An Improved Controller for the Rhino Robot Arm

by

Mark A. Hopkins

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Hugh VanLandingham, Chairman

D. William Luse

Charles Nunnally

December, 1984

Blacksburg, Virginia

An Improved Controller for the Rhino Robot Arm

by

Mark A. Hopkins

(ABSTRACT)

The study of robotics cannot be satisfactorily pursued without access to working robots. The inexpensive Rhino robot arm is one that academic institutions can easily obtain for educational purposes. This thesis presents a new controller that replaces the original Rhino controller, which in many ways was not suited, or was too limited, for experimentation.

A comparison of the old and new controllers is given, but the primary purpose of this thesis is to provide complete details of the new controller, and its use.

The conclusion discusses the performance of the new controller and areas of experimentation to which it might be applied.

ACKNOWLEDGEMENTS

I thank Dr. John Roach, who conceived this project three years ago, for his tremendous interest and support.

I also thank: Dr. Hugh VanLandingham for suggesting this project, and for his help and support from the outset; Dr. Will Luse for insights he shared into the theoretical aspects of the controller, and valuable suggestions in hardware debugging; and Dr. Charles Nunnally for help with questions on microprocessor operation, and for technical support on the Intel development system.

I also thank the many others who helped make completion of this work possible: provided design ideas for the mailboxes, the shared random access memory and the pulsewidth modulator; provided help with the PASCAL programming language, and general morale boosting; provided technical help concerning the motors; provided valuable tips about hardware construction and operation; kept the Intel development system functioning through some hard use; provided much-needed help in the technical library and the occasional odd part that made the system go; Dr. Roger Ehrich provided programs that helped

make sense of the VAX input/output system, and which served as kernels for the communication routines I used; Laura Hall hounded the parts suppliers until they came through with the goods I needed; and helped proof-read this rather long document.

I dedicate this thesis to my wife, whose love, understanding, and hard work made it possible to see this project through.

This material is based upon work supported under a National Science Foundation Graduate Fellowship, and supported in part by National Science Foundation Grant "Coordinating Robot Arms". Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

CONTENTS

Chapter 1. Introduction 1

Robots as Research Tools 1

Comparison of the Old and New Controllers 2

Possible Improvements on the New Controller 7

Chapter 2. System Overview of the Controller 9

Communications 9

Bookkeeping Tasks in the Intel 8751 12

Bookkeeping Tasks in the 8748 13

Keeping Track of Velocity 13

Keeping Track of Position 14

Control Considerations 15

Stability and Variable-Rate Sampling 15

Velocity Determination from Position Feedback . . . 18

Velocity Control by Pulsewidth Modulation 19

Chapter 3. Conclusion **21**

Performance 21

Areas of Experimentation 23

Appendix A. Protocol and Data Format in VAX/8751 Communications **26**

Communicating Command Codes 26
Communicating Data 27

Appendix B. 8751 Software for VAX Communication . . . 34
Scheme for VAX Communications 34
Processing Commands in the Interrupt Routine 35
Processing Data in the Interrupt Routine 36
Processing Communications in the Main Routine . . . 36
Transmitting Responses 38

Appendix C. Protocol and Data Format in 8751/8748 Communications 39
Communicating Command Codes 39
Communicating Data 40

Appendix D. The Mailboxes 44
8748 Access to the Mailboxes 44
8751 Access to the Mailboxes 46

Appendix E. The Shared Random Access Memory 52

Appendix F. The Commands from VAX 56
Commands Controlling the State 56
Commands Monitoring the State 58

Repetitive Command Function (Command-on-Interrupt) . . 59

Appendix G. Shaft Position Encoding 61

The Shaft Encoder Disk and Angular Measure 61

Encoder Output 62

Appendix H. 8748 Interrupt Generation 67

Motor Motion Interrupt Generation 67

Motor Stopped Interrupt Generation 68

Appendix I. Keeping Track of Direction 71

Appendix J. Position Bookkeeping in the 8748 73

The Absolute Position Counter 73

The Present Move Command (Position Error) 75

The Quarter-Hole Tail 77

Appendix K. Velocity Bookkeeping in the 8748 79

Measuring the Motor Speed 79

Clocking the Interrupt Interval 79

Taking the Reciprocal 80

Determining Velocity Error 84

Determining the Direction of Acceleration. 85

Appendix L. Bookkeeping Tasks in the Intel 8751 . . . 86

(Hardware) The Motor Linked Latch 86

(Hardware) The Baud Rate 87

(Software) The 8748 Status Flags 88

Not Ready for Next Move Flags 88

Motor Stalled Flags 89

Hardware Reset Flags 89

Appendix M. Control by Non-Linear Feedback 90

Appendix N. Determining the Pulsewidth 94

Appendix O. The Pulsewidth Modulator 98

Appendix P. Allocation of Random Access Memory . . . 101

Appendix Q. System Schematic Diagrams. 105

Appendix R. A Special Case: The Gripper Motor . . . 115

Appendix S. Sample VAX Program (PASCAL) 118

Appendix T. 8751 Assembly Language Program Outline . 125

The Initialization Routine 125

8751 Main Program 126
The Respond to 8748 Subroutine 130
The Decode 8748 Command Subroutine 130
The Write to VAX Subroutine 132
The Serial Output Subroutine 132
The Decode VAX Command Subroutine 134
The Global Write Subroutine 143
The Serial Port Interrupt Service Subroutine 144
The Interrupt Zero Service Subroutine 147

Appendix U. 8748 Assembly Language Program Outline . 148
The Initialization Routine 148
The Main Program 149
The Interrupt Service Subroutine 153
The Motor Stopped Subroutine 160
The Move Yet? Subroutine 162
The New Move Subroutine 163
The Position Update Subroutine 164
The Pulsewidth Update Subroutine 166
The Multiply Subroutine 168

Appendix V. Listing of the Sample Program on VAX . . 170

Appendix W. Listing of the 8751 Assembly Language

Program 185

Appendix X. Listing of the 8748 Assembly Language Program 211

Appendix Y. Listings of the Input/Output Procedures on VAX 235

Bibliography 242

Vita 244

LIST OF ILLUSTRATIONS

Figure 1. System block diagram. 20
Figure 2. Command codes from VAX to 8751. 30
Figure 3. Communication protocol for VAX-originated codes to 8751. 31
Figure 4. Command codes from 8751 to VAX. 32
Figure 5. Communication protocol for 8751-originated codes to VAX. 33
Figure 6. Command codes between 8751 and 8748. . . . 42
Figure 7. 8751/8748 Communications protocol and resulting actions. 43
Figure 8. The mailboxes and the 8748 device addressing scheme. 49
Figure 9. 8751 Port 0 Input Device Addressing Scheme. 50
Figure 10. 8751 Port 1 Input Device Addressing Scheme. 51
Figure 11. Block diagram of the shared RAM. 55
Figure 12. Typical shaft encoder disk and its natural angular measurement. 65
Figure 13. Block diagram of the shaft encoder and output conditioning. 66
Figure 14. 8748 interrupt generation and clocking. . . 70
Figure 15. Block diagram of the pulsewidth modulator. 100
Figure 16. Allocation of 8748 internal RAM. 102
Figure 17. Allocation of shared RAM. 103
Figure 18. Allocation of 8751 internal RAM. 104
Figure 19. Connections to the Intel 8751 microprocessor. 106
Figure 20. Connections for 8751 shared RAM enables, flag latches, ... 107
Figure 21. Connections to the shared RAM. 108
Figure 22. Connections for the 8751 mailbox enables and flag latches. 109
Figure 23. Connections to the Intel 8748, its mailbox, and device selection. 110
Figure 24. Connections for encoder bit shaping and data latching. 111
Figure 25. Connections for interrupt generation and interrupt clocking. 112
Figure 26. Connections for the Motor 1 pulsewidth modulator. 113
Figure 27. Oscillators for the 8748's (6 MHz) and interrupt clock (32 kHz). 114

CHAPTER 1. INTRODUCTION

ROBOTS AS RESEARCH TOOLS

The study of robotics is not complete without some practical experience in making robots move. Indeed, there is a wealth of research problems involving precisely that task in such areas as trajectory planning, collision avoidance, and sensory feedback. Powerful and flexible tools are necessary to enhance the research possibilities.

Few robots are designed specifically to be research tools. In the case of an industrial robot, information that would greatly aid in using it for experimentation is often proprietary, unavailable even to academic institutions. Thus, a robot in the research lab is often just a large 'black box' designed to move through a series of widely separated points specified by the user. The path that is followed from point to point is determined by computations over which the user has no control.

However, a researcher may want the robot to follow a path determined by computations being made on an entirely separate machine, but may be unable to circumvent the control circuitry

and software of the robot in his laboratory.

The Rhino[1] robot thus finds its place in the laboratory, since it was designed specifically for research. The details of its circuitry and software are provided by the manufacturer. But, as a research tool, it has some severe limitations.

COMPARISON OF THE OLD AND NEW CONTROLLERS

There are several problems with the design of the original controller for the Rhino robot. Because the controller implements what is generally called 'bang-bang' control, the robot cannot make velocity-controlled moves. Under that type of control, every move is made at top speed, often resulting in large, undesirable oscillation of the end effector (gripper). This limited capacity is not surprising, with only one microprocessor controlling all six motors.

The primary reason for designing a new controller was to implement velocity control during execution of move commands. Among the benefits expected under velocity control was the reduction

---

[1] The name Rhino is a registered trademark of Rhino Robots, Inc., 308 S. State, Champaign, IL, 61820.

of unwanted oscillations to a tolerable level. The major obstacle to achieving velocity control was the lack of tachometers on the motors of the robot. Velocity control had to be implemented using only position feedback.

The position feedback from each motor consists of two bits from a quadrature-encoded shaft encoder disk. The disk, itself, compounds the problem of obtaining velocity control from position feedback because it has very poor angular resolution (15 to 45 degrees).

The velocity control problem was solved, however, to the extent that the new controller can maintain a specified velocity to a reasonable degree of accuracy (about 8% in most cases) and can execute moves with no overshoot and a tolerably small amount of oscillation of the end effector.

One measure of the quality of a robot is the repeatability of a position (that is, how close can the robot come to assuming a specified previous position). Under the new controller, repeatability (typically on the order of 1 millimeter) has been improved by a factor of four. That is because the original controller senses angular error with only one-fourth the resolution of the new controller.

A further significant limitation of the original controller is the small angular distance that can be specified by a single move command. It is restricted to a signed one-byte number (+/- 127), corresponding to about ten degrees of motion at most joints. Thus, to implement a long move it is necessary to break the move up into several short, consecutive moves. That problem is compounded by the lack of a buffer for a follow-up move, causing the motors to pause slightly at the end of each short move while the follow-up move is communicated.

That communication, itself, is somewhat tedious since commands and data are passed using ASCII coding. For example, passing a number such as '-120' requires four bytes of data, one for each character.

In recognition of those limitations, the new controller accepts signed 16-bit moves (+/- 32767), as two 8-bit (serial) data bytes, allowing even very large moves (+/- 2500 degrees) to be communicated by just two bytes of data. Furthermore, it has the capability to receive data for a following move during current move execution, and the transition from one move to the next is smooth.

The new controller has an extensive communication language. It

allows the user (under dynamic computer control) to easily monitor and alter the internal states and control parameters of the system, thus providing additional research capability. That contrasts with the old controller, which has only four marginally useful (and somewhat clumsy) commands.

The improvements in communication and control capability were not cheap. Large amounts of processing and computational power are used. The final design is a three-level communications network consisting of a host computer (highest level), a communications facilitator (middle level computer), and up to seven computers that implement control on individual motors (lowest level).

The communications network problem was considerably reduced because communications originating at the lowest level can be serviced by polling, rather than by interrupt. That is due to the high speed at which the middle level computer (Intel 8751) is capable of polling and the relatively low priority of communications originating at the lowest level.

The solution of the velocity control problem is somewhat ad hoc. However, the control algorithm has been greatly generalized, allowing the user to specify twelve parameters for each

motor, and to change them at any time, even during the execution of move commands.

The software and hardware that were developed to implement the control are specified in detail in this document. However, theoretical details supporting the effectiveness of the control algorithm are not elaborated. The lack of theoretical details is mainly due to the lack of a well-developed and computationally tractable model for the system.

Sampled-data system modelling is generally simplified by making the initial assumption that the sample rate is constant. The poor resolution of the Rhino robot position encoders led to the use of an interrupt-driven control algorithm, where an interrupt is generated each time the position encoder output changes. Clearly, the faster a motor is turning, the faster is the rate of interrupt generation. Thus the simplifying assumption of a constant sample rate cannot be made for this system.

The theoretical implications of a variable-rate sampled-data system are not trivial. In fact, they could easily be the basis of another thesis. For that reason it was impossible, under the various constraints (particularly, the time constraint), to give that topic a just treatment here.

This document primarily explains how the new controller works and presents the communication format for running it from a host computer. During its development, the controller was run from a VAX/VMS 11/780, so all references to communication assume the VAX as host. However, any computer capable of unformatted 8-bit communication over an RS/232 interface, at either 4800 or 9600 baud, can be used as the host.

POSSIBLE IMPROVEMENTS ON THE NEW CONTROLLER

While the control exercised through the new controller is far better than what was available using the original controller, it is also far from perfect. That fact suggests that work could be done on the new system to improve it further.

In particular, the microprocessor used at the lowest level for motor control is very limited in computational power and speed, and in program length. If a more sophisticated microprocessor were used, then the velocity measurement could be made with greater resolution. Similarly, the pulsewidth power to the motor could be computed with greater resolution. Those two factors together would result in tighter control of the velocity.

It would be difficult to use the existing program to control motors having much finer or much coarser angular position resolution. In the latter case, sparseness of feedback would tend to destabilize the control at low speeds. In the former case, the long computation time would tend to destabilize the control at high speeds (although that could be remedied by using a faster microprocessor or fewer of the samples).

The program running at the lowest level was written in modules, primarily to facilitate debugging the control algorithm. It would therefore be relatively easy to make changes in, or even completely rewrite the control algorithm, not having to alter the communication and bookkeeping routines.

The program running at the middle level is primarily a collection of routines that handle special cases. It would be a simple matter to expand the communication codes if the need arose; for example, the seventh lower-level computer (the Rhino has only six motors), could have special functions.

The middle-level computer could also be incorporated into the control scheme. For example, it could provide feedback paths among the low-level computers. That suggests many interesting research topics in the area of large-scale systems.

CHAPTER 2. SYSTEM OVERVIEW OF THE CONTROLLER

COMMUNICATIONS

The system can be considered as a distributed network of computers, where the host computer (hereafter, assumed to be the VAX) communicates, over an RS/232 serial link, with the Intel 8751 microprocessor; and the 8751 communicates with up to seven Intel 8748 microprocessors, each of which controls one motor of the Rhino robot. A block diagram of the entire system is given in Figure 1 on page 20.

An overview of the system must include the program running on the VAX, which communicates various commands to the Rhino robot. For example, the program might be designed to compute the sequence of move commands that will cause the Rhino robot to move along some desired trajectory. Whatever its purpose, the program must use the communication protocol and data format expected by the Intel 8751, the details of which are given in "Appendix A. Protocol and Data Format in VAX/8751 Communications" on page 26. The VAX program must also respond to the condition of a motor stalling; and it must wait to send move data until the 8751 indicates that the Rhino is ready for

another move. A sample program for the VAX is given in "Appendix S. Sample VAX Program (PASCAL)" on page 118.

The Intel 8751 can be considered as a simple communication link between the VAX and the six Intel 8748's. However, it also serves:

- to format certain data passed between the VAX and the 8748's,
- to react swiftly to emergency conditions reported by the 8748's,
- to concatenate certain communications from the various 8748's before passing them on to the VAX,
- to report system states (such as positions and velocities of the motors) when requested by the VAX.
- to read data ('peek') from any system random access memory (RAM), and write data ('poke') to locations in individual 8748 internal RAM.

Each Intel 8748 serves primarily to control the position and velocity of one motor on the Rhino robot and to sense if that motor is stalled. Its secondary, but no less important, function is to communicate with the Intel 8751. The communication protocol and data formatting are discussed in "Appendix C. Pro-

tocol and Data Format in 8751/8748 Communications" on page 39.

Immediate commands between the 8751 and 8748's, also called codes (as distinct from data), are communicated through two 'mailboxes.' The 8748 writes commands to the 8751 through one mailbox and reads commands from the 8751 through the other.

Each mailbox is equipped with a flag that is set when a command is written by one microprocessor, and cleared when it is read by the other. Details of the mailboxes are given in "Appendix D. The Mailboxes" on page 44.

Data are passed through an external random access memory to which both the 8751 and the 8748 have read- and write-privilege (called the shared RAM). Details of the shared RAM are given in "Appendix E. The Shared Random Access Memory" on page 52.

While each 8748 must communicate with only one pair of mailboxes, and gain access to only one shared RAM, the 8751 has up to seven pairs of mailboxes and seven shared RAM's to service. It was therefore necessary to devise a straightforward addressing scheme that would:

- access the read-mailbox flags (to test for the presence of

commands from the various 8748's),

- access the read-mailboxes individually (to read in a command from an individual 8748),
- access the write-mailbox flags (to check that the old command has been read before writing the new one),
- access the write-mailboxes, either individually or globally,
- access an individual shared RAM, and
- access the shared-RAM-lockout flags.

The scheme used for mailbox addressing is detailed in "Appendix D. The Mailboxes" on page 44; the scheme for shared RAM addressing is detailed in "Appendix E. The Shared Random Access Memory" on page 52.

BOOKKEEPING TASKS IN THE INTEL 8751

The 8751 must keep track of several significant conditions in each 8748. At all times it must monitor:

- which 8748s are 'in service' (that is, expected to function);
- which 8748s' motors (if any) are stalled;
- which 8748s have lost power temporarily (causing an ini-

tializing 'hardware reset' to occur);

- which 8748s are ready for another move command.

The 'in-service' status of a particular 8748 determines if the 8751 will attempt to communicate with it.

In the cases of hardware reset and motor stall, the 8751 notifies the VAX immediately if the condition is reported from any 8748. However, the 8751 waits for all in-service 8748s to report being ready for another move before notifying the VAX.

Those bookkeeping details are given in "Appendix L. Bookkeeping Tasks in the Intel 8751" on page 86.

BOOKKEEPING TASKS IN THE 8748

Keeping Track of Velocity

It is evident that the direction as well as the speed of the motion must be determined to know the velocity. Not so evident, perhaps, is that it is also important to know the recent history of the velocity in order to perform the control task properly.

Of particular interest are the cases when

- motion is just beginning
- the direction of motion reverses
- the velocity is approaching the velocity set point
- the velocity is diverging from the velocity set point

Each of those cases requires special handling, so one of the bookkeeping tasks is to determine when they occur. The details of that task are given in "Appendix K. Velocity Bookkeeping in the 8748" on page 79.

Keeping Track of Position

There are several locations in 8748 RAM and shared RAM that keep track of position. Each of those locations serves a specific function in the overall task of position control. After each motion interrupt, the various position data must be updated. The details of that task are somewhat complicated; the interested reader is directed to "Appendix J. Position Bookkeeping in the 8748" on page 73.

CONTROL CONSIDERATIONS

Stability and Variable-Rate Sampling

In a system where position data are used to generate velocity feedback, it is a significant event when the position remains constant if it should be changing. Similarly, it is a significant event when the position changes if it should be constant. Therefore, it is important to report either case to the controller.

This argues for interrupt-driven control, where an interrupt is generated

- if the position has not changed after some standard period of time, T, or
- when a change in position occurs,

The controller must distinguish the cause of the interrupt in order to take the proper action. (The details of the interrupt generation are presented in "Appendix H. 8748 Interrupt Generation" on page 67.)

The result of this scheme is variable-rate sampling. When the

motor is stopped, samples occur at a fixed rate, 1/T. When the motor is moving, however, samples occur whenever a change of position is detected, which depends on the angular velocity and on the resolution of the shaft encoder disk that provides the position feedback (details of the shaft encoder disk and the position feedback are given in "Appendix G. Shaft Position Encoding" on page 61).

So there is a threshold of velocity detection established by picking the value of T (seconds). That is, velocities resulting in detection of fewer than 1/T position changes per second cannot be discriminated as uninterrupted motion.

That may not be a problem when the encoder disk is densely encoded, but those on the Rhino motors have a resolution of between 15 and 45 degrees of a motor shaft revolution, depending on the particular motor. So picking, for example, T = 1/16 second, the minimum velocity that can be discriminated is between 40 and 120 r/min.

Thus, it is not surprising that using this type of feedback for velocity control can result in unstable performance at set points below about 200 r/min. On the other hand, one might expect that choosing larger values of T would result in sparse

feedback, which can lead to sluggish response and to instability. That result was obtained experimentally.

During motor motion, the rate of interrupt generation is clearly dependent on the velocity, so there is no fixed sample rate. That fact makes the system difficult to model, since the simplifying assumption of a fixed sample rate cannot be made.

One way to model the system is to linearize it about some fixed velocity equilibrium points (system poles thus vary with the velocities). Representing each motor by a one-pole model, each pole can be imagined to vary (dependent on the associated velocity) in the z-plane along the real line from a point near the origin (good stability) to a point near the unit circle (marginal stability). Experimental results indicate tightest velocity control occurs at speeds above 20% of full speed, with actual instability occurring under certain types of loading at speeds below 15% of full speed. This is discussed in more detail in the section titled "Performance" on page 21.

Velocity Determination from Position Feedback

Since velocity is the derivative of position, obtaining velocity data from position feedback gives rise to derivative control. The simplest method, given that Intel 8748 microprocessors were to be used, was to measure the time period, t, between motion interrupts and to determine the reciprocal, 1/t, which is the instantaneous velocity in detected position changes per second. The angular velocity is directly proportional to that result, according to the angular resolution of the encoder disk.

That scheme is not without faults. It implicitly assumes perfect spacing and detection of the transition points on the shaft encoder disk. That is, of course, not possible; the result is measurement error, which can be considered as noise due to the differentiation.

Another problem is the reciprocal operation, itself. In order to obtain good timing resolution over the full range of motor speeds it is necessary to use a two-byte clock. The Intel 8748 has neither a divide nor a multiply command, so computing a two-byte reciprocal would be an extremely time-consuming operation. For that reason, a method of taking the reciprocal was

devised using a clock frequency that is a power of 2 and a reciprocal look-up table. The method is described in detail in "Appendix K. Velocity Bookkeeping in the 8748" on page 79.

Velocity Control by Pulsewidth Modulation

The 8748 functions as a velocity controller, and can be viewed as closing a feedback loop where input is a velocity set point and output is the motor velocity. The flexibility of software allows easy definition of special situations and corresponding linear or non-linear responses.

The system state is broken down into various special cases for which different control algorithms are used, as discussed in "Appendix M. Control by Non-Linear Feedback" on page 90.

In any case, the result of the algorithm is a new value for the level of power that is sent to the DC motor. Power to the motor is varied by a pulsewidth modulator (see "Appendix O. The Pulsewidth Modulator" on page 98). The pulsewidth (duty cycle) is directly proportional to a seven-bit number received from the 8748. The polarity of the voltage applied to the motor is determined by an associated sign bit.

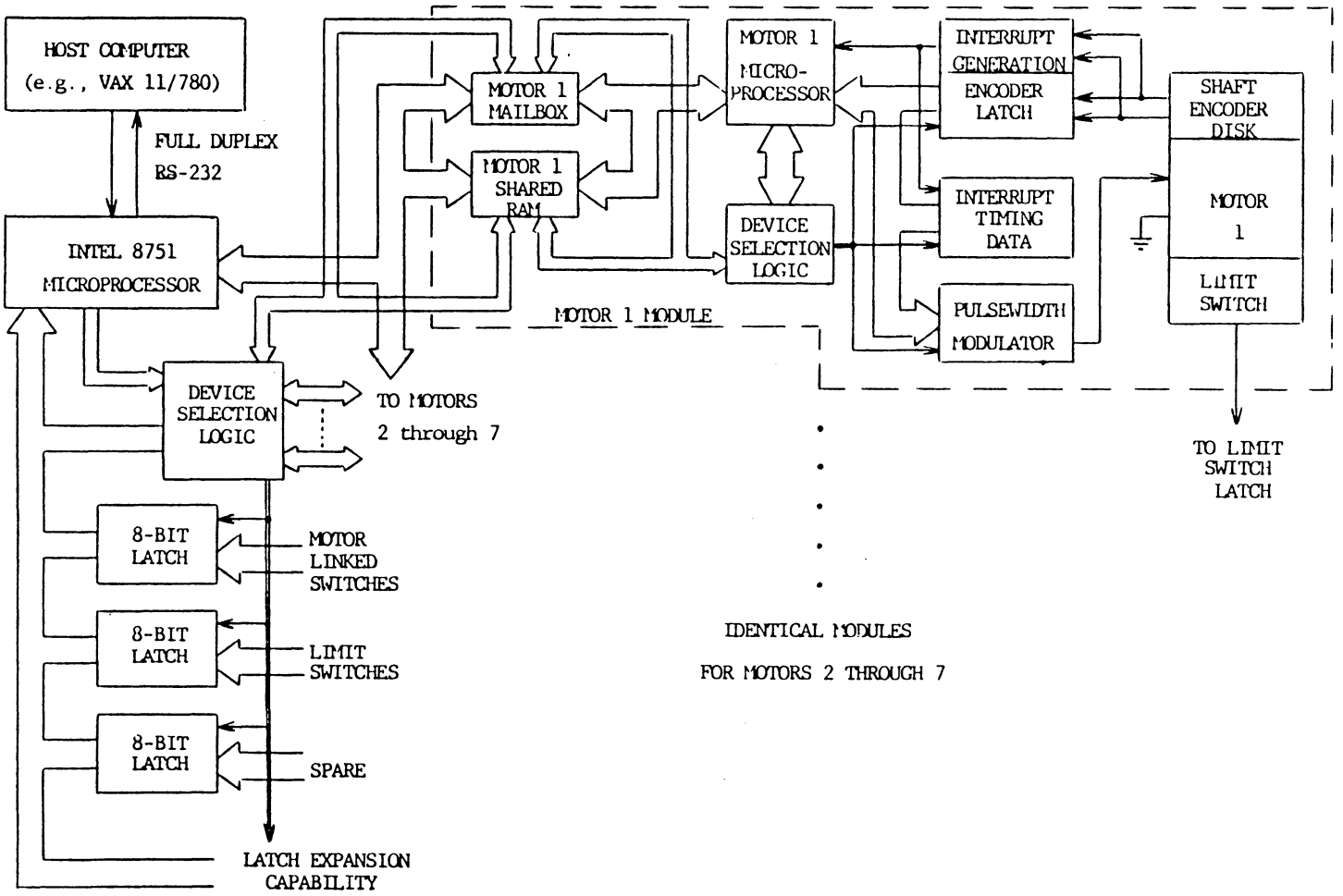

Figure 1. System block diagram.

CHAPTER 3. CONCLUSION

PERFORMANCE

Maximum motor angular speed is about 240 holes per second. Since that is slightly less than an exact power of two, representation of the velocity in binary form is optimized. (The maximum angular speed corresponds to about 5000 r/min; see "Appendix G. Shaft Position Encoding" on page 61 for details of the angular measurement in holes per second.)

Velocity measurements were made on all the motors under various loading conditions, and over the full range of velocity set points. Differences in motor loading affect performance significantly. Such differences involve changing torque, changing friction, changing load, and centrifugal forces. ;p.Some of the motors see a relatively constant load. In that category are the gripper motor, the roll and pitch motors, and the 'waist' motor. The others (the 'shoulder' and 'elbow' motors) are subject to widely varying loading, especially during maneuvers which extend or retract the robot arm.

The best results, of course, were for the four motors whose

loads are not greatly affected by configuration. In that case, for set points above 10% of maximum speed, speed was stable, fluctuating up to 8% about the set point, and exhibiting good recovery from sudden load changes and step changes of the set point. At the end of a move, regardless of the speed, position overshoot is less than thirty degrees of one motor shaft revolution (about 0.05 degrees at the joint), and velocity ramps down so smoothly that very little oscillation occurs.

For set points below 10% of maximum speed, stability was marginal, and was the worst at the slowest set point possible, which is 2% of maximum. At speeds below about 5% of maximum, many loading conditions caused instability, which manifests as starting and stopping the motor in brief bursts of motion.

The worst case was the shoulder motor at full extension under rated loading (four pounds) where instability occurred at set points below 15% of maximum when the load was being lowered, and the same move was only marginally stable at set points between 15% and 25% of maximum (marginally stable meaning the velocity fluctuates around 50% of the set point value, about the set point).

The performance results mean that for all motors the minimum

specified speed for a move will have to be greater than 5% of maximum speed (and, for the shoulder and elbow motors, greater than 15% of maximum speed) to ensure smooth motion of the robot arm. While that is somewhat restrictive, it still provides a good range of speeds for experimentation.

AREAS OF EXPERIMENTATION

Every stage of the design was carried out with the main consideration of creating a tool for experimentation. The internal workings of the system were intended to be as accessible as possible to the user.

The control algorithms of the 8748 programs were written to allow the user to change almost every significant parameter. In particular, the following are specified by the user for each motor:

- the coefficients (zero to 63/16 in increments of 1/16) multiplying the velocity error, for each of four different speed ranges;
- magnitude limits (0 to 127) on the velocity error for each of four different speed ranges, allowing non-linearities in the algorithm;

- a hysteresis (0 to 63), or minimum non-zero value of the pulsewidth (another non-linearity);
- the minimum overshoot correction power, (0 to 127) specified for overshoot less than one hole, and for overshoot greater than one hole (the same values are used for undershoot correction).

All system RAM can be read by the user. Furthermore, the user can write to the 8748 internal RAM, which is the significant area in terms of control.

The system can be run with zero to seven motors operational. Since the Rhino robot has only six motors, there is a spare axis available. It could be used for another axis of motion, or for something else entirely, such as a sensory-data processing microcomputer.

There is capability for up to six additional 8-bit-latch inputs, directly addressable from the host computer, similar to the latches used for the motor-linked switches and the limit switches.

The foregoing capabilities were designed into the system to make the research possibilities as wide-ranging as possible.

For instance, the Rhino robot with this controller could certainly be used in motion studies of trajectory control or of dynamics under load. The addressing and input capabilities make its use with sensory-feedback devices feasible. Adaptive control could be studied using a program that makes changes in the various control parameters and set points based on velocity and/or position feedback.

The user could obtain other 8748 and 8751 microprocessors and rewrite some or all of the software, to test entirely different algorithms or to expand capabilities, and plug them into the existing hardware.

In short, the system is a functional experimental tool with far more capability than existed with the old controller.

APPENDIX A. PROTOCOL AND DATA FORMAT IN VAX/8751 COMMUNICATIONS

COMMUNICATING COMMAND CODES

All command codes have a common format which promotes quick code validation. The format is easily seen if the codes are written as hexadecimal (hex) numbers. In all cases, the upper and lower hex digits are identical. In other words, from each hex code, the hex number 11 (decimal 17) can be factored. A complete list of the codes which the VAX can send to the 8751 is given in Figure 2 on page 30.

The communication originating from VAX initiates a predictable sequence of events in the 8751. For some commands, the 8751 will request data to be transmitted. For others, the 8751 sends a simple response or prepares data for transmission to VAX. This is called the communication protocol for VAX-originated commands, and is tabulated in Figure 3 on page 31.

The 8751 has a set of command codes which it can send to VAX.

Some of them are sent only in response to VAX commands, but there are some which can originate from the 8751. The code format is the same as that described above for the VAX codes. A complete list of the codes which the 8751 can send to VAX is given in Figure 4 on page 32.

Communication originating from the 8751 must elicit certain expected responses from VAX, in the cases where a response is required. In most cases, no response is expected from VAX, so the program running on the VAX has a bit of latitude in that area. The communication protocol for 8751-originated command codes is tabulated in Figure 5 on page 33.

COMMUNICATING DATA

Data are serially communicated between VAX and the 8751 only after a 'send data' code has been transmitted and received successfully. Data for each command have a unique format, as enumerated here.

Data from VAX to the 8751:

1. The data for a move command are transmitted in two groups, each group containing data for seven motors.
   a. The angular distance for each motor is formatted as a sixteen-bit number, high byte transmitted before low byte; motor seven data are transmitted first, motor 1 data last; the lower 15 bits are the (unsigned) magnitude of the move, in encoder holes; the highest, or sixteenth, bit is the direction of the move, where a zero indicates clockwise.
   b. The velocity for each motor is a number from 0 to 127, and represents one-half of the velocity set point in holes per second, rounded up; motor seven data are transmitted first, motor one data last.
2. A motor number is sent for several of the commands; it is essential that it not exceed seven (that is, the upper five bits of the byte must be clear).
3. Coefficients are from 0 to 63.
4. Addresses for 8751 peek and shared RAM peek are from 0 to 255.
5. Addresses for the 8748 peek/poke are between 0 and 63; the eighth bit of the address byte is also used, indicating 'peek' if it is clear and 'poke' if it is set.

Data from the 8751 to VAX:

1. Format for data in response to 48PK or SHRPK: first, the command code that elicited the data; next, the Rhino Clock count, a sixteen bit positive number, high byte first; the address of the peek comes next; finally, the data read at that address from each motor, with motor seven data first, motor one data last.
2. Format for data in response to 51PK: first, the 8751 peek code; next, the Rhino Clock count, a sixteen bit positive number, high byte first; the address of the peek comes next; finally, the data read from that address (from internal 8751 RAM if the address is 0-127; from the latch that is enabled by the address if the address is 128-255).
3. Format for data in response to VRQ: first, the Rhino Clock count, a sixteen bit positive number, high byte first; followed by the velocity of each motor, a one-byte number in two's-complement, representing one-half the velocity; motor seven's velocity first, motor one's last.
4. Format for data in response to PRQ: first, the Rhino Clock count, a sixteen bit positive number, high byte first; followed by the position of each motor, a two-byte number in two's-complement form, high byte first, with motor seven data first and motor one data last.

COMMAND CODES FROM VAX 11/780 TO INTEL 8751		
CODE	mnemonic	COMMAND
0	NOP	No operation
17	ECI	Enable 'command-on-interrupt'
34	DCI	Disable 'command-on-interrupt'
51	48PK	'Peek' into 8748 internal RAM
	WR48	'Poke' into 8748 internal RAM
68	51PK	'Peek' into 8751 internal RAM
85	VRQ	Velocity data request
102	PRQ	Position data request
119	SNDVX	Send data to VAX (from 8751)
136	OVR	Send previous code over again
153	CRM	Clear remaining moves
170	MOV	Move
187	HLT	Halt
204	AGO	All motors Go
221	SHRPK	'Peek' into shared RAM
238	RAP	Reset absolute position
255	KFF	Coefficients

Figure 2. Command codes from VAX to 8751.

VAX ORIGINATED CODE	8751 RESPONSE	VAX RESPONSE	8751 RESPONSE	VAX RESPONSE	8751 RESPONSE
NOP (0)					
DCI (34)	DCI (34)				
CRM (153)	CRM (153)				
HLT (187)	HLT (187)				
AGO (204)	AGO (204)				
RAP (238)	RAP (238)				
OVR (136)	1 byte data	depends on data			
VRQ (85)	VDT (85)	SNDVX (119)	9 bytes data		
PRQ (102)	PDT (102)	SNDVX (119)	16 bytes data		
ECI (17)	SND51 (17)	1 byte data	RCVD (119)		
			INV (255)		
KFF (255)	SND51 (17)	15 bytes data	RCVD (119)		
51PK (68)	SND51 (17)	1 byte data	DAT (68)	SNDVX(119)	5 bytes data
SHRPK (221)	SND51 (17)	1 byte data	DAT (68)	SNDVX(119)	11 bytes data
48PK (51)	SND51 (17)	1 byte data (high bit clear)	DAT (68)	SNDVX(119)	11 bytes data
WR48 (51)	SND51 (17)	1 byte data (high bit set)	SND51 (17)	2 bytes data	RCVD (119)
MOV (170)	SND51 (17)	14 bytes data	SND51 (17)	7 bytes data	RCVD (119)

Figure 3. Communication protocol for VAX-originated codes to 8751.

COMMAND CODES FROM INTEL 8751 TO VAX 11/780		
CODE	mnemonic	COMMAND
0	NOP	No operation
17	SND51	Send data to 8751 (from VAX)
34	DCI	Disabled command-on-interrupt
51	RDYNXT	Ready for next move
68	DAT	'Peek' data ready to transmit
85	VDT	Velocity data ready to transmit
102	PDT	Position data ready to transmit
119	RCVD	Data received
136	OVR	Send previous data over again
153	CRM	Clear remaining moves
170	HDWRST	Hardware reset
187	HLT	Halted
204	AGO	All motors in 'go' state
221	MTRSTL	At least one motor is stalled
238	RAP	Reset absolute position
255	INV	Invalid data

Figure 4. Command codes from 8751 to VAX.

INTEL 8751 ORIGINATED CODE	VAX 11/780 RESPONSE	INTEL 8751 RESPONSE
NOP (0)		
RDYNXT (51)		
HDWRST (170)		
HLT (187)		
MIRSTL (221)		
OVR (136)	1 byte data	depends on data
VDT (85)	SNDVX (119)	9 bytes data
PDT (102)	SNDVX (119)	16 bytes data
DAT (68)	SNDVX (119)	either 5 or 11 bytes data

Figure 5. Communication protocol for 8751-originated codes to VAX.

APPENDIX B. 8751 SOFTWARE FOR VAX COMMUNICATION

SCHEME FOR VAX COMMUNICATIONS

All communication with the host computer occurs over RS/232 serial link. The highest priority of the 8751 program is servicing the asynchronous receiver/transmitter. This is accomplished by setting the serial port interrupt to the highest priority and always having that interrupt enabled.

The 8751 distinguishes between 'code' and 'data' from the VAX. Command code is expected to be received in the symmetric format described in "Appendix A. Protocol and Data Format in VAX/8751 Communications" on page 26, while data is unformatted. The two different classes of communication are treated completely differently from the moment they are received.

The following discussion refers to various locations in 8751 internal RAM, the allocation of which is shown in Figure 18 on page 104.

Processing Commands in the Interrupt Routine

The serial-port-interrupt-service subroutine only distinguishes between the two classes of communication by the value found in the location denoted NR, which is short for 'Number of data bytes to Receive'. If NR is zero, then the communication from VAX is taken to be a command code.

Command codes received from the VAX are immediately moved into a holding location, called the Serial Port Read Buffer. The code is held there until all data associated with it have also been received, after which the code is moved to the VAX Command Buffer, where it will be detected by the main program and decoded.

The foregoing process may take several iterations of the serial-port-interrupt-service subroutine, depending on the amount of data expected with the particular command.

