.6 SYSTEM STATUS AND CONTROL

A number of commands can be used to determine the current state of the VAL system, to initialize it, and to terminate its execution. These commands are described below.

CALIBRATE

[.] Calibrates the joint-position sensors in the PUMA and PUMA 250 robot systems. Although the system can be run under manual or program control while uncalibrated, it is recommended that the CALIBRATE command be executed as soon as possible after power-up. (The robot becomes uncalibrated whenever system power is switched off; if the monitor program is restarted [see the DONE command below]; and, for the PUMA 250 robot, when the LIMP command is executed.)

For the PUMA robot, the CALIBRATE command can be issued when the robot is at any location within its working range, and causes all the robot joints to be driven a small amount. For the PUMA 250 robot, the robot must be in its nest, and the CALIBRATE command drives the robot out of the nest a short distance.

LIMP

[.] This command is used in the PUMA 250 robot system to assist with moving the robot into its nest. After requesting confirmation, all the robot joints become limp and can be manually moved to any position within their hardware limit stops. Control is returned to VAL by pressing the RETURN key, at which time the system assumes that the robot is in its nest and resets the joint servos.

CAUTION

If the robot is not in the nest when the RETURN key is pressed, subsequent motions will be erratic. Note that the robot is left uncalibrated after this command is executed.

STATUS

[.*] Displays status information for the user program being executed. This includes the name of the tool transformation in effect; the program (and subroutine) step(s) being executed; the speed set by the monitor SPEED command; and the number of program EXECUTE loops completed, and

those still remaining. Note that if the program is in an infinite EXECUTE loop (that is, if a negative value was given for "nloop"), the display of the number of loops completed can be off by 32,767*n counts [n = 1, 2, ...] since the loop counter is permitted to overflow.

FREE

[.*] Displays the percentage of available memory not currently in use. (As the available memory is being added up, a simple check of the whole area is made to ensure that the bookkeeping tags are consistent.)

ZERO

[.] Reinitializes the VAL system and deletes all defined locations and user programs. Since this is a totally destructive operation, it first asks for confirmation that the operation is to be performed.

DONE

[.] Causes the MONITOR program (that is, VAL) to stop, and the system to exit to the computer hardware debugging feature, the Octal Debugging Tool (ODT). The MONITOR program can be restarted by starting execution at memory location 173,000 (that is, by typing 173000G when the "@" prompt is displayed).

4.7 SYSTEM SWITCHES

Several system switches govern various features of the VAL system. The switches, and commands that set and display their states, are described below.

MESSAGES Controls whether output messages from TYPE and TYPEI instructions will be displayed on the terminal. Message output from PAUSE, HALT, and STOP instructions is not affected by this switch. The default setting of this switch is ENABLE'd.

CRT For terminals with a RUBOUT key instead of a BACKSPACE key this switch can be ENABLE'd to have RUBOUT have the desired effect. This switch is DISABLE'd at startup.

SRV.ERR Turns on the automatic monitoring of special hardware error conditions generated by the joint servos. Initially this switch is DISABLE'd.

CP — This switch can be used to turn off continuous-path processing (see Appendix C). This switch is initially ENABLE'd.

EHAND — For systems with a servo-controlled hand, this switch determines whether the system should send commands to the joint controller for the hand. When this switch is DISABLED'd, the system only responds to hand-control instructions by sending signals to the pneumatic solenoid valves.

VISION — Operates only when the VISION option is installed. ENABLE'ing this switch initializes the vision software and establishes the communication link to the vision subsystem. When this switch is DISABLE'd, which is the initial setting, vision instructions are prohibited from being executed.

HHT - HAND HELD TERMINAL

ENABLE <switch> or DISABLE <switch>

[.*] Turns system switches on and off, respectively. When referring to a switch, its name can be abbreviated to the minimum length required to define it uniquely.

ENABLE MESSAGES — Turns on the MESSAGES switch.

SWITCH [<switch>],...,[<switch>]

[.*] Displays the setting of one or more system switches. If no switches are specified, the settings of all switches are displayed.

SWITCH CRT,CP — Displays the settings of the CRT and CP switches.

CHAPTER 5

PROGRAM INSTRUCTIONS

The following sections describe the instructions which can be included in user-written programs.

5.1 ROBOT CONFIGURATION CONTROL

For an anthropomorphic, six-joint robot, most points in its workspace can be reached by assuming one of eight possible spatial configurations. Normally, the robot remains in the configuration in which it starts when the user program begins execution, or continues from a PAUSE instruction or run-time error. The only exceptions are (1) when a READY instruction is executed, (2) when a specific change in configuration is requested by the user program through execution of any of the instructions below, or (3) when the robot is forced to change from FLIP to NOFLIP or vice versa to keep joint 4 or 6 within stop limits.

When the user specifies a change in robot configuration, the requested change is accomplished during execution of the next motion instruction (that is, MOVE, MOVET, APPRO, or DEPART). Note that no configuration change is permitted during a straight-line motion, and that the DELAY instruction has the effect of cancelling a pending configuration change.

NOTE

Since the robot normally does not change configuration when a program begins execution, it is often good practice to begin programs with an initialization sequence that specifies the configuration to be used. (This is especially important when a PUMA 500 or 600 robot is going to start at the READY location, since the configuration at that location is ambiguous.)

RIGHTY or LEFTY

Requests a change in the robot configuration so that the first three joints of the robot resemble a human's right or left arm, respectively.

ABOVE or BELOW

Requests a change in robot configuration so that the "elbow" of the robot is pointed up (ABOVE) or down (BELOW).

FLIP or NOFLIP

Changes the range of operation of joint 5 to positive (NOFLIP) or negative (FLIP) angles. (Not applicable to the PUMA 500 robot.)

5.2 MOTION

In the following motion instructions, all distances are to be in millimeters and each "location" argument can be satisfied with either a precision point, a transformation, or a compound transformation.

Note that the DELAY instruction (described in Section 5.8, Miscellaneous) has an effect similar to a motion instruction because it is treated as a "move to nowhere" instruction.

MOVE <location>[!]

Moves the robot to the location and orientation specified by the variable <location>. Intermediate set points between the initial and final robot locations are computed by interpolating between the initial and final joint-variable values, resulting in a joint-interpolated motion. Any changes in configuration requested by the user are executed during the motion.

If the location name is followed by an exclamation point, <location> has its value set, while the instruction is being typed in, to the robot position and orientation at the instant the MOVE instruction is completed by the RETURN key.

MOVE #PICK! — Moves by joint-interpolated motion to the location described by the precision point PICK, which is defined as the location of the robot when the instruction is completed by a carriage return.

0=Closed 1=Open

MOVET <location>,<hand opening>

Generates a joint-interpolated motion (see the MOVE instruction above) to the location and orientation specified by the variable <location>. During the motion, the hand opening is changed to <hand opening> millimeters if a servo-controlled hand is used. Also, the pneumatic-control system receives an "open" signal if <hand opening> is greater than zero; otherwise it receives a "close" signal. A MOVET instruction can be

inserted into a program by pressing the RECORD button on the manual control while in "T" Teach Mode (refer to Section 4.2, Program Editing).

MOVET PART1,0 — Moves by joint-interpolated motion to the location described by the transformation PART1, closing the hand during the motion.

MOVES <location>[!]

Moves the robot to the location and orientation specified by the variable <location>. The tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion. (See MOVE for an explanation of the "!" option.)

MOVES PLACE — Moves along a straight-line path to the location described by the transformation PLACE.

MOVEST <location>,<hand opening>

Identical to the MOVET instruction, except that the tool is moved along a straight line and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion. A MOVEST instruction can be inserted into a program by pressing the RECORD button on the manual control while in "TS" Teach Mode (refer to Section 4.2).

MOVEST PART7,100 — Moves along a straight-line path to the location described by the transformation PART7, changing the hand opening to 100mm during the motion.

DRAW [<dX>],[<dY>],[<dZ>]

Moves the robot tool along a straight line, a distance dX in the X direction, dY in the Y direction and dZ in the Z direction. (Any omitted distances are assumed to be zero.) The robot configuration and tool orientation are maintained during this motion.

DRAW 100.5,,-40 — Moves the tool along a straight line 100.5mm in the X direction and 40 mm in the negative Z direction.

ALIGN

Causes the tool to be rotated so that its Z axis is aligned parallel to the nearest axis of the world coordinate system. This instruction is primarily useful for lining up the tool before a series of locations are taught. This is most easily done by using the monitor DO command (DO ALIGN).

APPRO <location>[!], <distance>

Moves the tool to the position and orientation defined by the variable <location> and an offset along the tool Z axis of the distance given. A positive distance sets the tool "back" (negative tool-Z) from the specified location, a negative distance offsets the tool "forward" (positive tool-Z). (See MOVE for an explanation of the "!" option.)

APPRO PLACE,75	Moves the tool, by joint-interpolated motion, to a location 75 mm from that defined by the transformation PLACE, with the offset along the resultant Z axis of the tool.

APPROS <location>[!], <distance>

Same as APPRO, but the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion. (See MOVE for an explanation of the "!" option.)

APPROS PLACE,-50	Moves the tool along a straight line to a location 50 mm from that defined by the transformation PLACE, with the offset along the resultant Z axis of the tool, and "beyond" the location PLACE.

DEPART <distance>

Moves the tool (by joint-interpolated motion) the distance given along the current Z axis of the tool. A positive distance moves the tool "back;" a negative distance moves the tool "forward."

DEPART 80	Moves the tool 80 mm back from its current location.

DEPARTS <distance>

Same as DEPART, but the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion.