Processing Data in the Interrupt Routine

If the number NR is not zero, then the communication from VAX is taken to be data. There is a 15-byte receive-data buffer in 8751 internal RAM. Data associated with a command code is held there until all of it is received, after which the main program is allowed to process it.

The data arriving over the serial port is put in the location computed by adding the number (NR - 1) to the lowest address of the receive-data buffer. After a data byte is received the number NR is decremented by one. Thus, the last byte of data received will always be at the bottom of the receive-data buffer.

Processing Communications in the Main Routine

If the main routine is not currently processing a command, then it will look for the arrival of a new command code in the VAX Command Buffer. When a code appears there, it is moved to the

VAX Command location, and the decode-VAX-command subroutine is called to take the appropriate action.

That subroutine will generate a response, and may also generate data, that must be transmitted to VAX. The response appears in the Response Code location, and the data appears in the 16-byte write-data buffer.

The number of data bytes generated appears in NTBUFF, where it is 'invisible' to the various transmit routines until the VAX has sent a request for data. At that time, it will be moved to the NT location, where NT is short for Number of data bytes to Transmit.

After a response to a command is generated, the Write-to-VAX subroutine is invoked. If the transmitter is idle, the code is sent immediately; otherwise, it is queued for transmission.

Transmitting Responses

The priority of transmission is this:

1. Data for transmission (determined by NT not zero, where NT denotes the Number of data bytes to Transmit).
2. Contents of the Serial Write Buffer
3. Contents of the Response Code location.
4. A request for data, if VAX has indicated it has data to transmit.

If the transmitter is not idle, but the Serial Write Buffer is empty, then the contents of the Response Code location are moved to the Serial Write Buffer (queued for transmission).

After a byte of data is transmitted, the number NT is decremented by one. Thus, the value of NT indicates not only how much data is left to transmit, but also points to the location of the next byte to transmit, in the same manner as NR, above.

For details, see "The Serial Output Subroutine" on page 132 and "The Serial Port Interrupt Service Subroutine" on page 144.

APPENDIX C. PROTOCOL AND DATA FORMAT IN 8751/8748 COMMUNICATIONS

COMMUNICATING COMMAND CODES

All command codes have a common format which promotes quick code validation. The format is easily seen if the codes are written as hexadecimal (hex) numbers. In all cases, the upper and lower hex digits are identical. In other words, from each hex code, the hex number 11 (decimal 17) can be factored.

Complete lists of the codes which the 8751 and the 8748 can send each other are given in Figure 6 on page 42.

The command codes sent between the 8751 and 8748 result in the completely predictable responses written into software. These are called the communication protocol, and are tabulated in Figure 7 on page 43.

COMMUNICATING DATA

All data passed between the 8751 and the 8748s is passed through the shared RAM. There is only one set of data for which the 8748 requires a different format from the 8751 than the 8751 requires from the VAX. That is, the 8751 passes on all other data to the 8748's in exactly the same form they were received from VAX.

The data that are reformatted are the move distance data. The main reason is to remove a considerable computational burden from the 8748 and put it on the 8751. Since the 8751 is a more powerful microprocessor, the same tasks are not as difficult to perform.

The reformatting of the move distance data is outlined here.

- The sign bit is stripped off of the high byte of the distance, leaving only a 15-bit magnitude, which is moved to the low and high byte 'next move' locations in shared RAM.
- The sign bit is loaded into the 'next move sign' location in shared RAM, where it is in the eighth bit position, with all other bits cleared.

- If the 15-bit move is not zero, the 'next move received' flag in shared RAM is set to a non-zero value; otherwise, it is set to zero.
- If the 15-bit move exceeds the number 31, then the 'next move infinite' flag in shared RAM is set to a non-zero value; otherwise, it is set to zero.

Data passed from the 8748 to the 8751 are forwarded to VAX in unchanged format, with the exception of velocity data. The velocity data come from the 8748's as seven bit magnitudes with the direction sign in the eighth bit positions. The 8751 reformats that data into 8-bit, two's complement numbers before sending them to VAX.

COMMAND CODES FROM INTEL 8751 TO INTEL 8748's		
CODE	MNEMONIC	COMMAND
0	NOP	no operation
136	OVR	Send previous code over again
153	CRM	Clear remaining moves
170	MOV	Move
187	HLT	Halt
204	AGO	Go
221	48PK	'Peek' into internal 8748 RAM
	WR48	'Poke' into internal 8748 RAM
238	RAP	Reset absolute position
255	KFF	Coefficients

COMMAND CODES FROM INTEL 8748 TO INTEL 8751		
CODE	MNEMONIC	COMMAND
0	NOP	no operation
136	OVR	Send previous code over again
153	MTRSTL	Motor is stalled
170	HDWRST	Hardware reset
187	RDYNXT	Ready for next move
204	DAT	'Peek' data ready

Figure 6. Command codes between 8751 and 8748.

8751 ORIGINATED CODE	8748 RESPONSE	8748 ACTION
NOP (0)	none	none
OVR (136)	1 byte data	clear previous code location
CRM (153)	none	clear all present move, next move, pulsewidth magnitude and velocity data
MOV (170)	none	read next move data in from shared RAM to internal RAM next move locations
HLT (187)	none	clear the 'go' flag
AGO (204)	none	set the 'go' flag
RAP (238)	none	clear the absolute position locations in shared RAM
KFF (255)	none	read coefficients in from shared RAM to the internal RAM coefficient locations
WR48 (221)	none	read in 'poke' data from shared RAM into the specified internal RAM location
48PK (221)	DAT (204)	read in 'peek' address from shared RAM and put data from that address in internal RAM in data passing location of shared RAM

8748 ORIGINATED CODE	8751 RESPONSE	8751 ACTION
NOP (0)	none	none
OVR (136)	1 byte data	clear previous code location
MTRSTL (153)	HLT (187)	set the motor's stall flag, halt all motors, and send 'motor stalled' code to VAX
HDWRST (170)	none	set the motor's 'hardware reset' flag, halt all motors, and send 'hardware reset' code to VAX
RDYNXT (187)	none	clear the motor's 'not ready for next move' flag

Figure 7. 8751/8748 Communications protocol and resulting actions.

APPENDIX D. THE MAILBOXES

8748 ACCESS TO THE MAILBOXES

There are two mailboxes associated with each 8748. The 8748 can read command codes from its read mailbox that were put there by the 8751; the 8748 can write command codes to its write-mailbox for the 8751 to read.

Some confusion may arise over the fact that the 8748-read-mailbox is one of the seven 8751-write-mailboxes; and the 8748-write-mailbox is one of the seven 8751-read-mailboxes. However, it need only be remembered that each microprocessor reads from its own read mailbox and writes to its own write mailbox.

The mailboxes are shown in functional block diagram form in Figure 8 on page 49. The diagram also shows the method by which the 8748 addresses its mailboxes. The same diagram is shown in schematic form in Figure 23 on page 110.

For the 8748, the binary address 100xxxxx (where x indicates

either a zero or a one), when written on Port 2, will cause the mailbox-enable line to become active (low). That line serves a dual purpose.

When it is active, the mailbox-enable gates the read- and write-pulses from the 8748 through to the read- and write-mailboxes. Notice that it also gates the write-mailbox-flag through to Test Input 1 of the 8748; when it is not enabled, the read-mailbox-flag is gated through to that same test input. So the read-mailbox-flag is tested when the mailboxes are not enabled, and the write-mailbox-flag is tested when they are enabled.

Thus, putting the mailbox-enable address on Port 2 is a device select operation. When they are enabled, the mailboxes act very much like random access memory (except, of course, no address need be multiplexed on the bus lines to read or write). The 8748 software access the mailbox only from the main routine, which is given in detail in the appendix, "Listing of the 8748 Assembly Language Program."

8751 ACCESS TO THE MAILBOXES

The 8751 has up to fourteen mailboxes to which it must gain access, two for each motor. The access scheme is similar to that of the 8748s, only on a larger scale. The details of addressing the mailboxes can be found in the Respond-to-8748, Decode-8748-Command, and Global-Write subroutines in the appendix, "Listing of the 8751 Assembly Language Program."

The hardware which implements the access scheme is shown in block diagram form in Figure 10 on page 51 and Figure 9 on page 50, and in schematic form in Figure 22 on page 109.

Two 74LS138 decoders are used to gate the 8751 read- and write-pulses through to the proper mailboxes. Note that 8751 Port 2 is used to output the device address.

The read-mailboxes are addressed by a binary number xaaa0xxx, where 'aaa' is the (binary) motor number assigned to the mailbox being addressed, and x indicates either a zero or a one.

The write-mailboxes are addressed by a binary number 1aaa0xxx, where 'aaa' is the (binary) motor number assigned to the mail-

box being addressed, and x indicates either a zero or a one. Note that if 'aaa' = '000', then all seven write-mailboxes are addressed simultaneously; thus, the write-mailboxes can be addressed either globally or locally.

The mailboxes behave like random access memory from the point of view of the 8751, except no address need be multiplexed on Port 0 to perform a read or write. However, the 8751 (unlike the 8748) cannot perform a read or write on Port 0 without multiplexing some address; so whenever a mailbox is read from or written to, a dummy address is specified.

The 8751 checks the mailbox flags, just as the 8748 does. With reference to Figure 10 on page 51, it can be seen that the same addresses that access the read-mailboxes on Port 0 also enable the output of the read-mailbox-flag latch onto Port 1. (Note that the address 00000000 on Port 2 also enables the read-mailbox-flag latch output.)

Similarly, the same addresses that access the write-mailboxes on Port 0 also enable the output of the write-mailbox-flag latch onto Port 1.

The system was designed that way to allow a mailbox's flag to be

checked using the same address that accesses the mailbox, where the state of the flag appears at Port 1 and the mailbox is accessed through Port 0.

The mailbox flags from motor 1 appear in the bit 1 position of Port 1 input, those from motor 2 appear in the bit 2 position, etc. This scheme allows quick isolation of the flag of interest within the 8751 software.

The schematic of the hardware for addressing the mailboxes and accessing the flags is given in Figure 22 on page 109.

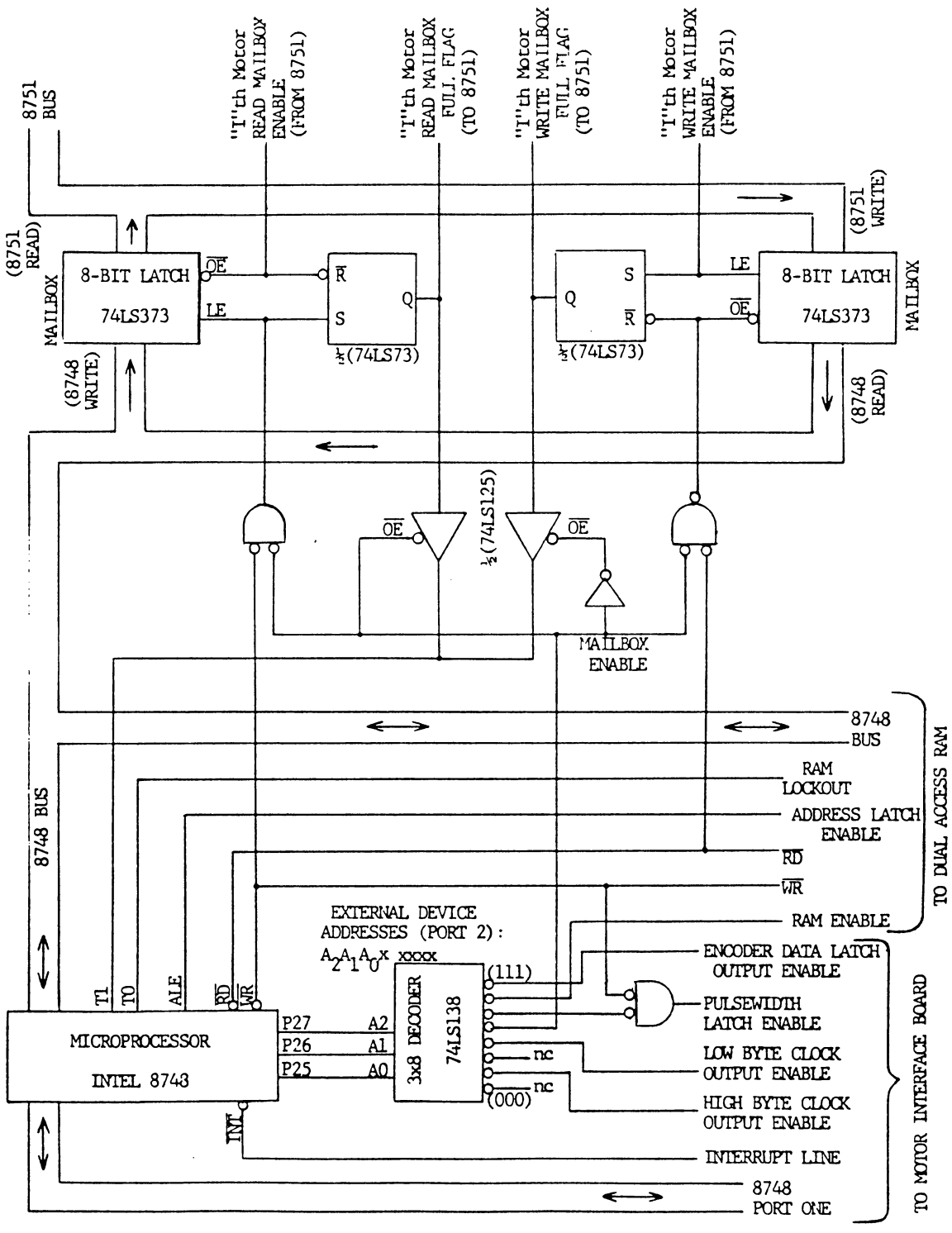

Figure 8. The mailboxes and the 8748 device addressing scheme.

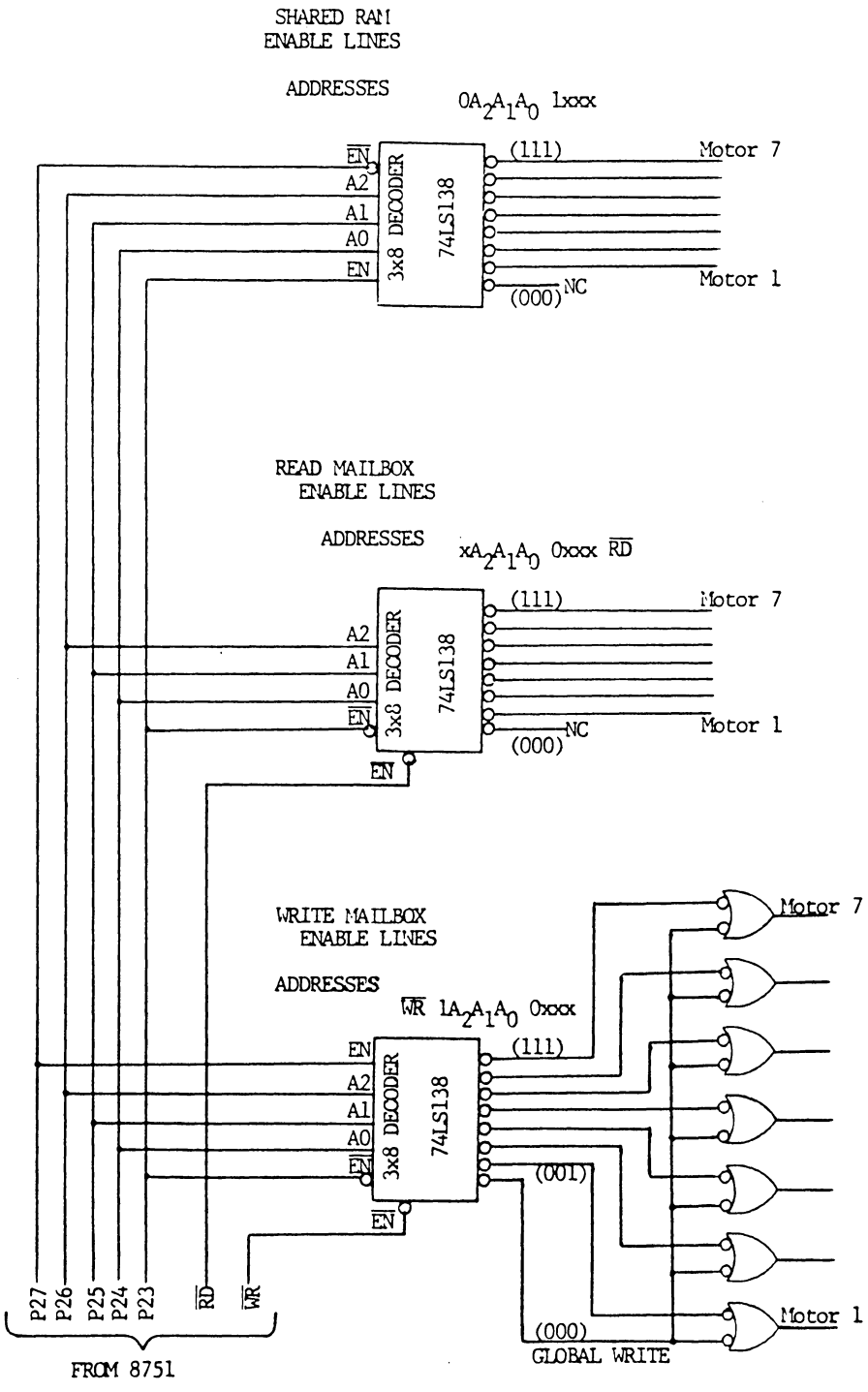

Figure 9. 8751 Port 0 Input Device Addressing Scheme.

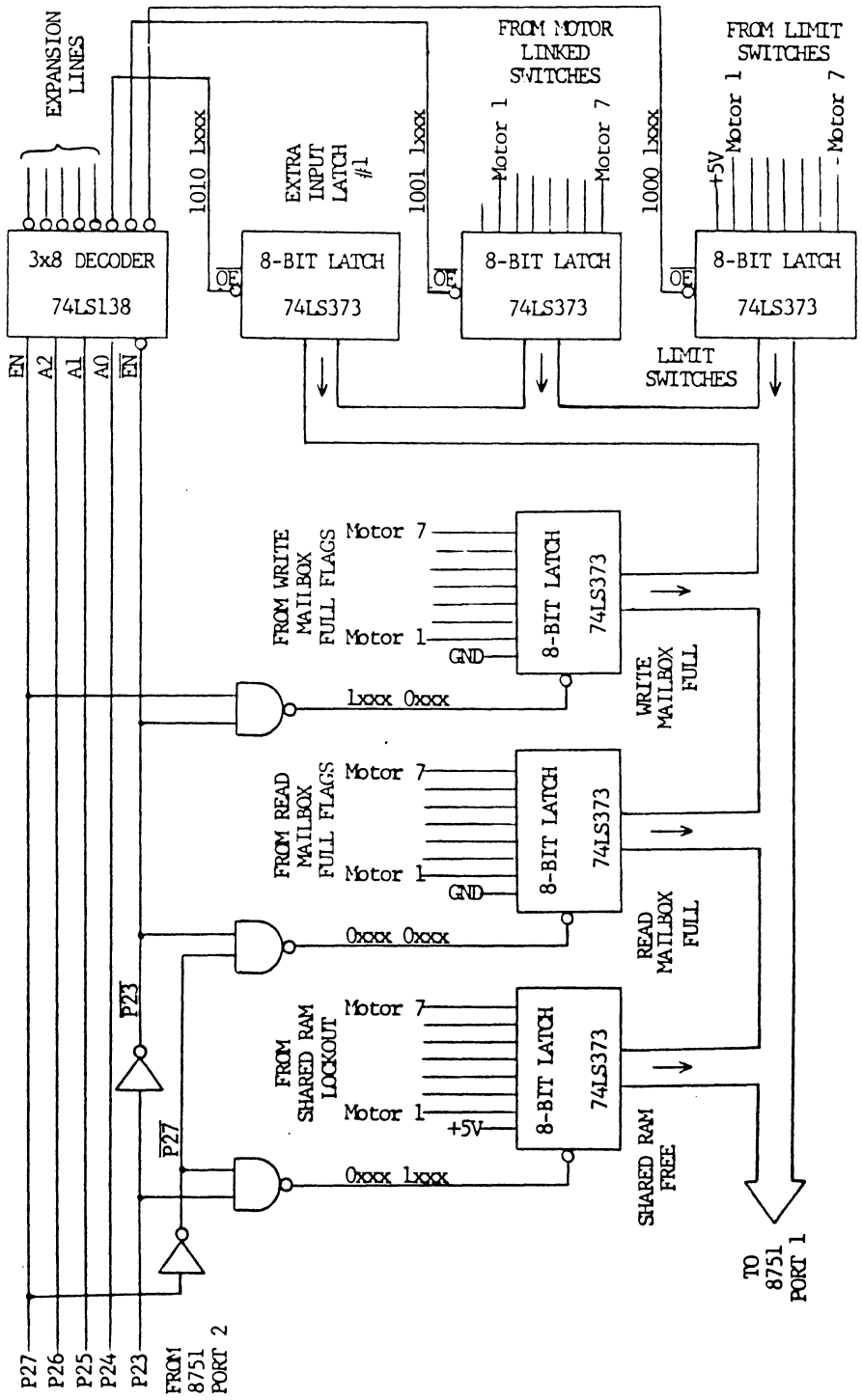

Figure 10. 8751 Port 1 Input Device Addressing Scheme.

APPENDIX E. THE SHARED RANDOM ACCESS MEMORY

The shared random access memory (RAM) appears to both the 8748 and 8751 as a normal external RAM. In practice, the only difference is that it must be ascertained that the other device with access is not using it before proceeding to read or write.

For the purpose of protecting the various hardware devices, the system was designed with an absolute 'hardware lockout'. That is, if one of the microprocessors is using the shared RAM, it is impossible for the other one to do so. So it is that the one which is locked out could go through the motions of reading or writing, but the result would be the same as if there were no RAM connected to it. That design was achieved by the use of tri-state buffers on every line that is needed to read and write to the RAM.

With reference to Figure 11 on page 55, it can be seen that the data bus lines from each microprocessor must be enabled through the 74LS245 tri-state bus transceiver before any address or data can be passed to or from the shared RAM. The control lines, consisting of the address latch enable and the read- and write-pulse lines, must also be enabled through tri-state buf-

fers, before a read or write can be performed.

The heart of the lockout system is the 74LS74 dual delay-type flip-flop with overriding presets. The sequence of events is as follows:

1. The microprocessor program puts the address on Port 2 which selects the shared RAM.
2. The device select logic decodes the Port 2 address and activates the (low active) shared RAM enable line.
3. The microprocessor attempting to gain access tests its lockout line: if it is high, then the other microprocessor is locked out, and the read or write can be performed; if it is low, then the other microprocessor is using the shared RAM, so a waiting loop is performed until the lockout line goes high.
4. When the lockout line is high, the delay flip-flop, which is being clocked at 6 MHz, gates the low enable through to the various tri-stated buffers, connecting the address/data bus to the address latch and to the shared RAM, and connecting the control lines to their appropriate points.
5. The active low output of the the delay flip-flop is also connected to the overriding preset of the other half of the

dual flip-flop, causing it to remain in the set (inactive) condition, regardless of the input, thus locking out the other microprocessor.

6. After the read or write is performed, the enable address is removed from Port 2 to allow the other microprocessor to have access to the shared RAM.

Note the necessity for the pull-up resistors at the tri-state outputs. This prevents spurious chip selecting and address latching, thus conserving power and extending the life of the memory chips.

A detailed schematic of the shared RAM can be seen in Figure 21 on page 108.

The scheme for 8748 access to the shared RAM is shown in block diagram form in Figure 8 on page 49 and in schematic form in Figure 23 on page 110.

The scheme for 8751 access to the shared RAM is shown in schematic form in Figure 20 on page 107, which corresponds to parts of the two block diagrams, Figure 9 on page 50 and Figure 10 on page 51.

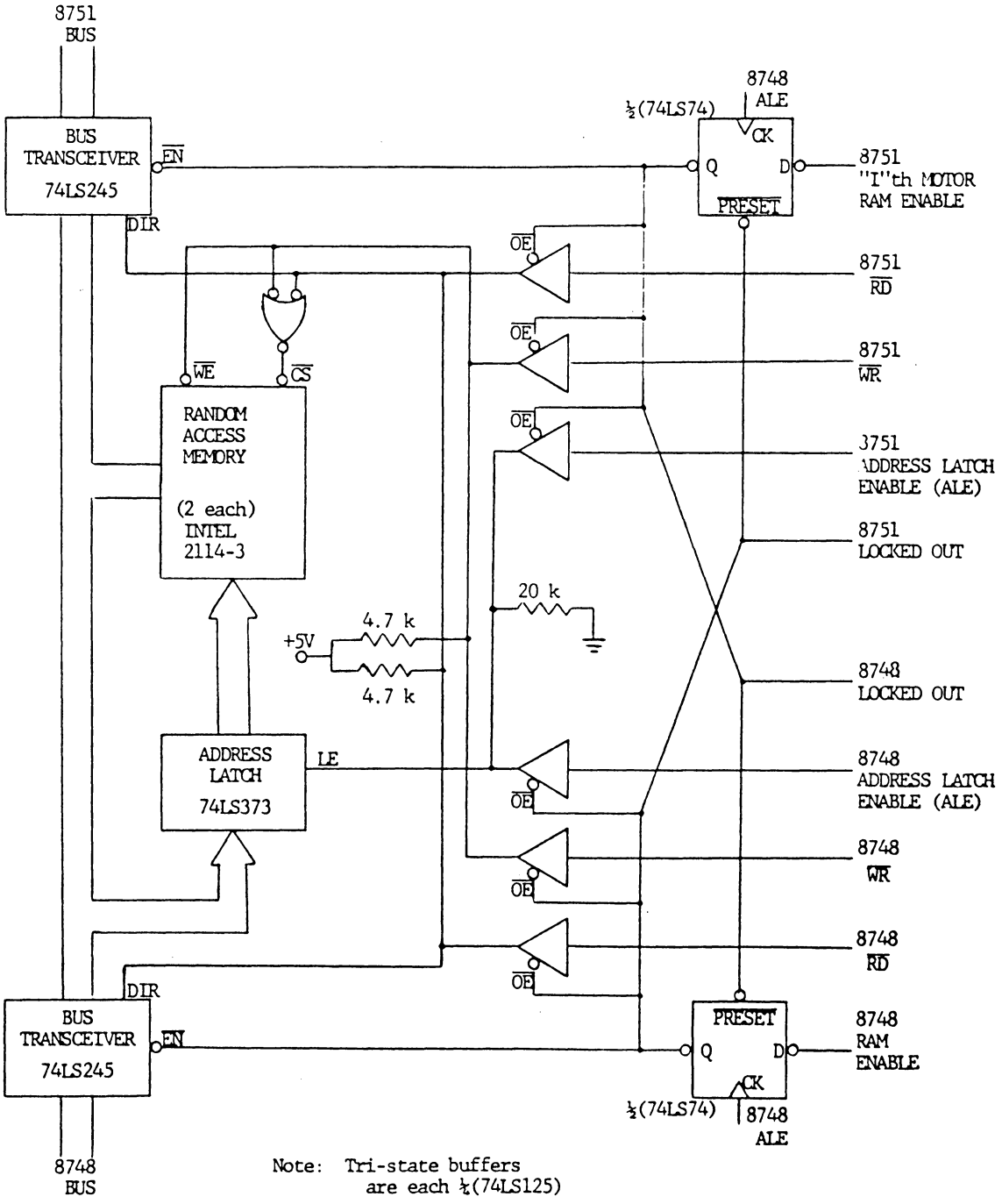

Figure 11. Block diagram of the shared RAM.

APPENDIX F. THE COMMANDS FROM VAX

The various commands which can be sent from the VAX are listed in Figure 2 on page 30. Those which can be considered to originate from the VAX, as opposed to being used only in response to 8751 codes, are shown along with their protocol in Figure 3 on page 31.

COMMANDS CONTROLLING THE STATE

Commands which change some state of the 8748's or the shared RAM are said to control the state. Commands in this category are somewhat self-explanatory. They are listed here:

- HALT (HLT) : This command clears the 'go' flags in all the 8748's, and can be considered to put the robot in the halt state; in that state no power is applied to any of the motors, but the 8748's keep track of changes in position.
- GO (AGO) : This command sets the 'go' flags in all the 8748's, and can be considered to put the robot in the go

state; in that state the 8748's attempt to reduce position error to zero, using the current value of the velocity set point for velocity control.

- RESET ABSOLUTE POSITION (RAP) : This command causes all 8748's to set their absolute position counters to zero.
- WRITE TO 8748 (WR48) : This command has data associated with it which will be written into the internal RAM of the specified 8748; it is referred to elsewhere in this document as the 'poke' command.
- COEFFICIENTS (KFF) : This command allows the user to write the various position error coefficients and magnitude limits, the gripper closed pulsewidth (ignored by all but the gripper controller), the starting pulsewidth, pulsewidth hysteresis value, and long and short overshoot pulsewidths to a specified 8748 internal RAM.
- MOVE (MOV) : this command allows the user to alter the values of the position error and velocity set point in all of the 8748's at one time.
- CLEAR REMAINING MOVES (CRM) : this command causes all 8748's to set their internal position error, next move data, and velocity data locations to zero; note that if the motors are in motion, the configuration when this command is received becomes the point of zero position error.

COMMANDS MONITORING THE STATE

Commands which are used to determine the state of the system are said to monitor the state. They are listed here:

- VELOCITY DATA REQUEST (VRQ) : This command results in a stream of data from the 8751 representing the current motor velocities and the internal time (Rhino Clock) of the system.
- POSITION DATA REQUEST (PRQ) : This command results in a stream of data from the 8751 representing the current configuration of the motors with respect to some calibration point, and the internal time (Rhino Clock) of the system.
- SEND LAST CODE OVER AGAIN (OVR) : This command causes the 8751 to repeat the most recent command code sent to VAX.
- PEEK 8748 (48PK) : This command results in a stream of data from the 8751 representing the data found at the specified location in each 8748, and the internal time (Rhino Clock) of the system.
- PEEK 8751 (51PK) : This command results in a stream of data from the 8751 representing the data found at the specified location in the 8751, and the internal time (Rhino Clock) of the system.

- PEEK SHARED RAM (SHRPK) : This command results in a stream of data from the 8751 representing the data found at the specified location in all of the shared RAM's, and the internal time (Rhino Clock) of the system.

REPETITIVE COMMAND FUNCTION (COMMAND-ON-INTERRUPT)

There is one pair of commands that does not fit into either of the preceding categories. The function of this command pair is to allow a state-monitoring command to be repeated quickly without actually sending it from the VAX.

When this function is invoked, it is accompanied by the code of the command that is to be repeated. Execution of the accompanying code then occurs whenever the state of 8751 Interrupt 0 falls from high to low. Whenever that interrupt condition occurs, the Rhino Clock is incremented as well (it is not incremented at any other time).

The intent of this design is that the Interrupt 0 line should be connected to a constant frequency square-wave generator. That

causes the Rhino Clock to be an accurate measure of the passage of time relative to the system. Then, whenever one of the monitor commands is repeated, there is a time-identifier along with the system data. Thus the Rhino Clock provides data for accurate time graphing of system states.

ENABLE COMMAND-ON-INTERRUPT (ECI) : This command is accompanied by one of the commands, VRQ, PRQ, PK48, PK51. The accompanying command will be queued for execution each time the Interrupt 0 line falls from high to low. In the case of the peek commands, the last address specified for a peek before the ECI command is used for the repetitive peek.

DISABLE COMMAND-ON-INTERRUPT (DCI) : This command causes the interrupt command buffer to be cleared (effectively loading the no-operation command). Thus, when Interrupt 0 transits high to low, the Rhino Clock is still incremented, but no command is queued for execution.

APPENDIX G. SHAFT POSITION ENCODING

THE SHAFT ENCODER DISK AND ANGULAR MEASURE

The shaft encoder disk is a disk that is attached to the motor shaft to provide position feedback. It has two perforated annular regions which allow light from light-emitting diodes to reach photo-sensitive transistors when the disk is in certain positions, and which block the light in other positions. A typical disk is shown in Figure 12 on page 65, where the perforations are arranged in what is called quadrature encoding.

The reason for quadrature encoding is that the succession of two-bit 'words' from the collectors of the photo-sensitive transistors when the motor is rotating clockwise is the opposite of the succession generated by counterclockwise motion. Thus the direction of motor motion can be deduced by inspection of the two-bit encoder output sequence.

It seems natural to use the encoder shaft to measure angular distance. In this document, the distance of motor shaft angular motion is discussed in terms of 'holes'. One hole is

defined as the angular distance from the center of a position where the encoder bits are (0,0) to the center of the next position where the encoder bits are (0,0). The three intermediate positions will be referred to as 'quarter-hole' positions, and the angular distance from one to the next is called one quarter-hole.

Note that the resolution of the encoder disk attached to a particular motor determines the constant of proportionality between velocity in holes per second and velocity in revolutions per minute (r/min) for that motor.

ENCODER OUTPUT

The output of the encoder comes from photo-sensitive transistors. Thus, when a void area of the encoder disk falls between the light source and the transistor, the transistor is switched on; when a solid area of the disk falls between them, the transistor is switched off.

Consider what happens when the transistor is switched on, but a

solid part of the disk is rotating into position between the light source and the photo-sensitive transistor. As the shadow of the solid area falls onto the transistor, the amount of light falling onto it decreases. However, the decrease in light is not abrupt, especially at slow speeds, so the transistor operates over its linear region for the period during which transition from light to dark is occurring.

Linear operation is not desirable in a digital system. It is therefore necessary to condition the data from the position encoder. In particular, the transitions between low and high levels should be quite abrupt; that is, the encoder output should be two square waves, regardless of motor speed. That is the function of the bit-shaping circuitry, shown in block form in Figure 13 on page 66 and in complete detail in Figure 24 on page 111.

Referring to the block diagram, it can be seen that each photo-sensitive transistor's output is compared to a +2.5-volt reference by an LM339 comparator. This causes the transitions between low (0 volts) and high (5 volts) levels to occur in only a few nanoseconds.

The 220 kilohm and 1 megohm resistors provide hysteresis in the

comparison process. A simple analysis shows that they cause the low-to-high transition to occur at 3.05 volts, while the high-to-low transition occurs at 1.95 volts. The point of introducing hysteresis is to prevent the comparator from oscillating when the input voltage from the photo-sensitive transistor is changing slowly and is close to the transition voltage.

Notice that 4.7 kilohm pull-up resistors are needed for both the open-collector photosensitive-transistors and the comparator outputs.

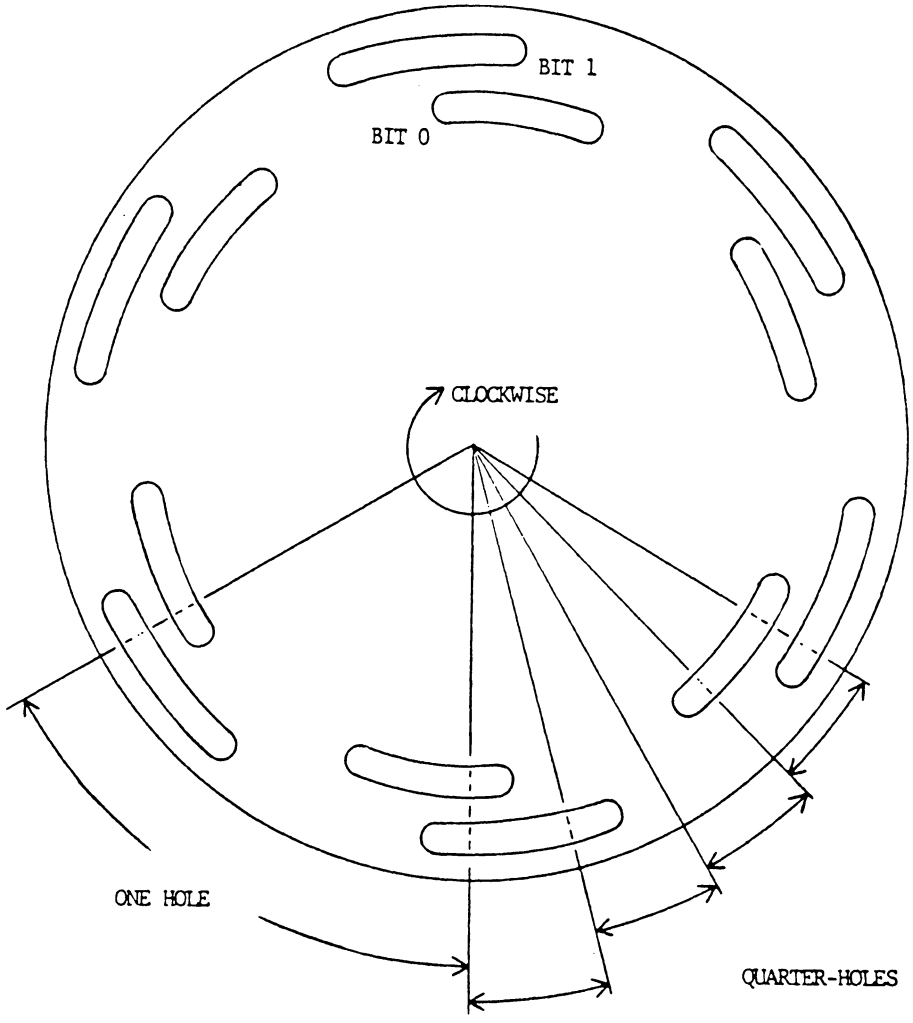

Figure 12. Typical shaft encoder disk and its natural angular measurement.

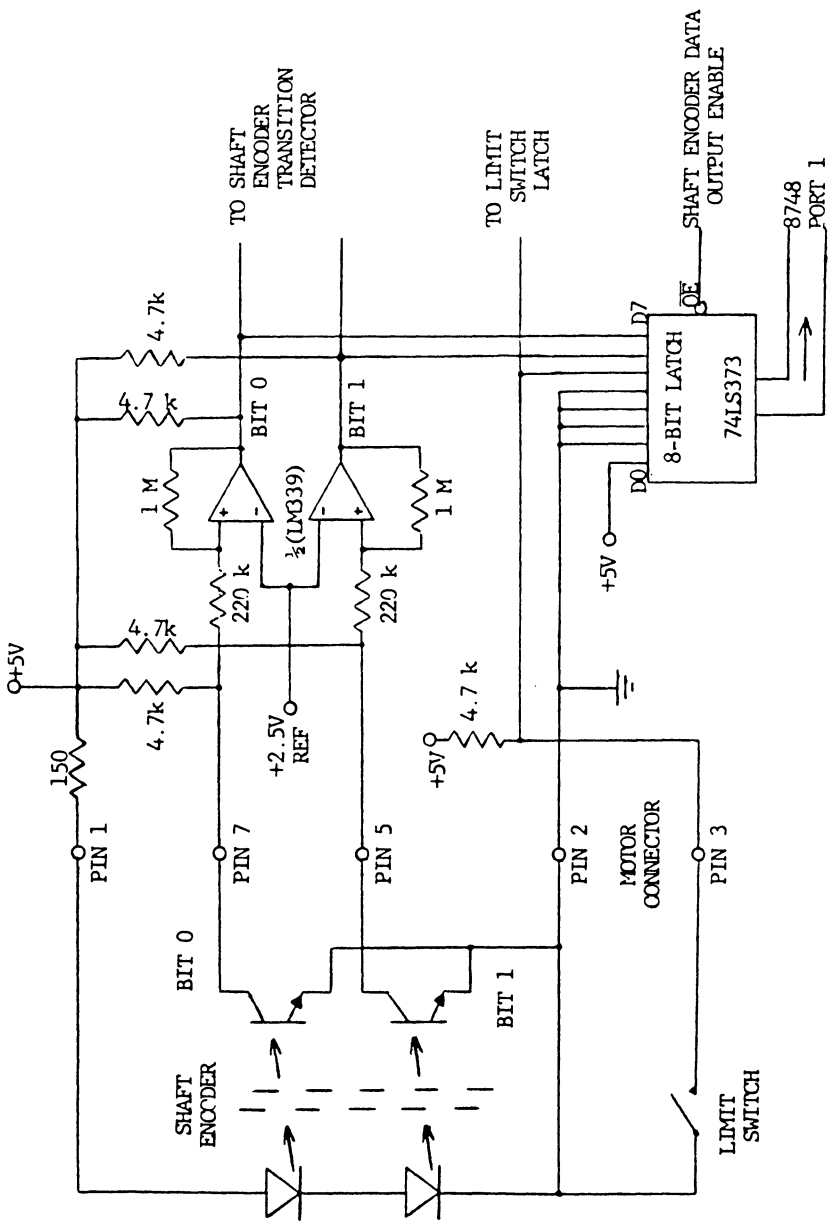

Figure 13. Block diagram of the shaft encoder and output conditioning.

APPENDIX H. 8748 INTERRUPT GENERATION

MOTOR MOTION INTERRUPT GENERATION

The position encoder output is used to detect motor motion. Referring to Figure 14 on page 70, it can be seen that any transition of either of the encoder bits will trigger one of the four edge-triggered, one-shot multivibrators. Once triggered, a multivibrator remains high for 700 microseconds (timing circuitry is shown in detail in Figure 25 on page 112), during all of which time it holds the 8748 interrupt line in the active condition.

For that reason, when the interrupt service routine in the 8748 finishes servicing a motion interrupt, it must wait until the interrupt line becomes inactive before returning from the routine. Otherwise, it could respond to the same interrupt more than once.

The point of that arrangement is to provide an easy distinction between an interrupt that is generated by motor motion, and one that is generated by lack of motor motion. In the latter case,

the interrupt line is held active only until an interrupt acknowledge signal is sent from the 8748; in the former case, the acknowledge signal has no effect on the state of the interrupt line.

MOTOR STOPPED INTERRUPT GENERATION

As shown in Figure 14 on page 70, the time elapsed since the last interrupt is measured by a two-byte clock. (For more details on interrupt clocking, see "Appendix K. Velocity Bookkeeping in the 8748" on page 79.) If the time since the last interrupt exceeds 1/16 second, then motion, if any, is so slow that the motor is assumed stopped. An interrupt must therefore be generated to notify the 8748 of the stopped condition.

That task is performed by the motor-stopped flip-flop, shown in the same figure. If the clock counts up to 1/16 second before a motion interrupt, then the motor stopped flip-flop is set, which holds the interrupt line active. That flip-flop is only cleared when an interrupt-acknowledge signal is sent from the 8748. Note that the interrupt-acknowledge signal also per-

forms the task of enabling the encoder-data-latch output onto the 8748 Port 1.

If a motion interrupt occurs during the time that the motor-stopped flip-flop is set, or while the motor-stopped interrupt is being serviced, that interrupt is not lost. It lasts for 700 microseconds, so it is detected when the motor-stopped subroutine finishes (unlike motion interrupt servicing, motor-stopped servicing does not wait for the interrupt line to become inactive before returning).

For details of those routines, see "The Motor Stopped Subroutine" on page 160 and "The Interrupt Service Subroutine" on page 153.

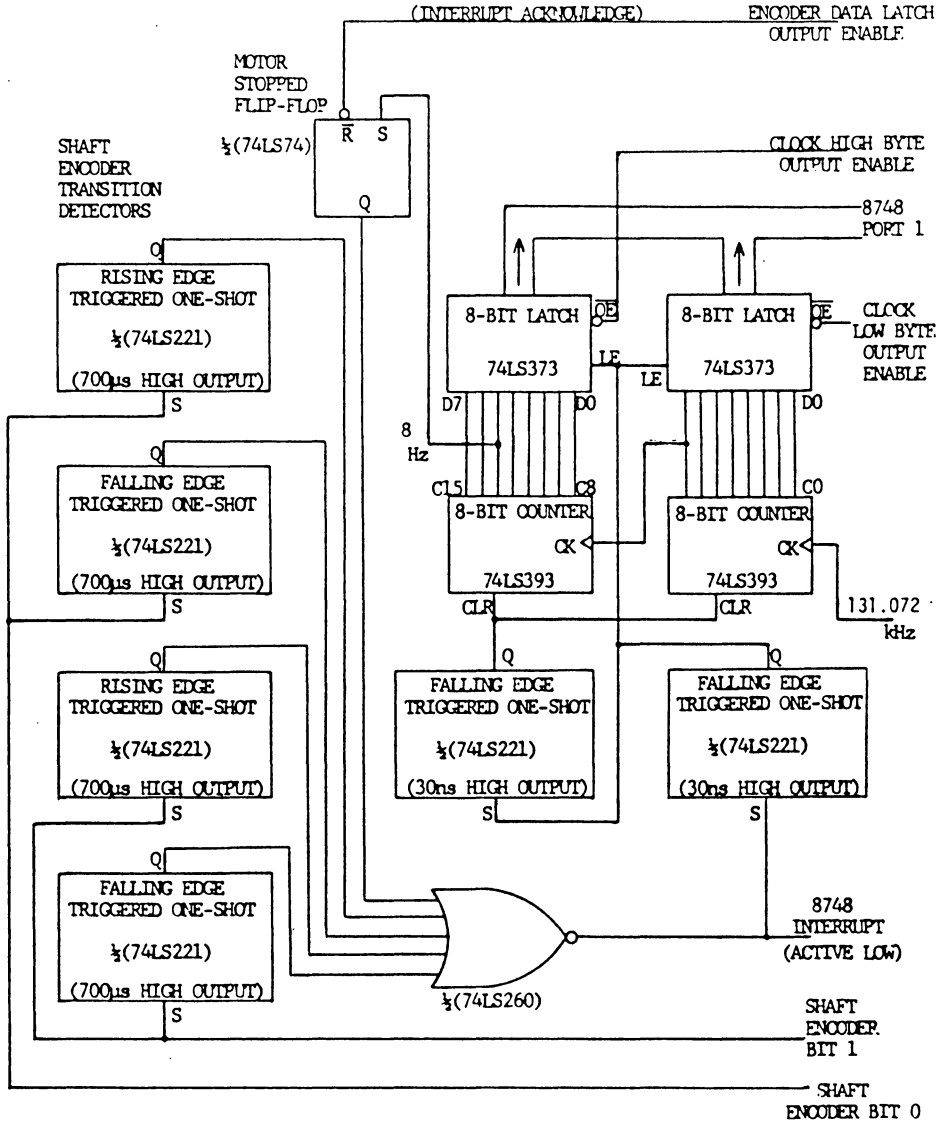

Figure 14. 8748 interrupt generation and clocking.

APPENDIX I. KEEPING TRACK OF DIRECTION

It is quite important to know the current direction of motion, but equally important is to know the direction of motion during the previous interrupt, and how the direction of motion relates to the commanded direction of motion.

As for the direction of motion, quadrature encoding allows a very simple detection scheme. The value of encoder bit 1 is taken 'exclusive-or' with the previous value of encoder bit 0. The result of that operation is zero if the direction is clockwise, and one if the direction is counter-clockwise, and can thus be used as the value for the absolute-direction flag.

At this point in the interrupt-service subroutine the value of the previous absolute-direction flag is still available. The 'exclusive-or' of the present and previous absolute-direction flags yields a one if a reversal has just occurred.

If reversal has just occurred, then the new-motion flag must be set. The reason is, of course, that the motor had to come to a full stop before reversing direction. Also, if the previous-position-(0,0) flag is clear, then the

reversal-since-last-update flag must be complemented. (For more details on the reversal-since-last-update flag, see "The Absolute Position Counter" on page 73.)

The absolute-position flag can now be taken 'exclusive-or' with the sign of the present move command (position error) to yield a value for the relative-direction flag. The relative-direction flag will thus be zero if motion is in the commanded direction, and will be one if motion is opposite the commanded direction.

All the flags mentioned in the foregoing are located in Registers 6 and 7 of 8748 Register Bank 1. For complete details on how the updating is accomplished, refer to "The Interrupt Service Subroutine" on page 153, step 4, and the associated section of the program listing, in the appendix "Listing of the 8748 Assembly Language Program".

APPENDIX J. POSITION BOOKKEEPING IN THE 8748

There are several locations in RAM that keep track of position. Each position data location serves a specific function in the overall task of position control. After each interrupt due to motor motion, the various position data locations must be updated.

THE ABSOLUTE POSITION COUNTER

The absolute position counter in shared RAM holds a signed, two-byte number representing the sum of all moves since the absolute position counter was last set to zero. (Note that the command 'reset absolute position' will set the absolute position counter to zero, as will a hardware reset.)

The number is in the angular measure of encoder disk holes, so it must be updated only when the encoder is in the (0,0) position. That is, on those interrupts occurring at quarter-hole positions that are not (0,0), the absolute position counter is

not updated. It is arbitrary which of the four quarter-hole positions is chosen as the one on which to update the counter; (0,0) was chosen because it is easy to detect. (For a discussion of angular measurement in 'holes', see "Appendix G. Shaft Position Encoding" on page 61.)

Notice that if the direction of motion reverses while the encoder disk is not in the (0,0) position, then the next time the encoder reaches the (0,0) position it would be a mistake to change the number in the absolute position counter, since the encoder disk has merely backtracked to its previous position. For that reason, it is necessary to keep track of reversals which occur when the motor is not in the (0,0) position.

That is the function of the reversal-since-last-update flag. For every reversal occurring in a position other than (0,0), that flag is complemented. That requires knowledge of the previous position, since a reversal is not detected until the position has changed.

That is the function of the present-position-(0,0) and the previous-position-(0,0) flags. When a motion interrupt occurs, the value of the former must be moved to the latter to update it. (Later, during the position-update subroutine, the pres-

ent-position-is-(0,0) flag is updated). So, if a reversal is detected and if the previous-position-(0,0) flag is clear, the reversal-since-last-update flag is complemented.

During the position-update subroutine, the absolute position counter is only updated if the present position is (0,0) and the reversal-since-last-update flag is clear. The updating depends on the absolute direction of motor motion. If the absolute-direction flag is set, then motor motion is counter-clockwise, and the absolute position is decremented. Otherwise, the absolute position is incremented.

THE PRESENT MOVE COMMAND (POSITION ERROR)

The present move data is a signed, two-byte number in internal RAM. For the purpose of the following discussion, it will be called the position error. It is measured in angular units of encoder holes, so (like the absolute position counter) it is only updated when the encoder position is (0,0) and the reversal-since-last-update flag is clear.

Unlike the absolute-position, however, the updating of the position error is dependent upon the direction of motion relative to the commanded direction. If the relative-direction flag is clear, then the motor is moving in the commanded direction, so the position error is decremented (since position error is being reduced). Otherwise, position error is incremented.

The position error can also be altered by issuing a new move command. When the position error is small and there is a new move command in the same direction waiting to execute, the new move command and the position error will be summed to yield a new value for the position error. If the position error is zero, then any new move is brought in immediately upon detection.

It should be added that new move commands are detected only during the interrupt service subroutine, to ensure that a standard measure of time is allowed for the motor to start moving after initial power is applied to it.

THE QUARTER-HOLE TAIL

It was considered desirable to control the position with quarter-hole resolution rather than, as the original controller did, with only full hole resolution. So, while the position counters only register full hole changes in position, the controller is operating at the quarter-hole level.

To reduce storage requirements, the quarter-hole tail was designed to represent only the number of quarter-holes moved since the last interrupt, relative to the commanded direction of motion. So it is, effectively, the low order two bits of the position error.

The quarter-hole tail, unlike the other position data locations, must be updated after every motion interrupt, regardless of reversals, etc. If the relative-direction flag is clear, the quarter-hole tail is decremented since the number of quarter-holes of position error is decreasing. Otherwise, it is incremented. However, every time the position is (0,0), the quarter-hole tail is cleared, forcing its magnitude to be modulo four, as it was designed to be.

The quarter-hole tail serves only one purpose. The Move-Yet? subroutine (see "The Move Yet? Subroutine" on page 162) uses it to detect small position errors whenever there is no large position error.

APPENDIX K. VELOCITY BOOKKEEPING IN THE 8748

MEASURING THE MOTOR SPEED

The motor speed is determined by measuring the time between motion interrupts, and finding the reciprocal of that time. The timing is done by two 8-bit dividers (74LS393s), as shown in Figure 14 on page 70.

Clocking the Interrupt Interval

When the interrupt line becomes active, its transition from high to low triggers a 30-nanosecond pulse from the one-shot multivibrator that latches the value of the clock bytes onto their respective (tri-state) bus buffer latches. As that 30-nanosecond pulse finishes, its falling edge triggers another 30-nanosecond pulse that clears the clock value to zero. In that way, the clock counts only the elapsed time between interrupts.

A detailed schematic of the timing and interrupt circuitry is given in Figure 25 on page 112.

Taking the Reciprocal

The following scheme was devised to simplify the task of determining the reciprocal of the clock time.

The most significant eight bits of the clock count can always be expressed in the form

(1.xxxxxxxx) X 2 ** m

where m is equal to the bit position of the most significant bit, from 0 to 15, and each 'x' can be either one or zero.

The elapsed time is simply the clock count divided by the clock input frequency. There is a distinct advantage to be gained by making the clock input frequency an exact power of 2. The advantage is that the quotient which yields the elapsed time can then be written (in terms of the eight most significant

bits) as

(1.xxxxxxx) X 2 ** (m-n)

where n is the base 2 logarithm of the clock input frequency, 2 ** n, and (m-n) is an integer.

The reciprocal is thus

(1/(1.xxxxxxx)) X 2 ** (n-m)

Finding the reciprocal of any elapsed time is therefore reduced to finding the binary reciprocal of 1.xxxxxxx and shifting the result n-m times. Since there are only 128 different values of the form 1.xxxxxxx, a 128-byte look-up table is used to determine the reciprocal.

Note that the reciprocal satisfies the binary inequality

0.1000000 < 1/1.xxxxxxx <= 1.0000000

However, the numbers in the look-up table are between 128 and 255, to maximize the resolution of the table. Therefore, the reciprocal from the look-up table has been pre-multiplied by

256, or 2 ** 8.

Thus, the equation for velocity, denoting the returned value from the look-up table by LTV, is

$$((LTV) \times 2^{**} -8) \times 2^{**} (n-m)$$
$$= LTV \times 2^{**} (n-m-8)$$

Therefore, the value returned from the look-up table is shifted (n-m-8) times to obtain the velocity value.

It remains to determine a suitable value of 2 ** n, the input clock frequency. The frequency 2 ** 17 was chosen for several reasons. First and foremost, it is fast enough to allow discrimination between speeds of 254 and 255 holes per second, which is slightly faster than any of the motors can move. However, another consideration was that it also produces subharmonics as slow as 2 Hz in the interrupt clock. Initially it was not known what the rate of motor stop interrupt generation should be, so it was deemed useful to have such slow frequencies available.

The value of input clock frequency chosen, then the velocity equation is

```
LTV X 2 ** (9 - m)
```

Note, however that the result is in interrupts per second, which is the same as quarter-holes per second. To get the velocity in holes per second, that result must be divided by four; or, equivalently, multiplied by 2 ** -2. Furthermore, the result is formatted for only seven bits, rather that eight, so the result must be divided once more, by two. Finally, then velocity is found by shifting the look-up table value, LTV, according to

```
LTV X 2 ** (6-m)
```

where m is the bit position of the most significant digit of the clock count, from 0 to 15. Implementation of this scheme is outlined in "The Interrupt Service Subroutine" on page 153, step 5, with assembly language details in the associated section of the appendix "Listing of the 8748 Assembly Language Program".

DETERMINING VELOCITY ERROR

Velocity error is determined very simply by subtracting the current velocity error from the velocity set point. While this actually gives the negative of the velocity error, that number indicates the direction in which power to the motor should be changed. That is, if velocity is larger than the set point, power to the motor should be reduced.

Before subtraction is performed, however, it is necessary to check the sign of the velocity relative to the set point. If the motor is moving in the wrong direction, then velocity is actually negative, relative to the set point. In that case, the magnitude of the velocity is added to the set point to obtain the velocity error.

DETERMINING THE DIRECTION OF ACCELERATION.

The direction of acceleration is determined simply by subtracting the previous velocity from the present velocity. The result is used to determine when to stop integrating the velocity error.

For example, if the velocity exceeds the set point, then the motor power is decreased by subtracting the product of the velocity error and a coefficient specified from VAX, from the present pulsewidth. That effectively integrates the velocity error, over a period of many motion interrupts. However, when the motor velocity begins to decrease towards the set point, there is no need to decrease the power to the motor any further, since the desired change has already begun to occur. (An analogous test is made if the velocity is below the set point.)

That scheme was found to have a stabilizing effect on the motor velocity at all set points.

APPENDIX L. BOOKKEEPING TASKS IN THE INTEL 8751

(HARDWARE) THE MOTOR LINKED LATCH

Occasions may arise when it is not desired (or not possible) to connect all motors to the system. Indeed, the 8751 is capable of facilitating seven motors, while the Rhino has only six. (The additional capability was provided in anticipation of sensor experimentation.) There must be a method for the 8751 to determine which motors are in service.

The function of the motor-linked switches is to provide that information to the 8751 through one of the spare latch inputs. The motor linked information is used in every communication with the 8748's to prevent the 8751 from waiting indefinitely for responses from 8748's that are not operational.

It should be noted that the motor linked latch is read only once, during the initialization routine executed on power-up. So if motors are to be connected or disconnected, the system power must be shut off.

The motor linked latch is shown in block form in Figure 10 on page 51, and detailed connections are shown in Figure 20 on page 107.

(HARDWARE) THE BAUD RATE

The baud rate can be set to either 4800 or 9600. That is done simply by wiring bit zero of the motor-linked latch to +5-volts for 4800 baud, or to ground for 9600 baud. In order to change the baud rate, system power must be turned off and back on again, since the baud rate is set during the initialization routine.

(SOFTWARE) THE 8748 STATUS FLAGS

Several flags are used by the 8751 to keep track of the status of the various linked 8748's and their motors.

Not Ready for Next Move Flags

In normal operation, the not-ready-for-next-move flags of all linked (operational) motors are set. However, if the 8751 receives code from an 8748 indicating that it is ready for the next move, the flag corresponding to that 8748 is cleared. When the condition occurs that all of the flags are clear, the VAX is notified that the system is ready for the next move, and the flags of all linked motors are reset.

Nothing prevents the 8751 from accepting another move command from the VAX when those flags are set, so the program running on the VAX must also keep track of this status to prevent it over-writing prior move commands before they are executed.

Motor Stalled Flags

A stall occurs when a motor has not moved for one second under at least 50% power. In normal operation, the motor-stalled flags are all clear. However, if the 8751 receives code from an 8748 indicating that its motor is stalled, the flag corresponding to that 8748 is set until the VAX clears the stall condition. The stall condition is cleared by VAX with the Clear Remaining Moves (CRM) command.

Hardware Reset Flags

Hardware reset occurs whenever power is turned off and back on. The significance of a hardware reset is that all data in the internal and shared RAMs of the 8748 are lost. In normal operation, the hardware-reset flags are all clear. However, if the 8751 receives code from an 8748 indicating that it has undergone hardware reset, the flag corresponding to that 8748 is set until the VAX is notified of the hardware reset condition.

APPENDIX M. CONTROL BY NON-LINEAR FEEDBACK

While the motor is in motion, the 8748 functions as a velocity controller, and can be viewed as closing a feedback loop where system input is a velocity set point and system output is the motor velocity. Unlike an ordinary feedback loop, however, it can respond to special situations in a most non-linear manner.

The situation is broken down into various cases, or system states, under which different control algorithms are applied.

1. the motor is not moving and the position error is zero
2. the motor is not moving and the position error is not zero
    a. initial power has not yet been applied
    b. the motor did not move under the initial application of power
    c. the motor did not move under further application of power
3. the motor is moving
    a. a move command is being executed
        1) position error is large
        2) position error is small
    b. all move commands completed

1) position error is large

2) position error is small

3) the position error is zero

Each of these states has significance in terms of velocity and position control, and the 8748 software is designed to recognize them and take the appropriate control action.

Note that state 1 is an equilibrium point, and does not require any action.

State 2 is typical of a start-up situation, and is handled by the motor-stopped subroutine, which is outlined in "The Motor Stopped Subroutine" on page 160.

In state 2a, the position error has just been detected. In that case, the starting pulsewidth, specified from the VAX, is applied.

In state 2b, the motor failed to respond to the initial pulsewidth, indicating that the motor is probably working against a load. In that case, the pulsewidth is increased in direct proportion to the velocity set point.

In state 2c, the motor is not moving even under larger amounts of power. Power is increased again, unless the pulsewidth is over 50% and the motor has not moved for one second, when the stall condition is assumed.

The system is in state 3 whenever there is a motion interrupt. The actions taken are outlined in "The Interrupt Service Subroutine" on page 153. Initially (assuming the system is not halted or stalled), the change in pulsewidth is determined strictly by the coefficient and maximum velocity error specified from VAX for the range in which the velocity set point falls, and by the direction of the acceleration, as described in "Appendix N. Determining the Pulsewidth" on page 94. Afterwards, the substate is determined and other changes are made, if necessary.

In state 3a1, the pulsewidth is not changed any further, since the state should be under velocity control only. In state 3a2, if no further move in the same direction follows the move being executed, then the velocity set point is simply ramped down as the position approaches the position set point. If a further move does follow, the state is treated as state 3a1.

In state 3b1, the long overshoot pulsewidth, specified from the

VAX, is applied in the direction that will correct the error. In addition to that, if the motor motion is in the direction of worsening position error, the corrective pulsewidth is increased in direct proportion to the motor speed. This is a very non-linear response, and it is one reason that overshoot is kept so low.

In state 3b2, the response is the same as for state 3b1, except that the short overshoot pulsewidth is applied, rather than the long overshoot pulsewidth. It, too, is specified from the VAX.

In state 3b3, the response should be a braking pulsewidth. However, in the case where the motor is moving relatively slowly, no response is required. The scheme chosen, and it works well, is to subtract one from the motor speed and use the result as the braking pulsewidth. This works particularly well in avoiding 'jitter' at the final position, since the value of speed assigned just after direction reverses is one; after subtracting one from that speed, zero pulsewidth is applied as braking.

APPENDIX N. DETERMINING THE PULSEWIDTH

The duty cycle of the rectangular pulse train supplying power to the motor is directly proportional to the pulsewidth written to the pulsewidth modulator. (For details of the pulsewidth modulator, see "Appendix O. The Pulsewidth Modulator" on page 98.) This appendix explains how the value of the pulsewidth is obtained under normal velocity feedback control.

When a position error is initially detected, the value which the user has specified as the 'starting pulsewidth' is used as the pulsewidth, and the sign of the position error is used as the pulsewidth sign. After this initial value of pulsewidth is applied to the motor, one of two things will occur: either the motor will move or it will not.

If the motor does not move, then the motor stopped subroutine will detect that exactly one-sixteenth of a second has passed since initial power was applied, and will increment the pulsewidth by the amount of the velocity set point, under the assumption that the lack of response indicates the motor is working against a heavy load. If the motor fails to move after that, then on each successive motor-stopped interrupt, the pulsew-

idth is incremented by the amount of the starting pulsewidth, plus the number of interrupts since power was first applied. That process will continue until either the motor moves or the pulsewidth exceeds 50% and the motor has not moved for more than one second under applied power. (The latter case is considered as the stall condition.)

If the motor does move, then on the first motion interrupt after power is applied, there is no way to determine the speed because the clock value is meaningless. The velocity error is set to be zero, since there is no way to determine a good value for it until another move interrupt yields a valid clock count.

On successive motion interrupts, the velocity is determined using the scheme given in "Appendix K. Velocity Bookkeeping in the 8748" on page 79. The velocity error is the velocity minus the velocity set point. The negative of the velocity error is used in the following way to obtain the pulsewidth increment:

1. The error is compared to the user-specified maximum error for the range in which the velocity set point falls, and is set equal to the maximum if it exceeds it (sign is preserved).
2. If the sign of the acceleration indicates that the velocity

is approaching the set point, the velocity error is set to zero.

3. The velocity error is multiplied by the user-specified coefficient for the range in which the velocity set point falls, yielding the pulsewidth increment.
4. If the increment exceeds a magnitude of 127, it is set equal to 127.
5. The increment is added to the pulsewidth.
6. If the new pulsewidth is negative, it is two's complemented and the pulsewidth sign is also complemented.
7. If the new pulsewidth is greater than 127, it is set equal to 127.
8. The user-specified hyteresis value is added to the magnitude of the pulsewidth.
9. If the result exceeds 127, it is set equal to 127.
10. The new pulsewidth and sign are written to the pulsewidth modulator, and to their internal 8748 RAM locations.

Note that when the pulsewidth sign changes, a pulsewidth of zero must be written to the pulsewidth modulator for about 30 microseconds to allow the excited optical isolator and power transistor pair to turn off before exciting the other pair. If both transistors were on at the same time, they would provide a low resistance path from the +12-volt supply to the -12-volt

supply, and would be burned out very quickly by excessive current (refer to Figure 26 on page 113).

The foregoing scheme for updating the pulsewidth does not hold for the special cases, such as overshoot and undershoot. They are explained in "Appendix M. Control by Non-Linear Feedback" on page 90.

APPENDIX O. THE PULSEWIDTH MODULATOR

The pulsewidth modulator is based on a very simple scheme. Referring to Figure 15 on page 100, the pulsewidth (duty cycle) is specified by a number from 0 to 127, which is latched into the pulsewidth latch from the 8748.

That magnitude is constantly compared to the magnitude of a counter which counts repetitively from 0 to 127. As long as the value in the latch exceeds the increasing number in the counter, power is applied to the motor. When the counter value equals or exceeds the pulsewidth latch value, power to the motor is cutoff.

The result is a rectangular wave, or pulse train, of power to the motor with constant period, P, which is the time it takes the counter to count from 0 to 127 (about 2 ** -8 second).

The rectangular output of the 7-bit comparison circuit is gated to one of two optical isolators by the pulsewidth sign bit, which is the eighth bit of the pulsewidth data, latched from the 8748. Each optical isolator is connected so that when it is turned on, biasing is applied to the base of the associated

power transistor, driving that transistor into saturation.

The power transistors are a complementary pair, with the PNP TIP-125 providing power for clockwise motor motion and the NPN TIP-120 providing power for counterclockwise motion.

Complete details of the pulsewidth modulator are presented in the schematic diagram, Figure 26 on page 113.

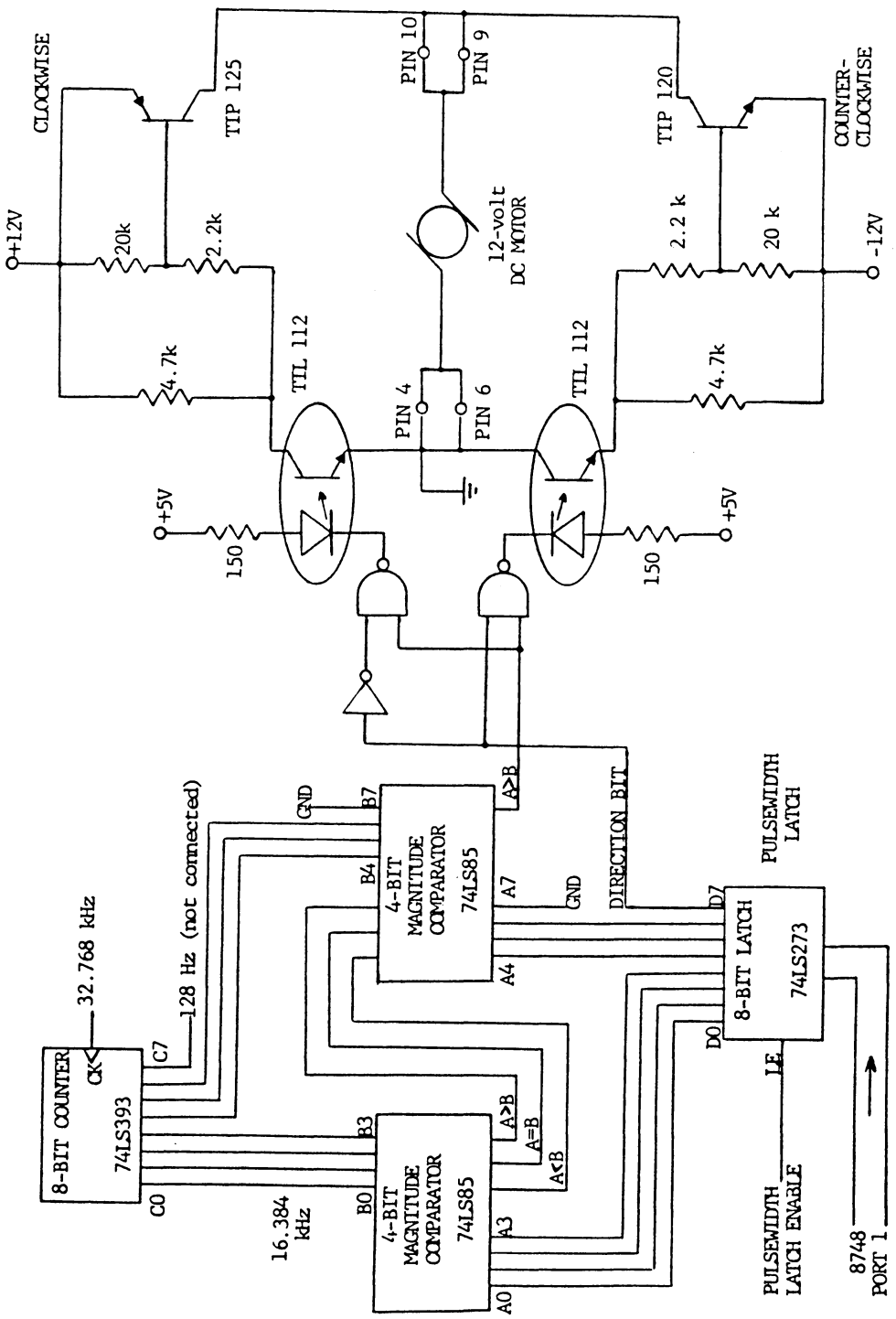

Figure 15. Block diagram of the pulsewidth modulator.

APPENDIX P. ALLOCATION OF RANDOM ACCESS MEMORY

The 8748 RAM, aside from being somewhat small, has a fairly difficult access scheme. Most of it can only be addressed indirectly, which quickly increases the number of program steps in even the simplest procedures. The 8748 program was written 'around' the RAM access problem. That accounts for the peculiar arrangement of the RAM allocation in the 8748, which is shown in Figure 16 on page 102.

The shared RAM allocations were made based on the same considerations, namely, how to allow access by the 8748 from its various routines in the smallest number of program steps (and, thus, the shortest amount of time as well). The result is that the shared RAM allocation is nearly identical to that of the 8748 internal RAM. Allocation of the shared RAM is shown in Figure 17 on page 103.

The 8751, having far more flexible addressing capability than the 8748, was open to a more logical allocation of internal RAM. That allocation is shown in Figure 18 on page 104.

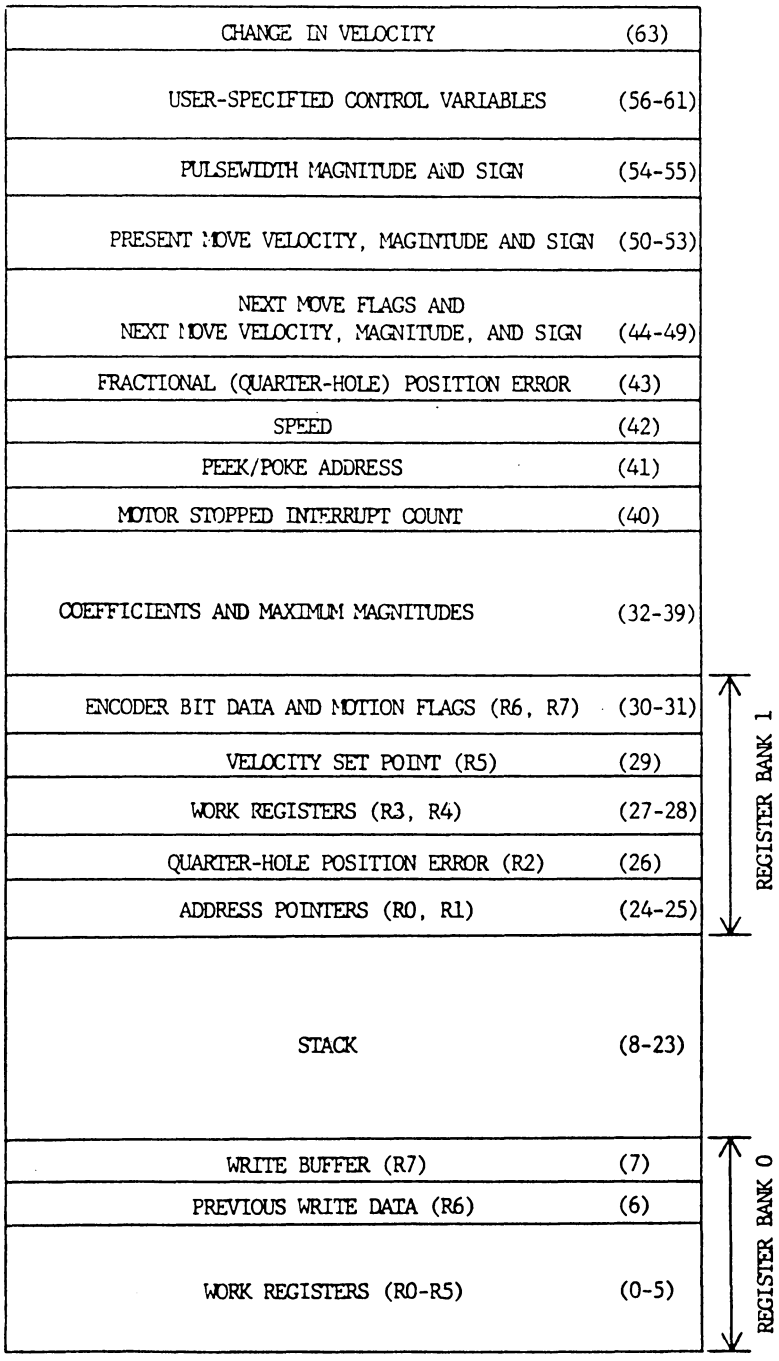

Figure 16. Allocation of 8748 internal RAM.

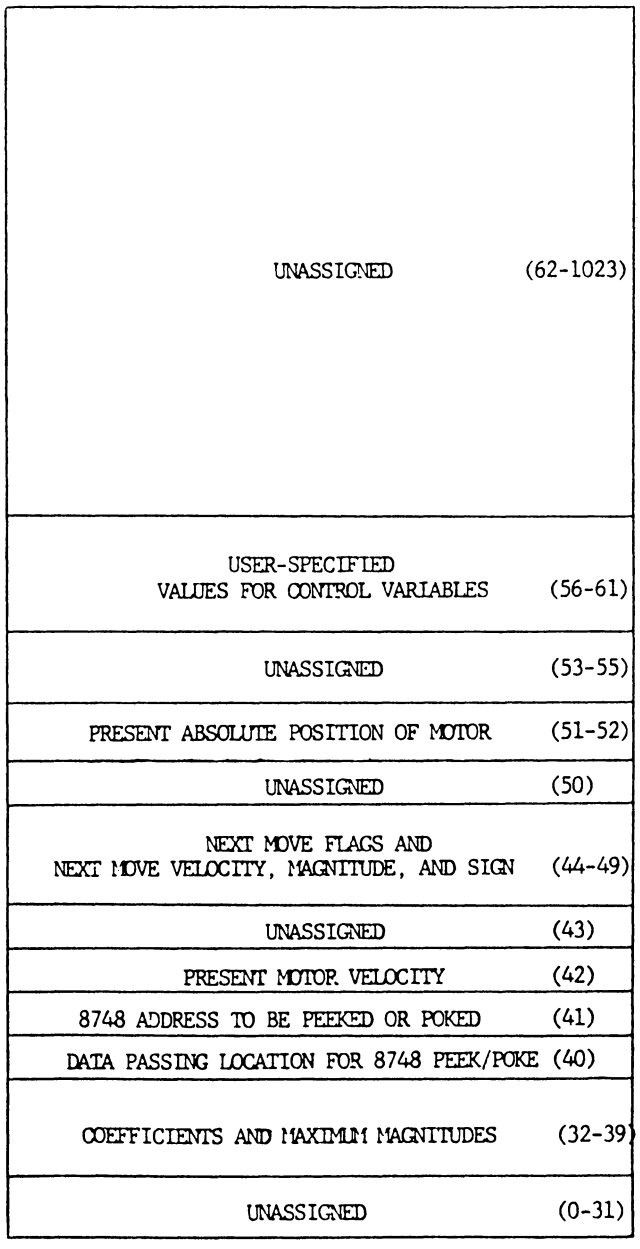

Figure 17. Allocation of shared RAM.

STACK	(80-127)
WRITE DATA BUFFER (TO VAX)	(64-79)
VAX COMMUNICATION FLAGS AND DATA	(56-63)
PREVIOUS COMMAND DATA	(48-55)
COMMAND DATA	(44-47)
RHINO CLOCK	(42-43)
8748 ADDRESSING DATA	(36-41)
8748 STATUS FLAGS	(32-35)
READ DATA BUFFER (FROM VAX)	(16-31)
REGISTER BANK 1	(8-15)
REGISTER BANK 0	(0-7)

Figure 18. Allocation of 8751 internal RAM.

APPENDIX Q. SYSTEM SCHEMATIC DIAGRAMS.