DEPARTS 200	Withdraws the tool 200 mm along a straight-line path from its current location.

DRIVE <jt>,<change>,<speed>

Operates the single specified joint, changing its joint variable by <change> units (degrees or ~~millimeters~~). The joint number, <jt>, can be 1, 2, ..., n, where n is the number of robot joints. (A proportional hand is considered here as an additional joint.) The speed of the motion is governed by a combination of the speed given in this instruction and the monitor SPEED setting.

DRIVE 4,-62.4,75 — Changes the angle of joint 4 by driving the joint 62.4 degrees in the negative direction at a speed of 75% of the monitor speed.

READY

Moves the robot to the READY location above the workspace, which forces the robot into a standard configuration. Regardless of where the robot is originally located, this instruction always succeeds.

This instruction is particularly useful with the DO command (DO READY) to check for proper calibration of the robot.

CAUTION

Be careful that the robot does not strike anything while moving to the READY location.

NEST

This instruction, which is only for the PUMA 250 robot system, moves the robot into its nest.

CAUTION

To avoid possible damage to the robot, the NEST instruction should be used only after a READY instruction, and only the CALIBRATE and READY instructions should be used to move the robot out of the nest.

5.3 HAND CONTROL

The robot tool frequently has the form of some kind of grasping device, referred to as a hand. Provision is made for pneumatically powered two-state hands, and electrically driven proportional hands. For the latter, the proportional-hand option must be installed in the robot controller, and the EHAND switch ENABLE'd. The following instructions provide control of

the tool. If the tool is not a hand, these instructions will have to be interpreted appropriately.

OPEN [<hand opening>] or CLOSE [<hand opening>]

These instructions cause the pneumatic control valves to receive a signal to "open" or "close," respectively, during the next motion.

Simultaneously, if the robot is equipped with a servo-controlled hand, and the EHAND switch is ENABLE'd, the hand opening is changed to <hand opening> millimeters. If the hand opening is omitted, or negative, the hand is closed as far as possible. If the hand opening is very large, the hand is opened as far as possible.

Note that the OPEN and CLOSE instructions produce exactly the same effect for proportional hands, and the hand-opening argument, if given, is ignored for control of the pneumatic control valves.

OPEN 75 — During the next robot motion, instructs the pneumatic control valves to assume the "open" state; if a servo-controlled hand is operational, changes the hand opening to 75 mm.

OPENI [<hand opening>] or CLOSEI [<hand opening>]

These instructions have the same effects as OPEN and CLOSE except that the operations are performed immediately instead of during the next robot motion. A small time delay occurs to allow the hand to operate before the program proceeds.

CLOSEI 50 — Immediately instructs the pneumatic control valves to assume the "close" state; if a servo-controlled hand is operational, changes it's opening to 50 mm.

RELAX - WON'T WORK AND MORE

Immediately turns off the open and close pneumatic control solenoid valves, causing the pneumatic hand to become limp. (This instruction has no effect on systems using a four-way pneumatic control valve.)

GRASP <hand opening>,[<label>] - SERVO HAND ONLY

If a servo-controlled hand is installed and the EHAND switch is ENABLE'd, this instruction causes the hand to close immediately, and then checks to see if the final opening is less than the specified amount. If it is, the program branches to the step specified by the program label. If the optional program label is not specified, the program halts and an error message is displayed. Thus, this instruction provides a simple one-step

method for grasping an object and testing to ensure that contact has been achieved.

This instruction also causes the pneumatic control valves to immediately receive a "close" signal.

GRASP 12.7,120 — Closes the servo-controlled hand and checks whether the final opening is less than 12.7 mm. If it is, have program execution continue at the step with the label "120;" otherwise continue with the next step in the program.

5.4 INTEGER VARIABLE

In most cases where an integer variable is indicated as an argument for a user program instruction, either the symbolic name of an integer variable can be given or an integer value can be specified. (The only time that an integer value cannot be used in place of a integer variable is when the specified operation would result in an attempt to alter the value of the constant; in which case an error message is generated.) Integer variable names conform to the rules for location-variable names and program names, and their values must be between -32,768 and 32,767.

SETI <i.var> = <i.var 2> [<operation> <i.var 3>]

Sets the value of <i.var> equal to the result of the expression on the right-hand side of the equal sign. The permitted operations (and the symbols used to represent them) are addition (+), subtraction (-), multiplication (*), integer division (/), and modulus determination (%). If the operation and the third variable are missing, the instruction is a direct assignment statement.

SETI N=N+1 — Redefines the value of the integer variable N to be one larger than its current value.

SETI M=N%2 — Sets the integer variable M equal to the remainder resulting from the operation N/2.

TYPEI <i.var>

Displays the name and value of the specified integer variable. No output is done, however, if the MESSAGES switch is DISABLE'd.

TYPE COUNTER — Displays the current value of the integer variable COUNTER on the system terminal.

5.5 LOCATION ASSIGNMENT AND MODIFICATION

These instructions define and modify the destinations of robot motion instructions during program execution.

It is especially important to understand these very powerful instructions because they provide significant flexibility in VAL programs. For example, by accepting external input signals and then altering the base or tool settings, or defining or modifying the values of location variables, a user program can adapt to the robot working environment.

HERE <location>

Sets the value of a transformation or precision point equal to the current robot location. (This instruction is equivalent to the monitor HERE command.)

Only the right-most transformation of a compound transformation is defined. An error message results if any of the other transformations in a compound transformation are not already defined.

HERE PART	Sets the transformation PART equal to the current robot location.
HERE #PART	Assigns the current location of the robot to the precision point PART.

UNKNOWN KNOWN

SET <transformation> = <transformation 2>[:<trans 3>] ... [:<trans n>]

or SET <precision point> = <precision point>

Sets the value of the location variable on the left equal to that on the right of the equal sign. If the right-hand side is not defined, an error message is generated.

SET PICK=START	Sets the value of the transformation PICK equal to that of the transformation START.
SET #PLACE=#POST	Sets the value of the precision point PLACE equal to that of the precision point POST.

SHIFT <transformation> BY [<dx>],[<dy>],[<dz>]

Modifies the X, Y, and Z components of the indicated transformation by adding distance changes <dx>, <dy>, and <dz>, respectively. If the value of the transformation has not yet been defined, or its new X, Y, Z location is too large to represent, an error message is generated.

SHIFT PICK BY 100.8,-35.1	Redefines transformation PICK to be shifted 100.8mm in the X direction and 35.1 mm in the negative Y direction.

TOOL [<transformation>]

Sets the value of the tool transformation equal to the value of <transformation>. Refer to the monitor TOOL command for a description of the effect of this instruction.

TOOL POINTER — Replaces the current tool transformation assignment with the value of transformation POINTER.

INVERSE <transformation> = <transformation 2>[:<trans 3>] ... [:<trans n>]

Similar to the SET instruction. Sets the value of <transformation> equal to the matrix inverse of the transformation on the right. That is, if the transformation on the right represents a location, A , relative to another location, B , then the inverse transformation represents the location B relative to A . If the right-hand side is not defined, an error message is generated.

INVERSE TRANS.INV=TRANS — Defines the transformation TRANS.INV to be equal to the inverse of the transformation TRANS.

FRAME <transformation> = <trans 2>,<trans 3>,<trans 4>

Assigns a value to <transformation> which describes the relationship of a secondary reference frame to that of the robot. The secondary reference frame is assumed to have (1) its origin at the point defined by <trans 2>, (2) its positive X axis passing through the point defined by <trans 3>, and (3) its X-Y plane containing the point defined by <trans 4>. This instruction is usually used to define a "base" transformation for relative locations.

To improve accuracy, the points should be as far apart as practical, and the location defined by <trans 4> should be near the Y axis, and as far as practical from the X axis.

FRAME BASE=A1,A2,A3 — Defines the value of tranformation BASE to be a description of the reference frame which has its origin at the point described by transformation A1 , with the positive X axis passing from A1 through the point given by A2 , and A3 defining another point which lies in the X-Y plane.

5.6 PROGRAM CONTROL

The following instructions alter the sequence in which user program instructions are executed, and interlock the VAL system with other devices.

In this section, "<channel>" designates one of the external signal input or output lines, and can be either an integer number or an integer variable. The absolute value of <channel> specifies a particular signal line. The magnitude of <channel> must be between 1 and n , where n is the number of hardware bidirectional interlocks provided with the system. For certain instructions, the sign of <channel> specifies high or low signal values. A positive value indicates a high signal and a negative value indicates a low signal.

GOTO <label>

Performs an unconditional branch to the program step identified by the given label.

GOTO 95	Causes program execution to continue at the step with the label "95."

GOSUB <program>

Execution of the current program is temporarily suspended, and execution continues at the first step of the indicated user program, which is then considered a subroutine. Execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Up to ten programs can be superseded at any given time.

GOSUB PALLET	Branches to the program named PALLET. When a RETURN instruction is executed, control returns to this program.

RETURN [<skip count>]

Terminates execution of the current subroutine and resumes execution of the last-suspended program at <skip count> + 1 steps (not counting REMARK instructions) following the instruction which caused the subroutine to be invoked. If the skip count is omitted, or a count less than zero is specified, a value of zero is used. A RETURN in a main program has the same effect as a STOP instruction.

If the subroutine started execution due to the triggering of a REACTI instruction (see below), a skip count of one can be used to resume execution of the interrupted routine without executing the interrupted program step again.

RETURN 2 — Returns to the previously suspended program, at the third (2+1) step following the instruction which invoked subroutine execution.