The complete system schematics are given in this appendix. An attempt was made to have the schematics correspond one-to-one with the block diagrams in the other appendices. This was managed for the most part, but the major exceptions are the schematics in Figure 20 on page 107 and Figure 22 on page 109, which, taken together, correspond to Figure 9 on page 50 and Figure 10 on page 51.

Also, there is no block diagram corresponding to Figure 19 on page 106, or Figure 27 on page 114.

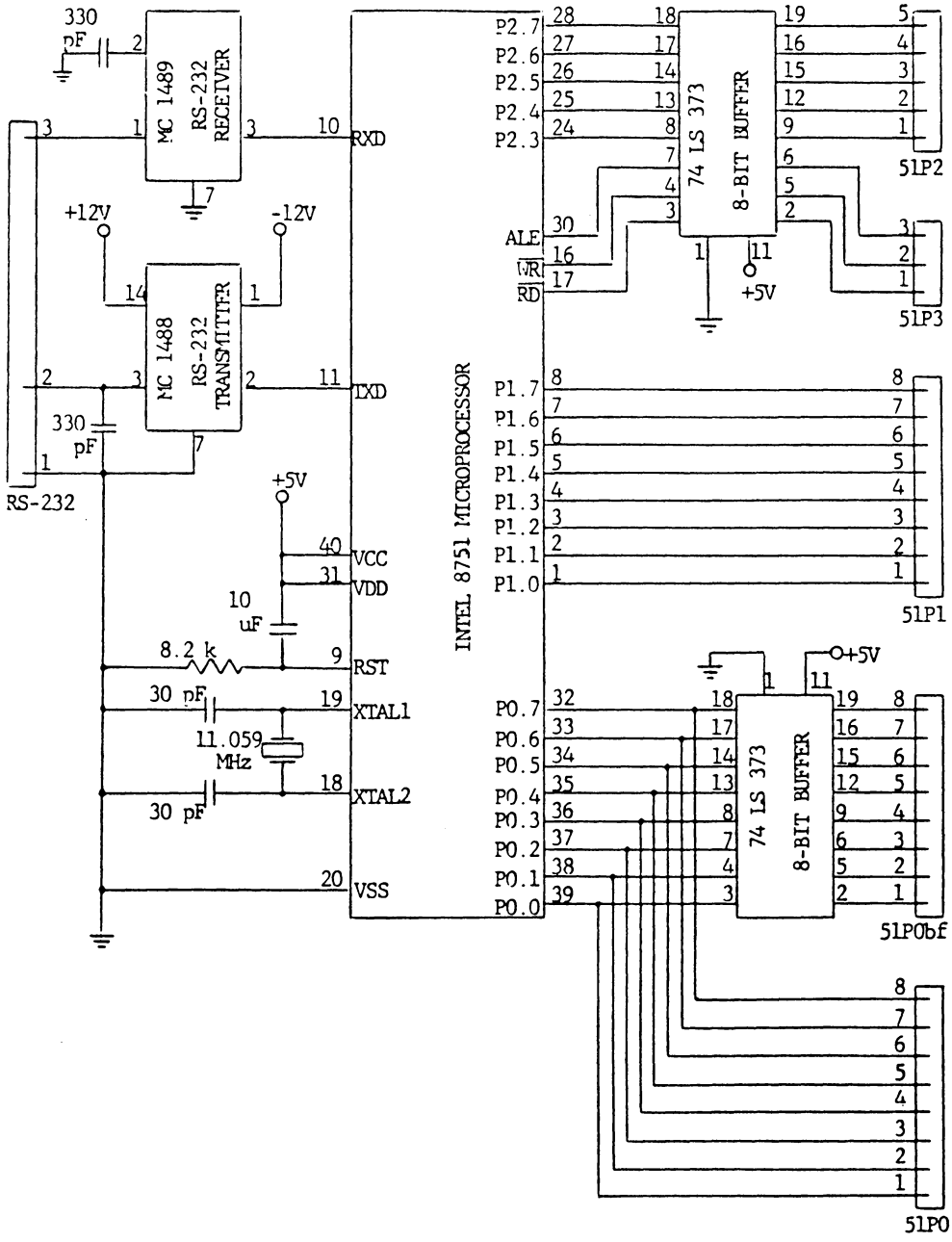

Figure 19. Connections to the Intel 8751 microprocessor.

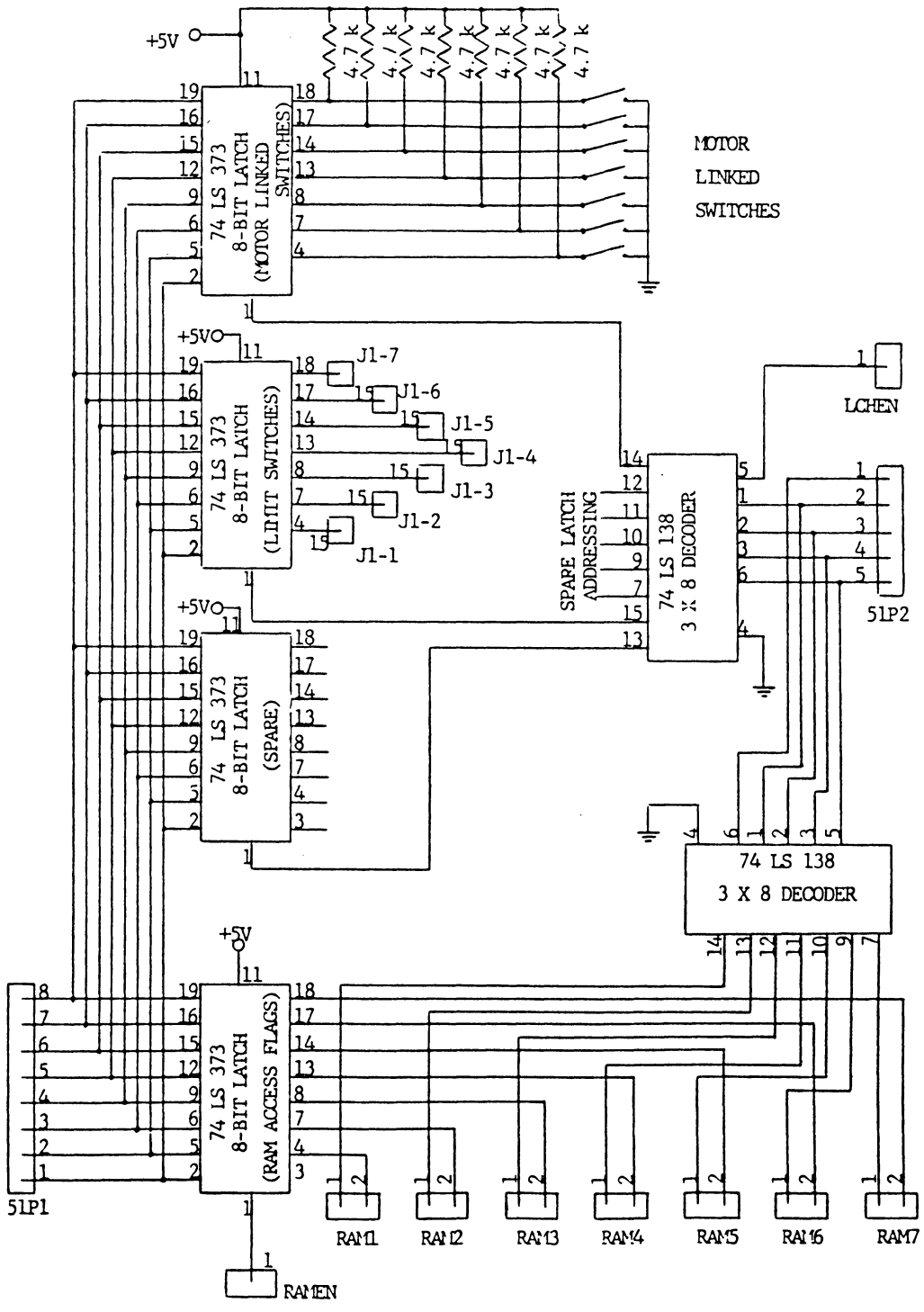

Figure 20. Connections for 8751 shared RAM enables, flag latches, and extra latches.

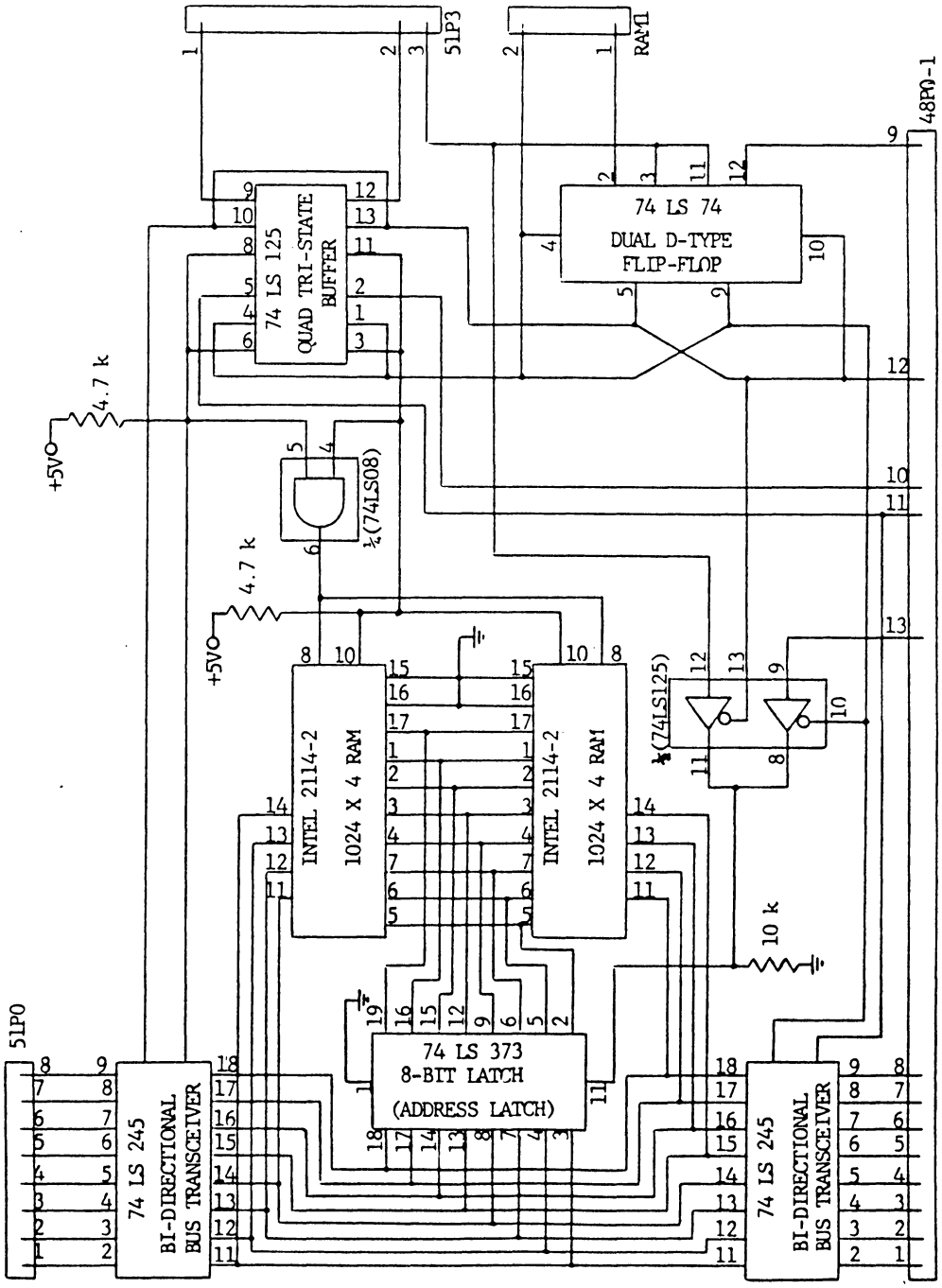

Figure 21. Connections to the shared RAM.

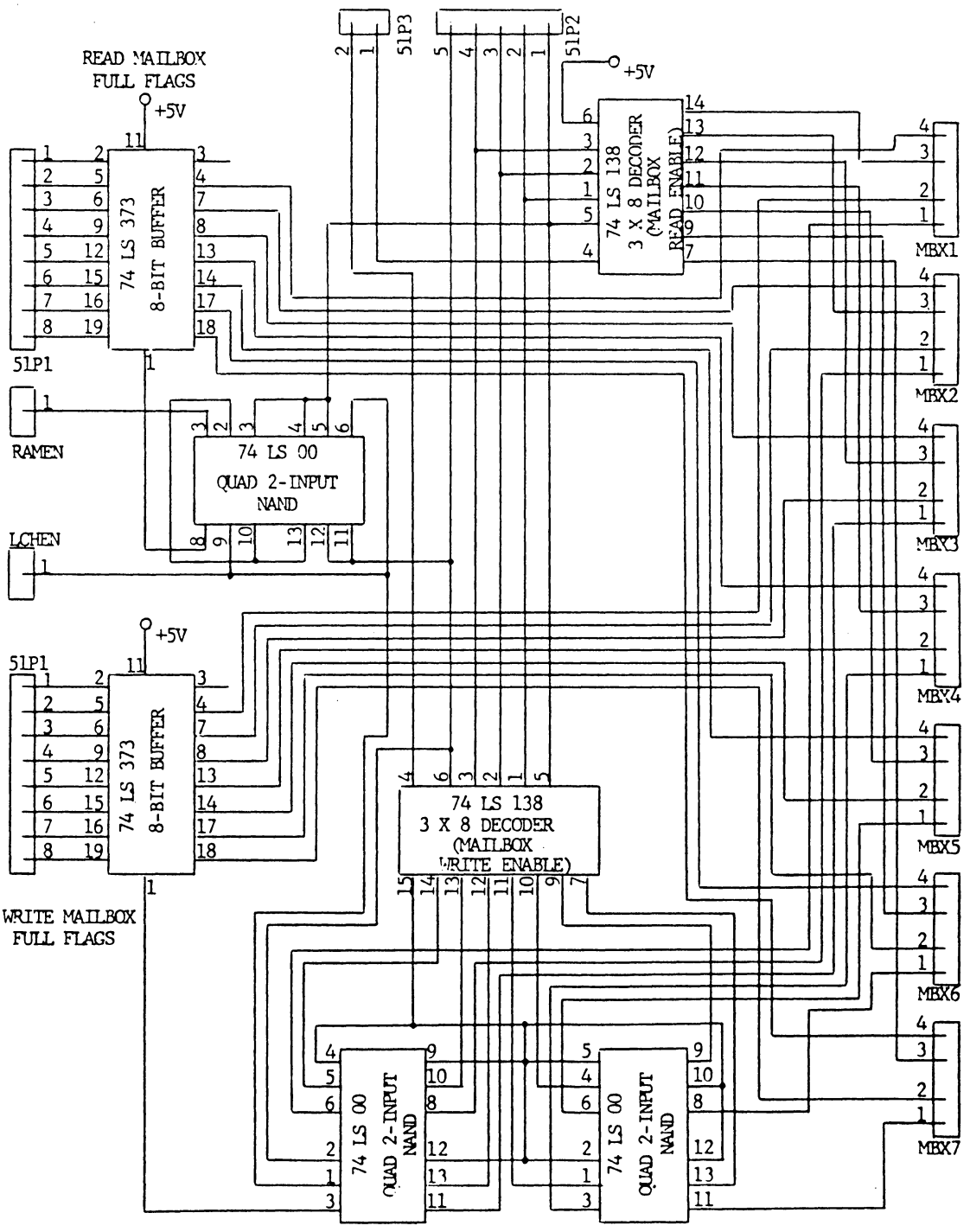

Figure 22. Connections for the 8751 mailbox enables and flag latches.

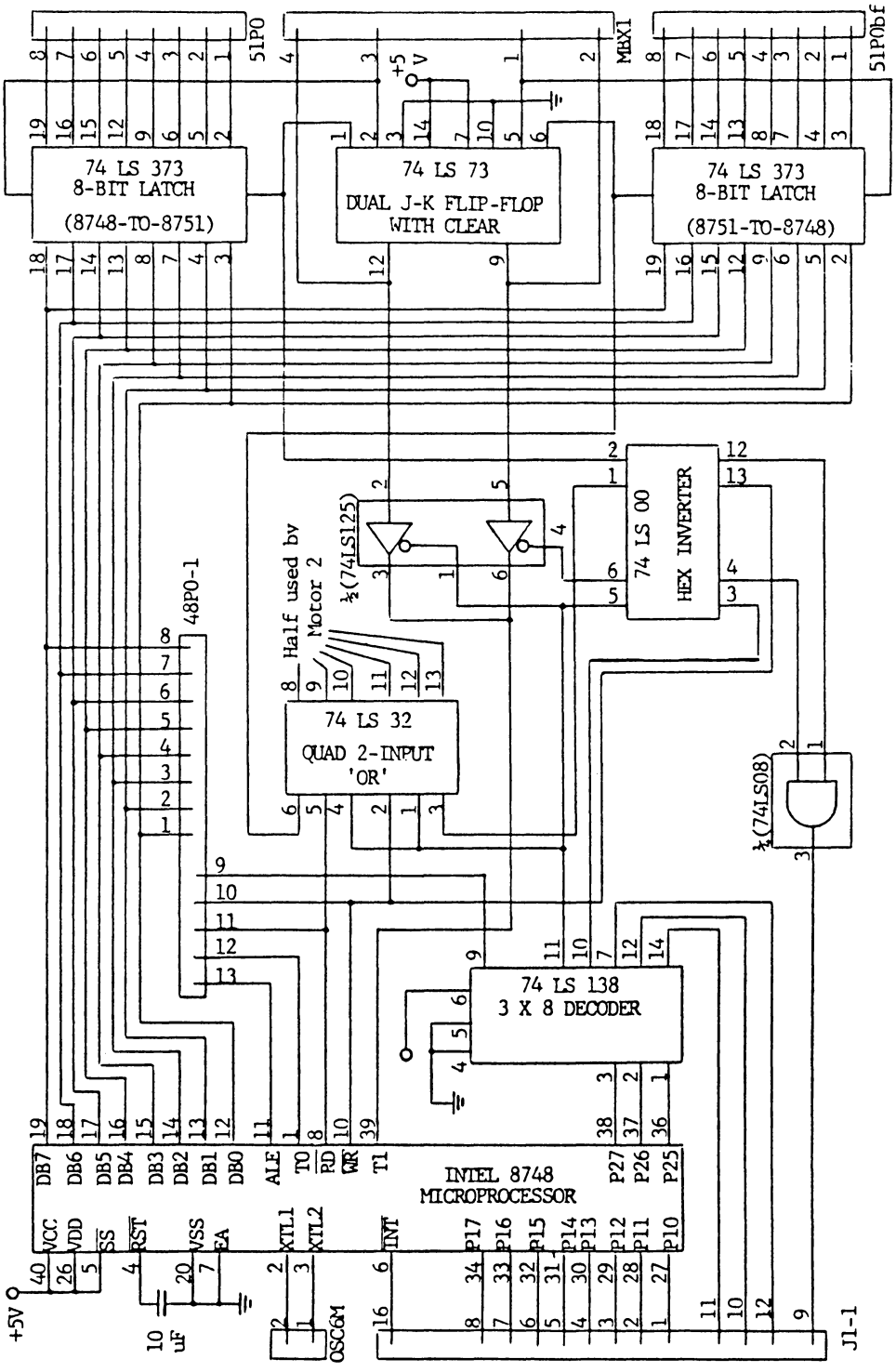

Figure 23. Connections to the Intel 8748, its mailbox, and device selection.

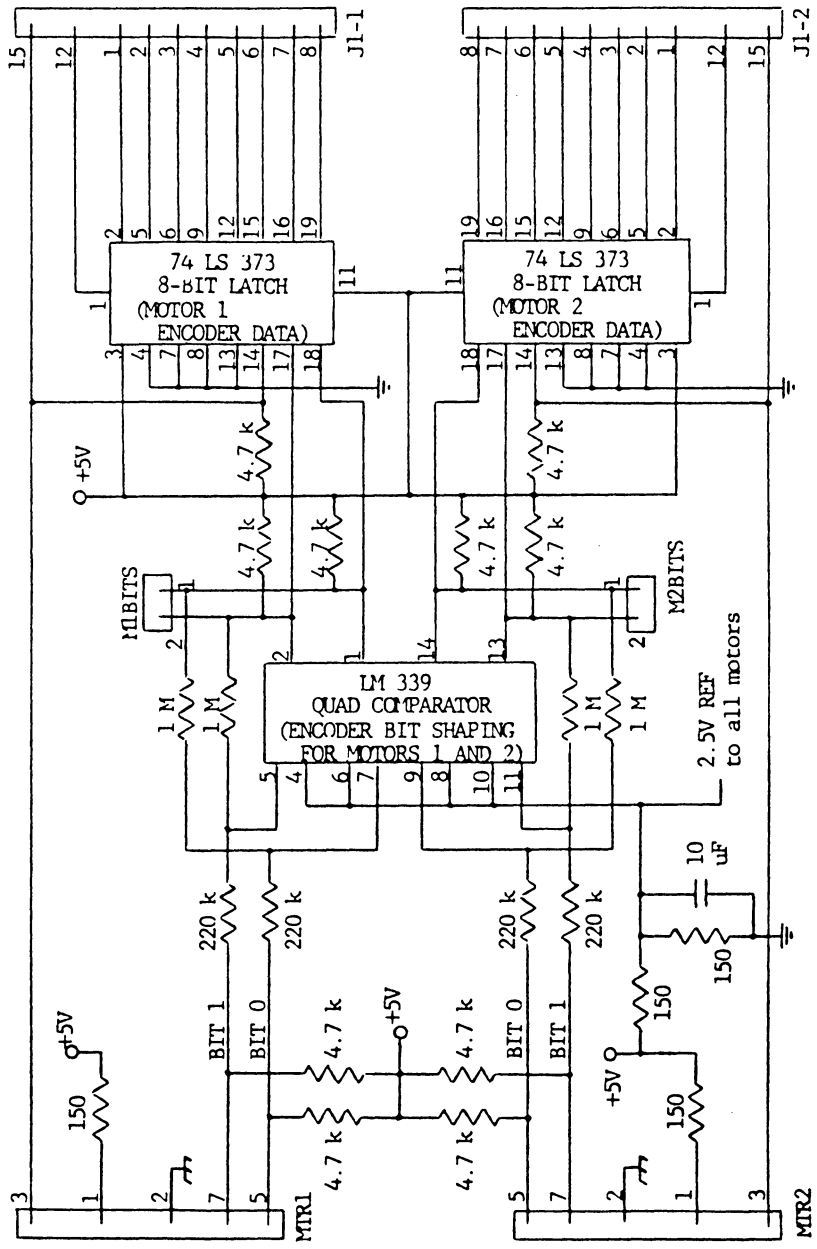

Figure 24. Connections for encoder bit shaping and data latching.

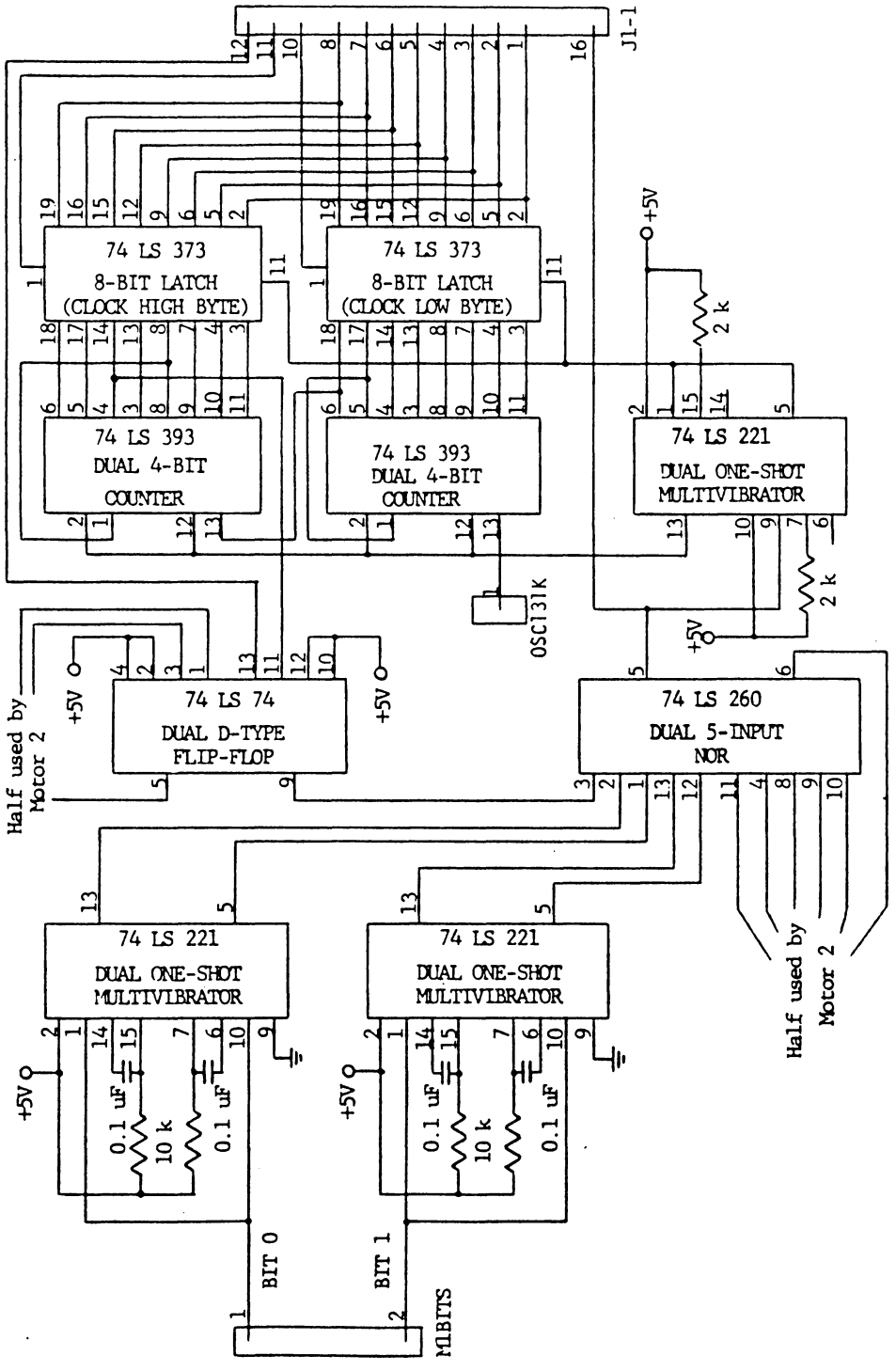

Figure 25. Connections for interrupt generation and interrupt clocking.

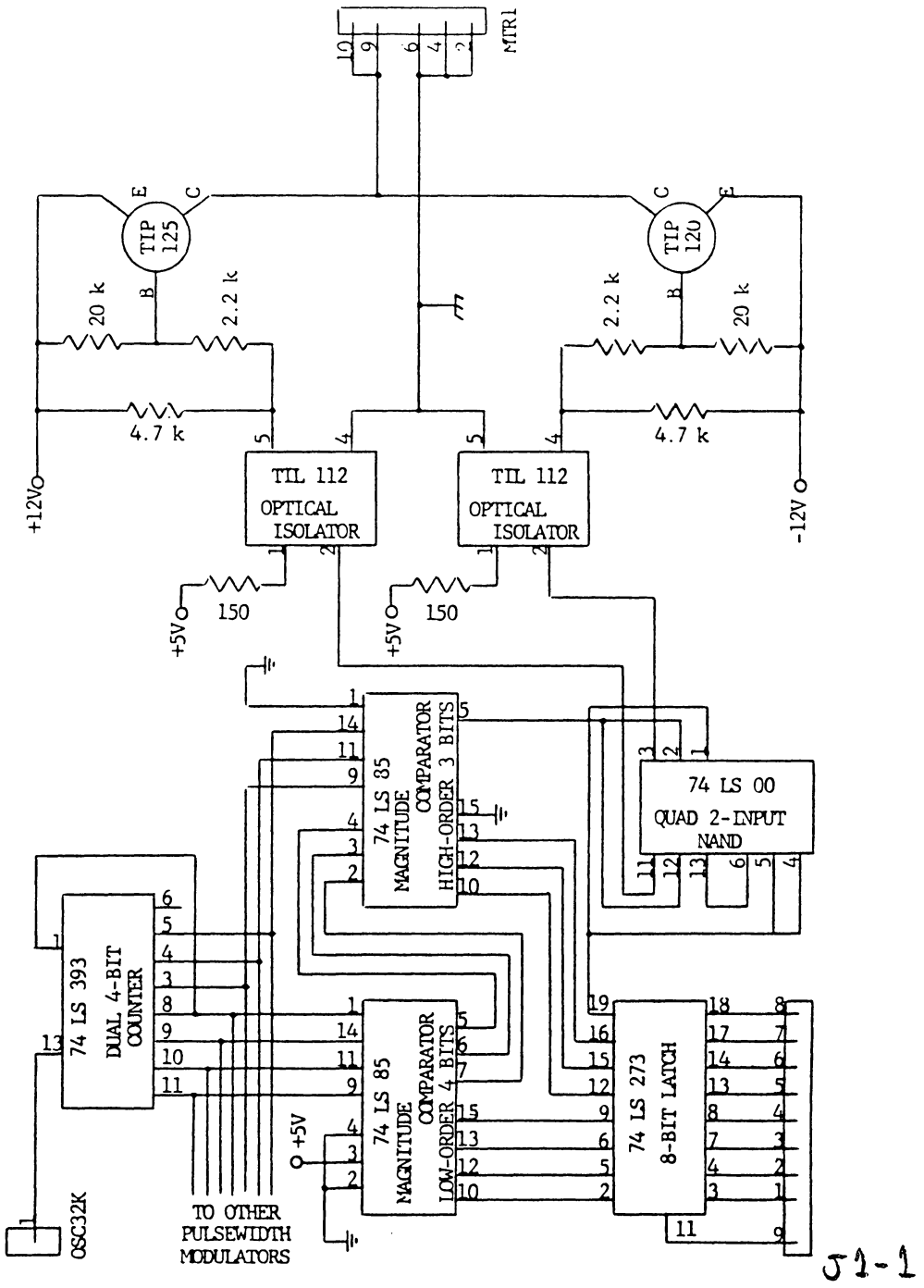

Figure 26. Connections for the Motor 1 pulsewidth modulator.

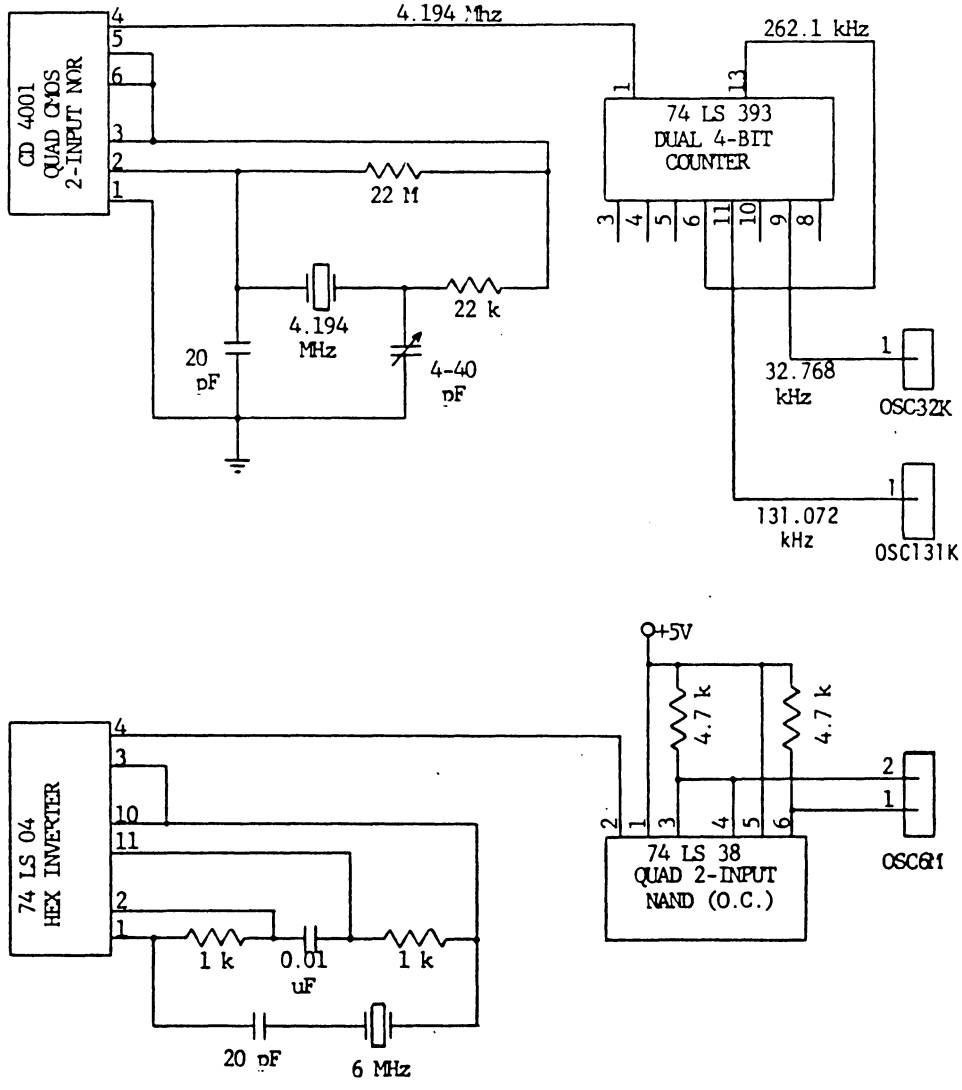

Figure 27. Oscillators for the 8748's (6 MHz) and interrupt clock (32 kHz).

APPENDIX R. A SPECIAL CASE: THE GRIPPER MOTOR

The program for gripper motor control differs somewhat from the program for the other motors. The main difference is that velocity control is disabled. Velocity control is both unnecessary and undesirable for the gripper.

If the gripper motor had velocity control, then when it closed on an object more and more power would be applied in an attempt to maintain the set point velocity. That might damage the object. A more appropriate program would be set up to sense the moment of contact and apply some designated amount of gripping power to the motor.

The point of contact is sensed by a method similar to that used to sense motor stalling in the other program. If the motor does not move for more than 1/8 second under application of the specified closing power, then it is assumed that the jaws are closed on some object (or completely closed). In that case, the user-specified gripping power is applied to the motor.

If the motor moves beyond a small amount after contact is made, power is discontinued. It is assumed that the motor moves

after making contact only if the object is deforming or slipping, or if contact has been lost. The discontinuation of power is designed to prevent damage.

In the program for the gripper, the user specifies several parameters.

- the power to be applied during the opening or closing of the gripper 'jaw' (0 to 63),
- the hysteresis (minimum non-zero) power during opening or closing,
- the gripping power (applied after contact is made) (0 to 63), and
- the overshoot (undershoot) correction power.

Another difference is that all position data locations are kept in terms of quarter-holes. That simplifies the bookkeeping tasks and maximizes position resolution and repeatability.

When a move is commanded, the gripper motor will move all the way to the new position or to a point of contact, whichever occurs first. In either case, the position error is set to zero where motion stops. Thus, a very long move can be specified if it is desired to have the jaws close on an object, and the part

of the move left after contact is made is simply deleted.

The program used is not specified in detail due to the fact that it is identical in most respects to the other program, and due to space limitations.

APPENDIX S. SAMPLE VAX PROGRAM (PASCAL)

This program was created specifically to aid in the development and debugging of the new controller. It is useful in making the robot run, and for helping anyone interested to see one way the bookkeeping and communication tasks can be accomplished. It is written in PASCAL primarily because of the availability of the 'case' statement in that language.

Some people claim that PASCAL is self-documenting. Indeed, the freedom of choice in naming procedures, and the unrestricted use of indentation are tremendous aids in making a program understandable. For that reason, a detailed outline was not considered necessary. A brief outline is given at the end of this appendix, and the complete program listing is given in "Appendix V. Listing of the Sample Program on VAX" on page 170.

In order to fully understand this program, one must study the communication protocol and the communication subroutines in the 8751 assembly language program thoroughly. In fact, the sole function of this program is to translate the user's commands into a format that fits the protocol, and to provide the inverse operation on communication and data from the 8751.

Some would say it makes the Rhino robot user-friendly.

Before using this program, one would do well to understand the function of the following command entries:

- '?' : This command causes the menu of valid commands to be displayed to the user, so it acts like a help command.
- 'x' : This command acts like an escape character. It is followed immediately on the command line by a number from 0 to 255 that will be sent directly to the Intel 8751. Essentially, 'x' disables all the prompting and formatting procedures of the program and allows the user to send codes directly to the 8751.
- 'f' : This command allows the user to change the value of three Boolean flags used in the program:
  - The auto-respond flag is the complement of the 'x' command. If it is cleared, all of the automatic response features of the program are disabled so codes and data from the 8751 are displayed exactly as they are received, with no translation or formatting, and no prompting of the 8751.
  - The ready flag is set whenever a 'ready for next move' code (RDYNXT) is received, and is cleared whenever a 'move' command (MOV) is transmitted. A move command

will not be processed if this flag is not set. It is initially set.

- The go flag is set whenever the 'go' command (AGO) is transmitted, and is cleared whenever the 'halt' command (HLT) or the 'motor stalled' command (MTRSTL) is received. It is initially cleared.

- 'z' : This command causes program execution to end.

When a command mnemonic is entered (only the first letter is required), the program will prompt the user for all the data that must accompany the command. The entire communication sequence, including formatting of the data, is then carried out according to the protocol for the particular command, with no further effort on the part of the user.

If the 'auto-respond' flag is set, then all communications originating in the 8751 are also handled automatically. Notice of the communication (and its meaning) is displayed to the user.

Thus, the program allows the user to control the Rhino robot from a keyboard with simple command mnemonics and data entries.

A brief outline of the sample VAX program follows.

Variables and Constants

1. Define the nonstandard types for variables.

2. Declare the program variables and their types.

3. Declare the program constants.

External Procedures

1. The Rhino Read procedure looks for 8-bit input from both the Rhino (Intel 8751) and the keyboard.

2. The Rhino Write procedure writes 8-bit codes to the Rhino (Intel 8751).

Internal Procedures

1. The Read Two RS/232 Devices procedure assigns the correct device address to the Rhino and calls the external 'rhino read' procedure.

2. The Get Response procedure waits for an expected code from the Rhino, and interprets any code arriving before the

expected code.

3. The Get Data procedure prompts the Rhino for data and waits for the appropriate number of data bytes to be received (it also formats the received data).

4. The Get Screen Data procedure prompts the user for the appropriate data to accompany to command entered by the user, and formats them for transmission to the Intel 8751.