IF <i.var> <relationship> <i.var 2> THEN <label>

Compares the value of <i.var> to the value of <i.var 2>; if the stated relationship is true, program execution branches to the program step identified by the given label. Otherwise, the next step of the program is executed as usual. The possible relationships that can be specified are EQ (equal), NE (not equal), LT (less than), GT (greater than), LE (less than or equal), and GE (greater than or equal). Note that either variable can be replaced by an integer value.

IF N GT 3 THEN 25 — If the current value of the integer variable N is greater than 3, continue program execution at the step with the label "25," otherwise continue with the next step.

PAUSE [<string>]

Terminates execution of the user program, and displays the message <string>. Execution can be continued from this point by typing PROCEED.

PAUSE POSITION THE PART — Stops program execution and displays "POSITION THE PART" on the system terminal. The operator can have program execution continue by typing PROCEED.

STOP [<string>]

Terminates execution of the user program unless more program loops (see the EXECUTE command) are to be completed, in which case execution of the program continues at its first step. If provided, the message <string> is displayed on the terminal. The STOP instruction is used to mark the end of a program execution pass. Note that the HALT instruction has a different effect.

STOP END OF TEST — Terminates a program execution loop and displays "END OF TEST."

HALT [<string>]

Terminates execution of the user program regardless of any program loops (see the EXECUTE command) remaining to be completed, and displays the optional message <string> on the terminal. After termination by a HALT instruction, program execution cannot be resumed with a PROCEED command.

HALT END OF PROGRAM — Stops program execution and displays "END OF PROGRAM."

IFSIG <channel>,[<channel>],[<channel>],[<channel>] THEN <label>

If the state(s) of the indicated external input signal(s) exactly match the state(s) specified, the program branches to the instruction identified by the given label. That is, if any mismatch is detected, the next program step is executed. Channels are specified as signed values to indicate whether the tests are to be made for high or low signals. Omitted channel numbers or channel numbers with a value of zero always satisfy the matching test.

IFSIG 2,-3,, THEN 15 — If external input signal line #2 is high and line #3 is low, continues program execution at the step with the label "15," otherwise execution continues with the next step. Note that the commas after the "3" are required to correctly complete the rest of the instruction.

NOTE TRAILING COMAS

SIGNAL <channel>,[<channel>,...,<channel>]

CAN BE VARIABLE

Turns the signal(s) on or off at the specified output channel(s). Positive channel numbers turn the corresponding signals on, negative numbers turn the signals off.

SIGNAL -1,4 — Turns off the signal at output channel #1, and turns on the signal at output channel #4.

REACT <channel>,[<program>] [ALWAYS]

Initiates continuous monitoring of the external signal at the specified input channel. If a high value is detected, the program "reacts" by altering the sequence in which the following program steps are executed. The reaction is equivalent to performing a "GOSUB <program>" after completion of the program step during which the high signal first occurs. If ALWAYS is specified, the signal monitoring is active until an IGNORE instruction (see below) is executed, or until the reaction is triggered (in which case the equivalent of an "IGNORE <channel> ALWAYS" is automatically performed). If ALWAYS is omitted, the signal is only checked until completion of the next motion instruction. If no program is referenced, the program step following that during which the first high value occurs is skipped and execution continues at the next program step. Only one REACT or REACTI command can be associated with any channel at any given time.

REACT 3,DELAY — Monitors external input signal #3; if during the next motion instruction it becomes high, branch to subroutine DELAY after the motion.

REACTI <channel>,[<program>] [ALWAYS]

Like REACT, this instruction initiates the continuous monitoring of the external signal at the specified input channel. However, in this case when a high value is detected, the current instruction is immediately aborted and the equivalent of a "GOSUB <program>" is executed. The subroutine call to <program> is performed such that if a "RETURN 0" is encountered, the interrupted program step is executed once again. To return control to the interrupted program and skip further execution of the interrupted step, a "RETURN 1" must be used to exit from <program>. If no program is referenced, the current program step is immediately aborted and the following step is executed when a high signal on the channel is detected. The optional argument ALWAYS and the IGNORE instruction have exactly the same effect on REACTI as they do on the REACT instruction. Only one REACT or REACTI command can be associated with any channel at any given time.

REACTI 1,ALARM ALWAYS	Monitors external input signal #1; if it ever becomes high, immediately branch to subroutine ALARM.

IGNORE <channel> [ALWAYS]

Disables the REACT or REACTI instruction associated with the specified external-signal input channel. If ALWAYS is specified, the reaction is permanently disabled; otherwise it is disabled only until completion of the next motion instruction. The value <channel> must always be greater than zero.

IGNORE 5 ALWAYS	Stops monitoring of external input signal #5.

WAIT <channel>

Puts the program into a "wait loop" until the desired sense of the external signal at the specified input channel is detected. Positive and negative channel numbers indicate waiting should be done until the external signal goes true (high) or false (low), respectively. A signal wait loop can be aborted by using the monitor PROCEED command.

WAIT -4	Stops program execution until external input signal #4 becomes low.

5.7 TRAJECTORY CONTROL

The following instructions are used for enabling and disabling special features of the hardware position servo and the software trajectory generator. As indicated below, these instructions can include the word ALWAYS if it is desired that the requested option affect all successive motions. Whenever an instruction does not include ALWAYS, it is assumed that it is only to affect the next motion. Note that a DELAY instruction cancels the effects of these instructions when ALWAYS is not specified.

Since these instructions have the effect of turning features on and off, those features are sometimes called "switches." They should not be confused with the switches described in Section 4.7, System Switches.

SPEED <value> [ALWAYS]

Requests that robot motion be performed at the specified speed ("normal" = 100),. which can range from 0.01 (extremely slow) to 327.67 (extremely fast). (If no SPEED instruction is executed, a speed of 100 is applied.)

The speed at which motions are performed is determined by combining the value specified in this instruction and the monitor SPEED setting. Note that motion speed has different meanings for joint-interpolated motions and straight-line motions, as explained in Section 3.6, Trajectory Control.

SPEED 60 ALWAYS	Sets the program motion speed to 60 until changed by another SPEED command.

COARSE [ALWAYS]

Enables a low-tolerance feature in the hardware servo so that larger errors in the final position of the robot joints are permitted at the ends of motions. This results in faster motion execution when high accuracy is not required.

COARSE ALWAYS	Enables the low-tolerance feature until explicitly disabled.

FINE [ALWAYS]

Opposite of the COARSE instruction, this is the default state of the hardware servo system.

NONULL [ALWAYS]

When NONULL is in effect, commanded motions are terminated without waiting for the electronics to signal that all moving joints have reached their specified positions. Like the COARSE command, this mode allows faster motion if high final position accuracy is not required. However, since no position-error checking is done, motion execution times are fixed and final position errors can be quite large.

NULL [ALWAYS]

Opposite of the NONULL instruction, this is the default state of the software control system.

INTOFF [ALWAYS]

Turns off the hardware position-error-integration feature during path-controlled robot motions. However, hardware integration is always enabled during the final position-error nulling period.

INTON [ALWAYS]

Opposite of the INTOFF instruction. Causes the hardware to integrate position error throughout path-controlled robot motions. This is the default state of the software control system.

5.8 MISCELLANEOUS

The following instructions make a variety of capabilities available to the programmer.

BASE [<dX>],[<dY>],[<dZ>],[<Z rotation>]

Redefines the reference frame of the robot in the same way as the monitor BASE command.

BASE 100,,-50 — Redefines the world reference frame to effectively shift all locations 100 mm in the negative X direction and 50 mm in the positive Z direction from their current location. Note that the arguments for this instruction describe movement of the robot reference frame relative to its present location, and thus have an opposite effect on locations relative to the robot.

DELAY <time>

Puts the program into an idle loop for the specified period of time. The duration can be given as any value between 0.01 and 327.67 and is interpreted in seconds.

It should be noted that DELAY is interpreted as a "move-to-nowhere" motion instruction. In particular, (1) if there is a pending hand operation, the hand motion takes place during the execution of the DELAY instruction; (2) if any temporary trajectory switches have been specified, they are cleared after the conclusion of the DELAY; and (3) if there is a pending configuration change, it is canceled.

DELAY 2.5	Causes all robot activity to stop for 2.5 seconds, any pending hand operation occurs, clears any temporary trajectory switches that may be set, and cancels any pending configuration change requests.

ENABLE <switch> or DISABLE <switch>

These instructions have the same effects on the system switches as the corresponding monitor commands.

ENABLE MESSAGES	Turns on the MESSAGES switch.

REMARK [<string>]

Provides a "comment" line in a program. That is, REMARK instructions are used only for the programmer's benefit, and are ignored when a program is executed.

REMARK COUNT LOOPS	Includes a reminder to the programmer in the program.

TYPE [<string>]

Displays the message <string> on the terminal; a blank line is output if no string is provided. If the MESSAGES switch is DISABLE'd no output is done.

TYPE Program MOTOR	Display the message "Program MOTOR" on the system terminal to direct the operator.

CHAPTER 6

SAMPLE PROGRAMS

The examples in this chapter are intended to demonstrate the use of various VAL instructions. The tasks used in the examples have been chosen because they are representative of common types encountered.

6.1 PROGRAM INITIALIZATION

It is often useful to begin a program with an initialization sequence to make sure certain conditions are as required for proper execution of the program. The following example establishes the configuration the robot is to assume, and makes sure the hand is open.

Since one never knows when it might become desirable to have a program invoked as a subroutine by another program, it is good practice to terminate a program with a RETURN instruction unless a STOP or HALT is required for some reason.

```
REMARK  START OF PROGRAM TO ...
REMARK
REMARK  ASSURE THE PROPER ROBOT CONFIGURATION
RIGHTY
ABOVE
NOFLIP
REMARK  ASSURE CORRECT INITIAL HAND OPENING
OPENI   100.00
REMARK  START OF TASK INSTRUCTIONS
  .
  .
  .
REMARK  END OF TASK INSTRUCTIONS
REMARK
REMARK  END WITH "RETURN" SO THIS ROUTINE CAN BE USED AS A
REMARK          SUBROUTINE LATER IF DESIRED
RETURN  0
REMARK
REMARK  -- END OF PROGRAM --
```

6.2 PALLETIZING

This example demonstrates the use of the SETI, IF, SET, and SHIFT instructions. The task to be accomplished is to pick objects from a tray (pallet) with six rows (50 mm apart) and 12 columns (30 mm apart) of bins. The instructions shown here only define the locations of successive objects. Other instructions would be included in the indicated space to perform the desired operations on the objects.

The plane of the pallet is assumed to be parallel to the X, Y plane of the robot world coordinate system. The rows are assumed to have a direction 30° from the normal X axis, as indicated by the BASE instruction below. If these assumptions are not correct, the arguments in the SHIFT instructions would have to be changed to represent the correct X, Y, Z displacements between pallet positions.

A "main" program that will administer this task follows.

```
      REMARK  PROGRAM TO PICK OBJECTS FROM A PALLET
      REMARK
      REMARK  INITIALIZE VARIABLES FOR THE PALLET SUBROUTINE
      REMARK
      REMARK  SET THE REFERENCE FRAME AXES PARALLEL TO PALLET
      BASE    0.00, 0.00, 0.00, 30.00
      REMARK  SET VALUES OF MAXIMUM ROW AND COLUMN COUNTS
      SETI    MAX.ROW = 6
      SETI    MAX.COL = 12
   10 REMARK  SET ROW COUNTER TO FIRST ROW
      SETI    ROW = 1
      REMARK  SET COLUMN COUNTER TO FIRST COLUMN
      SETI    COLUMN  = 1
      REMARK  SET OBJECT LOCATION TO TAUGHT LOCATION IN THE PALLET
      SET     PICK = CORNER
   20 REMARK  INSTRUCTIONS DESCRIBING MOTIONS TO AND FROM LOCATION
      REMARK          "PICK" AND ACTIONS TO BE PERFORMED TO THE OBJECT
        .
        .
        .
      REMARK  REDEFINE "PICK" TO BE NEXT PALLET LOCATION
      GOSUB   PALLET
      REMARK  GO BACK AND PROCESS NEW PALLET LOCATION
      GOTO    20
      REMARK  GET HERE WHEN WHOLE PALLET HAS BEEN PROCESSED
      REMARK          BECAUSE OF "RETURN 1" IN SUBROUTINE
      REMARK  RETURN TO CALLING PROGRAM OR MONITOR
      RETURN  0
      REMARK  -- END OF PROGRAM --
```

The subroutine PALLET will define the successive values for location variable PICK.

```
        REMARK  SUBROUTINE PALLET
        REMARK
        REMARK  PURPOSE:  CALCULATE PALLET LOCATIONS
        REMARK
        REMARK  INCREMENT COLUMN COUNTER
        SETI    COLUMN = COLUMN + 1
        REMARK  TEST FOR END OF ROW
        IF      COLUMN GT MAX.COL THEN 10
        REMARK  SHIFT ALONG ROW
        SHIFT   PICK BY 30.00, 0.00, 0.00
        REMARK  RETURN TO MAIN PROGRAM AND PROCESS NEXT OBJECT
        RETURN  0
        REMARK
     10 REMARK  ROW COMPLETED, INCREMENT ROW COUNTER
        SETI    ROW = ROW + 1
        REMARK  TEST FOR ALL ROWS DONE
        IF      ROW GT MAX.ROW THEN 20
        REMARK  SHIFT BACK TO START OF ROW, AND DOWN TO NEXT ROW
        SHIFT   PICK BY -330.00, 50.00, 0.00
        REMARK  RESET COLUMN COUNTER
        SETI    COLUMN = 1
        REMARK  RETURN TO MAIN PROGRAM AND PROCESS NEXT OBJECT
        RETURN  0
        REMARK
     20 REMARK  ALL PALLET POSITIONS HAVE BEEN USED, RETURN TO MAIN
        REMARK          PROGRAM AND SKIP THE "GOTO" INSTRUCTION
        RETURN  1
        REMARK  -- END OF SUBROUTINE PALLET --
```

Before these routines can be used, location CORNER must be defined, as well as any locations referenced by the omitted instructions. Location PICK is redefined to correspond the each pallet location during execution. If the pallet is later moved, only location CORNER need be redefined--all the other pallet locations are calculated by PALLET.

6.3 COMMUNICATING WITH EXTERNAL SIGNAL LINES

This sample program performs the following task: (1) the robot waits for a part to be in place in a feeder; (2) after picking up the part, the robot carries it to an inspection station, and signals the station that a part is in place; (3) the station determines whether the part is type "A" or "B" and sets the states of signal lines accordingly; (4) based on the output of the inspection station, the robot is directed by one of three subroutines to process the part; (5) the cycle repeats indefinitely.

The program will immediately branch to subroutine EMERGENCY if an emergency condition is indicated by input signal line #7 at any time from the start of the program until the IGNORE instruction.

```
    REMARK  START OF PROGRAM
    REMARK
    REMARK  INITIALIZE SIGNAL LINE
    SIGNAL  -2
    REMARK  MAKE SURE HAND IS INITIALLY OPEN
    OPENI   100.00
    REMARK
 10 REMARK  START OF LOOP TO PROCESS PARTS
    REMARK
    REMARK  START LOOKING AT "EMERGENCY" SIGNAL ON INPUT CHANNEL 7
    REACTI  7,EMERGENCY ALWAYS
    REMARK  WAIT FOR "PART IN PLACE" SIGNAL ON INPUT CHANNEL 1
    WAIT    1
    REMARK  PICK UP PART FROM FEEDER
    SPEED   200.00
    APPRO   PART, 50.00
    MOVES   PART
    CLOSEI  0.00
    DEPARTS 50.00
    REMARK  MOVE TO INSPECTION STATION
    APPRO   TEST, 75.00
    MOVES   TEST
    REMARK  STOP CHECKING FOR EMERGENCY CONDITION
    IGNORE  7 ALWAYS
    REMARK  SIGNAL THAT PART IS IN PLACE
    SIGNAL  2
    REMARK  WAIT FOR "INSPECTION DONE" SIGNAL ON INPUT CHANNEL 6
    WAIT    6
    REMARK  WITHDRAW FROM INSPECTION STATION
    DEPART  100.00
    REMARK  RESET "PART IN PLACE" SIGNAL
    SIGNAL  -2
    REMARK  TEST RESULTS OF INSPECTION; FIRST FOR PART "A"
    IFSIG   -3,4,-5,  THEN 20
    REMARK  THEN FOR PART "B"
    IFSIG   3,-4,-5,  THEN 30
    REMARK  PART IS NEITHER "A" OR "B" -- PROCESS REJECT
    GOSUB   REJECT
    GOTO    40
 20 REMARK  PROCESS PART "A"
    GOSUB   PART.A
    GOTO    40
 30 REMARK  PROCESS PART "B"
    GOSUB   PART.B
 40 REMARK  PART PROCESSING COMPLETED, GET ANOTHER PART
    GOTO    10
    REMARK  -- END OF PROGRAM --
```

The inspection station is assumed to perform tests that identify the part. The results are reported to the robot by the status of input signal lines 3, 4 and 5.

6.4 USE OF TOOL TRANSFORMATIONS

Suppose that in the last example the parts are the have a hole drilled in them, and it is to be in different places on the two parts. Thus, the part processing involves holding the parts under a drill in the proper position and orientation.

One way to program this task would be to have both subroutines PART.A and PART.B contain identical motion sequences, with the only difference being the locations involved. This approach will work, but it may be undesirable because of the duplication of instructions required. Also, this approach will get very complicated if several other work stations are involved.

Another way to program the task is to use the same instruction sequence for both parts, and use tool transformations to compensate for the differences between the parts. The instructions to do this might be as follows (omitted instructions are the same as in the preceding section).

```
          .
          .
          .
        REMARK  CALCULATE THE TOOL TRANSFORMATIONS
        INVERSE PART.A = INV.A
        INVERSE PART.B = INV.B
     10 REMARK  START OF LOOP TO PROCESS PARTS
        REMARK
        REMARK  SET THE TOOL TRANSFORMATION FOR THE NULL TOOL
        TOOL
          .
          .
          .
     20 REMARK  PROCESS PART "A"
        TOOL    PART.A
        GOTO    40
        REMARK
     30 REMARK  PROCESS PART "B"
        TOOL    PART.B
        REMARK
     40 REMARK  PROCESS THE PARTS
        APPRO   DRILL, 100.00
        MOVES   DRILL
        REMARK  SIGNAL THAT PART IS UNDER DRILL
        SIGNAL  3
        REMARK  WAIT FOR "DRILL DONE" SIGNAL
        WAIT    6
        REMARK  WITHDRAW FROM THE DRILL AND DROP OFF PART
```

```
WAIT     6
REMARK   WITHDRAW FROM THE DRILL AND DROP OFF PART
DEPART   100.00
MOVE     RELEASE
OPEN     100.00
DEPART   80.00
REMARK   GET ANOTHER PART
GOTO     10
REMARK   -- END OF PROGRAM --
```

Note that both types of part will be moved through the same sequence. The following sequence should be used to define the necessary transformations:

1. Move the empty hand to a location that is fixed relative to the drill. (This location should be well defined so the hand can be returned to it exactly if the drill should later be moved.)
2. Type "HERE DRILL" to define location DRILL.
3. Put a type-A part in the hand and position it under the drill.
4. Type "HERE DRILL:INV.A" to define the transformation INV.A .
5. Put a type-B part in the hand and position it under the drill.
6. Type "HERE DRILL:INV.B" to define the transformation INV.B .
7. Move to the drop-off location and type "HERE RELEASE".