5. The Get All Data procedure sends a command to the Rhino which will elicit data, then uses the Get Data procedure to properly handle the data when the Rhino has it ready to transmit.

6. The Display 8751 Data procedure is used to display data returned from an 8751 peek command.

7. The Display 8748 Data procedure is used to display data returned from an 8748 peek command.

8. The Display Shared RAM Data procedure is used to display data returned from a shared ram peek command.

9. The Display Velocities procedure is used to display data returned from a velocity request.

10. The Display Positions procedure is used to display data returned from a position request.

11. The Send Command Get Echo procedure is used to send commands to the 8751 when the only expected response is an 'echo' of the same code (for example, the 'halt' command).

12. The Send 8748 Data procedure is used to send the last two bytes of data for an 8748 'poke' command.

13. The Send Moves procedure is used to send the move and velocity data to the 8751 for a 'move' command.

14. The Load Coefficients procedure is used to send the data accompanying a 'coefficient' command to the 8751.

15. The Enable C-O-I procedure is used to send the command code and data for the 'enable command-on-interrupt' command.

16. The Code From 8751 procedure is used to decode and respond to incoming code from the Intel 8751.

17. The Keyboard Command procedure is used to interpret the command entered by the user from the keyboard, and to take appropriate action to implement that command.

18. The User Help procedure is used to display to the user the menu of valid commands that can be entered from the keyboard.

19. The Change Flags procedure allows the user to alter the values (TRUE or FALSE) of the three Boolean flags in the VAX program from the keyboard.

The Main Program

1. Prompt for command and data input from the user, and call the appropriate procedures based on the input.

2. Monitor input from the Rhino (Intel 8751 serial port), indicating to the user when it occurs, what it means, and what actions have been or should be taken in response.

APPENDIX T. 8751 ASSEMBLY LANGUAGE PROGRAM OUTLINE

THE INITIALIZATION ROUTINE

The initialization routine is executed on power up.

1. Put the 'no operation' code in all mailboxes.
2. Clear internal RAM locations zero through 127.
3. Put the data byte from the 'motor linked' latch in the motor link location in internal RAM.
4. Wait for a 'hardware reset' code from every linked motor.
5. Initialize the stack pointer to just below the bottom of the stack area of internal RAM.
6. Set up the serial port for communication with VAX.
   a. Set the serial port mode for a ten-bit frame with no parity bit (this is called Mode 1).
   b. Set up Timer 1 to generate 9600 baud.
   c. Clear the serial port transmit and receive interrupt flags.
   d. Set the serial port interrupt priority to the highest level.
7. Put the 'hardware reset' code into the serial port write buffer.

8. Initialize the Rhino Clock to unity (this clock counts the number of times external interrupt zero occurs).
9. Set interrupt zero to the positive-edge-triggered mode (this interrupt increments the Rhino Clock and causes execution of the command-on-interrupt code).
10. Enable the serial port interrupt and interrupt zero.
11. Start the main routine.

8751 MAIN PROGRAM

The main routine monitors communication from the 8748 mailboxes and the VAX, and responds appropriately to commands from any of them.

1. If any 'read mailbox full' flag is set, call the respond-to-8748 subroutine.
2. Check the various **8748 status flags**.
   a. If the peek data ready flag of every linked motor is set, then all the peek data is ready to send to the VAX.
      1) Clear the peek data ready flags.
      2) Read 1 byte of peek data in from each of the seven shared RAM's to the write buffer, with the Motor 1

data at the bottom of the buffer.

3) Above the peek data in the write buffer put the address from which the data was obtained in the 8748's.

4) Above the address put the low and high byte of the Rhino Clock.

5) Finally, above the clock data, put the '8748 peek' code to indicate to VAX which peek command is being answered.

6) Put the 'peek data ready' code in the response code buffer.

7) Load the number 11 as the pending number of data bytes to transmit (i.e., into NTBUFF).

8) Call the write-to-VAX subroutine.

9) Restart the main routine.

b. If any 8748 hardware reset flag is set, then several steps are taken.

1) Clear the 8748 hardware reset flags.

2) Put the 'hardware reset' code in the response code buffer.

3) Call the write-to-VAX subroutine.

4) Restart the main routine.

c. If any 8748 motor stalled flag is set, then several steps are taken.

1) If the 'new stall' flag is set, then a stall has just occurred.
   a) Reset the 'new stall' flag.
   b) Put the 'motor stalled' code in the response code buffer.
   c) Call the write-to-VAX subroutine.
2) If the VAX command is 'no operation' (i.e., cleared), then move in the contents of the read command buffer to the VAX command location, and clear the buffer.
3) The VAX command must be executable during a stall.
   a) If the command is either 'go' or 'move', then it must not be executed, since at least one motor is stalled.
      i. Replace the VAX command by the 'halt' command.
      ii. Put the 'motor stalled' code in the response code buffer.
      iii. Call the write-to-VAX subroutine.
   b) Call the decode-VAX-command subroutine.
4) If no data is to be transmitted or received (that is, if NT, NTBUFF, NR, and NRBUFF are zero), then check for an interrupt command.
   a) Move the contents of the interrupt command

buffer to the VAX command location, and clear the buffer.

b) Call the decode-VAX-command subroutine.

5) Restart the main routine.

d. If the not ready for next move flags are all clear, then the system is ready for the next move.

1) Reset the 'not ready for next move' flags of all linked motors.

2) Put the 'ready for next move' code in the serial port write buffer.

3) Call the write-to-VAX subroutine.

3. If no transmit data remains in the write buffer (that is, if NT is zero) then check for a VAX command.

a. If the VAX command is 'no operation', then move the contents of the read command buffer to the VAX command location, and clear the buffer.

b. Call the decode-VAX-command subroutine.

4. If there is no data to be transmitted or received (that is, if NT, NTBUFF, NR, and NRBUFF are zero), then check for an interrupt command.

a. Move the contents of the interrupt command buffer into the VAX command location, and clear the buffer.

b. Call the decode-VAX-command subroutine.

5. Restart the main routine.

THE RESPOND TO 8748 SUBROUTINE

The respond-to-8748-mailbox subroutine checks each motor's read mailbox individually, starting with motor seven.

1. If the motor's 'read mailbox full' flag is set, then enable the motor's read mailbox.
    a. Read in the command code from the 8748 read mailbox.
    b. Call the decode-8748-command subroutine.
2. Return from the respond-to-8748-mailbox subroutine.

THE DECODE 8748 COMMAND SUBROUTINE

The decode-8748-command subroutine determines the appropriate response to a command from an Intel 8748.

1. If the command is not 'no operation', then several steps are taken.
    a. Enable the motor's write mailbox.
    b. If the code is not valid, request a resend of the command.
    c. If the code is valid, then decode the command.

1) If the command is send previous code over again, then put the contents of the motor's previous code location into the motor's write mailbox, and clear the previous code location.

2) If the command is motor stalled, then several steps are taken.

   a) Set the motor's 'stall' flag.

   b) Set the 'new stall' flag.

   c) Put the 'halt' code in all motors' write mailboxes.

3) If the command is hardware reset, then several steps are taken.

   a) Set the motor's 'hardware reset' flag.

   b) Put the 'halt' code in all motors' write mailboxes.

4) If the command is ready for next move, then clear the motor's 'not ready for next move' flag.

5) If command is peek data ready, then set the motor's 'peek data ready' flag.

2. Return from the decode-8748-command subroutine.

THE WRITE TO VAX SUBROUTINE

The write-to-VAX subroutine transmits the command in the response code buffer.

1. If the 'transmitter idle' flag is set, then call the serial-output subroutine.
2. If the 'transmitter idle' flag is clear, wait for one of the following two occurrences.
   a. If the 'transmitter idle' flag becomes set, then call the serial-output subroutine.
   b. If the response code buffer is cleared, then the response code was transmitted from the serial-port-interrupt-service subroutine.
3. Return from the write-to-VAX subroutine.

THE SERIAL OUTPUT SUBROUTINE

The serial-output subroutine puts the appropriate data byte into the transmit serial port buffer.

1. Clear the 'transmitter idle' flag.

2. If there is data to transmit (that is, if NT is not zero), then several steps are taken.
   a. Compute the location of the data byte to transmit.
   b. Move the byte from that location to the serial port transmit buffer.
   c. Decrement the data byte counter, NT.
   d. Set the previous write to VAX location equal to one to indicate 'no operation'.
   e. Return from the serial-output subroutine.
3. If there is no data to transmit, then check for a command to transmit.
   a. If the write code buffer is not zero, then its contents must be transmitted.
   b. If the write code buffer is zero, then check the response code location.
      1) If the code is not equal to zerø, then put it directly into the write code buffer.
      2) If the code is equal to zero, and the 'request data from VAX' flag is set, then several steps are taken.
         a) Put the 'send data to 8751' code into the write code buffer.
         b) Move the contents of NRBUFF to NR (update the current number of data bytes to transmit).

c) Put the 'send data to 8751' code in the previous write to VAX location.

3) Clear the response code buffer.

c. If the write code buffer is not zero, then its contents must be transmitted.

1) If the code equals one, then put the 'no operation' code into the seial port transmitter buffer.

2) If the code is not equal to one or to zero, then put it directly into the serial port transmitter buffer.

3) If the code is zero, do not transmit it, and reset the 'transmitter idle' flag.

4) Clear the write code buffer location.

d. Return from the serial-output subroutine.

THE DECODE VAX COMMAND SUBROUTINE

The decode-VAX-command subroutine executes the appropriate response to a command from the VAX.

1. If the command is no operation, no action is taken.
2. If the code is not valid, then several steps are taken.

a. If the previous code to VAX was one which required the 'send data' response from VAX, then the VAX may be expecting data to be sent, so that a resend request would be seen as data.

   1) Send a string of 16 zero bytes to VAX to indicate a data communication fault.

   2) Put unity in the response code buffer to indicate 'no operation' should be sent to VAX.

b. If the previous code to VAX did not require the 'send data' response from VAX, then put the 'send previous command over again' code in the response buffer.

3. If the command is enable command-on-interrupt, then several steps are taken.

   a. If the command to be executed on interrupt zero is not one of the four commands allowed for that purpose, then put the 'invalid data' code into the response code buffer.

   b. If the command is allowable on interrupt, then move it to the interrupt command location and put the 'received data' code in the response code buffer.

4. If the command is disable command-on-interrupt, clear the interrupt command location.

5. If the command is 8748 peek/poke then several steps are taken.

a. If the high bit of the address to peek/poke is set, then the desired operation is a poke.

   1) Request more data from the VAX (the motor number for the poke operation and the data byte to be poked).

   2) Wait for the data to be received from the VAX.

   3) Obtain access to the shared RAM of the 8748 to be poked.

   4) Put the address and the data byte in their locations there.

   5) Enable that motor's write mailbox.

   6) Wait for the write mailbox full flag to be clear.

      a) If the write mailbox is full, check that motor's read mailbox.

      b) If the read mailbox is full, call the decode-8748-command subroutine.

   7) Put the '8748 peek/poke' command code into the write mailbox.

   8) Put the 'received data' code into the response code buffer.

b. If the high bit of the peek/poke address is clear, then the desired operation is a peek.

   1) Gain access to each shared RAM individually, and write the address of the peek in its location in

shared RAM.

2) After all shared RAM's have the peek address loaded, set the '8748 peek in progress' flag (bit zero of the byte containing the 'peek data ready' flags).

3) Call the global-write subroutine to write the '8748 peek/poke' command into all write mailboxes.

4) Clear the response code buffer (the main routine concatenates the 8748 peek data when it is ready).

5) Return from the decode-VAX-command subroutine.

6. If the command is 8751 peek then several steps are taken.

a. If the high bit of the address of the peek is set, then one of the external latches is to be read.

1) Put the address out on Port 2 to enable the appropriate latch output.

2) Read the latch output in from Port 1 to the bottom of the write buffer.

b. If the high bit of the address of the peek is clear, then one of the internal RAM addresses is to be read.

1) Put the address into an address pointer.

2) Read the internal RAM address by indirect addressing and put the data in the bottom of the write buffer.

c. Above the data in the write buffer, put the address

from which it was read, the low and high bytes of the Rhino Clock, and the '8751 peek' code.

d. Set the pending number of data bytes for transmission to five.

e. Put the 'peek data ready' code in the response code buffer.

7. If the command is velocity request, then several steps are taken.

a. Gain access to the shared RAM of each motor, individually, and move in the velocity data to the write buffer (Motor 1 data at the bottom of the buffer).

b. Above the data in the write buffer, put the low and high bytes of the Rhino Clock.

c. Set the pending number of data bytes to transmit (NTBUFF) to nine.

d. Put the 'velocity data ready' code in the response code buffer.

8. If the command is absolute position request, then several steps are taken.

a. Gain access to the shared RAM of each motor, individually, and move in the low and high bytes of position data to the write buffer (Motor 1 data at the bottom of the buffer).

b. Above the data in the write buffer, put the low and

high bytes of the Rhino Clock.

c. Set the pending number of data bytes to transmit (NTBUFF) to sixteen.

d. Put the 'position data ready' code in the response code buffer.

9. If the command is send data to VAX, then just return, since that should never reach this level in the program (it should be intercepted by the serial-port-interrupt-service subroutine).

10. If the command is send previous code over again, then two steps are taken.

a. Move the contents of the previous write to VAX location to the response code buffer.

b. Set the previous write to VAX location to unity, indicating a 'no operation' code.

11. If the command is clear remaining moves, then several steps are taken.

a. Clear the '8748 motor stalled' flags.

b. Call the global-write subroutine to put the 'clear remaining moves' code in all write mailboxes.

c. Put the 'cleared remaining moves' code in the response code buffer.

12. If the command is move, then several steps are taken.

a. If no motors are linked, then put 'ready for next move'

code in the response code buffer.

b. If at least one motor is linked, then gain access to each shared RAM, individually.

   1) For the particular motor, if the move is zero, clear the 'next move received' flag in shared RAM, and clear the motor's 'not ready for next move' flag in 8751 internal RAM.

   2) Strip off the sign bit (bit 15) of the move and write it to the 'move sign' location in shared RAM.

   3) For the particular motor, if the move exceeds 31, then set the 'infinite move' flag in shared RAM.

   4) Put the unsigned two byte move in its locations in shared RAM.

c. Set the receive data pending buffer to seven (for seven bytes of velocity data).

d. Set the 'request data' flag.

e. Call the write-to-VAX subroutine to transmit the 'send data to 8751' code to the VAX.

f. Wait until all seven bytes of velocity data have been received from the VAX.

g. Move the velocity data from the read buffer to the shared RAM velocity location for each motor.

h. Call the global-write subroutine to put the 'move' command in all of the write mailboxes.

i. Put the 'received data' code into the response code location.

13. If the command is halt, then two steps are taken.
    a. Call the global-write subroutine to pass the 'halt' command on to all the write mailboxes.
    b. Put the 'motors halted' code in the response code location.
14. If the command is all motors go, then two steps are taken.
    a. Call the global-write subroutine to pass the 'go' command on to all the write mailboxes.
    b. Put the 'motors in go state' code in the response code location.
15. If the command is shared RAM peek, then several steps are taken.
    a. Gain access to each motor's shared RAM, individually.
    b. Move the data from the indicated address into the write buffer, with Motor 1 data at the bottom of the buffer.
    c. Above the data in the write buffer, put the address from which the data came, the low and high bytes of the Rhino Clock, and the command to which the data is the response.
    d. Set the pending transmit data byte counter to eleven.
    e. Put the 'peek data ready' code in the response code buffer.

16. If the command is reset absolute position counters, then two steps are taken.
    a. Call the global-write subroutine to pass the 'reset absolute position' command on to all the write mailboxes.
    b. Put the 'absolute positions reset' code in the response code location.
17. If the command is coefficients, then several steps are taken.
    a. Gain access to the shared RAM for the motor number indicated.
    b. Move the coefficients from the read buffer to the corresponding shared RAM locations.
    c. Check that the motor's write mailbox is not full.
        1) If it is full, and if the read mailbox is full, call the respond-to-8748 subroutine.
        2) Wait until the write mailbox is not full.
    d. Put the 'coefficients' command code into the write mailbox.
    e. Put the 'received data' code in the response code buffer.
18. Clear the VAX command location.
19. Jump to the write-to-VAX subroutine (that is, return from this subroutine by way of the write-to-VAX subroutine).

THE GLOBAL WRITE SUBROUTINE

The global-write subroutine writes a specified command code to all write mailboxes simultaneously.

1. Enable all write mailboxes.
2. If any write mailboxes is full, then several steps are taken.
   a. If any read mailbox is full, call the respond-to-8748 subroutine.
   b. Keep checking the write and read mailboxes until all of the write mailboxes are empty.
3. Write the code from the 'global' location to all the mailboxes simultaneously.
4. Put the same code in each of the 'previous write to 8748' locations.
5. Return from the global-write subroutine.

THE SERIAL PORT INTERRUPT SERVICE SUBROUTINE

The serial-port-interrupt-service subroutine is called whenever the serial port finishes transmitting or receiving a byte of communication data.

1. Save the processor status word and the accumulator contents on the stack.
2. Select register bank 1.
3. If the receive interrupt flag is set, then a byte of communication data has just been received from the VAX.
    a. Clear the receive interrupt flag.
    b. If any receive data is pending (that is, if NR is not zero), then the byte just received is unformatted data.
        1) Compute the read buffer location in which to put the incoming data.
        2) Decrement the receive data pending counter (NR).
            a) If the result is NR equals zero, then all data associated with the command in the input code buffer has been received.
                i. Move the command from the input code buffer to the read code buffer, so it will be processed by the main routine.

ii. Clear the input code buffer.

iii. If the command is always followed by an address (for example, the peek commands), then the lowest byte in the read buffer is an address to be moved to the 'latch address' location.

b) If the result is NR not equal zero, then more data is yet to be received.

c. If no receive data is pending (that is, if NR equals zero), then the incoming communication byte is a command code.

1) Move the code from the serial port receiver buffer to the input code buffer.

2) If the number of data bytes which are associated with the incoming command code is not zero, then several steps are taken.

a) Set the number of receive data bytes pending (NRBUFF) to the correct value for the particular command.

b) Set the 'request data from VAX' flag.

c) Call the serial-output subroutine.

d. Recover the processor status word and accumulator contents from the stack.

e. Return from the serial-port-interrupt-service subrou-

tine.

4. If the transmit interrupt flag is set, then several steps are taken.
   a. Clear the transmit interrupt flag.
   b. Call the serial-output subroutine.
   c. Recover the processor status word and the accumulator contents from the stack.
   d. Return from the serial-port-interrupt-service subroutine.

THE INTERRUPT ZERO SERVICE SUBROUTINE

The interrupt-zero-service subroutine is called whenever there is a low-to-high transition of the interrupt zero line of the Intel 8751.

1. Save the processor status word and accumulator contents on the stack.
2. Increment the two-byte Rhino Clock.
3. If the '8748 RAM peek in progress' flag is clear, move the interrupt command into the interrupt command buffer for processing by the main routine.
4. Recover the processor status word and accumulator contents from the stack.
5. Return from the interrupt-zero-service subroutine.

APPENDIX U. 8748 ASSEMBLY LANGUAGE PROGRAM OUTLINE

THE INITIALIZATION ROUTINE

The initialization routine is executed on power up.

1. Obtain the present shaft encoder disk position.
2. Set the 'motor stopped' flag.
3. Gain access to the shared RAM.
4. Clear shared RAM locations zero through 63.
5. Clear internal RAM locations zero through 63.
6. Test the shared RAM by a write and read of location zero.
7. Put the 'hardware reset' code in the write mailbox.
8. Write a zero pulsewidth to the pulsewidth latch.
9. Jump to the main program.

THE MAIN PROGRAM

The main routine services the mailboxes and responds to commands from the 8751.

1. Strobe the interrupt enable to pick up the interrupt condition.
2. If the write buffer is full, jump to the write routine.
3. If the read mailbox is full, read in the command and interpret it.
   a. If the command is the no operation, then restart the main routine.
   b. If the command code is not in valid format, put the 'resend request' code in the write buffer and go to the write routine.
   c. If the command is resend request then several steps are taken.
      1) Put the code from the 'previous write' buffer in the write mailbox.
      2) Put the 'no operation' code in the 'previous write' buffer, in case the previously saved code was garbled before transmission (this avoids an infinite loop of resend requests and retransmissions of garbage).

3) Restart the main routine.

d. If the command is clear remaining moves, then several steps are taken.

1) Clear the 'motor stalled' flag.

2) Clear all next move locations, all present move locations, the present pulsewidth location, and the velocity set point location.

3) Put the 'ready for next move' code into the write buffer.

4) Restart the main routine.

e. If the command is move, then several steps are taken.

1) Gain access to the shared RAM.

2) Move in the 'next move infinite' and 'received next move' flags, the velocity, the low and high bytes of the next move, and the sign of the next move from the shared RAM.

3) Restart the main routine.

f. If the command is halt, clear the 'go' flag, and restart the main routine.

g. If the command is go, set the 'go' flag and restart the main routine.

h. If the command is 8748 peek/poke, then several steps are taken.

1) Gain access to the shared RAM.

2) Read in the 1-byte address to be 'peeked' or 'poked'.

3) If the high bit of the address is set, then a 'poke' is to be executed (only the low six bits of the address are significant in internal RAM addresses).

   a) Read in the data to be 'poked' from the data passing location in shared RAM.

   b) Put that data into the location specified by the low six bits of the address byte.

   c) Restart the main routine.

4) If the high bit of the address is clear, then a 'peek' is to be executed.

   a) Read data from the location specified by the low six bits of the address byte.

   b) Move that data to the data passing location in shared RAM.

   c) Put the 'peek complete' code into the write buffer.

   d) Go to the write routine.

i. If the command is reset absolute positions, then several steps are taken.

1) Gain access to the shared RAM.

2) Clear the low and high bytes of the absolute posi-

tion.

3) Restart the main routine.

j. If the command is none of the above, then it must be coefficients.

1) Gain access to the shared RAM.

2) Move the six coefficients from their locations in shared RAM to the corresponding locations in internal RAM.

3) Restart the main routine.

4. The **write routine** is part of the main routine.

a. Move the code from the write buffer to the previous write buffer.

b. Clear the write buffer.

c. Strobe the interrupt enable line while waiting for the write mailbox to become empty (without this step, response to an interrupt might be delayed unduly).

d. Once the write mailbox is empty, put the code from the previous write buffer into the write mailbox.

e. Restart the main routine.

THE INTERRUPT SERVICE SUBROUTINE

The interrupt-service subroutine is executed whenever the interrupt line is active and the interrupt is enabled.

1. Put the 'interrupt acknowledge' address on Port 2 (this is also the address that enables the shaft encoder data latch output).
2. Read in the shaft encoder data.
3. If the position is unchanged since the last interrupt, call the motor-stopped subroutine, then return from the interrupt service routine.
4. Update the register 6 and 7 flags and data.
    a. Exchange the present shaft encoder data for the contents of register 7.
    b. Restore the previous state of the 'reversal since last update' flag (register 7, bit 0).
    c. Exclusive-or the present encoder bit 0 with the previous encoder encoder bit 1 to obtain the absolute direction bit.
    d. Update the 'absolute direction' flag by exchanging the new direction bit for the old flag value.
    e. Exclusive-or the new and old values of the 'absolute direction' flag to see if the direction has just

changed.

f. If the direction has just changed, then two steps are taken.

   1) Set the 'new motion' flag, since motor had to come to a stop before reversing.

   2) Complement the 'reversal since last update' flag, unless the 'previous position 0,0' flag is set (since, in that case, the last update occurred just as the new direction began).

g. Exclusive-or the 'absolute direction' flag with the present move sign to obtain the new value for the 'relative direction' flag.

5. Determine the velocity of the motor.

   a. If the 'new motion' flag is set, then the velocity is taken to be 1, since the interrupt clock value is meaningless, but there is motion.

   b. If the 'new motion' flag is clear, then use the interrupt clock value to determine the velocity.

      1) Effectively shift the 16-bit clock value left until it is left-adjusted, keeping track of the number of shifts required.

      2) The high byte of this left-adjusted number is used to obtain a value from the velocity look-up table.

      3) The look-up table value is shifted right a number

of times equal to eleven minus the number of left shifts required to left-adjust the clock value (note that this is based on the fact that the relationship between the clock value and the look-up table value is reciprocal and logarithmic base 2).

4) If a set bit is shifted out on the last right shift, round the result upward.

5) If the result of the right-shifting is zero, set it to one to indicate that the motor is not stopped.

6. Copy the velocity value into the 'speed' location in internal RAM.

7. Determine the velocity error.

   a. If the 'relative motion' flag is set, then the speed is negative with respect to the velocity set point.

   b. Subtract the speed from the velocity set point to obtain the velocity error.

   c. Determine which maximum velocity error magnitude applies for the present velocity set point.

   d. Limit the velocity error by the relationship: -1 * (maximum velocity error) <= velocity error <= (maximum velocity error).

   e. If velocity change is towards the set point, then set the velocity error to zero.

8. If the 'go' flag is not set or if the 'stall' flag is set, then the motor is either halted or stalled.
    a. Call the position-update subroutine.
    b. Write zero pulsewidth to the pulsewidth modulator latch and to the internal 'pulsewidth' location.
    c. Wait for the interrupt line to become inactive before returning from the interrupt.
9. If the 'go' flag is set and the 'stall' flag is clear, then several steps are taken.
    a. Determine which velocity coefficient to use, based on the current velocity set point.
    b. Call the pulsewidth-update subroutine.
    c. Call the position-update subroutine.
    d. Handle special cases of the state.
        1) If the position error is negative, then overshoot is occurring.
            a) If the overshoot is at least one full hole, load the user-specified pulsewidth for a 'long' overshoot as the base pulsewidth.
            b) If the overshoot is less than one full hole, load the user-specified pulsewidth for a 'short' overshoot as the base pulsewidth.
            c) If motor motion is in the direction of worse overshoot, add the value of the 'speed'

location to the base pulsewidth to give additional braking that is proportional to the overshoot speed.

d) Write the computed pulsewidth to the pulsewidth modulator latch with the pulsewidth sign opposite to the commanded move sign.

e) Wait for the interrupt line to become inactive before returning from the interrupt-service routine.

2) If the position error is zero, then several steps are taken.

a) Clear the velocity set point, since move is finished.

b) If the speed equals one, then set pulsewidth to zero to avoid end position jitter.

c) If the speed is greater than one, then set the pulsewidth equal to the speed, thus applying braking power proportional to the speed.

d) Write the computed pulsewidth to the pulsewidth modulator latch with the pulsewidth sign opposite to the 'absolute direction' sign.

e) Wait for the interrupt line to become inactive before returning from the interrupt-service routine.

3) If the move error is positive and the velocity set point equals zero, then the motor is undershooting (that is, it has completed the last move command but is out of position in the undershoot direction).

   a) If the undershoot is at least one full hole, load the user-specified pulsewidth for a 'long' overshoot as the pulsewidth.

   b) If the undershoot is less than one full hole, load the user-specified pulsewidth for a 'short' overshoot as the pulsewidth.

   c) Write the computed pulsewidth to the pulsewidth modulator latch with the pulsewidth sign the same as the 'absolute direction' sign.

   d) Wait for the interrupt line to become inactive before returning from the interrupt-service routine.

4) If none of the cases above is true (that is, the move error is positive and the velocity set point is non-zero), then several cases must be considered.

   a) If the position error greater then 127 holes then the state should be under ordinary velocity control (already determined); so just wait

for the interrupt line to become inactive and return from the interrupt service routine.

b) If the error is less than 128 holes, then several flags must be tested.

i. If the 'next move received' flag is set and the next move sign is the same as the present move sign, then the next move is like an extension of the present move.

i) If the 'next move infinite' flag is set, then the extension of the present move is long (at least 32 holes), so no preparation for stopping is required; just wait for the interrupt line to become inactive and return from the interrupt service routine.

ii) If the 'next move infinite' flag is clear, then the extension of the present move is short (less than 32 holes); if the present move (position error) is also small, then add the next move to the present move.

ii. Decelerate by ramping down the velocity set point: if the velocity set point exceeds the position error plus fifteen,

then the set point is replaced by the the position error plus fifteen.

iii. Wait until the interrupt line is inactive, then return from the interrupt service routine.

THE MOTOR STOPPED SUBROUTINE

The motor-stopped subroutine is called from the interrupt-service subroutine whenever there is an interrupt with no motor motion.

1. If the interrupt line is still active, then motor motion has occurred (since the flip-flop that is set after 1/16 second of time passes without an interrupt should have been reset by the interrupt acknowledge), so return from the motor-stopped subroutine.
2. If the interrupt line is no longer active then the motor has not moved in the past 1/16 second.
   a. Increment the 'motor stopped interrupt count', which keeps track of the number of times the motor stop interrupt has occurred since the last motion.

b. Set the 'motor stopped' flag.

c. Put zero in the 'speed' location.

d. Call the position-update subroutine.

e. If the 'stall' flag is set or the 'go' flag is clear, write zero to the pulsewidth modulator latch and the pulsewidth location in internal RAM, and return from the motor-stopped subroutine.

f. If the 'stall' flag is clear and the 'go' flag is set, check to see if the motor is stalled.

    1) If the pulsewidth is at least 50% and the 'motor stopped interrupt count' is at least 8 (that is, motor has been stopped for at least one second), then the motor is stalled.

        a) Set the 'stall' flag.

        b) Write zero to the pulsewidth modulator latch and to the internal pulsewidth location.

        c) Return from the motor-stopped subroutine.

    2) If the pulsewidth is less than 50% or the 'motor stopped interrupt count' is less than 8, then the motor is not stalled.

        a) Call the move-yet? subroutine to determine the appropriate pulsewidth (which depends on the position error).

        b) Write the pulsewidth to the pulsewidth modula-

tor latch.

c) Return from the motor-stopped subroutine.

THE MOVE YET? SUBROUTINE

The move-yet? subroutine checks the remaining position error and determines the appropriate pulsewidth.

1. If the position error is non-zero or if the 'quarter-hole tail' is non-zero, then the pulsewidth will be non-zero.
   a. If the motor stopped interrupt count is exactly one, then add the velocity set point to the starting pulsewidth.
   b. Add the user-specified starting pulsewidth to the current pulsewidth.
   c. Obtain the pulsewidth sign by exclusive-or of the present move sign with the sign of the position error.
   d. Return from the move-yet? subroutine with the accumulator cleared.
2. If the present position error is zero, then call the new-move subroutine.
   a. If a non-zero new move is returned, determine the

starting pulsewidth.

1) Set the pulsewidth equal to the user-specified starting pulsewidth.

2) Set the pulsewidth sign equal to the sign of the new move (position error).

3) Return from the move-yet? subroutine with the accumulator cleared.

b. If a zero new move is returned, do not change the pulsewidth, and return from the move-yet? subroutine with a non-zero value in the accumulator.

THE NEW MOVE SUBROUTINE

The new-move subroutine moves the data for a non-zero next move into the present move locations.

1. If the 'next move received' flag is not set, then return from the new-move subroutine with a non-zero value in the accumulator.
2. If the 'next move received' flag is set, then several steps are taken.
   a. Clear the 'next move received' flag.

b. Move the velocity command into the velocity set point location.

c. Move the velocity, distance, and sign data from the next move locations to the present move locations.

d. Put the 'ready for next move' code into the write buffer.

e. Return from the new-move subroutine with the accumulator cleared.

THE POSITION UPDATE SUBROUTINE

The position-update subroutine updates the position error and quarter-hole error locations in internal RAM and the velocity and absolute position locations in shared RAM.

1. Gain access to the shared RAM.
2. Concatenate the sign bit from the 'absolute direction' flag and the present speed, storing the result in the velocity location of the shared RAM.
3. If the speed is zero, then the motor has not moved and no position updating is required, so just return from the position-update subroutine.

4. If the speed is not zero, then update the quarter-hole position error, the position error, and the absolute position.

   a. If the 'relative direction' flag is set, then motion is in the direction opposite that specified by the move sign, so increment the quarter-hole position error.

   b. If the 'relative direction' flag is clear, then motion is in the same direction as that specified by the move sign, so decrement the quarter-hole position error.

   c. If the 'present position 0,0' flag is clear, then no further position updating is required, so return from the position-update subroutine.

   d. If the 'present position 0,0' flag is set, then all position data locations are updated.

      1) Clear the quarter-hole position error location.

      2) If 'relative direction' flag is set, then increment the position error (present move) in internal RAM.

      3) If 'relative direction' flag is clear, then decrement the position error (present move) in internal RAM.

      4) If 'absolute direction' flag is set, then decrement the absolute position in shared RAM.

      5) If 'absolute direction' flag is clear, then incre-

ment the absolute position in shared RAM.

6) Return from the position-update subroutine.

THE PULSEWIDTH UPDATE SUBROUTINE

1. The pulsewidth-update subroutine determines the appropriate new pulsewidth (based on the present pulsewidth, the velocity error, and the velocity error coefficient), and writes the new pulsewidth to the pulsewidth modulator and to the internal RAM locations for the pulsewidth.
2. Call the multiply subroutine to multiply the velocity error and the velocity error coefficient (the returned quotient is therefore the pulsewidth increment).
3. If present pulsewidth (power) is in the direction opposite the commanded direction of motion, then it is negative with respect to the pulsewidth increment, so two's complement the pulsewidth increment before adding it to the pulsewidth.
4. Add the pulsewidth increment to the pulsewidth.
5. If the new pulsewidth exceeds 127, then it has overflowed.
   a. If the pulsewidth increment was positive, then the

overflow is due to the magnitude exceeding the maximum allowed (127).

1) Set the pulsewidth equal to the maximum, 127.
2) Concatenate the pulsewidth sign.
3) Write the signed pulsewidth to the pulsewidth modulator.
4) Return from the pulsewidth-update subroutine.

b. If the pulsewidth increment was negative, then the overflow is due to the value of the pulsewidth going negative.

1) Two's complement the pulsewidth value.
2) Complement the pulsewidth sign.
3) Write zero to the pulsewidth modulator.
4) Enter a delay loop to allow time for the power transistor that was turned on to reach the cutoff state before the other power transistor is turned on.
5) Add the user-specified hysteresis value to the pulsewidth.
6) If the pulsewidth now exceeds 127, set the it to the maximum value, 127.
7) Concatenate the pulsewidth sign.
8) Write the signed pulsewidth to the pulsewidth modulator.

9) Return from the pulsewidth-update subroutine.

6. If the new pulsewidth does not exceed 127, and if it is not zero, then add the user-specified hysteresis value to it.
7. If the pulsewidth now exceeds 127, set it to the maximum value, 127.
8. Concatenate the pulsewidth sign.
9. Write the signed pulsewidth to the pulsewidth modulator.
10. Return from the pulsewidth-update subroutine.

THE MULTIPLY SUBROUTINE

1. The multiply subroutine is a fast subroutine for multiplying the argument passed to it in the accumulator by the number in the location passed to it in address pointer RB1 R0 (where the number at RB1 R0 is taken to have only six significant digits, and taken to be divided by 16).
2. Save the original argument in RB1 R3, to preserve the sign data.
3. If the argument is negative (high bit set) then two's complement it.
4. Use RB0 R0 through RB0 R5 to store the argument multiplied

by powers of 2, from 2 ** 1 through 2 ** -4, respectively.

5. Obtain the coefficient from the location specified by the address pointer RB1 R0.
6. For each bit of the coefficient that is not set, clear the corresponding register in RB0 (for example, if bit 0 is not set, clear RB0 R5 and if bit 3 is not set, clear RB0 R2), for bits zero through five of the coefficient.
7. Add the resulting contents of RB0 R0 through RB0 R5 to obtain the product.
8. If the original argument was negative, two's complement the product.
9. Return from the multiply subroutine.

APPENDIX V. LISTING OF THE SAMPLE PROGRAM ON VAX

This program was compiled by the VAX/VMS 11/780 resident PASCAL compiler. Page headings have been deleted for the sake of brevity.

```
0001
0002    program rhinocom (input,output);
0003
0004
0005
0006
0007       {*************************************}
0008       {**     define the variable types     **}
0009       {*************************************}
0010
0011    TYPE
0012         smallint  = 0..255;
0013         bytl      = [byte] smallint;
0014
0015         code set type      = set of smallint;
0016         mnemonic set type  = set of char;
0017
0018         string5  = packed array [1..5] of char;
0019         string80 = varying [80] of char;
0020
0021
0022
0023       {*****************************************}
0024       {**          define the variables       **}
0025       {*****************************************}
0026
0027    var
0028         command, prevcommand, coi command  : bytl;
0029         code, termcode, expected code      : bytl;
0030         pkaddress, dest                    : bytl;
0031         termflag                           : bytl;
0032         junkcode                           : bytl;
0033         movesign                           : bytl;
0034
0035         datatype             : integer;
0036         movetime             : integer;
0037         modulomove           : integer;
0038         length               : integer;
0039         loopcount            : integer;
0040         status, delaystatus  : integer;
0041         test1,test2          : integer;
0042         ef1,chan1            : integer;
0043
0044         poke48data      : array [0..2]  of bytl;
0045         latch51         : array [0..4]  of bytl;
0046         latch48         : array [0..10] of bytl;
0047         latchex         : array [0..10] of bytl;
0048         velocities      : array [0..8]  of bytl;
0049         veloc cmd       : array [0..6]  of bytl;
0050         coefficients    : array [0..14]  of bytl;
0051         movedata        : array [0..13] of bytl;
0052         positiondata    : array [0..15] of bytl;
0053
0054         moves       : array [0..6] of integer;
0055         positions   : array [0..7] of integer;
0056         velocity    : array [0..6] of integer;
0057
0058         autorespond         : boolean;
0059         go flag             : boolean;
0060         readyflag           : boolean;
0061
0062         code set          : code set type;
0063         mnemonic set      : mnemonic set type;
0064
0065         source, rhino, keyboard   : string5;
0066         junkinput                 : string80;
0067
0068         chrc   : char;
0069
0070
0071       {*************************************}
0072       {**   define the communication codes  **}
0073       {**      between VAX and the 8751     **}
0074       {*************************************}
0075
0076    const
0077
0078            nop    = 0;     {no operation}
0079            eci    = 17;    {enable command-on-interrupt}
0080            dci    = 34;    {disable command-on-interrupt}
0081            pk48   = 51;    {peek/poke 8748 internal ram}
0082            pk51   = 68;    {peek 8751 internal ram or latch}
0083            vrq    = 85;    {velocity request}
0084            prq    = 102;   {position request}
0085            sndvx  = 119;   {send to VAX}
0086            crm    = 153;   {clear remaining moves}
0087            mov    = 170;   {move}
0088            hlt    = 187;   {halt}
0089            ago    = 204;   {all motors go}
0090            shrpk  = 221;   {shared RAM peek}
0091            rap    = 238;   {reset absolute position}
0092            kff    = 255;   {coefficients}
0093
0094            ovr    = 136;   {send over (resend request)}
0095
0096            snd51  = 17;    {send to 8751}
0097            rdynxt = 51;    {ready for next move}
0098            dat    = 68;    {peek data ready}
0099            vdt    = 85;    {velocity data ready}
```

```
0100          pdt    = 102;   {position data ready}
0101          rcvdat = 119;   {received data}
0102          hdwrst = 170;   {hardware reset of Rhino}
0103          mtrstl = 221;   {motor stall}
0104          inv    = 255;   {invalid data}
0105
0106
0107      {*****************************************}
0108      {**      define external procedures:    **}
0109      {*****************************************}
0110
0111
0112    procedure rhinord
0113                    (%stdescr ttline:string5;
0114                     %ref keybuff:bytl;
0115                      %ref rhinobuff:bytl;
0116                       %ref termflag:bytl;
0117                        %ref rhinoflag:integer;
0118                         %ref rhinochannel:integer); extern;
0119
0120
0121    procedure rhinowr
0122                    (%ref rhinoflag:integer;
0123                         %ref rhinochannel:integer;
0124                              %ref buffer:bytl); extern;
0125
0126
0127      {************************************************}
0128      {**      define internal procedures :          **}
0129      {************************************************}
0130
0131
0132
0133    procedure ttread2(var dest:bytl);          {This procedure calls}
0134                                               {the rhinord external}
0135                                               {procedure, with the }
0136        begin                                  {correct device name.}
0137              status:=0;
0138
0139          if (rhino='ttb0:') or (rhino='TTB0:') then
0140                    rhinord('ttb0:',termcode,dest,termflag,efl,chanl);
0141          if (rhino='ttb1:') or (rhino='TTB1:') then
0142                    rhinord('ttb1:',termcode,dest,termflag,efl,chanl);
0143          if (rhino='ttb2:') or (rhino='TTB2:') then
0144                    rhinord('ttb2:',termcode,dest,termflag,efl,chanl);
0145          if (rhino='ttb7:') or (rhino='TTB7:') then
0146                    rhinord('ttb7:',termcode,dest,termflag,efl,chanl);
0147          if (rhino='tte7:') or (rhino='TTE7:') then
0148                    rhinord('tte7:',termcode,dest,termflag,efl,chanl);
0149
0150          if termflag=0 then status:=10;
0151
0152          end;
0153
0154       {************************************************************}
0155
0156
0157    procedure get response(expected code:bytl);
0158                                          {this procedure looks for code}
0159                                          {from the rhino, then tests to}
0160                                          {see if it matches the code   }
0161                                          {expected.  A status code is  }
0162          begin                           {returned indicating what code}
0163                                          {was actually received, if any}
0164
0165             code set:=[nop,hdwrst,rdynxt,dat,
0166                        rcvdat,vdt,pdt,mtrstl,inv,ovr];
0167
0168             ttread2(code);
0169                                           { status = 10 only if a byte  }
0170             if status=10 then             {is received from the Keyboard}
0171               writeln('No response...');{before a byte from the Rhino }
0172
0173             if status<10 then
0174                if code = expected code then status:=0
0175                  else if code in code set then
0176                    case code of
0177
0178                      nop         : begin
0179                                      writeln('   NO OPERATION code          ');
0180                                      writeln('    received from Rhino       ');
0181                                      writeln('     before expected response ');
0182                                      writeln('                         .....');
0183
0184                                      get response(expected code);
0185                                    end;
0186
0187                      hdwrst      : begin
0188                                      status:=8;          {hardware reset}
0189                                      go flag:=false;
0190                                      readyflag:=true;
0191                                    end;
0192
0193                      mtrstl      : begin
0194                                      status:=7;          {motor stalled}
0195                                      go flag:=false;
0196                                      readyflag:=false;
0197                                    end;
0198
```

```
0199                    inv          : begin
0200                                     writeln('   INVALID DATA code          ');
0201                                     writeln('    received from Rhino       ');
0202                                     writeln('     before expected response ');
0203                                     writeln('                         .....');
0204
0205                                     get response(expected code);
0206                                   end;
0207
0208                    rcvdat       : begin
0209                                     writeln('    RECEIVED DATA  code       ');
0210                                     writeln('     received from Rhino      ');
0211                                     writeln('                   .....      ');
0212
0213                                     get response(expected code);
0214                                   end;
0215
0216
0217                    ovr          : begin
0218                                     writeln('    RESEND REQUEST code       ');
0219                                     writeln('     received from Rhino      ');
0220                                     writeln('                              ');
0221                                     writeln('. . resending: ',prevcommand  );
0222                                     writeln('                   .....      ');
0223
0224                                     rhinowr(efl,chanl,prevcommand);
0225                                     get response(expected code);
0226                                   end;
0227
0228                    rdynxt       : begin
0229                                     writeln('   READY FOR NEXT MOVE code   ');
0230                                     writeln('    received from Rhino       ');
0231                                     writeln('     before expected response');
0232                                     writeln('                .............');
0233
0234                                     readyflag:=true;
0235                                     get response(expected code);
0236                                   end;
0237
0238
0239                    dat          : status:=-4;           {peek data ready}
0240
0241
0242                    vdt          : status:=-9;           {velocity data ready}
0243
0244
0245                    pdt          : status:=-16;          {position data ready}
0246
0247
0248                  end  {various cases of code}
0249
0250                else  status:=2;
0251
0252
0253               if status<0 then    {in this case, the 8751 has data to transmit}
0254                     begin
0255                       delaystatus:=status;          {delay asking for data}
0256                       status:=0;                    {and go look again for}
0257                       get response(expected code);  {  the expected code  }
0258                     end;
0259
0260
0261                if (status=2) and (expected code=snd51) then
0262                    begin
0263                       command:=nop;                      {in this case, 8748 may}
0264                       for loopcount:=1 to 17 do          {be expecting data, so }
0265                           rhinowr(efl,chanl,command);    {send 17 zero bytes to }
0266                       status:=6;                         {clear communication and}
0267                    end;                                {set status to data fault}
0268
0269             end;
0270
0271         {*************************************************************}
0272
0273
0274       procedure get data;
0275                                                  {this procedure gets a data}
0276           begin                                  {   stream from the rhino  }
0277               code:=sndvx;   {send data to vax}
0278               rhinowr(efl,chanl,code);
0279               test1:=0;
0280               test2:=0;
0281               length:=0;
0282
0283               case datatype of
0284
0285                  1  : begin                    {Datatype of 1 means that the data}
0286                         ttread2(code);         {  from the 8751 are 'peek' data. }
0287                         case code of           {Peek data are always preceded by }
0288                                                { the original peek request code. }
0289                           pk51  : begin
0290                                     for loopcount:=3 downto 0 do
0291                                     begin
0292                                       length:=length+1;
0293                                       ttread2(latch51[loopcount]);
0294                                       if status<10 then test1:=
0295                                                          test1 + latch51[loopcount];
0296                                       if status=10 then test2:=10;
0297                                     end;
```

```
0298                              latch51[4]:=code;
0299                            end;
0300
0301                    pK48  : begin
0302                              for loopcount:=9 downto 0 do
0303                               begin
0304                                 length:=length+1;
0305                                 ttread2(latch48[loopcount]);
0306                                 if status<10 then test1:=
0307                                                   test1 + latch48[loopcount];
0308                                 if status=10 then test2:=10;
0309                               end;
0310                              latch48[10]:=code;
0311                            end;
0312
0313                    shrpK  : begin
0314                              for loopcount:=9 downto 0 do
0315                               begin
0316                                 length:=length+1;
0317                                 ttread2(latchex[loopcount]);
0318                                 if status<10 then test1:=
0319                                                   test1 + latchex[loopcount];
0320                                 if status=10 then test2:=10;
0321                               end;
0322                              latchex[10]:=code;
0323                            end;
0324
0325                      end; {various cases of original peeK code}
0326
0327                end; {case of datatype = 1}
0328
0329            2  : begin                                  {datatype of 2 means that}
0330                   for loopcount:=8 downto 0 do         { the data from 8751 are }
0331                     begin                              {       velocities       }
0332                       length:=length+1;
0333                       ttread2(velocities[loopcount]);
0334                       if status<10 then
0335                         begin
0336                           test1:=test1 + velocities[loopcount];
0337                           if loopcount<7 then
0338                             if velocities[loopcount]>127 then
0339                                 velocity[loopcount]:=
0340                                   -2 * ( 256 - velocities[loopcount] )
0341                               else velocity[loopcount]:=
0342                                             2 * velocities[loopcount];
0343                         end;
0344                       if status=10 then test2:=10;
0345
0346                     end;
0347
0348                end; {case of datatype = 2}
0349
0350            3: begin                                    {Datatype of 3 means that the}
0351                 for loopcount:=15 downto 0 do          {   data from the 8751 are   }
0352                   begin                                {        positions.          }
0353                     length:=length+1;
0354                     ttread2(positiondata[loopcount]);
0355                     if status<10 then test1:=test1 + positiondata[loopcount];
0356                     if status=10 then test2:=10;
0357                   end;
0358                if (not(status=10)) then
0359                  for loopcount:=0 to 7 do
0360                    begin
0361                      positions[loopcount]:=positiondata[2*loopcount]
0362                                          + 256*positiondata[2*loopcount+1];
0363                      if (positions[loopcount]>32767)
0364                       and (loopcount<>7) then
0365                          positions[loopcount]:=
0366                                       -1 * (65536-positions[loopcount]);
0367                    end;
0368
0369              end; {case of datatype = 3}
0370
0371         end; {various cases of datatype}
0372
0373         if (status<10) and (test1=0) then  {8751 sent all zeros, a data fault}
0374                  begin
0375                     length:=17-length;
0376                     writeln('reading',length,'data to dummy location');
0377                     for loopcount:=1 to length do
0378                           ttread2(code);
0379                     if status<10 then status:=6;   {data communication fault}
0380                   end;
0381         if test2=10 then
0382                  begin
0383                     status:=10;
0384                     writeln('                                                ');
0385                     writeln('Data stream from Rhino contaminated             ');
0386                     writeln('before completion by entries from Keyboard      ');
0387                     writeln('     ..... resulting data field is invalid.     ');
0388                     writeln('                                                ');
0389                   end;
0390       end;
0391
0392      {*************************************************************}
0393
0394
0395   procedure get screendata;
0396     begin                                          {this procedure prompts the}
```

```
0397        if status=0 then                                { user for data to send to }
0398          case command of                               {        the 8751          }
0399
0400            eci  : begin
0401                     writeln('Interrupt commands are executed by Rhino whenever ');
0402                     writeln('the Interrupt Zero line has a low-high transition.');
0403                     writeln('Valid interrupt commands are:                      ');
0404                     writeln;
0405                     writeln('.         51PK, 48PK, VRQ, and PRQ                 ');
0406                     writeln;
0407                     write  ('                          Interrupt command:  ');
0408
0409                     read(chrc);
0410
0411                     if (not(chrc='4')) and (not(chrc='5'))
0412                       and (not(chrc='v')) and (not(chrc='V'))
0413                       and (not(chrc='p')) and (not(chrc='P')) then
0414                         coi command:=nop;
0415
0416                     if chrc='4' then coi command:=pk48;
0417                     if chrc='5' then coi command:=pk51;
0418                     if (chrc='v') or (chrc='V') then coi command:=vrq;
0419                     if (chrc='p') or (chrc='P') then coi command:=prq;
0420
0421                     readln(junkinput);
0422
0423                   end;
0424
0425            kff  : begin
0426                     writeln;
0427                     write('                    Enter MOTOR NUMBER (1 to 7) : ');
0428                     readln(coefficients[0]);
0429                     coefficients[0]:=coefficients[0] mod 8;
0430
0431                     writeln('For velocity set point from 1 to 31,');
0432                     write('                      velocity error coefficient: ');
0433                     readln(coefficients[1]);
0434                     write('                      maximum magnitude of error: ');
0435                     readln(coefficients[5]);
0436                     writeln('For velocity set point from 32 to 63,');
0437                     write('                      velocity error coefficient: ');
0438                     readln(coefficients[2]);
0439                     write('                      maximum magnitude of error: ');
0440                     readln(coefficients[6]);
0441                     writeln('For velocity set point from 64 to 127,');
0442                     write('                      velocity error coefficient: ');
0443                     readln(coefficients[3]);
0444                     write('                      maximum magnitude of error: ');
0445                     readln(coefficients[7]);
0446                     writeln('For velocity set points above 127,');
0447                     write('                      velocity error coefficient: ');
0448                     readln(coefficients[4]);
0449                     write('                      maximum magnitude of error: ');
0450                     readln(coefficients[8]);
0451                     writeln(' ');
0452                     write('                        Enter gripper pulsewidth: ');
0453                     readln(coefficients[9]);
0454                     write('                    Enter hysteresis pulsewidth: ');
0455                     readln(coefficients[10]);
0456
0457                     write('                     Enter starting pulsewidth: ');
0458                     readln(coefficients[11]);
0459
0460                     write('                Enter long overshoot pulsewidth: ');
0461                     readln(coefficients[12]);
0462
0463                     write('               Enter short overshoot pulsewidth: ');
0464                     readln(coefficients[13]);
0465
0466                   end;
0467
0468            pk51 : begin              {peek into 8751 ram or latch}
0469                     writeln('                                                   ');
0470                     writeln('Data can be read from the following locations:    ');
0471                     writeln;
0472                     writeln('   - 8751 internal RAM, addresses 0 to 127        ');
0473                     writeln('   - limit switch latch, address 136              ');
0474                     writeln('   - motors linked latch, address 152             ');
0475                     writeln;
0476                     write('                Enter address to be read from 8751: ');
0477
0478                     readln(pkaddress);
0479                   end;
0480
0481            pk48 : begin
0482
0483                     if chrc='4' then begin
0484                             writeln(' ');
0485                             writeln('8748 RAM addresses are  0 - 63.        ');
0486                             writeln(' ');
0487                             write  ('                  Enter peek address: ');
0488
0489                             readln(pkaddress);
0490                             pkaddress:=pkaddress mod 128;
0491                           end;
0492
0493                     if chrc='W' then begin
0494                             writeln;
0495                             write('Motor number for the poke (1 to 7): ');
```

```
0496                            readln(poke48data[1]);
0497                            poke48data[1]:=poke48data[1] mod 8;
0498
0499                            writeln(' ');
0500                            write('       Enter poke address (0 to 63): ');
0501                            readln(poke48data[2]);
0502                            poke48data[2]:=poke48data[2] mod 128;
0503                            poke48data[2]:=poke48data[2] + 128;
0504
0505                            writeln(' ');
0506                            write('  Enter data to be written (0 to 255): ');
0507                            readln(poke48data[0]);
0508                           end;
0509
0510                  end;
0511
0512
0513          shrpK : begin
0514                     writeln(' ');
0515                     writeln('Enter the location');
0516                     write  ('          to be read (0-255): ');
0517
0518                     readln(pkaddress);
0519                     pkaddress:=pkaddress mod 128;
0520                    end;
0521
0522
0523          mov     : begin
0524                     writeln(' ');
0525                     writeln('Enter TIME move should take                ');
0526                     write  ('              (1 to 127 seconds): ');
0527
0528                     readln(movetime);
0529
0530                     writeln(' ');
0531                     writeln('Enter the MOVES: ');
0532                     writeln;
0533
0534                     for loopcount:=0 to 6 do
0535                       begin
0536                          write('          Motor', loopcount+1, ' : ');
0537
0538                          readln(moves[loopcount]);
0539
0540                          if moves[loopcount]<0 then
0541                              begin
0542                                   movesign:=128;
0543                                   moves[loopcount]:= -1 * moves[loopcount];
0544                                end
0545                           else movesign:=0;
0546
0547                          if (movetime<>0) then
0548                                veloc cmd[loopcount]:=
0549                                             moves[loopcount] div movetime
0550                           else veloc cmd[loopcount]:=200;
0551
0552                          veloc cmd[loopcount]:=veloc cmd[loopcount] div 2;
0553                          veloc cmd[loopcount]:=veloc cmd[loopcount] mod 128;
0554
0555                          if (veloc cmd[loopcount]<=0)
0556                                        and (moves[loopcount]>0)  then
0557                                                 veloc cmd[loopcount]:=1;
0558
0559                          if veloc cmd[loopcount]>100 then
0560                                                 veloc cmd[loopcount]:=100;
0561
0562                          movedata[2*loopcount]:=moves[loopcount] mod 256;
0563
0564                          movedata[2*loopcount+1]:=
0565                                       (moves[loopcount] div 256) + movesign;
0566
0567                          if movesign=128 then
0568                            moves[loopcount]:= -1 * moves[loopcount];
0569
0570                        end;
0571
0572                    end   {case of command = mov}
0573
0574                end;  {various cases of command mnemonic}
0575
0576            writeln(' ');
0577
0578      end;
0579
0580      {*************************************************************************}
0581
0582
0583  procedure get alldata;                    {  this procedure obtains }
0584                                            {    data from the rhino  }
0585      begin
0586         rhinowr(etl,chanl,command);
0587         prevcommand:=command;
0588         if status=0 then
0589           begin
0590                 case command of            {set the datatype for use }
0591                   pK48:  datatype:=1;      {in the get data procedure}
0592                   pK51:  datatype:=1;
0593                   shrpK: datatype:=1;
0594                   vrq:   datatype:=2;
```

```
0595                    prg:   datatype:=3;
0596                   end;
0597                  if datatype=1 then        {a peek command must be}
0598                    begin                   {followed by an address}
0599                     get response(snd51);
0600                     if status=0 then rhinowr(efl,chanl,pkaddress);
0601                     if status=0 then get response(dat);   {expect 'data ready'}
0602                    end
0603                   else get response(command);
0604                  if status=0 then get data;
0605              end;
0606          end;
0607      {*****************************************************************}
0608
0609    procedure display 51data;               {  this procedure displays the  }
0610                                            {data obtained from an 8751 peek}
0611        begin
0612           writeln(' 8751 PEEK:');
0613           writeln(' ');
0614           writeln('  rhino time    address     data');
0615           writeln(latch51[3]*256+latch51[2],latch51[1],latch51[0]);
0616           writeln(' ');
0617        end;
0618
0619      {*****************************************************************}
0620
0621    procedure display 48data;               {  this procedure displays the  }
0622                                            {data obtained from an 8748 peek}
0623        begin
0624           writeln(' 8748 PEEK:');
0625           writeln(' ');
0626           writeln('rhino time        address');
0627           writeln(latch48[9]*256+latch48[8],' ',latch48[7]);
0628           writeln(' ');
0629           writeln('   motor 1   motor 2   motor 3   motor 4',
0630                                              '   motor 5   motor 6   motor 7');
0631           writeln(latch48[0],latch48[1],latch48[2],latch48[3],
0632                                            latch48[4],latch48[5],latch48[6]);
0633           writeln(' ');
0634
0635         end;
0636
0637      {*****************************************************************}
0638
0639    procedure display shrdata;              {this procedure displays the data}
0640                                            {obtained from a shared RAM peek.}
0641        begin
0642           writeln(' SHARED RAM PEEK:');
0643           writeln(' ');
0644           writeln('rhino time        address');
0645           writeln(latchex[9]*256+latchex[8],' ',latchex[7]);
0646           writeln(' ');
0647           writeln('   motor 1   motor 2   motor 3   motor 4',
0648                                              '   motor 5   motor 6   motor 7');
0649           writeln(latchex[0],latchex[1],latchex[2],latchex[3],
0650                                            latchex[4],latchex[5],latchex[6]);
0651           writeln(' ');
0652
0653         end;
0654
0655      {*****************************************************************}
0656
0657    procedure display velocities;           {   this procedure displays  }
0658                                            {the velocity data from rhino}
0659        begin
0660           writeln(' VELOCITIES:');
0661           writeln('                ');
0662           writeln('rhino time   motor 1   motor 2   motor 3   motor 4',
0663                                              '   motor 5   motor 6   motor 7');
0664           writeln(velocities[8]*256+velocities[7],velocityl[0],
0665                             velocityl[1],velocityl[2],velocityl[3],
0666                             velocityl[4],velocityl[5],velocityl[6]);
0667           writeln('                ');
0668        end;
0669
0670      {*****************************************************************}
0671
0672    procedure display positions;            {  this procedure displays   }
0673                                            {the position data from rhino}
0674        begin
0675           writeln(' POSITIONS:');
0676           writeln('                ');
0677           writeln('rhino time   motor 1   motor 2   motor 3',
0678                             '   motor 4   motor 5   motor 6   motor 7');
0679           writeln(positions[7],positions[0],positions[1],positions[2],
0680                             positions[3],positions[4],positions[5],positions[6]);
0681           writeln('                ');
0682        end;
0683
0684      {*****************************************************************}
0685
0686    procedure send command get echo;        {this procedure transmits the}
0687                                            {command code and obtains the}
0688        begin                               { resulting 8751 response if }
0689           rhinowr(efl,chanl,command);      { an echo of the command is  }
0690           prevcommand:=command;            {   the expected response.   }
0691           get response(command);
0692        end;
0693
```

```
0694      {*****************************************************************}
0695
0696    procedure send 48data;                      {this procedure sends the}
0697                                                { appropriate data for an}
0698         begin                                  {8748 poke command (WR48)}
0699            rhinowr(efl,chanl,command);         {       to the 8751       }
0700            prevcommand:=command;
0701            get response(snd51);
0702            if status=0 then
0703              begin
0704                rhinowr(efl,chanl,poke48data[2]);
0705                get response(snd51);
0706                if status=0 then
0707                  begin
0708                    for loopcount:=1 downto 0 do
0709                      rhinowr(efl,chanl,poke48data[loopcount]);
0710                    get response(rcvdat);
0711                  end
0712              end
0713       end;
0714
0715      {*****************************************************************}
0716
0717    procedure send moves;                       {this procedure sends the data }
0718                                                {in the MOVES array to the 8751}
0719         begin
0720               rhinowr(efl,chanl,command);     {move command to 8751}
0721               prevcommand:=command;
0722               get response(snd51);    {should be 'send data'}
0723               if status=0 then
0724                 begin
0725                    for loopcount:=13 downto 0 do
0726                            rhinowr(efl,chanl,movedata[loopcount]);
0727
0728                    get response(snd51); {should be 'send data to 8751'}
0729                    if status=0 then
0730                       begin
0731                         for loopcount:=6 downto 0 do
0732                             rhinowr(efl,chanl,veloc cmd[loopcount]);
0733                         get response(rcvdat);
0734                       end;
0735                 end;
0736
0737               if status=0 then readyflag:=false;
0738           end;
0739
0740      {*****************************************************************}
0741
0742    procedure load coefficients;                    {this procedure sends the data}
0743                                                    {from the COEFFICIENTS array  }
0744         begin                                      {         to the 8751         }
0745
0746            rhinowr(efl,chanl,command);   {coefficient command to 8751}
0747            prevcommand:=command;
0748            get response(snd51);    {should be 'send data'}
0749            if status=0 then
0750              begin
0751                for loopcount:=14 downto 0 do
0752                        rhinowr(efl,chanl,coefficients[loopcount]);
0753                get response(rcvdat);   {should be  'data received'}
0754              end;
0755          end;
0756
0757      {*****************************************************************}
0758
0759    procedure enable coi;                       { this procedure sends the }
0760                                                {interrupt command to rhino}
0761         begin
0762            rhinowr(efl,chanl,command);         {send 'enable command on interrupt'}
0763            prevcommand:=command;
0764            get response(snd51);                {expected response is 'send data'}
0765            if status=0 then
0766               begin
0767                   rhinowr(efl,chanl,coi command);   {send the interrupt command}
0768                   get response(rcvdat);   {expected response is 'data received'}
0769                end;
0770         end;
0771
0772      {*****************************************************************}
0773
0774
0775    procedure code from 8751;                   {this procedure tests the code }
0776                                                {     received from the 8751    }
0777         begin
0778          status:=0;
0779          code set:=[nop,rdynxt,hdwrst,mtrstl,ovr,dat,vdt,pdt,
0780                     rap,dci,hlt,ago,crm,rcvdat,snd51,inv];
0781          if code in code set then
0782
0783            case code of
0784
0785              nop     :  writeln('               NO OPERATION');
0786
0787              rdynxt  :  begin
0788                           writeln('           READY FOR NEXT MOVE');
0789                           readyflag:=true;
0790                         end;
0791
0792              hdwrst  :  begin
```

```
0793                      writeln('            RHINO HARDWARE RESET    ');
0794                      writeln('              System halted.       ');
0795                      writeln('         Reload control coefficients.');
0796                      go flag:=false;
0797                      readyflag:=true;
0798                     end;
0799
0800          mtrstl  :  begin
0801                      writeln('            RHINO STALLED           ');
0802                      writeln('              System halted.        ');
0803                      writeln('  Send CRM to clear stall condition.');
0804                      go flag:=false;
0805                     end;
0806
0807          ovr     :  begin
0808                      writeln('          RESEND LAST COMMAND       ');
0809                      writeln('Previous command code : ',prevcommand);
0810                     end;
0811
0812          dat     :  begin
0813                      if autorespond=true then
0814                        begin
0815                          datatype:=1;
0816                          get data;
0817                          if status=0 then
0818                             case code of
0819                                pk48  :  display 48data;
0820                                pk51  :  display 51data;
0821                                shrpk :  display shrdata;
0822                             end
0823                        end
0824                       else writeln('Peek data is ready');
0825                     end;
0826
0827          vdt     :  begin
0828                      if autorespond=true then
0829                        begin
0830                          datatype:=2;
0831                          get data;
0832                          if status=0 then display velocities;
0833                        end
0834                       else writeln('Velocity data ready');
0835                     end;
0836
0837          pdt     :  begin
0838                      if autorespond=true then
0839                        begin
0840                          datatype:=3;
0841                          get data;
0842                          if status=0 then display positions;
0843                        end
0844                       else writeln('Position data ready');
0845                     end;
0846
0847          rap     :  writeln('   ABSOLUTE POSITION COUNTER RESET');
0848
0849          ago     :  begin
0850                      go flag:=true;
0851                      writeln('       RHINO IN GO MODE');
0852                     end;
0853
0854          hlt     :  begin
0855                      go flag:=false;
0856                      writeln('      RHINO IN HALT MODE');
0857                     end;
0858
0859          crm     :  begin
0860                      readyflag:=true;
0861                      writeln('         REMAINING MOVES CLEARED');
0862                     end;
0863
0864          rcvdat  :  begin
0865                      writeln('         RECEIVED ALL DATA       ');
0866                      writeln('                                 ');
0867                      writeln('  previous code = ',prevcommand  );
0868                     end;
0869
0870          dci     :  writeln('   DISABLED COMMAND-ON-INTERRUPT');
0871
0872          snd51   :  writeln('   READY TO RECEIVE DATA          ');
0873
0874          inv     :  writeln('    INVALID DATA                  ');
0875
0876
0877                end   {cases of command code from 8751}
0878
0879
0880        else   {if the code is not one of the known commands}
0881          begin
0882            if autorespond=true then
0883                 writeln('  Unknown code from Rhino:', code  );
0884            if autorespond=false then
0885                 writeln('Data:', code                       );
0886                 end;
0887
0888      writeln(' ');
0889    end;
0890
0891  {*****************************************************************}
```

```
0892
0893    procedure Rhino command(mnemonic:char); {this procedure tests the mnemonic}
0894                                            {received from the user's keyboard}
0895        begin
0896
0897          status:=0;
0898          mnemonic set:=['G','N','D','C','K','W',
0899                         'S','X','H','A','R','E','V','P','5','4','M'];
0900
0901          if mnemonic in mnemonic set then
0902              begin
0903               case mnemonic of
0904
0905                'G'  :  begin
0906                          writeln('unassigned command');
0907                        end;
0908
0909                'N'  :  begin
0910                          command:=nop;
0911                          writeln('          NOP      ');
0912                          rhinowr(efl,chanl,command);
0913                          prevcommand:=command;
0914                        end;
0915
0916                'D'  :  begin
0917                          command:=dci;
0918                          writeln('          DCI      ');
0919                          send command get echo;
0920                        end;
0921
0922                'C'  :  begin
0923                          command:=crm;
0924                          writeln('          CRM      ');
0925                          send command get echo;
0926                          if status=0 then readyflag:=true;
0927                        end;
0928
0929                'K'  :  begin
0930                          command:=kff;
0931                          get screendata;
0932                          if status=0 then writeln ('          KFF    ');
0933                          if status=0 then load coefficients;
0934                        end;
0935
0936                'S'  :  begin
0937                          command:=shrpk;
0938                          get screendata;
0939                          if status=0 then writeln('         SHRPK   ');
0940                          if status=0 then get alldata;
0941                          if status=0 then display shrdata;
0942                        end;
0943
0944                'H'  :  begin
0945                          command:=hlt;
0946                          writeln('          HLT      ');
0947                          send command get echo;
0948                          if status=0 then go flag:=false;
0949                        end;
0950
0951                'A'  :  begin
0952                          command:=ago;
0953                          writeln('          AGO      ');
0954                          send command get echo;
0955                          if status=0 then go flag:=true;
0956                        end;
0957
0958                'X'  :  begin
0959                          readln(command);
0960                          rhinowr(efl,chanl,command);
0961                          status:=11;
0962                        end;
0963
0964                'R'  :  begin
0965                          command:=rap;
0966                          writeln('          RAP      ');
0967                          send command get echo;
0968                        end;
0969
0970                'E'  :  begin
0971                          command:=eci;
0972                          get screendata;
0973                          if status=0 then writeln ('          ECI      ');
0974                          if status=0 then enable coi;
0975                        end;
0976
0977                'V'  :  begin
0978                          command:=vrq;
0979                          writeln('        VRQ      ');
0980                          get alldata;
0981                          if status=0 then display velocities;
0982                        end;
0983
0984                'P'  :  begin
0985                          command:=prq;
0986                          writeln('        PRQ      ');
0987                          get alldata;
0988                          if status=0 then display positions;
0989                        end;
0990
```

```
0991                '5'  :  begin
0992                           command:=pk51;
0993                           get screendata;
0994                           if status=0 then writeln ('       51PK      ');
0995                           if status=0 then get alldata;
0996                           if status=0 then display 51data;
0997                          end;
0998
0999                '4'  :  begin
1000                           command:=pk48;
1001                           get screendata;
1002                           if status=0 then writeln ('       48PK      ');
1003                           if status=0 then get alldata;
1004                           if status=0 then display 48data;
1005                          end;
1006
1007                'W'  :  begin
1008                           command:=pk48;
1009                           get screendata;
1010                           if status=0 then writeln ('      WR48       ');
1011                           if status=0 then send 48data;
1012                          end;
1013
1014                'M'  :  begin
1015                           if readyflag=false then status:=3
1016                            else
1017                              begin
1018                              command:=mov;
1019                              get screendata;
1020                              if status=0 then writeln('        MOV     ');
1021                              if status=0 then send moves;
1022                              if status=0 then
1023                                begin
1024                                  writeln(' COMPUTED SPEEDS:');
1025                                  writeln
1026    ('     motor 1   motor 2   motor 3   motor 4   motor 5   motor 6   motor 7');
1027                                  writeln(2*veloc cmd[0],
1028                                      2*veloc cmd[1],2*veloc cmd[2],2*veloc cmd[3],
1029                                      2*veloc cmd[4],2*veloc cmd[5],2*veloc cmd[6]);
1030                                  writeln(' ');
1031                                  writeln('GO FLAG=',go flag);
1032                                 end
1033                              end
1034
1035                         end;   {the case of mnemonic = 'M'}
1036
1037               end; {the various cases of mnemonic}
1038
1039           writeln(' ');
1040
1041           case status of
1042
1043             0    :  writeln('communication sequence completed');
1044
1045             1    :  begin
1046                       writeln('Rhino requests resend.');
1047                       writeln('    Previous command was: ',prevcommand);
1048                      end;
1049
1050             2    :  writeln('Rhino response unrecognized:',code);
1051
1052             3    :  writeln('System not ready for next move.');
1053
1054             4    :  writeln('Command-on-interrupt enabled.');
1055
1056             5    :  writeln('Invalid data.');
1057
1058             6    :  writeln('Data communication fault.');
1059
1060             7    :  begin
1061                       writeln('At least one motor is STALLED ...Rhino halted.');
1062                       writeln('              Issue CRM to clear stall condition.');
1063                       go flag:=false;
1064                      end;
1065
1066             8    :  begin
1067                       writeln('HARDWARE RESET occurred ...Rhino halted.');
1068                       writeln('         Reload control coefficients.     ');
1069                       go flag:=false;
1070                      end;
1071
1072             9    :  writeln('8748 RAM PEEK in progress...');
1073
1074             10   :  writeln('Rhino communication interrupted from Keyboard.');
1075
1076             11   :  write('           ');
1077
1078            end; {various cases of status}
1079
1080          end {close the if statement for mnemonic in mnemonic set}
1081
1082         else   {mnemonic not in mnemonic set}
1083
1084              writeln('UNKNOWN RHINO COMMAND. RETRY.');
1085
1086       writeln(' ');
1087
1088       if delaystatus<0 then      {Rhino has data to transmit}
1089          begin
```

```
1090            writeln(' ..Rhino originated communication...');
1091            if delaystatus=-4 then
1092              begin
1093                datatype:=1;
1094                get data;
1095                if status=0 then
1096                   case code of
1097                     pk48  :  display 48data;
1098                     pk51  :  display 51data;
1099                     shrpk :  display shrdata;
1100                   end;
1101              end;
1102
1103            if delaystatus=-9 then
1104              begin
1105                 datatype:=2;
1106                 get data;
1107                 if status=0 then display velocities;
1108               end;
1109
1110            if delaystatus=-16 then
1111              begin
1112                 datatype:=3;
1113                 get data;
1114                 if status=0 then display positions;
1115               end;
1116
1117            delaystatus:=1;
1118
1119         end;
1120       end;
1121
1122      {*****************************************************************}
1123
1124   procedure user help;                        {this procedure displays the}
1125                                               {  meanings of the various  }
1126       begin                                   {      command mnemonics    }
1127
1128          writeln('The following MNEMONICS represent valid commands:      ');
1129          writeln('                                                       ');
1130          writeln('  ECI   = enable command-on-interrupt                  ');
1131          writeln('  DCI   = disable command-on-interrupt                 ');
1132          writeln('                                                       ');
1133          writeln('  AGO   = all motors go                                ');
1134          writeln('  HLT   = halt all motors                              ');
1135          writeln('  RAP   = reset absolute position counters             ');
1136          writeln('                                                       ');
1137          writeln('  KFF   = coefficients for control algorithm           ');
1138          writeln('  CRM   = clear remaining moves                        ');
1139          writeln('  MOV   = move                                         ');
1140          writeln('                                                       ');
1141          writeln('  WR48 = write to 8748 internal ram (poke)             ');
1142          writeln('  48PK  = 8748 internal ram peek                       ');
1143          writeln('  51PK  = 8751 internal ram peek                       ');
1144          writeln('  SHRPK  = shared RAM peek                             ');
1145          writeln('                                                       ');
1146          writeln('  VRQ   = velocity data request                        ');
1147          writeln('  PRQ   = position data request                        ');
1148          writeln('                                                       ');
1149          writeln('  X   = escape character: send numeric code directly   ');
1150        end;
1151
1152      {*****************************************************************}
1153
1154   procedure change flags;
1155
1156       begin
1157             writeln('The following flags can be altered:');
1158             writeln;
1159             writeln('     1  =  READYFLAG   ');
1160             writeln('     2  =  GO FLAG     ');
1161             writeln('     3  =  AUTORESPOND ');
1162             writeln;
1163             write  ('Enter number of flag to change: ');
1164
1165             read(chrc);
1166             readln(junkinput);
1167
1168             case chrc of
1169
1170               '1'  :  begin
1171                          writeln('READYFLAG = ',readyflag);
1172                          write  ('Do you want to change it? (Y/N) ');
1173                          read(chrc);
1174                          readln(junkinput);
1175                          if (chrc='y') or (chrc='Y') then
1176                                  if readyflag=true then readyflag:=false
1177                                     else readyflag:=true;
1178                       end;
1179
1180
1181               '2'   :  begin
1182                          writeln('GO FLAG = ',go flag);
1183                          write  ('Do you want to change it? (Y/N) ');
1184                          read(chrc);
1185                          readln(junkinput);
1186                          if (chrc='y') or (chrc='Y') then
1187                                  if go flag=true then go flag:=false
1188                                     else go flag:=true;
```

```
1189                        end;
1190
1191
1192                '3'   :  begin
1193                           writeln('AUTORESPOND = ',autorespond);
1194                           write  ('Do you want to change it? (Y/N) ');
1195                           read(chrc);
1196                           readln(junkinput);
1197                           if (chrc='y') or (chrc='Y') then
1198                                   if autorespond=true then autorespond:=false
1199                                      else autorespond:=true;
1200                        end;
1201
1202                end; {case statement}
1203
1204          end; {change flags procedure}
1205
1206
1207
1208                      {*****************************************}
1209                      {**                                     **}
1210                      {**      THE MAIN PROGRAM BEGINS HERE   **}
1211                      {**                                     **}
1212                      {*****************************************}
1213
1214
1215                           {INITIALIZE THE VARIABLES AND FLAGS}
1216
1217   begin
1218
1219      writeln('Enter device name');                  {get the name of the terminal}
1220      write('  for the RHINO port (TTBn:) .. ');     {line that the Rhino is con- }
1221                                                     {          nected to.        }
1222      readln(rhino);
1223
1224      modulomove:=32768;   {Maximum allowable move command is +/- 32767.}
1225
1226      status:=0;                    {Initially, status indicates no errors}
1227      delaystatus:=1;               { and no data waiting for transmission}
1228
1229      readyflag:=true;              {Initially, system is ready for a move.}
1230      go_flag:=false;               {Initially, motors are all halted.}
1231      autorespond:=true;            {Start program in automatic response mode.}
1232
1233
1234                    {INITIALIZE THE COMMUNICATION LINK}
1235
1236
1237      writeln(' To initialize communication,              ');
1238      writeln('                    press any key ...      ');
1239      writeln('                                           ');
1240
1241      ttread2(code);          {When first called, channels and event flags}
1242                              {are assigned for the Rhino and the keyboard}
1243
1244      writeln('Rhino assigned channel number = ',chanl);
1245      writeln;
1246
1247      if status=0 then  {a code from Rhino was received first}
1248           begin
1249             writeln('Code from Rhino = ',code  );
1250             writeln('                          ');
1251             code from 8751;
1252            end
1253
1254        else       {input from the keyboard was received first}
1255           begin
1256             writeln('  Sending HALT command to Rhino  ');
1257             writeln('                                 ');
1258             command:=hlt;
1259             send command get echo;
1260             if status<10 then
1261                 begin
1262                   writeln('Rhino communication established.');
1263                   writeln('                                 ');
1264                   writeln('     Code from Rhino = ', code   );
1265                   writeln('                                 ');
1266                  end;
1267           end;
1268
1269                           {INITIALIZATION IS COMPLETE}
1270
1271
1272
1273     {The next part of the main program monitors input from both the}
1274       {      user's keyboard and the Intel 8751 serial port.      }
1275
1276
1277      repeat
1278       begin                                  { Prompt the user for a command }
1279       writeln('COMMAND:');                   {and call the procedure to await}
1280       ttread2(code);                         {    code from either source.   }
1281                                              { Upon return from ttread2, one }
1282                                              {  of them has transmitted code.}
1283
1284       if termflag=0 then {code is from the keyboard rather than the Rhino}
1285         begin
1286           if (termcode>96) and (termcode<123) then        {change characters}
1287
```

```
1288                    termcode:=termcode-32;                  {  to upper case  }
1289
1290            chrc:=chr(termcode);                        {convert the 8-bit number }
1291                                                        {     to ASCII code.      }
1292
1293                                                      {Normally, the first character of}
1294                                                      {  the keyboard entry uniquely    }
1295                                                      {  identifies the command, but    }
1296            if chrc<>'X' then                         { if the entry is 'X', the escape}
1297                    readln(junkinput);                { character, what follows is the }
1298                                                      { numerical command code to send.}
1299
1300            mnemonic set:=['?','Z','F'];    {see if entry is from the set of}
1301            if chrc in mnemonic set then    {   internal program    commands.  }
1302               case chrc of
1303
1304                '?'  :  user help;
1305
1306                'F'  :  change flags;
1307
1308                'Z'  :  writeln('MAIN PROGRAM EXECUTION ENDS');
1309
1310                end  {cases of internal program commands}
1311
1312                                                {if command is not from the set of }
1313              else Rhino command(chrc);         {internal program commands, then go}
1314                                                {  see if it is a Rhino command.   }
1315
1316            end  {the if..begin statement for input from the user's keyboard}
1317
1318          else {input is from the Rhino rather than the keyboard}
1319             begin
1320                  if autorespond=true then {Rhino input is command code}
1321                    begin
1322                     writeln(' Rhino originated code ...    ');
1323                     writeln('                              ');
1324                     code from 8751;  {go interpret the Rhino}
1325                    end;              { communication code   }
1326
1327                if autorespond=false then {Rhino input may be command or data}
1328                     writeln(' code from rhino: ', code      );
1329
1330                writeln;
1331             end;
1332
1333         end; {the repeat loop}
1334
1335        until chrc='Z';   {return to start of repeat loop}
1336                          {unless keyboard entry was 'Z' }
1337
1338         end.
1339
1340
1341
1342
1343
1344
COMMAND QUALIFIERS
  PAS/LIS RHINOCOM
  /CHECK=(BOUNDS,NOCASE_SELECTORS,NOOVERFLOW,NOPOINTERS,NOSUBRANGE)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /NOENVIRONMENT
  /LIST=CS:[HOPKINS]RHINOCOM.LIS;81
  /OBJECT=CS:[HOPKINS]RHINOCOM.OBJ;89
  /NOCROSS_REFERENCE /ERROR_LIMIT=30 /NOG_FLOATING /NOMACHINE_CODE
  /NOOLD_VERSION /OPTIMIZE /NOSTANDARD /WARNINGS
COMPILER INTERNAL TIMING
  Phase          Faults        CPU Time       Elapsed Time
  Initialization     97        00:00.2          00:00.7
  Source Analysis   344        00:04.3          00:05.3
  Source Listing     25        00:01.8          00:02.8
  Tree Construction 397        00:02.2          00:04.8
  Flow Analysis      69        00:01.1          00:02.6
  Profit Analysis    75        00:01.2          00:02.0
  Context Analysis  454        00:13.0          00:23.9
  Name Packing       11        00:00.6          00:00.9
  Code Selection    109        00:03.0          00:05.4
  Final             170        00:06.3          00:08.6
  TOTAL            1759        00:33.7          00:57.2
COMPILATION STATISTICS
  CPU Time: 00:33.7       (2390 Lines/Minute)
  Elapsed Time:   00:57.2
  Page Faults:    1759
  Compilation Complete
```