Note that the transformations INV.A and INV.B are the inverses of the tool transformations used in the program. The INVERSE instructions in the program calculate the tool transformations from these transformations.

Once the transformations PART.A and PART.B are defined, the relationships between the robot and the two hole locations are established. This could be especially useful if the parts are to be processed at several machining stations. To teach the locations corresponding to other stations, the following sequence could be used:

1. Type "TOOL PART.A" to set the tool transformation for part A .
2. Put a type-A part in the hand.
3. Position the part at the desired work station.
4. Type "HERE <station name>" to record the location.
5. Repeat steps 3 and 4 for other the stations.

This not only teaches the locations for processing type-A parts, but also automatically includes type-B parts. This is because when type-B parts are processed, the tool transformation PART.B correctly represents the relationship between the hand and the hole in those parts.

APPENDIX A

COMPOUND TRANSFORMATIONS

Compound transformations provide a means of specifying tool locations and orientations relative to the locations and orientations represented by other transformations. This is a very useful facility in situations when several object locations have to be taught relative to a reference location. If the reference is moved, only the transformation specifying the reference need be updated. Then, all locations defined relative to the reference are automatically corrected for the change of the reference the next time they are used.

For example, if PLATE is the name of the transformation specifying the location of a base plate relative to the reference frame of the robot, and OBJECT is the relative transformation for the location of an object relative to the location of the plate, the compound transformation PLATE:OBJECT defines the location of the object relative to the reference frame of the robot.

It is possible to string together any number of relative transformations in this way. Hence, in the above example, if the object were to be grasped at a particular point, GRASP, which is defined relative to the location of OBJECT, a program instruction to move the robot tool to the grasp location could be written as follows:

```
MOVE PLATE:OBJECT:GRASP
```

It should be noted that the order in which relative transformations are specified is generally crucial. For example, if transformation B is defined relative to transformation A, then A:B yields the desired result and B:A does not.

It should also be noted that the use of compound transformations introduces positioning errors because of the cumulative computational errors involved; thus they should be used sparingly.

A.1 DEFINING RELATIVE TRANSFORMATIONS

In VAL, a number of methods are available for defining the values of relative transformations. It is normally most convenient to define relative transformations starting with the reference transformation and working out to the last relative transformation. It is possible to define each of the relative transformations using the POINT command, explicitly typing in the value of each of the transformation coordinates. However, it is normally

much easier to teach each of the relative transformations using the HERE command.

Again, turning to the example, each of the relative transformations can be defined in the manner which follows. First, the location of the PLATE is defined, and then one works outward to the grasp location. If the tool of the robot is positioned at PLATE,

```
HERE PLATE
```

defines the location and orientation of the PLATE transformation. Next, if the tool is positioned at the location of OBJECT,

```
HERE PLATE:OBJECT
```

sets the value of the relative transformation OBJECT equal to the location and orientation of the tool relative to the location of PLATE. In this case, if PLATE was undefined, the value of OBJECT could not be determined and an error message would be generated. Finally, the tool must be moved to the grasp point, and the following command issued:

```
HERE PLATE:OBJECT:GRASP
```

This last command sets the value of GRASP equal to the location of the tool relative to the location of OBJECT in the reference frame of the robot. As an alternative to this last command, one could also define the value of GRASP while entering a program by typing the instruction:

```
MOVE PLATE:OBJECT:GRASP!
```

assuming that the values of PLATE and OBJECT were already defined.

A.2 COMPUTATIONAL CONSIDERATIONS

In situations where the same compound transformation is to be used several times, it is computationally advantageous to make use of a SET instruction. For example, to approach GRASP and then move to the final location, instead of using

```
APPRO PLATE:OBJECT:GRASP, 100.0
MOVE PLATE:OBJECT:GRASP
```

one could use

```
SET X = PLATE:OBJECT:GRASP
APPRO X, 100.0
MOVE X
```

While these two programs are functionally equivalent, the second version would take less time to execute since the compound transformation has to be evaluated only once.

A.3 EXAMPLE

This example will be another method of programming the palletizing task in Chapter 6. In this case the pallet will be considered a reference frame, with the pallet locations defined relative to that frame.

The first part of the program leads the user through the steps needed to define the pallet reference frame. These steps only have to be executed once each time the pallet is relocated relative to the robot. Thus, these steps would normally be stored as a separate program.

The "main" program for the task becomes:

```
        REMARK  PROGRAM TO PICK OBJECTS FROM A PALLET
        REMARK
        REMARK  -- PROCEDURE TO CALCULATE THE PALLET REFERENCE FRAME --
        REMARK
        TYPE    POSITION THE ROBOT AT THE PALLET "ORIGIN"
        PAUSE   TYPE "PROCEED<RETURN>" WHEN READY
        REMARK  RECORD THIS LOCATION
        HERE    ORIGIN
        TYPE    POSITION THE ROBOT AT THE FURTHEST PALLET LOCATION
        TYPE            ALONG THE PALLET  +X  DIRECTION
        PAUSE   TYPE "PROCEED<RETURN>" WHEN READY
        REMARK  RECORD THIS LOCATION
        HERE    X
        TYPE    POSITION THE ROBOT AT THE FURTHEST PALLET LOCATION
        TYPE            ALONG THE PALLET  +Y  DIRECTION
        PAUSE   TYPE "PROCEED<RETURN>" WHEN READY
        REMARK  RECORD THIS LOCATION
        HERE    Y
        REMARK  CALCULATE THE PALLET REFERENCE FRAME
        FRAME   PALLET = ORIGIN, X, Y
        REMARK  CALCULATE TRANSFORMATION FOR STARTING LOCATION IN PALLET
        INVERSE TEMP = PALLET
        SET     START = TEMP:ORIGIN
        REMARK
        REMARK  -- END OF PROCEDURE TO CALCULATE REFERENCE FRAME --
        REMARK
        REMARK  INITIALIZE VARIABLES FOR THE PALLET SUBROUTINE:
        REMARK  (1) SET VALUES OF MAXIMUM ROW AND COLUMN COUNTS
        SETI    MAX.ROW = 6
        SETI    MAX.COL = 12
     10 REMARK  (2) SET ROW COUNTER TO FIRST ROW
        SETI    ROW = 1
```

```
    REMARK  (3) SET COLUMN COUNTER TO FIRST COLUMN
    SETI    COLUMN  = 1
    REMARK  (4) SET OBJECT LOCATION TO FIRST LOCATION IN THE PALLET
    SET     HOLE = START
    REMARK  DEFINE LOCATION OF OBJECT IN ROBOT REFERENCE FRAME
    SET     PICK = PALLET:HOLE
 20 REMARK  INSTRUCTIONS DESCRIBING MOTIONS TO AND FROM LOCATION
    REMARK          "PICK" AND ACTIONS TO BE PERFORMED TO THE OBJECT
    .
    .
    .

    REMARK  REDEFINE "PICK" TO BE NEXT PALLET LOCATION
    GOSUB   PALLET
    REMARK  GO BACK AND PROCESS NEW PALLET LOCATION
    GOTO    20
    REMARK  GET HERE WHEN WHOLE PALLET HAS BEEN PROCESSED
    REMARK          BECAUSE OF "RETURN 1" IN SUBROUTINE
    REMARK  RETURN TO TOP OF LOOP FOR NEW PALLET
    GOTO    10
    REMARK  -- END OF PROGRAM --
```

The subroutine PALLET will define the successive values for location variable PICK. It does this by shifting location HOLE around the pallet and defining PICK as the location defined by HOLE relative to PALLET.

```
    REMARK  SUBROUTINE PALLET
    REMARK
    REMARK  PURPOSE:  CALCULATE PALLET LOCATIONS USING COMPOUND TRANS.
    REMARK
    REMARK  INCREMENT COLUMN COUNTER
    SETI    COLUMN = COLUMN + 1
    REMARK  TEST FOR END OF ROW
    IF      COLUMN GT MAX.COL THEN 10
    REMARK  SHIFT ALONG ROW
    SHIFT   HOLE BY 30.00, 0.00, 0.00
    GOTO    20
 10 REMARK  ROW COMPLETED, INCREMENT ROW COUNTER
    SETI    ROW = ROW + 1
    REMARK  TEST FOR ALL ROWS DONE
    IF      ROW GT MAX.ROW THEN 30
    REMARK  SHIFT BACK TO START OF ROW, AND DOWN TO NEXT ROW
    SHIFT   HOLE BY -330.00, 50.00, 0.00
    REMARK  RESET COLUMN COUNTER
    SETI    COLUMN = 1
    REMARK  RETURN TO MAIN PROGRAM AND PROCESS NEXT OBJECT
 20 REMARK  SET THE VALUE OF PICK
    SET     PICK = PALLET:HOLE
    RETURN  0
    REMARK
```

```
30 REMARK  ALL PALLET LOCATIONS HAVE BEEN USED, RETURN TO MAIN
   REMARK           PROGRAM AND SKIP THE "GOTO" INSTRUCTION
   RETURN  1
   REMARK  -- END OF SUBROUTINE PALLET --
```

As in the program in Chapter 6, the location variable HOLE is shifted around to each pallet location during execution. The difference here is that HOLE only defines a location relative to the pallet and is combined with PALLET to define PICK, the location relative to the robot.