APPENDIX W. LISTING OF THE 8751 ASSEMBLY LANGUAGE PROGRAM

This program was assembled on the Intel MDS-80 Development System, using the MCS-51 Macro Assembler, Version 2.0. The cross-reference and no-page-numbering options were specified.

```
ISIS-II MCS-51 MACRO ASSEMBLER V2.0
OBJECT MODULE PLACED IN .F1.P51V17.OBJ
ASSEMBLER INVOKED BY   ASM51 .F1.P51V17.ASM XREF NOPAGING

LOC  OBJ             LINE     SOURCE

                        1     ;               COMMUNICATION CODES FROM 8751 TO VAX
                        2     ;               -----------------------------------
                        3     ;
                        4     ; NOP   EQU   0H            ; NO OPERATION.
                        5     ;
  0011                  6     SND51   EQU   11H           ; SEND DATA TO 8751.
                        7     ;
  0022                  8     DCI     EQU   22H           ; COMMAND ON INTERRUPT DISABLED.
                        9     ;
  0033                 10     RDYNXT  EQU   33H           ; RHINO READY FOR NEXT MOVE.
                       11     ;
  0044                 12     DAT     EQU   44H           ; DATA FROM 8751 RAM, 8748 RAM, SHARED RAM,
                       13     ;                                       OR EXTERNAL LATCH FOLLOWS.
                       14     ;
  0055                 15     VDT     EQU   55H           ; VELOCITY DATA READY.
  0066                 16     PDT     EQU   66H           ; POSITION DATA READY.
                       17     ;
  0077                 18     RCVDAT  EQU   77H           ; DATA RECEIVED.
                       19     ;
  0088                 20     OVR     EQU   88H           ; SEND PREVIOUS CODE OVER AGAIN.
                       21     ;
  0099                 22     CRM     EQU   99H           ; CLEARED REMAINING MOVES.
                       23     ;
  00AA                 24     HDWRST  EQU   0AAH          ; HARDWARE RESET.
                       25     ;
  00BB                 26     HLT     EQU   0BBH          ; INSTRUCTED ALL MOTORS TO HALT.
  00CC                 27     AGO     EQU   0CCH          ; INSTRUCTED ALL MOTORS TO GO.
                       28     ;
  00DD                 29     MTRSTL  EQU   0DDH          ; RHINO MOTOR(S) STALLED.
                       30     ;
  00EE                 31     RAP     EQU   0EEH          ; ABSOLUTE POSITIONS RESET.
  00FF                 32     INV     EQU   0FFH          ; INVALID DATA.
                       33     ;
                       34     ;.............................
                       35     ;
                       36     ;               COMMUNICATION CODES FROM VAX TO 8751
                       37     ;               -----------------------------------
                       38     ;
                       39     ; NOP   EQU   00H           ;    NO OPERATION.
                       40     ;
  0011                 41      ECI    EQU   11H           ; ENABLE COMMAND ON INTERRUPT.
                       42     ;DCI    EQU   22H           ; DISABLE COMMAND ON INTERRUPT.
                       43     ;
  0033                 44      PK48   EQU   33H           ; 8748 INTERNAL RAM PEEK.
  0044                 45      PK51   EQU   44H           ; 8751 INTERNAL RAM (OR EXT LATCH) PEEK.
  0055                 46      VRQ    EQU   55H           ; REQUEST VELOCITY DATA.
  0066                 47      PRQ    EQU   66H           ; REQUEST POSITION DATA.
                       48     ;
  0077                 49      SNDVX  EQU   77H           ; SEND DATA FROM I/O BUFFER TO VAX.
                       50     ;OVR    EQU   88H           ; SEND PREVIOUS CODE OVER AGAIN.
                       51     ;
                       52     ;CRM    EQU   99H           ; CLEAR REMAINING MOVES IN ALL MOTORS.
                       53     ;
  00AA                 54      MOVE   EQU   0AAH          ; MOVE DATA FOLLOWS.
                       55     ;
                       56     ;HLT    EQU   0BBH          ; HALT ALL MOTORS.
                       57     ;AGO    EQU   0CCH          ; INSTRUCT ALL MOTORS TO GO.
                       58     ;
  00DD                 59      SHRPK  EQU   0DDH          ; RAM PEEK (TO SHARED RAMS).
                       60     ;
                       61     ;RAP    EQU   0EEH          ; RESET ABSOLUTE POSITIONS OF ALL MOTORS.
                       62     ;
```

```
00FF                63      KFF     EQU   0FFH          ; COEFFICIENTS FOLLOW.
                    64      ;
                    65      ;.........................
                    66      ;
                    67      ;             COMMUNICATION CODES FROM 8751 TO 8748
                    68      ;             -------------------------------------
                    69      ;
                    70      ;NOP    EQU   0H            ; NO OPERATION.
                    71      ;                           ;
                    72      ;OVR    EQU   88H           ; RESEND LAST CODE BYTE.
                    73      ;                           ;
                    74      ;CRM    EQU   99H           ; CLEAR ALL REMAINING MOVES.
                    75      ;MOVE   EQU   0AAH          ; GET MOVE DATA FROM RAM.
                    76      ;                           ;
                    77      ;HLT    EQU   0BBH          ; HALT.
                    78      ;AGO    EQU   0CCH          ; GO.
                    79      ;                           ;
00DD                80      PK48RM  EQU   0DDH          ; READ 8748 INTERNAL RAM ADDRESS.
                    81      ;                           ;
                    82      ;RAP    EQU   0EEH          ; RESET ABSOLUTE POSITION.
                    83      ;KFF    EQU   0FFH          ; GET COEFFICIENT DATA FROM RAM.
                    84      ;                           ;
                    85      ;.........................
                    86      ;
                    87      ;          COMMUNICATION CODES FROM 8748 TO 8751
                    88      ;          -------------------------------------
                    89      ;
                    90      ; NOP       EQU   00H       ; NO OPERATION
                    91      ;                           ;
                    92      ; OVR       EQU   88H       ; SEND PREVIOUS OVER AGAIN.
                    93      ; 48STAL    EQU   99H       ; MOTOR IS STALLED
                    94      ; HDWRST    EQU   0AAH      ; HARDWARE RESET UNDERWAY
                    95      ; RDYMOV    EQU   0BBH      ; READY FOR NEXT MOVE
                    96      ; RDYPK     EQU   0CCH      ; PEEK DATA LOADED INTO SHARED RAM.
                    97      ;
                    98      ;.........................
                    99      ;
                   100      ;          INTERNAL RAM LOCATION SYMBOLS
                   101      ;          -----------------------------
                   102      ;
0010               103      SRRDBF      EQU   10H       ; (16) INPUT BUFFER FOR CODE FROM VAX.
0011               104      RDBOT       EQU   11H       ; (17) RAM LOCATION 9H TO 17H ARE THE INPUT
001F               105      RDTOP       EQU   1FH       ; (31) DATA BUFFERS FOR VAX/8751 DATA.
                   106      ;                           ;
0020               107      RSET48      EQU   20H       ; (32) 8748 HARDWARE RESET STATUS BYTE.
0021               108      RDY48       EQU   21H       ; (33) 8748 READY (FOR NEXT MOVE) STATUS BYTE.
0022               109      STLL48      EQU   22H       ; (34) 8748 STALL STATUS BYTE.
                   110      ;                           ;
0023               111      NEWSTL      EQU   23H       ; (35) 'NEW STALL HAS OCCURED' FLAG.
                   112      ;                           ;
0024               113      MTRLNK      EQU   24H       ; (36) EACH MOTOR LINKED HAS CORRESP BIT SET.
0025               114      MTRNMR      EQU   25H       ; (37) MOTOR NUMBER.
0026               115      BITLOC      EQU   26H       ; (38) BIT LOCATION.
                   116      ;                           ;
0028               117      RAMRDY      EQU   28H       ; (40) '8748 PEEK DATA READY' FLAGS.
0029               118      RAMADR      EQU   29H       ; (41) ADDRESS TO BE READ FOR PEEK COMMANDS.
                   119      ;                           ;
002A               120      RNOCKL      EQU   2AH       ; (42) RHINO CLOCK LOW BYTE.
002B               121      RNOCKH      EQU   2BH       ; (43) RHINO CLOCK HIGH BYTE.
                   122      ;                           ;
002C               123      INTCMD      EQU   2CH       ; (44) INTERRUPT COMMAND LOCATION.
002D               124      INTCDB      EQU   2DH       ; (45) INTERRUPT COMMAND BUFFER LOCATION.
                   125      ;                           ;
002E               126      VAXBUF      EQU   2EH       ; (46) VAX COMMAND BUFFER LOCATION.
002F               127      VAXCMD      EQU   2FH       ; (47) VAX COMMAND LOCATION.
                   128      ;                           ;
0030               129      GLOBL       EQU   30H       ; (48) GLOBAL 8748 WRITE BUFFER.
0030               130      PRVMTR      EQU   30H       ; (LOCATIONS 31H-37H HOLD PREVIOUS
                   131      ;                                          CODES TO THE 8748'S)
                   132      ;                           ;
0038               133      XMTFLG      EQU   38H       ; (56) THIS LOCATION HOLDS THE TRANSMIT FLAGS.
                   134      ;                           ;
```

```
0039             135     NTBUFF      EQU   39H         ; (57) 'NT' BUFFER (SEE NEXT LOCATION).
003A             136     NT          EQU   3AH         ; (58) NUMBER REMAINING DATA BYTES TO TRANSMIT.
003B             137     NRBUFF      EQU   3BH         ; (59) 'NR' BUFFER (SEE NEXT LOCATION).
003C             138     NR          EQU   3CH         ; (60) NUMBER REMAINING DATA BYTES TO RECEIVE.
                 139                                   ;
003D             140     PRVXWR      EQU   3DH         ; (61) PREVIOUS WRITE TO VAX.
003E             141     RESPCD      EQU   3EH         ; (62) RESPONSE CODE LOCATION.
003F             142     SRWRBF      EQU   3FH         ; (63) WRITE BUFFER LOCATION.
                 143                                   ;
0040             144     WRBOT       EQU   40H         ; (64) LOCATIONS 40H-4FH ARE THE DATA OUTPUT
004F             145     WRTOP       EQU   4FH         ;   BUFFERS FOR 8751/VAX COMMUNICATIONS.
                 146                                   ;
004F             147     STKBAS      EQU   4FH         ; (79) LOCATION BELOW START OF STACK.
                 148     ;
                 149     ;.............................
                 150     ;
                 151     ;               DEVICE ADDRESSES (THROUGH PORT 2)
                 152     ;               ---------------------------------
                 153     ;
0000             154     IDLE    EQU   00H             ; DURING 'IDLE', READ MAILBOX FULL FLAGS ENABLED.
                 155                                   ;
0018             156     MTRAM1  EQU   18H             ; ADDRESS TO ENABLE RAM OF MOTOR ONE.
                 157                                   ;
0080             158     GLBLWR  EQU   80H             ; ADDRESS TO WRITE GLOBALLY TO PIGEON-HOLES.
                 159                                   ;
0088             160     LIMLCH  EQU   88H             ; ADDRESS TO ENABLE LIMIT SWITCH LATCH.
0098             161     LNKLCH  EQU   98H             ; ADDRESS OF MOTORS LINKED LATCH.
                 162                                   ;
                 163     ;XTLCH1 EQU   10101XXXB       ; ADDRESS OF EXTRA LATCH 1.
                 164     ;XTLCH2 EQU   10111XXXB       ; ADDRESS OF EXTRA LATCH 2.
                 165     ;XTLCH3 EQU   11001XXXB       ; ADDRESS OF EXTRA LATCH 3.
                 166     ;XTLCH4 EQU   11011XXXB       ; ADDRESS OF EXTRA LATCH 4.
                 167     ;XTLCH5 EQU   11101XXXB       ; ADDRESS OF EXTRA LATCH 5.
                 168     ;XTLCH6 EQU   11111XXXB       ; ADDRESS OF EXTRA LATCH 6.
                 169     ;
                 170     ;.............................
                 171     ;
                 172     ;               MISCELLANEOUS BIT MASKS AND VALUES:
                 173     ;               -----------------------------------
                 174     ;
00FD             175     BAUD96  EQU   0FDH            ; THIS RELOAD VALUE COMPUTED FOR
                 176                                   ; TIMER ONE BAUD RATE GENERATION OF 9600.
                 177                                   ;
00FA             178     BAUD48  EQU   0FAH            ; THIS RELOAD VALUE COMPUTED FOR
                 179                                   ; TIMER ONE BAUD RATE GENERATION OF 4800.
                 180                                   ;
0002             181     MTR1BT  EQU   2H              ; MOTOR 1 BIT LOCATION IS BIT ONE.
                 182     ;
                 183     ;.............................
                 184     ;
                 185     ;               ALLOCATION OF SHARED RAM
                 186     ;               ------------------------
                 187     ;
0020             188     CF1     EQU   20H             ; VELOCITY ERROR COEFFICIENT 1 (1-31 HPS)
0021             189     CF2     EQU   21H             ; VELOCITY ERROR COEFFICIENT 2 (32-63 HPS)
0022             190     CF3     EQU   22H             ; VELOCITY ERROR COEFFICIENT 3 (64-127 HPS)
0023             191     CF4     EQU   23H             ; VELOCITY ERROR COEFFICIENT 4 (128+ HPS)
                 192                                   ;
0024             193     MG1     EQU   24H             ; MAX ERROR MAGNITUDE (1-31 HPS).
0025             194     MG2     EQU   25H             ; MAX ERROR MAGNITUDE (32-63 HPS)
0026             195     MG3     EQU   26H             ; MAX ERROR MAGNITUDE (64-127 HPS)
0027             196     MG4     EQU   27H             ; MAX ERROR MAGNITUDE (128+ HPS)
                 197                                   ;
0028             198     RAMDAT  EQU   28H             ; DATA FOR 8748 POKE OR FROM 8748 PEEK.
                 199     ;RAMADR EQU   29H             ; 8748 RAM ADDRESS TO BE PEEKED OR POKED.
                 200                                   ;
002A             201     SPEED   EQU   2AH             ; MOTOR VELOCITY.
                 202                                   ;
002C             203     NXMVNF  EQU   2CH             ; 'NEXT MOVE INFINITE' FLAG.
002D             204     RCVDNM  EQU   2DH             ; 'RECEIVED NEXT MOVE' FLAG.
002E             205     RAMVEL  EQU   2EH             ; VELOCITY COMMAND LOCATION.
002F             206     LOBTMV  EQU   2FH             ; LOW BYTE MOVE LOCATION.
```

```
0030                207     HIBTMV  EQU   30H             ; HIGH BYTE MOVE LOCATION.
0031                208     NXMVSN  EQU   31H             ; NEXT MOVE SIGN LOCATION.
                    209                                   ;
0033                210     LOBTPS  EQU   33H             ; LOW BYTE ABSOLUTE POSITION.
0034                211     HIBTPS  EQU   34H             ; HIGH BYTE ABSOLUTE POSITION.
                    212                                   ;
0038                213     GRIPPR  EQU   38H             ; GRIPPER PULSEWIDTH.
                    214                                   ;
0039                215     HYSTPW  EQU   39H             ; HYSTERESIS VALUE FOR MOTOR PULSEWIDTH.
003A                216     STRTPW  EQU   3AH             ; STARTING PULSEWIDTH.
003B                217     LONGOV  EQU   3BH             ; PULSEWIDTH VALUE FOR LONG OVERSHOOT.
003C                218     SHRTOV  EQU   3CH             ; PULSEWIDTH VALUE FOR SHORT OVERSHOOT.
                    219     ;
                    220     ;.....................................................................
                    221     ;.....................................................................
                    222     ;
                    223     ;            INTERRUPT VECTORING:
                    224     ;
                    225             ORG   0H              ; VECTOR LOCATION FOR HARDWARE RESET.
0000 012C           226             AJMP  BEGIN0          ; JUMP TO INITIALIZATION ROUTINE.
                    227                                   ;
                    228                                   ;
0003                229             ORG   3H              ; VECTOR LOCATION FOR INTERRUPT ZERO.
0003 C136           230             AJMP  INTRP0          ; JUMP TO INTERRUPT 0 SERVICE ROUTINE.
                    231                                   ;
0008                232             ORG   8H              ; UNUSED INTERRUPT VECTOR (TIMER 0).
0008 32             233             RETI                  ;
                    234                                   ;
0013                235             ORG   13H             ; UNUSED INTERRUPT VECTOR (INTERRUPT 1).
0013 32             236             RETI                  ;
                    237                                   ;
001B                238             ORG   1BH             ; UNUSED INTERRUPT VECTOR (TIMER 1).
001B 32             239             RETI                  ;
                    240                                   ;
                    241                                   ;
0023                242             ORG   23H             ; VECTOR LOCATION FOR SERIAL PORT INTERRUPT.
0023 A194           243             AJMP  SRLISR          ; JUMP TO INTERRUPT SERVICE ROUTINE.
                    244                                   ;
                    245                                   ;
002B                246             ORG   2BH             ; UNUSED INTERRUPT VECTOR (ON 8752).
002B 32             247             RETI                  ;
                    248     ;
                    249     ;.....................................................................
                    250     ;
                    251     ;            INITIALIZATION PROCEDURE
                    252     ;                        (ENTERED ON POWER-UP)
                    253     ;
002C 75A080         254     BEGIN0: MOV   P2,#GLBLWR      ; WRITE
002F E4             255             CLR   A               ;     'NO OPERATION' CODE
0030 F2             256             MOVX  @R0,A           ;          GLOBALLY (I.E., TO ALL MAILBOXES)
                    257                                   ;
0031 F8             258             MOV   R0,A            ; CLEAR INTERNAL RAM
0032 F6             259     BEGLP1: MOV   @R0,A           ;     LOCATIONS 0 THROUGH 127
0033 D8FD           260             DJNZ  R0,BEGLP1       ;
                    261                                   ;
0035 75A098         262             MOV   P2,#LNKLCH      ; READ IN 'MOTORS LINKED' DATA
0038 E590           263             MOV   A,P1            ;
003A 7FFA           264             MOV   R7,#BAUD48      ;
003C 20E002         265             JB    ACC.0,BEG48     ; IF BIT ZERO IS SET, BAUD RATE IS 4800.
003F 7FFD           266             MOV   R7,#BAUD96      ;  OTHERWISE, IT IS 9600.
0041 54FE           267     BEG48:  ANL   A,#0FEH         ;
0043 F524           268             MOV   MTRLNK,A        ; STORE MOTOR LINKED DATA.
0045 75A000         269             MOV   P2,#IDLE        ;
                    270                                   ;
0048 852420         271             MOV   RSET48,MTRLNK   ; INITIALIZE THE HARDWARE RESET BYTE.
                    272     ;
                    273     ;           THE FOLLOWING LOOP IS ITERATED UNTIL 'HARDWARE RESET'
                    274     ;           CODE IS RECEIVED FROM ALL LINKED MOTORS:
                    275     ;
004B 752507         276     BEGLP2: MOV   MTRNMR,#7H      ; START WITH MOTOR NUMBER SEVEN,
004E 752680         277             MOV   BITLOC,#80H     ;    WHICH CORRESPONDS TO BIT LOCATION 7.
                    278                                   ;
```

```
0051 E590          279  BEGLP3: MOV   A,P1          ; ISOLATE THE
0053 5524          280          ANL   A,MTRLNK      ;     'READ MAILBOX FULL' FLAG
0055 5526          281          ANL   A,BITLOC      ;        OF THE 'N'TH MOTOR, IF IT IS LINKED.
0057 6011          282          JZ    NXMTR1        ; IF NOT FULL, GO ON TO NEXT MOTOR.
                   283                             ;
0059 E525          284          MOV   A,MTRNMR      ; BUT IF FULL,
005B C4            285          SWAP  A             ;     READ IN THE CODE FROM THE
005C F5A0          286          MOV   P2,A          ;        'N'TH MOTOR'S MAILBOX.
005E E2            287          MOVX  A,@R0         ;
005F 64AA          288          XRL   A,#HDWRST     ; IF CODE IS NOT 'HARDWARE RESET',
0061 7007          289          JNZ   NXMTR1        ;        GO ON TO NEXT MOTOR.
                   290                             ;
0063 E526          291          MOV   A,BITLOC      ; IF CODE IS 'HARDWARE RESET',
0065 F4            292          CPL   A             ;     CLEAR THE 'N'TH MOTOR'S
0066 5520          293          ANL   A,RSET48      ;        RESET FLAG.
0068 F520          294          MOV   RSET48,A      ;
                   295                             ;
006A E520          296  NXMTR1: MOV   A,RSET48     ; CHECK IF ALL MOTORS HAVE SENT RESET CODE.
006C 600C          297          JZ    BEGFIN        ; IF SO, CONTINUE THE INITIALIZATION PROCEDURE.
006E D52502        298          DJNZ  MTRNMR,NXMTR2 ; IF NOT, GO CHECK THE NEXT MOTOR'S MAILBOX.
0071 8008          299          JMP   BEGLP2        ; IF RESULT IS ZERO, GO START WITH MOTOR 7 AGAIN.
                   300                             ;
0073 E526          301  NXMTR2: MOV   A,BITLOC      ; DETERMINE THE BIT WHICH CORRESPONDS
0075 03            302          RR    A             ;    TO THE PARTICULAR MOTOR NUMBER
0076 F526          303          MOV   BITLOC,A      ;        WHICH IS TO BE CHECKED NEXT.
0078 0151          304          AJMP  BEGLP3        ;
                   305                             ;
007A 75814F        306  BEGFIN: MOV   SP,#STKBAS    ; INITIALIZE THE STACK POINTER AT 79.
                   307          ;.....................
                   308          ;                        INITIALIZE THE SERIAL PORT:
                   309          ;                        ---------------------------
                   310          ;
007D D29E          311          SETB  SCON.6        ; SET SERIAL PORT TO MODE 1.
                   312                             ;
007F 438920        313          ORL   TMOD,#20H     ; SET TIMER 1 MODE TO 8-BIT, AUTO-RELOAD.
0082 8F8D          314          MOV   TH1,R7        ; PUT THE RELOAD VALUE INTO TIMER 1 HIGH BYTE.
0084 D28E          315          SETB  TCON.6        ; START TIMER 1 (STARTS BAUD RATE GENERATION).
                   316                             ;
0086 C299          317          CLR   SCON.1        ; CLEAR TRANSMIT INTERRUPT FLAG.
                   318                             ;
0088 C298          319          CLR   SCON.0        ; CLEAR RECEIVE INTERRUPT FLAG.
008A D29C          320          SETB  SCON.4        ; ENABLE SERIAL PORT RECEIVER (REN).
                   321                             ;
008C D2BC          322          SETB  IP.4          ; SET SERIAL PORT INTERRUPT TO HIGHEST PRIORITY.
008E D2AC          323          SETB  IE.4          ; ENABLE THE SERIAL PORT INTERRUPT.
                   324          ;.....................................................
                   325          ;
0090 7599AA        326          MOV   SBUF,#HDWRST  ; PUT HARDWARE RESET CODE IN SERIAL BUFFER.
0093 753DAA        327          MOV   PRVXWR,#HDWRST; ALSO PUT IT INTO PREVIOUS VAX WRITE.
                   328                             ;
0096 852421        329          MOV   RDY48,MTRLNK  ; INITIALIZE MOTOR READY FLAGS (NOT READY).
0099 052A          330          INC   RNOCKL        ; INCREMENT RHINO CLOCK TO 1.
                   331                             ;
009B D288          332          SETB  TCON.0        ; SET INTERRUPT ZERO TO EDGE-TRIGGERED MODE.
009D D2A8          333          SETB  IE.0          ; ENABLE INTERRUPT ZERO.
                   334                             ;
009F D2AF          335          SETB  IE.7          ; ENABLE ALL INTERRUPTS.
                   336          ;
                   337          ;.....................................................................
                   338          ;.....................................................................
                   339          ;
                   340          ;              MAIN ROUTINE POLLS 'READ MAILBOX FULL'
                   341          ;              FLAGS, AND EXECUTES ANY VAX COMMANDS.
                   342          ;              INTERRUPTED BY SERIAL PORT OR INTERRUPT ZERO.
                   343          ;
00A1 75A000        344  MAIN:   MOV   P2,#IDLE     ; CHECK THE
00A4 E590          345          MOV   A,P1          ;     'READ MAILBOX FULL' FLAGS
00A6 5524          346          ANL   A,MTRLNK      ;        OF ALL LINKED MOTORS.
00A8 6002          347          JZ    NOCOMM        ;
00AA 3180          348          ACALL RESPND        ; IF ANY MAILBOX IS FULL, CALL 'RESPOND'.
                   349                             ;
00AC E524          350  NOCOMM: MOV   A,MTRLNK      ; IF NO MOTORS ARE LINKED,
```

```
00AE 6023              351              JZ    INCODE          ;    THEN SYSTEM IS IN DEBUG MODE,
                       352                                    ;         SO SKIP THE FOLLOWING TESTS.
00B0 E528              353              MOV   A,RAMRDY        ; CHECK 'PEEK DATA READY' FLAGS.
00B2 6524              354              XRL   A,MTRLNK        ;   IF ALL LINKED MOTORS HAVE PEEK
00B4 54FE              355              ANL   A,#0FEH         ;      DATA READY, THEN GO SEND
00B6 6047              356              JZ    GETRAM          ;         PEEK DATA TO THE VAX.
                       357                                    ;
00B8 E520              358              MOV   A,RSET48        ; IF ANY LINKED MOTOR HAS
00BA 6002              359              JZ    STLCHK          ;    RESET, GO NOTIFY THE VAX.
00BC 2140              360              AJMP  NEWRST          ;
                       361                                    ;
00BE E522              362    STLCHK:   MOV   A,STLL48        ; IF ANY MOTOR HAS STALLED,
00C0 6002              363              JZ    RDYCHK          ;   SPECIAL HANDLING IS REQUIRED.
00C2 214D              364              AJMP  STAL48          ;
                       365                                    ;
00C4 E521              366    RDYCHK:   MOV   A,RDY48         ; IF ALL MOTORS
00C6 700B              367              JNZ   INCODE          ;    ARE READY FOR
00C8 753E33            368              MOV   RESPCD,#RDYNXT  ;      NEXT MOVE, THEN
00CB 853E3D            369              MOV   PRVXWR,RESPCD   ;        NOTIFY THE VAX.
00CE 852421            370              MOV   RDY48,MTRLNK    ;
00D1 31F4              371              ACALL WRITVX          ;
                       372                                    ;
00D3 E53A              373    INCODE:   MOV   A,NT            ; IF ANY DATA REMAINS TO TRANSMIT TO VAX,
00D5 70CA              374              JNZ   MAIN            ;       DO NOT DECODE NEXT COMMAND.
                       375                                    ;
00D7 E52F              376              MOV   A,VAXCMD        ; IF A VAX COMMAND IS WAITING
00D9 7006              377              JNZ   DECODE          ;   TO EXECUTE, GO DECODE IT.
00DB 852E2F            378              MOV   VAXCMD,VAXBUF   ; ELSE, MOVE IN ANY NEW COMMAND FROM VAX BUFFER,
00DE 752E00            379              MOV   VAXBUF,#0H      ;          AND CLEAR THE READ BUFFER.
                       380                                    ;
00E1 E52F              381    DECODE:   MOV   A,VAXCMD        ; CHECK THE VAX COMMAND LOCATION.
00E3 6002              382              JZ    BKCOM1          ; IF ZERO, THERE IS NO COMMAND TO EXECUTE.
00E5 5144              383              ACALL DCDVAX          ; IF NOT ZERO, CALL DECODE VAX COMMAND.
                       384                                    ;
00E7 E53A              385    BKCOM1:   MOV   A,NT            ; BEFORE EXECUTING THE 'INTERRUPT COMMAND',
00E9 453C              386              ORL   A,NR            ;     ALL PRESENT AND PENDING
00EB 4539              387              ORL   A,NTBUFF        ;        DATA COMMUNICATIONS MUST BE
00ED 453B              388              ORL   A,NRBUFF        ;           COMPLETED.
00EF 70B0              389              JNZ   MAIN            ;
                       390                                    ;
00F1 E52D              391              MOV   A,INTCDB        ; LOAD ANY NON-ZERO INTERRUPT COMMAND
00F3 60AC              392              JZ    MAIN            ;    INTO THE VAX COMMAND LOCATION,
00F5 852D2F            393              MOV   VAXCMD,INTCDB   ;       AND GO DECODE IT.
00F8 752D00            394              MOV   INTCDB,#0H      ;
00FB 5144              395              ACALL DCDVAX          ;
00FD 01A1              396              AJMP  MAIN            ;
                       397                                    ;
                       398    ;.......................
                       399
00FF 75A018            400    GETRAM:   MOV   P2,#MTRAM1      ; LOAD THE 8748 PEEK DATA INTO THE WRITE
0102 752602            401              MOV   BITLOC,#MTR1BT  ;   BUFFER AREA, WITH MOTOR 1 DATA AT THE
0105 7A07              402              MOV   R2,#7H          ;    BOTTOM OF THE BUFFER.
0107 7840              403              MOV   R0,#WRBOT       ;
0109 7928              404              MOV   R1,#RAMDAT      ; THE DATA FROM EACH 8748 IS AT
                       405                                    ;    THE SAME ADDRESS IN EACH SHARED RAM.
010B E590              406    GETLP1:   MOV   A,P1            ;
010D F4                407              CPL   A               ; GET THE DATA FROM ONE SHARED RAM
010E 5524              408              ANL   A,MTRLNK        ;    AT A TIME.  THIS INVOLVES POLLING
0110 5526              409              ANL   A,BITLOC        ;      AN INDIVIDUAL SHARED RAM LOCKOUT FLAG
0112 70F7              410              JNZ   GETLP1          ;         UNTIL ACCESS IS GAINED.
                       411                                    ;
0114 E3                412              MOVX  A,@R1           ; ONCE ACCESS IS GAINED, MOVE IN THE PEEK
0115 F6                413              MOV   @R0,A           ;    DATA TO THE SPECIFIED WRITE BUFFER LOCATION.
                       414                                    ;
0116 E526              415              MOV   A,BITLOC        ; COMPUTE THE BIT CORRESPONDING TO THE NEXT
0118 23                416              RL    A               ;  MOTOR NUMBER OF THE NEXT SHARED RAM TO BE READ.
0119 F526              417              MOV   BITLOC,A        ;
                       418                                    ;
011B E5A0              419              MOV   A,P2            ; COMPUTE THE ADDRESS OF THE
011D 2410              420              ADD   A,#10H          ;   SHARED RAM ENABLE FOR THE NEXT
011F F5A0              421              MOV   P2,A            ;      MOTOR TO BE READ.
                       422                                    ;
```

```
0121 08          423             INC   R0              ; INCREMENT R0 TO NEXT WRITE BUFFER ADDRESS.
                 424             ;
0122 DAE7        425             DJNZ  R2,GETLP1       ; SEE IF ALL SEVEN RAMS HAVE BEEN READ.
                 426             ;
0124 A629        427             MOV   @R0,RAMADR      ; ABOVE THE RAM PEEK DATA IN THE WRITE
0126 08          428             INC   R0              ;  BUFFER, PUT THE ADDRESS THAT WAS READ,
0127 A62A        429             MOV   @R0,RNOCKL      ;     THE LOW AND HIGH BYTES OF THE
0129 08          430             INC   R0              ;        'RHINO CLOCK', AND THE COMMAND
012A A62B        431             MOV   @R0,RNOCKH      ;           WHICH INITIATED THE PEEK (PK48).
012C 08          432             INC   R0              ;
012D 7633        433             MOV   @R0,#PK48       ;
                 434             ;
012F 75390B      435             MOV   NTBUFF,#0BH     ; SET NUMBER OF BYTES TO TRANSMIT TO 11.
0132 752800      436             MOV   RAMRDY,#0       ; CLEAR THE RAM READY FLAGS.
                 437             ;
0135 753E44      438             MOV   RESPCD,#DAT     ; LOAD PEEK DATA READY AS RESPONSE.
0138 853E3D      439             MOV   PRVXWR,RESPCD   ; ALSO SAVE IT IN PREVIOUS WRITE TO VAX.
                 440             ;
013B 1201F4      441             CALL  WRITVX          ; GO WRITE THE CODE TO VAX.
                 442             ;
013E 01A1        443             AJMP  MAIN            ;
                 444             ;
                 445             ;
                 446             ;
                 447     ;.....................;
                 448             ;
0140 753EAA      449     NEWRST: MOV   RESPCD,#HDWRST  ; IN CASE SOME 8748 HAS UNDERGONE A HARDWARE
0143 853E3D      450             MOV   PRVXWR,RESPCD   ;   RESET, SEND THE 'HARDWARE RESET' CODE
0146 752000      451             MOV   RSET48,#0H      ;     TO THE VAX, AND CLEAR THE RESET FLAGS.
0149 31F4        452             ACALL WRITVX          ;
014B 01A1        453             AJMP  MAIN            ;
                 454             ;
                 455     ;.....................;
                 456             ;
014D E523        457     STAL48: MOV   A,NEWSTL        ; INSPECT THE 'NEW STALL' FLAG.
014F 600B        458             JZ    STLRSP          ; IF NOT SET, GO CHECK VAX STALL RESPONSE.
                 459             ;
0151 752300      460             MOV   NEWSTL,#0H      ; IF 'NEW STALL' FLAG IS SET, THEN
0154 753EDD      461             MOV   RESPCD,#MTRSTL  ;   NOTIFY THE VAX OF THE STALL.
0157 853E3D      462             MOV   PRVXWR,RESPCD   ;
015A 31F4        463             ACALL WRITVX          ;
                 464             ;
015C E52F        465     STLRSP: MOV   A,VAXCMD        ; GET VAX STALL RESPONSE COMMAND,
015E 7006        466             JNZ   CHKCMD          ;    EITHER FROM THE VAX COMMAND LOCATION
0160 852E2F      467             MOV   VAXCMD,VAXBUF   ;       OR FROM THE VAX COMMAND BUFFER.
0163 752E00      468             MOV   VAXBUF,#0H      ;
                 469             ;
0166 E52F        470     CHKCMD: MOV   A,VAXCMD        ; DURING A STALL, THE 'MOVE' AND
0168 7002        471             JNZ   ISITGO          ;   THE 'GO' COMMANDS ARE NOT ALLOWED.
016A 01E7        472             AJMP  BKCOM1          ;
                 473             ;
016C 64CC        474     ISITGO: XRL   A,#AGO          ; IF EITHER OF THOSE
016E 6006        475             JZ    NOTALW          ;   COMMANDS IS RECEIVED
                 476             ;                     ;      FROM THE VAX, THEN
0170 E52F        477             MOV   A,VAXCMD        ;        RESEND THE 'MOTOR STALLED' CODE
0172 64AA        478             XRL   A,#MOVE         ;           AND SUBSTITUTE THE 'HALT'
0174 7008        479             JNZ   CODEOK          ;              FOR THE ONE SENT FROM VAX.
                 480             ;                     ; IF CODE IS NEITHER 'MOVE' NOR 'GO',
0176 752FBB      481     NOTALW: MOV   VAXCMD,#HLT     ;   THEN DECODE IT AS USUAL.
0179 753EDD      482             MOV   RESPCD,#MTRSTL  ;
017C 31F4        483             ACALL WRITVX          ;
017E 01E1        484     CODEOK: AJMP  DECODE          ;
                 485     ;
                 486     ;.....................................................................
                 487     ;.....................................................................
                 488     ;
                 489     ;        'RESPOND TO 8748 MAILBOX CODE' SUBROUTINE:
                 490     ;
                 491                                   ; PROVIDE:  NOTHING
                 492                                   ;
                 493                                   ; RETURNS:  APPROPRIATE CHANGES TO 8748
                 494                                   ;           STATUS FLAGS (RSET48, STLL48, AND
```

```
                                          ;       RDY48).  HALTS ALL MOTORS IF ONE OF
                                          ;       THEM IS STALLED, AND SETS STALL
                                          ;       FLAG TO INDICATE A NEW STALL.
                                          ;       RESENDS PREVIOUS CODE TO 8748 IF
                                          ;       REQUESTED.
                                          ;
                                          ; CHANGES:  ACCUM, P2, PSW, BITLOC, MTRNMR
                                          ;
                                          ; CALLS:  'DECODE 8748 CODE'   ('DCD48')
                                          ;
                                          ; CALLED BY:  MAIN ROUTINE
                                          ;             'GLOBAL WRITE'   ('GLOBAL')