There are two advantages to using compound transformations for this task: the pallet can have any orientation relative to the robot, and it is easier to compensate for a relocation of the pallet. If the pallet is later moved, the instructions which define the PALLET reference frame must be executed, but no changes need to be made to the program. In the earlier example, however, it might be necessary to modify the BASE and SHIFT instructions to compensate for a repositioning of the pallet.

APPENDIX B

DEFINING A TOOL TRANSFORMATION

If the dimensions of a tool are known, the POINT command can be used to define the corresponding tool transformation. Recall that the "null tool" has its center at the surface of the tool mounting flange and its coordinate axes parallel to that of the last joint of the robot, and is represented by the transformation [0,0,0,90,-90,0].

For example, if a hand is mounted to the flange and it is desired to change the tool setting to compensate for a 100 millimeter distance from the flange out to the fingers, one would type the following lines. (Recall that underlined characters indicate those typed by the user, and "®" represents the RETURN key.)

```
.POINT HAND®
   X/JT1     Y/JT2     Z/JT3     O/JT4     A/JT5     T/JT6
    0.00      0.00      0.00     90.000   -90.000     0.000
CHANGE?
,,100®
    0.00      0.00    100.00     90.000   -90.000     0.000
CHANGE?
®
.TOOL HAND®
OK
.
```

In cases where a tool with an unusual configuration is to be used, or when the tool dimensions must be compensated for tool-tip wear, it is convenient to use compound transformations to define and update the tool transformation. A short VAL program is presented below that can be used to define tool transformations. This program only requires that the operator be able to accurately position the robot at a point in the robot workspace with the same orientation using two different tools. The selected reference point can be located anywhere in the robot workspace. However, locations with the robot fully outstretched or tucked in should be avoided to improve computational accuracy.

In the following program, it is assumed that the robot has a tool attached whose transformation is already known.

```
 1.   TYPE     MOVE THE TOOL TIP TO THE SELECTED REFERENCE LOCATION,
 2.   TYPE     SET "REF.TOOL" EQUAL TO THE TRANSFORMATION FOR THIS
 3.   PAUSE    KNOWN TOOL, THEN TYPE "PROCEED<RETURN>".
 4.   REMARK   HERE'S WHERE THE REFERENCE LOCATION IS DEFINED
 5.   TOOL     REF.TOOL
 6.   HERE     REF.LOC
 7.   TYPE     INSTALL THE NEW TOOL, MOVE ITS TIP BACK TO THE
 8.   PAUSE    REFERENCE LOCATION, AND TYPE "PROCEED<RETURN>".
 9.   REMARK   NOW COMPUTE THE NEW TOOL TRANSFORMATION, "NEW.TOOL".
10.   REMARK   DEFINE "ACT" TO BE THE CURRENT ROBOT LOCATION.
11.   TOOL     REF.TOOL
12.   HERE     ACT
13.   INVERSE  ACTI = ACT
14.   SET      NEW.TOOL = REF.TOOL:ACTI:REF.LOC
15.   TOOL     NEW.TOOL
16.   TYPE     ALL DONE, THE TOOL TRANSFORMATION HAS BEEN SET EQUAL
17.   TYPE     TO THE VALUE OF "NEW.TOOL".
18.   STOP
```

Because of computational errors introduced when compound transformations are utilized, the accuracy of the program presented above can be improved by using a simple tool with no oblique rotations as the reference tool. In fact, if the mounting flange of the robot can be used without a tool as the initial pointer, the most computationally accurate results can be obtained. In this case, the reference tool would be the default "null tool" and the program above can be simplified by deleting the references to "REF.TOOL" in lines 2, 5, 11, and 14.

After this program has been executed once, the tool transformation can be updated by simply positioning the new tool at the same reference location as before and starting execution of the above program at step 11. As long as the values of "REF.TOOL" and "REF.LOC" have not been altered, a new tool transformation will automatically be computed and asserted. This is a convenient method for occasionally altering the tool transformation to account for tool wear.

APPENDIX C

CONTINUOUS PATH MOTION

The Continuous Path feature is provided for those applications which require that the robot not stop at each location to which it moves. Instead, the motion of the robot smoothly transitions between motion commands, subject to the maximum allowable accelerations of the robot.

Such transitions can occur between any combination of straight-line and joint-interpolated motions. Thus, for example, a continuous motion could consist of a straight-line motion followed by a joint-interpolated motion and a final straight-line motion. Any number of motion segments can be strung together.

C.1 Controlling Continuous Path

The Continous Path feature is activated whenever the system switch CP is ENABLE'd, which is the default condition when VAL is initialized. While continuous path is activated, all robot motions are smoothly transitioned unless a program instruction is executed. This forces an interruption of the continuous path. After being interrupted, continuous path resumes with the next motion instruction executed.

The following instructions cause a continuous motion to be interrupted:

BASE,	BREAK,	CLOSEI,	DELAY',	DISABLE,	ENABLE,	FRAME,
GRASP,	HALT,	HERE,	IGNORE',	INVERSE,	OPENI,	PAUSE,
REACT,	REACTI,	RELAX,	SET,	SETI,	SHIFT,	STOP,
TOOL,	TYPE,	TYPEI,	WAIT,	WEAVE 0 (if weaving is on)		

C.2 Considerations During Continuous Motion

There are certain considerations that should be kept in mind while using continuous motions because they can affect the performance of the system.

Because of the computation time needed to plan the transitions between motion segments, there is a limit on how closely spaced commanded locations can be. If locations are too close together, there is not be enough time for VAL to plan the transition from one motion to the next, and there is a break in the continuous motion. This means that the robot stops instantaneously at the intermediate location. The minimum spacing that can be used between locations before this effect occurs is determined by the time

taken to complete the motion from one location to the next. Straight-line motion can be used if the motion takes more than about 140 ms. Joint-interpolated motion can be used with motion times down to about 60 ms. Thus, joint-interpolated motions should be used if possible when closely spaced locations are desired.

During continuous motion, a SIGNAL instruction has its effect at the beginning of the next transition.

C.3 Program Instructions

The following two instructions are available if the VAL system in use includes the Continuous Path feature.

BREAK

Interrupts the continuous path before the next motion instruction. This instruction has no effect if continuous path is not active.

WEAVE <distance>,[<cycle time>],[<dwell>]

This instruction initializes the parameters for a sawtooth weaving motion to be superimposed on the following straight-line motion. Once initiated, weaving continues until (1) it is terminated with a WEAVE 0 instruction, (2) a joint-interpolated motion is performed, or (3) when the robot stops due to a break in continuous path or the end of a regular motion. Note that turning off weaving causes a BREAK in a continuous motion.

The distance argument specifies the peak-to-peak amplitude of the weave, and is limited to a maximum of 256 mm; weaving is disabled if the distance is zero. The second argument specifies the weave cycle time, which is limited to a maximum of 327.67 seconds. If the cycle time is omitted, or is less than the minimum possible, the minimum possible cycle time is used. The final argument can be specified to have all robot motion stop at the extreme weave positions, the apexes of the sawtooth pattern. This dwell period is limited to 327.67 seconds at each extreme.

WEAVE 55,10	Initialize the parameters for a weaving motion with a peak-to-peak amplitude of 55 mm, a 10-second cycle time, and no delays at the extreme positions.

APPENDIX D

SYSTEM DIAGNOSTICS AND MODIFICATON

The following monitor commands provide the ability to test various components in the robot system, and to use the results to modify certain parameters stored in VAL. These commands can be issued only when no program is being executed. Diagnostic programs are written by Unimation.

DIAGNOSTIC <file name>

[.] Reads in the specified system-diagnostic file from the floppy diskette and starts its execution. Certain diagnostic programs may require that VAL be started in a special manner. Refer to the documentation for the program to be used.

DIAGNOSTIC POTCAL — Read a diagnostic program from the floppy-disk file named POTCAL.

OVERLAY <file name>

[.] Reads in new calibration data from the specified floppy-disk file (previously written by a diagnostic program) and uses it to replace the robot calibration parameters contained in VAL. This command must be followed immediately by a CALIBRATE command for the data to be correctly utilized. (Power to the robot is automatically turned off by the OVERLAY operation.)

OVERLAY CALIB — Read new calibration data from the floppy-disk file named CALIB.

APPENDIX E

CHANGES MADE TO VAL

Version	Changes From Earlier Versions
12	Changed message displayed at turn-on New monitor commands: DIAGNOSTIC, LOAD, OVERLAY, STORE, SWITCH, and TEACH New program instructions: ALIGN, BASE, DEPARTS, DISABLE, ENABLE, HALT, MOVEST, MOVET, and NEST New switches: MESSAGES, CRT, SRV.ERR, CP, EHAND, and VISION * Changed the editor's T teach mode Added TS teach mode to the editor Changed messages displayed when execution terminates CALIB can be entered as CALIBRATE STATUS information includes tool transformation * Eliminated GO and MOVEI instructions Old DEPART instruction renamed to DEPARTS DEPART instruction now gives joint-interpolated motion * DRIVE instruction now takes speed instead of time Effects of OPEN, CLOSE, OPENI, and CLOSEI expanded GRASP now acts on the pneumatic control system Added optional string to STOP instruction Added continuous path feature, which includes: CP switch, BREAK and WEAVE instructions
11	* Added <dZ> argument to BASE command * TOOL command and instruction can now take a transformation as argument Changed argument list for LISTP command SET instruction now works with precision points as well as transformations Added INVERSE instruction Added FRAME instruction * Deleted SIGON and SIGOFF instructions Added SIGNAL instruction Added compound transformations

---

* The indicated change could significantly affect programs and procedures prepared for an earlier version of VAL.

APPENDIX F

VAL MESSAGES

While using the VAL system, it is possible for certain abnormal conditions to occur. For example, if commands or instructions are not entered in the correct way, or if some other indiscretion is attempted, VAL rejects the input. The usual response is to output to the system terminal an indication of what is wrong so the user can correct the error. The following list contains all the error messages produced by VAL, explains what they mean, and indicates what should be done in response.

This list also includes a variety of informational messages which VAL displays under certain circumstances.

ABORTED

Explanation: The last command requested, or the user program which was executing, has been aborted at your request.
User action: None.

AMBIGUOUS COMMAND

Explanation: VAL could not interpret the last command or instruction because not enough characters were entered for the name of the function.
User action: Retype the line, entering more characters.

ARE YOU SURE (Y/N)?

Explanation: The requested command will have a significant effect on the state of your system, and VAL wants to know if you really want it to happen.
User action: To have VAL continue, type "Y" followed by a carriage return. An "N" followed by a carriage return, or just a carriage return, causes the command to be aborted.

ARM DISABLED FOR TESTING

Explanation: This is a warning to indicate that VAL has been started in its diagnostic-test mode. In this mode, power to the robot cannot be turned on and user programs cannot be executed.
User action: Continue with the test procedure. If this message occurs during normal system use, refer to your system equipment manual.

ARITHMETIC OVERFLOW

Explanation: The result of a calculation was too large for the computer to handle, possibly because of division by zero. Keep in mind that integer variables can only assume values in the range from -32768 to +32767.
User action: Modify the program as required.

BAD ARITHMETIC OPERATION

Explanation: An invalid arithmetic operator was encountered, probably because of a typing error.
User action: Retype the line, correcting the error.

BAD COMPARISON OPERATOR

Explanation: An invalid comparison operator was encountered, probably because of a typing error.
User action: Retype the line, correcting the error.

BAD POT, JT n (For the PUMA robot only)

Explanation: The potentiometer circuitry for the indicated joint has malfunctioned. Normally, this error indicates that the pot reading did not change as the corresponding joint was moved.
User action: Refer to your system equipment manual.

BAD SWITCH

Explanation: An invalid switch name was encountered, probably because of a typing error.
User action: Check the switch name and retype the line.

BRANCH TO UNDEFINED LABEL

Explanation: A program instruction references a program label which is not defined in the program. Either the label is missing, or was mistyped when defined or in the reference.
User action: Check the label definition and reference.

CAN'T GO ALONG STR LINE, USE JOINT MOTION

Explanation: Movement to the requested location cannot be done along a straight line because too large of an orientational change of the outer joints would be required.
User action: Use joint-interpolated motion instead of straight-line motion, or break the motion up into a series of motions requiring less rotation of the outer joints.

CAN'T GO ON, USE "EXEC"

Explanation: Program execution cannot be initiated in the manner attempted. Normally, this error is signalled when an attempt is made to execute a PROCEED, RETRY, or NEXT command after a program has completed all its cycles.

User action: Use the EXECUTE command to restart the program from the desired instruction.

CAN'T INTERPRET LINE

Explanation: The argument list of the last command or instruction could not be interpreted by VAL; or, if the message occurred while loading a location file from the floppy disk, the file was probably created by a VAL system for a different type of robot and the indicated location is not compatible with the VAL system in use.

User action: Check the syntax of the function and enter the line again. In the case of an error while loading from the floppy disk, either change to the correct diskette or define the indicated location(s) after loading is

CHECKSUM ERROR

Explanation: While transferring a file to or from the floppy disk, a transmission error occurred.

User action: Attempt the transfer again.

CLOCK OVER RUN

Explanation: VAL ran out of time while trying to compute the next set of position commands for the joint servoes. This only happens if a hardware failure has occurred.

User action: Refer to your system equipment manual.

COMMUNICATION ERROR

Explanation: The floppy-disk drive received an invalid command from VAL.

User action: Try your command again.

COMPUTER MODE DISABLED

Explanation: The command attempted requires computer control of the robot, but "Computer Mode" was not selected on the manual control.

User action: Select computer mode on the manual control and reissue the command.

DIRECTORY ERROR

Explanation: An error occurred while accessing the floppy-disk directory, possibly because the diskette was not formatted or the diskette has been damaged in some way.
User action: Make sure the correct diskette is being used, that it is properly installed in the drive, and that it is formatted.

DISK FULL

Explanation: There is no more space available on the floppy diskette for file storage.
User action: Use the COMPRESS command to regain space previously occupied by deleted files, delete unneeded files and COMPRESS, or use another diskette.

"DO" NOT PRIMED

Explanation: A DO command was attempted without specifying a program instruction to be executed, and no previous DO had been issued.
User action: Provide the desired instruction with the DO command.

DUPLICATE LABEL

Explanation: The same program label is used more than once in a user program.
User action: Change one of the duplicate labels.

EXITING TO ODT!

Explanation: In response to the DONE command, VAL is stopping and control is reverting to the computer hardware Octal Debugging Tool (ODT).
User action: Issue the desired ODT commands. (To restart VAL type "173000G".)

[FATAL] ADC DEAD, JT n (For the PUMA robot only)

Explanation: The analog-to-digital converter for the indicated robot joint is not responding.
User action: Refer to your system equipment manual.

[FATAL] CPU ERROR, TRAP TO nn FROM mmmmmm

Explanation: A computer error occurred because of a bad read from memory or because of noise on the data bus. (Such errors occur infrequently even when the system is functioning normally.)
User action: Restart VAL by typing "173000G" and respond with a "N" to the inquiry regarding initialization. If this error occurs repeatedly, refer to your system equipment manual.

[FATAL] ENVELOP ERROR, JT n (For the PUMA and PUMA 250 robots)

Explanation: The indicated joint was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system.
User action: Try to perform the motion at a slower speed. If the error reoccurs, refer to your system equipment manual.

[FATAL] LOST ENCODER SYNC, JT n (For the PUMA and PUMA 250 robots)

Explanation: The encoder circuitry for the indicated joint is loosing or gaining counts.
User action: Refer to your system equipment manual.

[FATAL] OUT OF RANGE, JT n

Explanation: During its initialization sequence, VAL found the indicated joint out of its allowed software-limited range.
User action: Move the indicated joint manually toward the middle of its range and restart VAL by typing "173000G", or powering up again.

[FATAL] SERVO DEAD, JT n

Explanation: The joint controller for the indicated joint is not responding to commands from VAL.
User action: Refer to your system equipment manual.

[FATAL] SERVO RAM ERROR, JT n (For the PUMA and PUMA 250 robots)

Explanation: During its power-up self-check, the indicated joint processor found an error in its RAM memory.
User action: Refer to your system equipment manual.

FILE ALREADY EXISTS

Explanation: There is already a file on the floppy diskette with the name supplied to the storage request.
User action: Reissue the storage request with a different file name, or delete the old file on the diskette.

FILE ALREADY OPEN

Explanation: The floppy-disk cannot access another file because the last file access was not successfully completed, possibly because the disk door was opened too soon.
User action: Unplug the disk from the controller momentarily so that it will undergo a power-up reset.

FILE FORMAT ERROR

Explanation: The requested floppy-disk file is not in a format acceptable to VAL. Either because it was not created by VAL, or the file has been corrupted.
User action: Use another diskette or reference another file.

FLOPPY NOT READY

Explanation: The floppy-disk drive is not prepared to communicate with the controller.
User action: Make sure the drive is plugged in, the diskette is correctly inserted, and the door is fully closed.

FUNCTION TIME OUT, JT n

Explanation: The indicated joint took too long to complete the last motion, possibly because the robot is blocked and cannot reach its destination.
User action: Attempt the motion again, with any necessary changes.

HALT: STOPPED AT STEP n

Explanation: A HALT instruction has been executed, and thus execution of the current user program has terminated. If the user program included a text string in the HALT instruction, it is displayed on the line above the HALT message.
User action: Any monitor command can be issued, but PROCEED cannot be used to resume progam execution.

HAND CLOSED TOO FAR

Explanation: The hand closed more than the specified amount during a GRASP instruction, either because the object to be grasped was not present or because the specification was incorrect.
User action: Change the location of the object, the grasp distance, or whatever else is necessary.

HARDWARE SERVO DISABLED

Explanation: ARM POWER is not turned on.
User action: Turn on ARM POWER and reenter the last command.

ILLEGAL CHANNEL

Explanation: The external signal channel number specified is not in the allowed range of one to eight.
User action: Correct the channel number.

ILLEGAL FUNCTION

Explanation: The command or instruction attempted was not recognized by VAL, because of a error in the function name. (This error is often caused by mistyping the function name, or trying to use a command as an instruction or vice versa.) If the message occurred while loading a file from the floppy disk, the file was probably created by a different VAL system (different version or options) and the indicated line is not compatible with the VAL system in use.

User action: Check the function spelling/usage and re-enter the command. In the case of an error while loading from the floppy disk, either change to the correct diskette or, after loading is completed, edit the effected program(s) to restore the indicated instruction(s).

ILLEGAL IN WORLD OR TOOL MODE

Explanation: The command attempted can not be processed when the manual control has TOOL or WORLD mode selected.

User action: Select another mode on the manual control and retry the command.

ILLEGAL JOINT

Explanation: The joint number specified is not in the allowed range.

User action: Enter the command again with a correct joint number.

INITIALIZATION ERROR

Explanation: VAL was not able to FORMAT the floppy-disk directory because the diskette is defective.

User action: Use another diskette.

INITIALIZE (Y/N)?

Explanation: During its startup sequence VAL is asking whether it is okay to go ahead and initialize the system. This will cause any user information currently in memory to be lost.

User action: If the system has just had power applied, you MUST respond with a "Y"; otherwise you can decide whether you want the RAM memory to be erased and have the various system status conditions reset to their default settings.

INPUT BLOCK ERROR

Explanation: A read error has occurred while using the DIAGNOSTIC or OVERLAY command to read a binary-data file from the floppy disk. This indicates that the data in the file is corrupted.

User action: Try the operation again. If the error recurs use another diskette.

INPUT ERROR, TRY AGAIN

Explanation: The input provided was not consistent with what VAL expected.
User action: Provide another response.

INVALID LABEL

Explanation: The program label was not an integer from 0 to 32767.
User action: Reenter the line with a valid label.

INVALID NUMBER

Explanation: The line contains a number which is not an integer between -32768 and 32767.
User action: Reenter the line with a valid number.

INVALID SOLUTION, ERROR CODE n

Explanation: For codes 1 to 7, the requested motion was beyond the software-limited range of motion for the indicated joint. Codes 8 and 9 indicate that the requested location was too close to the column of the robot, or to far out, respectively.
User action: Redefine the destination location.

LOCATION OUT OF RANGE

Explanation: The X, Y, or Z coordinate of the destination location had a value too large to be represented in the computer. For example, a sequence of SHIFT instructions has moved the location very far. The permitted range for the coordinates is -1023 mm to 1024 mm for the PUMA robot, and -699 mm to 700 mm for the PUMA 250 robot.
User action: Correct the user program.

LOCKED IN RUN HOLD MODE!

Explanation: Program step execution is blocked because the RUN/HOLD/RESTART selector is in the HOLD position.
User action: Set the RUN/HOLD/RESTART selector to RUN.

WARNING

The executing user program will immediately resume execution. This may cause the robot to move suddenly.

MEMORY ERROR, LOC nnnnnn

Explanation: A hardware failure was detected at the indicated location during the check of RAM by VAL.
User action: Refer to your system equipment manual.

MISSING ARGUMENT

Explanation: One or more of the arguments required for the requested command or instruction was not provided.
User action: Check the function syntax and reenter the line.

NO PROGRAM SPECIFIED

Explanation: No program was specified for an EXECUTE or NEXT command.
User action: Type the line again, providing a program name.

NO PROGRAM STEPS DEFINED

Explanation: The program referenced in an EXECUTE or NEXT command does not exist, possibly because the program name was mistyped.
User action: If necessary, create the program with the editor or by reading it from the floppy disk; then reenter the line.

NONEXISTENT FILE

Explanation: The requested file is not stored on the floppy diskette currently in the disk drive. Either the name was mistyped or the wrong diskette is being read.
User action: Check the file name; use the LISTF command to display the directory of the diskette.

NOT AT READY LOCATION (For the PUMA 250 robot)

Explanation: The NEST instruction was attempted with the robot at some location other than the READY location.
User action: Insert a READY instruction before the NEST instruction and reexecute the program or type DO READY to move the robot to the required location.

NOT CALIBRATED, GO ON (Y/N)? (For the PUMA and PUMA 250 robots)

Explanation: The robot servo system is not calibrated, and thus subsequent motions may not go to the expected locations.
User action: If it's okay to proceed, respond with a "Y". If not, respond with "N" and then issue a CALIBRATE command.

NO ZERO INDEX, JT n (For the PUMA and PUMA 250 robots)

Explanation: During execution of the CALIBRATE command, the control system did not detect a zero-index mark for the indicated joint.
User action: Refer to your system equipment manual.

OK

Explanation: VAL is acknowledging the completion of the last command.
User action: None.

PANIC BUTTON HIT

Explanation: The OFF button on the manual control was pressed.
User action: Reselect COMPUTER mode on the manual control before resuming program execution.

PAUSE: STOPPED AT STEP n

Explanation: A PAUSE instruction has been executed, and thus the current user program has suspended execution. Program execution will start at the indicated step number if resumed. If the user program included a text string in the PAUSE instruction, it is displayed on the line above the PAUSE message.
User action: Any monitor command can be issued. To continue execution of the program, type "PROCEED".

PROGRAM COMPLETED: STOPPED AT STEP n

Explanation: The user program has been executed the number of times specified in the EXECUTE command which initiated execution. If the user program included a text string in the STOP instruction at the logical end of the program, it is displayed on the line above the completion message.
User action: Any monitor command can be issued, but PROCEED cannot be used to resume progam execution.

PROM ERROR, BANK n

Explanation: The indicated section of the VAL PROM memory has a value which is changed from when the PROM was created.
User action: Refer to your system equipment manual.

STORAGE AREA FORMAT ERROR

Explanation: A momentary hardware failure has corrupted the user data in RAM memory. The user programs and locations currently in memory may no longer be valid.
User action: Attempt to save as much as possible onto a floppy diskette. Then restart VAL, responding with a "Y" to the initialize query.

STORAGE SPACE EXHAUSTED

Explanation: There is no more space in RAM memory for user programs or location variables.
User action: Delete unused programs and location variables.

STR LINE MOTION CAN'T ALTER CONFIGURATION

Explanation: A change in configuration was requested during a straight-line motion. This is not allowed.
User action: Delete the configuration change request, or use a joint-interpolated-motion instruction.

TOO MANY SUBROUTINES CALLED

Explanation: An attempt to branch to a subroutine was made while there were already ten programs suspended.
User action: Reorganize the program logic to eliminate one or more subroutine calls.

UNDEFINED LOCATION

Explanation: The location variable specified has not been given a value.
User action: If the variable name was correctly entered, give the variable a value by issuing the appropriate command, or by reading a value from the floppy disk.

UNEXPECTED END OF FILE

Explanation: While reading a file from the floppy disk, the end of the file was encountered unexpectedly. The file may not have been correctly closed when it was written, possibly because the disk was disrupted before the writing was completed.
User action: Try again to read the file.

VARIABLE TYPE MISMATCH

Explanation: One or more of the variables in the line is of a type inconsistent with the other variables or with the type required by the command or instruction.
User action: Check the syntax for the function and reenter the line, correcting the mismatch.

WARNING: NO ARM CALIB (For the PUMA and PUMA 250 robots)

Explanation: The robot servo system is not calibrated, and thus, any location variables that are taught will not accurately represent the location desired.
User action: Issue the CALIBRATE command and reteach the location variables.

$.1 \times 25.4 = 2.54$

$\frac{.1}{2.54} = .064$

APPENDIX G

SUMMARY OF VAL COMMANDS AND INSTRUCTIONS

G.1 MONITOR COMMANDS

In the following commands, "location" refers to a "precision point", "transformation", or "compound tranformation" unless otherwise noted.

Program and Location-Data Storage

Program Control

System Status and Control

System Switches

System Diagnostics and Modificaton

G.2 PROGRAM INSTRUCTIONS

In the following instructions, "location" refers to a "precision point", "transformation", or "compound tranformation".

Location Assignment and Modification

		Page
FR	FRAME <transformation> = <trans 2>,<trans 3>,<trans 4>	5-9
HE	HERE <location>	5-8
INV	INVERSE <transform> = <trans 2>[:<t 3>] ... [:<t n>]	5-9
SET	SET <transformation> = <trans 2>[:<t 3>] ... [:<t n>] or SET <precision point> = <precision point>	5-8
SH	SHIFT <transformation> BY [<dx>],[<dy>],[<dz>]	5-8
TO	TOOL [<transformation>]	5-9

Program Control

GOS	GOSUB <program>	5-10
GOT	GOTO <label>	5-10
HA	HALT [<string>]	5-11
IF	IF <i.var> <relationship> <i.var 2> THEN <label>	5-11
IFS	IFSIG [<channel>],[<channel>], [<channel>],[<channel>] THEN <label>	5-12
IG	IGNORE <channel> [ALWAYS]	5-13
PA	PAUSE [<string>]	5-11
REACT	REACT <channel>,[<program>] [ALWAYS]	5-12
REACTI	REACTI <channel>,[<program>] [ALWAYS]	5-13
RET	RETURN [<skip count>]	5-10
SI	SIGNAL <channel>,[<channel>,...,<channel>]	5-12
ST	STOP [<string>]	5-11
WA	WAIT <channel>	5-13

Trajectory Control

CO	COARSE [ALWAYS]	5-14
FI	FINE [ALWAYS]	5-14
INTOF	INTOFF [ALWAYS]	5-15
INTON	INTON [ALWAYS]	5-15
NON	NONULL [ALWAYS]	5-15
NU	NULL [ALWAYS]	5-15
SP	SPEED <value> [ALWAYS]	5-14

Miscellaneous

BA	BASE [<dX>],[<dY>],[<dZ>],[<Z rotation>]	5-15
DEL	DELAY <time>	5-16
DI	DISABLE <switch>	5-16
EN	ENABLE <switch>	5-16
REM	REMARK [<string>]	5-16
TYPE	TYPE [<string>]	5-16

Continuous Path Motion

BR	BREAK	C-2
WE	WEAVE <distance>,[<cycle time>],[<dwell>]	C-2

INDEX