                                          ;
0180 752507    RESPND: MOV   MTRNMR,#7H   ; START WITH MOTOR NUMBER SEVEN MAILBOX.
0183 752680            MOV   BITLOC,#80H  ; LOAD IN CORRESPONDING BIT LOCATION.

0186 E590      RESLP1: MOV   A,P1         ; CHECK IF THE 'N'TH MOTOR'S
0188 5524              ANL   A,MTRLNK     ;   'READ MAILBOX FULL' FLAG IS SET.
018A 5526              ANL   A,BITLOC
018C 600B              JZ    NXMTR3       ; IF NOT FULL, GO CHECK NEXT MOTOR'S MAILBOX.
                                          ;
018E E525              MOV   A,MTRNMR     ; IF FULL, COMPUTE THE
0190 C4                SWAP  A            ;   ADDRESS TO ENABLE THIS
0191 F5A0              MOV   P2,A         ;     MOTOR'S READ MAILBOX, AND
0193 E3                MOVX  A,@R1        ;       READ IN THE CODE FROM IT.
                                          ;
0194 31A2              ACALL DCD48        ; CALL THE DECODE 8748 SUBROUTINE.
                                          ;
0196 75A000            MOV   P2,#IDLE     ; CLEAR THE ENABLE ADDRESS FROM PORT 2.
                                          ;
0199 E526      NXMTR3: MOV   A,BITLOC     ; COMPUTE THE BIT LOCATION
019B 03                RR    A            ;   CORRESPONDING TO THE NEXT MOTOR.
019C F526              MOV   BITLOC,A     ;
                                          ;
019E D525E5            DJNZ  MTRNMR,RESLP1 ; DECREMENT MOTOR NUMBER
01A1 22                RET                ;   UNTIL ALL READ MAILBOXES HAVE BEEN CHECKED.
               ;
               ;...........................................................
               ;...........................................................
               ;
               ;        'DECODE 8748 CODE' SUBROUTINE:
               ;
                                          ; PROVIDE:  BYTE TO BE DECODED IN ACCUM,
                                          ;           CORRECT MOTOR NUMBER IN MTRNMR,
                                          ;       CORRECT MOTOR BIT LOCATION IN BITLOC,
                                          ;       CORRECT 'READ MBX' ENABLED ON P2.
                                          ;
                                          ; RETURNS:  APPROPRIATE ACTION TO CODE.
                                          ;
                                          ; CHANGES:  P2, R0, R3, A, PSW,
                                          ;           NEWSTL, GLOBL, RSET48, STLL48
                                          ;           RDY48
                                          ;
                                          ; CALLS:  'GLOBAL WRITE' ('GLOBAL')
                                          ;                      (VECTORS TO 'CLRWRT')
                                          ;
                                          ; CALLED BY:  'RESPOND TO 8748'  ('RESPND')
                                          ;
                                          ;
01A2 601F      DCD48:  JZ    DCDFIN       ; IF CODE IS NON-ZERO,
01A4 43A080            ORL   P2,#80H      ;  ENABLE THE MOTOR'S WRITE MAILBOX.
                                          ;
01A7 FB                MOV   R3,A         ; SAVE ASIDE A COPY OF 8748 CODE.
                                          ;
01A8 7430              MOV   A,#PRVMTR    ; COMPUTE ADDRESS OF LOCATION
01AA 2525              ADD   A,MTRNMR     ;   WHERE CODE PREVIOUSLY SENT
01AC F8                MOV   R0,A         ;     TO THIS MOTOR IS STORED.
                                          ;
01AD EB                MOV   A,R3         ; VALIDATE 8748 CODE BY
01AE C4                SWAP  A            ;  CHECKING THAT BOTH
01AF 6B                XRL   A,R3         ;   NIBBLES ARE THE SAME.
```

```
01B0 7012          567             JNZ   RSNRQ1          ; IF NOT VALID, GO ASK FOR RESEND.
                   568                                   ;
01B2 EB            569             MOV   A,R3            ; IF VALID, DECODING DEPENDS ONLY
01B3 5407          570             ANL   A,#7H           ;    ON THE LOWEST THREE BITS.
01B5 601A          571             JZ    RSND48          ; IF CODE WAS 88H, GO RESEND LAST CODE.
01B7 14            572             DEC   A               ;
01B8 601A          573             JZ    STL48           ; IF CODE WAS 99H, MOTOR IS STALLED.
01BA 14            574             DEC   A               ;
01BB 6026          575             JZ    HWRST           ; IF CODE WAS OAAH, 8748 HAS RESET.
01BD 14            576             DEC   A               ;
01BE 6029          577             JZ    RDYNX           ; IF CODE WAS OBBH, READY FOR NEXT MOVE.
01C0 14            578             DEC   A               ;
01C1 602C          579             JZ    PKRDY           ; IF CODE WAS OCCH, RAM PEEK IS COMPLETE.
                   580                                   ;
01C3 22            581 DCDFIN:     RET                   ; IF NONE OF ABOVE, CODE HAS NO MEANING.
                   582                                   ;
01C4 E590          583 RSNRQ1:     MOV   A,P1            ; OBTAIN 'WRITE MAILBOX FULL' FLAG
01C6 5524          584             ANL   A,MTRLNK        ;    FOR THIS MOTOR.  IF IT IS FULL,
01C8 5526          585             ANL   A,BITLOC        ;        WAIT FOR IT TO BE READ BY THE 8748.
01CA 70F8          586             JNZ   RSNRQ1          ;
                   587                                   ;
01CC 7488          588             MOV   A,#OVR          ; IF IT IS EMPTY,
01CE F6            589             MOV   @R0,A           ;   PUT 'RESEND REQUEST' CODE INTO MAILBOX,
01CF F2            590             MOVX  @R0,A           ;     AND INTO 'PREVIOUS WRITE' LOCATION.
01D0 22            591             RET                   ;
                   592                                   ;
01D1 C6            593 RSND48:     XCH   A,@R0           ; IF A RESEND WAS REQUESTED BY THE 8748,
01D2 F2            594             MOVX  @R0,A           ;   WRITE PREVIOUS CODE TO MAILBOX, AND CLEAR
01D3 22            595             RET                   ;     PREVIOUS WRITE LOC TO AVOID INFINITE LOOP.
                   596                                   ;
01D4 E526          597 STL48:      MOV   A,BITLOC        ; IF THE MOTOR IS STALLED,
01D6 4222          598             ORL   STLL48,A        ;   SET THE MOTOR'S STALL' FLAG,
01D8 7523FF        599             MOV   NEWSTL,#0FFH    ;     SET THE 'NEW STALL' FLAG,
01DB 75A080        600 SNDHLT:     MOV   P2,#GLBLWR      ;       AND WRITE 'HALT' TO ALL MOTORS.
01DE 74BB          601             MOV   A,#HLT          ;
01E0 B187          602             ACALL CLRWR2          ;
01E2 22            603             RET                   ;
                   604                                   ;
01E3 E526          605 HWRST:      MOV   A,BITLOC        ; IF THE 8748 HAS UNDERGONE HARDWARE RESET,
01E5 4220          606             ORL   RSET48,A        ;   SET THE MOTOR'S RESET FLAG,
01E7 21DB          607             AJMP  SNDHLT          ;     AND GO WRITE 'HALT' TO ALL MOTORS.
                   608                                   ;
01E9 E526          609 RDYNX:      MOV   A,BITLOC        ; IF THE 8748 IS READY FOR ANOTHER MOVE,
01EB F4            610             CPL   A               ;    CLEAR THE MOTOR'S 'NOT READY' FLAG.
01EC 5221          611             ANL   RDY48,A         ;
01EE 22            612             RET                   ;
                   613                                   ;
01EF E526          614 PKRDY:      MOV   A,BITLOC        ; IF THE 8748 HAS JUST COMPLETED A PEEK,
01F1 4228          615             ORL   RAMRDY,A        ;   SET THE MOTOR'S 'PEEK COMPLETE' FLAG.
01F3 22            616             RET                   ;
                   617 ;
                   618 ;.....................................................................
                   619 ;.....................................................................
                   620 ;
                   621 ;         WRITE TO VAX SUBROUTINE:
                   622 ;
                   623                                   ; PROVIDE: CODE FOR TRANSMISSION IN 'RESPCD'.
                   624                                   ;
                   625                                   ; RETURNS:  IF TRANSMITTER IS IDLE, SENDS
                   626                                   ;         CODE DIRECTLY.  IF NOT IDLE, CODE
                   627                                   ;         IS LEFT IN THE WRITE BUFFER FOR
                   628                                   ;         TRANSMISSION.
                   629                                   ;
                   630                                   ; CHANGES:  RESPCD, WRBUF, ACCUM, PSW
                   631                                   ;
                   632                                   ; CALLS:  'SERIAL OUTPUT'   ('SEROUT')
                   633                                   ;
                   634                                   ; CALLED BY:  MAIN ROUTINE
                   635                                   ;               (NOTE, ALSO USED AS TAIL END
                   636                                   ;                   OF DECODE VAX SUBROUTINE.)
                   637                                   ;
01F4 E538          638 WRITVX:     MOV   A,XMTFLG        ; IF 'TRANSMITTER IDLE' FLAG IS SET,
```

```
01F6 20E005   639             JB    ACC.0,WRIRDY  ;   GO CALL THE SERIAL OUTPUT SUBROUTINE.
              640                                 ;
01F9 E53E     641             MOV   A,RESPCD      ; IF TRANSMITTER IS NOT IDLE, AND IF CODE
01FB 70F7     642             JNZ   WRITVX        ;  IS WAITING TO BE SENT, STAY IN THIS LOOP.
              643                                 ;   (THIS PREVENTS OVERWRITING THE RESPONSE CODE)
              644                                 ;
01FD 22       645             RET                 ; RETURN AFTER RESPONSE CODE IS WRITTEN.
              646                                 ;
01FE 5101     647   WRIRDY:   ACALL SEROUT        ;
0200 22       648             RET                 ;
              649   ;
              650   ;.....................................................................
              651   ;.....................................................................
              652   ;
              653   ;         SERIAL OUTPUT SUBROUTINE:
              654   ;
              655                                 ; PROVIDE:  CORRECT VALUE 'NT' OF DATA BYTES LEFT
              656                                 ;          TO BE TRANSMITTED FROM WRITE BUFFER,
              657                                 ;          AND THE 'TRANSMITTER IDLE' FLAG SET.
              658                                 ;
              659                                 ; RETURNS:    IF DATA ARE TO BE TRANSMITTED:
              660                                 ;          TRANSMISSION IS DIRECT, AND 'NT'
              661                                 ;          IS DECREMENTED.
              662                                 ;             IF CODE IS TO BE TRANSMITTED, THEN:
              663                                 ;          -IF CODE IS ZERO, NO TRANSMISSION.
              664                                 ;          -IF CODE EQUALS UNITY, THE NO-OP
              665                                 ;           CODE (ZERO) IS TRANSMITTED.
              666                                 ;          -ALL OTHER VALUES ARE TRANSMITTED
              667                                 ;           DIRECTLY WITH NO VALIDITY CHECK.
              668                                 ;             IN ANY CASE, IF A TRANSMISSION
              669                                 ;           OCCURS, THEN THE 'TRANSMITTER IDLE'
              670                                 ;           FLAG IS CLEARED.
              671                                 ;
              672                                 ; CHANGES:  ACCUM, PSW, R0, IE.4,
              673                                 ;          WRBUF, NT, TRFLAG
              674                                 ;
              675                                 ; CALLS:  NONE
              676                                 ;
              677                                 ; CALLED BY:  'WRITE TO VAX'   ('WRITVX')
              678                                 ;             'SERIAL PORT ISR'  ('SRLISR')
              679                                 ;
              680                                 ;
0201 5338FE   681   SEROUT:   ANL   XMTFLG,#0FEH ; CLEAR THE 'TRANSMITTER IDLE' FLAG (BIT 0).
              682                                 ;
0204 E53A     683             MOV   A,NT          ; IF THE 'DATA BYTES REMAINING' COUNTER
0206 702B     684             JNZ   MRDATA        ;  IS NOT ZERO, THERE IS MORE DATA TO TRANSMIT.
              685                                 ;
0208 E53F     686             MOV   A,SRWRBF      ; IF CODE IS TO BE TRANSMITTED, THEN
020A 7019     687             JNZ   PROCSS        ;  CODE IN THE WRITE BUFFER HAS PRIORITY.
020C E53E     688             MOV   A,RESPCD      ;  IF WRITE BUFFER IS EMPTY, CODE IN THE
020E 753E00   689             MOV   RESPCD,#0H    ;   'RESPONSE CODE' LOCATION IS TRANSMITTED.
0211 7012     690             JNZ   PROCSS        ;
0213 E538     691             MOV   A,XMTFLG      ; IF BOTH THE ABOVE LOCATIONS ARE EMPTY, THEN
0215 30E128   692             JNB   ACC.1,SETFLG  ;  CHECK THE 'EXPECTING DATA' FLAG.
0218 5338FD   693             ANL   XMTFLG,#0FDH  ;   IF IT IS SET, THEN LOAD THE
021B 7411     694             MOV   A,#SND51      ;    'READY TO RECEIVE' CODE FOR TRANSMISSION.
021D 853B3C   695             MOV   NR,NRBUFF     ; MOVE PENDING NUMBER OF RECEIVE DATA BYTES
0220 753B00   696             MOV   NRBUFF,#0H    ;  INTO NUMBER OF RECEIVE DATA BYTES LOCATION.
0223 F53D     697             MOV   PRVXWR,A      ;
              698                                 ;
0225 F53F     699   PROCSS:   MOV   SRWRBF,A      ; MOVE TRANSMIT CODE INTO WRITE BUFFER.
0227 14       700             DEC   A             ;  IF NO-OP CODE IS TO BE TRANSMITTED, THEN THE
0228 7002     701             JNZ   NOTNOP        ;   NUMBER LOADED WILL BE THE NUMBER ONE.
022A F53F     702             MOV   SRWRBF,A      ;    IF IT IS 1, PUT NO-OP CODE IN WRITE BUFFER.
              703                                 ;
022C 853F99   704   NOTNOP:   MOV   SBUF,SRWRBF  ; MOVE CODE TO SERIAL PORT OUTPUT BUFFER,
022F 753F00   705             MOV   SRWRBF,#0H    ;     AND CLEAR WRITE BUFFER.
0232 22       706             RET                 ;
              707                                 ;
0233 743F     708   MRDATA:   MOV   A,#SRWRBF     ; IF DATA IS TO BE SENT,
0235 253A     709             ADD   A,NT          ;  COMPUTE THE DATA BUFFER LOCATION
0237 F8       710             MOV   R0,A          ;     WHERE THE BYTE TO TRANSMIT IS FOUND.
```

```
0238 8699          711             MOV   SBUF,@R0       ;        LOAD THE DATA BYTE TO THE SERIAL
023A 153A          712             DEC   NT             ;          PORT TRANSMIT BUFFER.
023C 753D01        713             MOV   PRVXWR,#1      ; DATA TRANSMISSION DOES NOT BEGIN UNLESS
023F 22            714             RET                  ;   PREVIOUS CODE HAS BEEN RECEIVED BY VAX.
                   715                                  ;
0240 433801        716     SETFLG: ORL   XMTFLG,#1H     ; IN THE CASE OF NO TRANSMISSION,
0243 22            717             RET                  ;   SET THE 'TRANSMITTER IDLE' FLAG.
                   718     ;
                   719     ;.....................................................................
                   720     ;.....................................................................
                   721     ;
                   722     ;         'DECODE VAX COMMAND' SUBROUTINE:
                   723     ;
                   724                                  ; PROVIDE:  VAX COMMAND IN 'VAX COMMAND'
                   725                                  ;                  ('VAXCMD') LOCATION.
                   726                                  ;
                   727                                  ; RETURNS:  APPROPRIATE ACTION BASED ON
                   728                                  ;         VAX COMMAND, RETURN CODE FOR
                   729                                  ;       TRANSMISSION TO VAX, CORRECT VALUE
                   730                                  ;     ENTERED INTO 'PREVIOUS WRITE TO VAX'
                   731                                  ;     LOCATION ('PRVXWR').
                   732                                  ;     LAST PART OF SUBROUTINE IS ALWAYS A
                   733                                  ;     JUMP TO 'WRITE TO VAX'.
                   734                                  ;
                   735                                  ; CHANGES:  R0,R1,R2,R3,R4,IE.0,ACCUM,PSW,
                   736                                  ;         GLOBL,VAXCMD,INTCMD,INTCDB,
                   737                                  ;       BITLOC,NTBUFF,NT,PRVXWR,WRBUF
                   738                                  ;
                   739                                  ; CALLS:  'LOAD 8748 RAMS'  ('LOAD48')
                   740                                  ;         'GLOBAL WRITE'  ('GLOBAL')
                   741                                  ;         'COMPUTE VELOCITIES'  ('VELOCS')
                   742                                  ;
                   743                                  ; CALLED BY:  MAIN ROUTINE
                   744                                  ;
0244 E52F          745     DCDVAX: MOV   A,VAXCMD       ; IF THE CODE IS ZERO,
0246 6053          746             JZ    DCVFIN         ;          NO OPERATION.
                   747                                  ;
0248 C4            748             SWAP  A              ; TEST CODE FOR VALIDITY
0249 652F          749             XRL   A,VAXCMD       ;   ('NIBBLE'-WISE SYMMETRY)
024B 6002          750             JZ    DCDOK          ; IF CODE IS NOT VALID,
024D 419F          751             AJMP  RSNRQ2         ;           GO REQUEST A RESEND.
                   752                                  ;
024F E52F          753     DCDOK:  MOV   A,VAXCMD       ; DECODE THE CODE BY TESTING
0251 540F          754             ANL   A,#0FH         ;   ONLY THE FOUR LOWER BITS.
                   755                                  ;
0253 14            756             DEC   A              ;
0254 7002          757             JNZ   DCNXT0         ;
0256 41C4          758             AJMP  ENINT          ; IF CODE WAS 11H, INTERRUPT COMMAND FOLLOWS.
0258 14            759     DCNXT0: DEC   A              ;
0259 7002          760             JNZ   DCNXT1         ;
025B 41F0          761             AJMP  DISINT         ; IF 22H, DISABLE EXTERNAL INTERRUPT ZERO.
025D 14            762     DCNXT1: DEC   A              ;
025E 7002          763             JNZ   DCNXT2         ;
0260 41FB          764             AJMP  RAM48          ; IF 33H, PERFORM 8748 RAM PEEK/POKE.
0262 14            765     DCNXT2: DEC   A              ;
0263 7002          766             JNZ   DCNXT3         ;
0265 618E          767             AJMP  RAM51          ; IF 44H, PERFORM 8751 RAM OR LATCH PEEK.
0267 14            768     DCNXT3: DEC   A              ;
0268 7002          769             JNZ   DCNXT4         ;
026A 61B5          770             AJMP  VLCREQ         ; IF 55H, SEND VELOCITY DATA.
026C 14            771     DCNXT4: DEC   A              ;
026D 7002          772             JNZ   DCNXT5         ;
026F 61F1          773             AJMP  ABSREQ         ; IF 66H, SEND ABSOLUTE POSITION DATA.
0271 14            774     DCNXT5: DEC   A              ;
0272 7002          775             JNZ   DCNXT6         ;
0274 812B          776             AJMP  SNDDAT         ; IF 77H, SEND DATA FROM I/O BUFFER.
0276 14            777     DCNXT6: DEC   A              ;
0277 7002          778             JNZ   DCNXT7         ;
0279 812C          779             AJMP  RSNDVX         ; IF 88H, RESEND CODE PREVIOUSLY SENT TO VAX.
027B 14            780     DCNXT7: DEC   A              ;
027C 7002          781             JNZ   DCNXT8         ;
027E 8137          782             AJMP  CR48MV         ; IF 99H, PASS 'CLEAR MOVES' CODE TO 8748'S.
```

```
0280 14          783   DCNXT8: DEC   A                ;
0281 7002        784           JNZ   DCNXT9           ;
0283 813C        785           AJMP  MOVE48           ; IF OAAH, MOVE DATA IS IN THE READ BUFFER.
0285 14          786   DCNXT9: DEC   A                ;
0286 7002        787           JNZ   DCNX10           ;
0288 81D0        788           AJMP  PASSON           ; IF OBBH, PASS 'HALT' CODE TO 8748'S.
028A 14          789   DCNX10: DEC   A                ;
028B 7002        790           JNZ   DCNX11           ;
028D 81D0        791           AJMP  PASSON           ; IF OCCH, PASS 'GO' CODE TO 8748'S.
028F 14          792   DCNX11: DEC   A                ;
0290 7002        793           JNZ   DCNX12           ;
0292 81DA        794           AJMP  SHRDRM           ; IF ODDH, PERFORM SHARED RAM PEEK.
0294 14          795   DCNX12: DEC   A                ;
0295 7002        796           JNZ   DCNX13           ;
0297 81D0        797           AJMP  PASSON           ; IF OEEH, PASS 'RESET ABSOLUTE POSITION'
                 798                                  ;                          CODE TO 8748'S.
                 799                                  ; IF NONE OF ABOVE,
0299 A115        800   DCNX13: AJMP  COEF48           ;    THEN CONTROL COEFFICIENTS ARE IN READ BUFFER.
                 801                                  ;
029B 752F00      802   DCVFIN: MOV   VAXCMD,#0H       ; CLEAR THE VAX COMMAND LOCATION.
029E 22          803           RET                    ;
                 804                                  ;
                 805   ;.....................................
                 806   ;.....................................
                 807                                  ;
029F E53D        808   RSNRQ2: MOV   A,PRVXWR         ; IF CODE PREVIOUSLY SENT TO VAX
02A1 6444        809           XRL   A,#DAT           ;  IS ONE OF THE THREE CODES THAT
02A3 6011        810           JZ    FAULT            ;   REQUIRE 'SEND DATA' AS A RESPONSE,
02A5 E53D        811           MOV   A,PRVXWR         ;  THEN DO NOT REQUEST A RESEND FROM VAX,
02A7 6455        812           XRL   A,#VDT           ;   SINCE THE 'RESEND REQUEST' CODE MIGHT
02A9 600B        813           JZ    FAULT            ;     BE TREATED AS UNFORMATTED DATA BY VAX.
02AB E53D        814           MOV   A,PRVXWR         ;  INSTEAD, JUMP TO THE 'FAULT' LOCATION,
02AD 6466        815           XRL   A,#PDT           ;   WHERE A STRING OF ZEROS IS SENT TO VAX
02AF 6005        816           JZ    FAULT            ;     TO RESTART THE COMMUNICATION SEQUENCE.
                 817                                  ;
02B1 753E88      818           MOV   RESPCD,#OVR      ; OTHERWISE, DO SEND
02B4 41E8        819           AJMP  PREWRI           ;   'RESEND REQUEST' CODE TO VAX.
                 820                                  ;
02B6 7A10        821   FAULT:  MOV   R2,#10H          ; SEND SIXTEEN ZEROS TO INDICATE
02B8 753E01      822   FAULP1: MOV   RESPCD,#1H       ;  A COMMUNICATION FAULT AND TO RESTART
02BB 31F4        823           CALL  WRITVX           ;   THE COMMUNICATION SEQUENCE.
02BD DAF9        824           DJNZ  R2,FAULP1        ;
                 825                                  ;
02BF 853D3E      826           MOV   RESPCD,PRVXWR    ; RESTART THE DATA COMMUNICATION SEQUENCE
02C2 41E8        827           AJMP  PREWRI           ;    BY RESENDING THE LAST CODE SENT TO VAX.
                 828                                  ;
                 829   ;.....................................
                 830                                  ;
02C4 E511        831   ENINT:  MOV   A,RDBOT          ; GET THE COMMAND TO EXECUTE ON INTERRUPT.
02C6 6433        832           XRL   A,#PK48          ;  TEST WHETHER IT IS ONE OF THE FOUR
02C8 6018        833           JZ    VALCMD           ;   COMMANDS THAT ARE ALLOWED TO BE
02CA E511        834           MOV   A,RDBOT          ;     EXECUTED ON INTERRUPT 1.
02CC 6444        835           XRL   A,#PK51          ;
02CE 6012        836           JZ    VALCMD           ;
02D0 E511        837           MOV   A,RDBOT          ;
02D2 6455        838           XRL   A,#VRQ           ;
02D4 600C        839           JZ    VALCMD           ;
02D6 E511        840           MOV   A,RDBOT          ;
02D8 6466        841           XRL   A,#PRQ           ;
02DA 6006        842           JZ    VALCMD           ;
02DC 753EFF      843           MOV   RESPCD,#INV      ; IF IT IS NOT ONE OF THE FOUR, RETURN AN
02DF 0202E8      844           JMP   PREWRI           ;    'INVALID DATA' CODE AS RESPONSE.
                 845                                  ;
02E2 85112C      846   VALCMD: MOV   INTCMD,RDBOT     ; OTHERWISE, STORE IT IN THE INTERRUPT
                 847                                  ;   COMMAND LOCATION, AND SEND
02E5 753E77      848           MOV   RESPCD,#RCVDAT   ;       'RECEIVED DATA' CODE AS RESPONSE.
                 849                                  ;
02E8 752F00      850   PREWRI: MOV   VAXCMD,#0H       ; CLEAR VAX COMMAND LOCATION,
02EB 853E3D      851           MOV   PRVXWR,RESPCD    ;   COPY RESPONSE CODE INTO 'PREVIOUS WRITE',
02EE 21F4        852           AJMP  WRITVX           ;     AND GO WRITE RESPONSE TO VAX.
                 853                                  ;
                 854   ;.....................................
```

```
                        855
                        856     DISINT: MOV   INTCMD,#0H     ; DISABLE THE INTERRUPT-ON-COMMAND BY
02F0 752C00             857             MOV   INTCDB,#0H     ;   PUTTING NO-OP CODE INTO INTERRUPT COMMAND
                        858                                  ;     LOCATIONS.
02F6 852F3E             859             MOV   RESPCD,VAXCMD  ; ECHO 'INTERRUPT DISABLED' CODE AS RESPONSE.
02F9 41E8               860             AJMP  PREWRI         ;
                        861                                  ;
                        862     ;.....................................
                        863                                  ;
02FB E529               864     RAM48:  MOV   A,RAMADR       ; GET THE 8748 ADDRESS OF INTEREST.
02FD 20E72F             865             JB    ACC.7,POKE     ; IF THE HIGH BIT IS SET, COMMAND IS A POKE.
                        866                                  ;
                        867                                  ; OTHERWISE, COMMAND IS A PEEK, SO
0300 75A018             868             MOV   P2,#MTRAM1     ;  IT IS NECESSARY TO WRITE THE
0303 752602             869             MOV   BITLOC,#MTR1BT ;   ADDRESS TO EACH SHARED RAM INDIVIDUALLY.
0306 7A07               870             MOV   R2,#7H         ; START WITH MOTOR SEVEN.
0308 7829               871             MOV   R0,#RAMADR     ; SET R1 TO ADDRESS OF 'RAM ADDRESS'.
                        872                                  ;
030A E590               873     RAMLP1: MOV   A,P1           ; TEST THE PARTICULAR MOTOR'S
030C F4                 874             CPL   A              ;  'SHARED RAM LOCKOUT' FLAG.
030D 5524               875             ANL   A,MTRLNK       ;    WAIT IN THIS LOOP UNTIL
030F 5526               876             ANL   A,BITLOC       ;      TO THIS MOTOR'S SHARED RAM.
0311 70F7               877             JNZ   RAMLP1         ;
                        878                                  ;
0313 E529               879             MOV   A,RAMADR       ; MOVE THE ADDRESS OF INTEREST
0315 F2                 880             MOVX  @R0,A          ;   INTO THE PROPER LOCATION IN SHARED RAM.
                        881                                  ;
0316 E526               882             MOV   A,BITLOC       ; COMPUTE THE BIT LOCATION
0318 23                 883             RL    A              ;   CORRESPONDING TO
0319 F526               884             MOV   BITLOC,A       ;     THE NEXT MOTOR NUMBER.
                        885                                  ;
031B E5A0               886             MOV   A,P2           ; COMPUTE THE SHARED RAM ENABLE ADDRESS
031D 2410               887             ADD   A,#10H         ;   FOR THE NEXT MOTOR.
031F F5A0               888             MOV   P2,A           ;
                        889                                  ;
0321 DAE7               890             DJNZ  R2,RAMLP1      ; CONTINUE UNTIL SHARED RAMS OF ALL LINKED
                        891                                  ;  MOTORS HAVE BEEN WRITTEN TO.
0323 752801             892             MOV   RAMRDY,#1      ; SET '8748 PEEK IN PROGRESS' FLAG.
                        893                                  ;
0326 7530DD             894             MOV   GLOBL,#PK48RM  ; WRITE 'READ RAM' TO ALL MAILBOXES.
0329 B16F               895             ACALL GLOBAL         ;
                        896                                  ;
032B 752F00             897             MOV   VAXCMD,#0H     ; CLEAR VAX COMMAND LOCATION.  THERE IS
032E 22                 898             RET                  ; NO IMMEDIATE RESPONSE TO THIS COMMAND.
                        899                                  ;   (RESPONSE IS SENT AFTER 8748'S HAVE ALL
                        900                                  ;      PERFORMED THEIR INDIVIDUAL RAM PEEKS)
                        901                                  ;
032F 753B02             902     POKE:   MOV   NRBUFF,#2H     ; FOR POKE COMMAND, ANOTHER TWO BYTES OF DATA
0332 433802             903             ORL   XMTFLG,#2H     ;  MUST BE OBTAINED FROM THE VAX.
0335 753E00             904             MOV   RESPCD,#0H     ;   GO ASK FOR THEM.
0338 31F4               905             ACALL WRITVX         ;
                        906                                  ;
033A E53C               907     RAMLP2: MOV   A,NR           ; WAIT UNTIL BOTH BYTES OF
033C 453B               908             ORL   A,NRBUFF       ;      DATA HAVE BEEN RECEIVED.
033E 70FA               909             JNZ   RAMLP2         ;
                        910                                  ;
0340 E512               911             MOV   A,RDBOT+1      ; GET MOTOR NUMBER OF 8748 TO BE POKED.
0342 F525               912             MOV   MTRNMR,A       ;  THIS NUMBER IS USED BELOW IN A LOOP TO
0344 FA                 913             MOV   R2,A           ;    GENERATE THE CORRESPONDING BIT LOCATION.
0345 C4                 914             SWAP  A              ; COMPUTE THE SHARED RAM ENABLE ADDRESS
0346 4408               915             ORL   A,#8H          ;  FOR THIS MOTOR, AND ENABLE IT.
0348 F5A0               916             MOV   P2,A           ;
                        917                                  ;
034A 7401               918             MOV   A,#1H          ; COMPUTE THE CORRESPONDING BIT LOCATION.
034C 23                 919     RAMLP3: RL    A              ;
034D DAFD               920             DJNZ  R2,RAMLP3      ;
034F F526               921             MOV   BITLOC,A       ;
                        922                                  ;
0351 E590               923     RAMLP4: MOV   A,P1           ; GET 'SHARED RAM LOCKOUT' FLAG
0353 F4                 924             CPL   A              ;  FOR THE PARTICULAR MOTOR.
0354 5524               925             ANL   A,MTRLNK       ;   STAY IN THIS LOOP UNTIL
0356 5526               926             ANL   A,BITLOC       ;     ACCESS IS GAINED TO THE SHARED RAM.
```

```
0358 70F7          927             JNZ   RAMLP4          ;
                   928                                   ;
035A 7929          929             MOV   R1,#RAMADR      ; PUT THE ADDRESS OF INTEREST
035C E7            930             MOV   A,@R1           ;  AND THE DATA BYTE TO BE POKED TO THAT ADDRESS
035D F3            931             MOVX  @R1,A           ;   IN THEIR PROPER LOCATIONS IN SHARED RAM.
035E 19            932             DEC   R1              ;
035F E511          933             MOV   A,RDBOT         ;
0361 F3            934             MOVX  @R1,A           ;
                   935                                   ;
0362 E5A0          936             MOV   A,P2            ; COMPUTE THE WRITE MAILBOX ENABLE ADDRESS
0364 6488          937             XRL   A,#88H          ;  FOR THIS MOTOR, AND ENABLE IT.
0366 F5A0          938             MOV   P2,A            ;
                   939                                   ;
0368 E590          940   RAMLP5:   MOV   A,P1            ; CHECK THAT THIS MOTOR'S
036A 5524          941             ANL   A,MTRLNK        ;   WRITE MAILBOX IS NOT FULL.
036C 5526          942             ANL   A,BITLOC        ;
036E 6013          943             JZ    RAMWRT          ; IF IT IS NOT FULL, GO WRITE TO THE MAILBOX.
                   944                                   ;
0370 53A070        945             ANL   P2,#70H         ; BUT IF IT IS FULL, THEN
0373 E590          946             MOV   A,P1            ;  CHECK IF THIS MOTOR'S
0375 5524          947             ANL   A,MTRLNK        ;   READ MAILBOX IS FULL, SINCE IF IT IS FULL,
0377 5526          948             ANL   A,BITLOC        ;    THE 8748 MAY BE WAITING TO WRITE TO IT.
0379 6003          949             JZ    NOTFLL          ;     IN THAT CASE, BOTH MICROPROCESSORS WOULD
037B E2            950             MOVX  A,@R0           ;      GET STUCK WAITING TO WRITE FOREVER.
037C 31A2          951             ACALL DCD48           ; SO, IF IT IS FULL, READ IT AND GO DECODE IT.
                   952                                   ;
037E 43A080        953   NOTFLL:   ORL   P2,#80H         ; IF READ MAILBOX IS NOT FULL, GO BACK AND
0381 6168          954             AJMP  RAMLP5          ;  SEE IF THE WRITE MAILBOX IS STILL FULL.
                   955                                   ;
0383 74DD          956   RAMWRT:   MOV   A,#PK48RM       ; WHEN WRITE MAILBOX IS EMPTY,
0385 F2            957             MOVX  @R0,A           ;   GO AHEAD AND WRITE THE 'PEEK RAM' CODE TO IT.
0386 75A000        958             MOV   P2,#IDLE        ; CLEAR ENABLE ADDRESS FROM PORT 2.
                   959                                   ;
0389 753E77        960             MOV  RESPCD,#RCVDAT   ; LOAD 'RECEIVED DATA' AS RESPONSE TO VAX.
038C 41E8          961             AJMP  PREWRI          ;
                   962                                   ;
                   963   ;.....................          ;
                   964                                   ;
038E E529          965   RAM51:    MOV   A,RAMADR        ; TEST THE 8751 ADDRESS TO BE PEEKED.
0390 20E705        966             JB    ACC.7,LCHEXT    ;  IF IT EXCEEDS 128, IT IS A LATCH ADDRESS.
                   967                                   ;   OTHERWISE, IT IS AN INTERNAL RAM ADDRESS.
0393 F8            968   RAMINT:   MOV   R0,A            ; PUT DATA FROM INTERNAL RAM LOCATION
0394 8640          969             MOV   WRBOT,@R0       ;  IN LOWEST WRITE BUFFER LOCATION.
0396 61A0          970             AJMP  RAMFIN          ;
                   971                                   ;
0398 F5A0          972   LCHEXT:   MOV   P2,A            ; PUT DATA FROM EXTERNAL LATCH
039A 859040        973             MOV   WRBOT,P1        ;  IN LOWEST WRITE BUFFER LOCATION.
039D 75A000        974             MOV   P2,#IDLE        ;
                   975                                   ;
03A0 7941          976   RAMFIN:   MOV   R1,#WRBOT+1     ; ABOVE THE PEEK DATA, IN THE WRITE BUFFER,
03A2 A729          977             MOV   @R1,RAMADR      ;  PUT THE ADDRESS WHERE DATA WAS OBTAINED,
03A4 09            978             INC   R1              ;    THE LOW AND HIGH BYTES OF 'RHINO CLOCK',
03A5 A72A          979             MOV   @R1,RNOCKL      ;      AND THE COMMAND TO WHICH THIS DATA
03A7 09            980             INC   R1              ;        IS THE RESPONSE.
03A8 A72B          981             MOV   @R1,RNOCKH      ;
03AA 09            982             INC   R1              ;
03AB 7744          983             MOV   @R1,#PK51       ;
                   984                                   ;
03AD 753905        985             MOV   NTBUFF,#5H      ; SET 'NT BUFFER' FOR FIVE BYTES OF DATA.
                   986                                   ;
03B0 852F3E        987             MOV   RESPCD,VAXCMD   ; ECHO '51RQ' CODE AS RESPONSE.
03B3 41E8          988             AJMP  PREWRI          ;
                   989                                   ;
                   990   ;.....................          ;
                   991                                   ;
03B5 75A018        992   VLCREQ:   MOV   P2,#MTRAM1      ; TO SEND VELOCITIES OF ALL MOTORS, IT IS
03B8 752602        993             MOV  BITLOC,#MTR1BT   ;  NECESSARY TO GAIN ACCESS TO EACH MOTOR'S
03BB 7840          994             MOV   R0,#WRBOT       ;   SHARED RAM, INDIVIDUALLY, TO READ ITS
03BD 792A          995             MOV   R1,#SPEED       ;    VELOCITY LOCATION.  START WITH MOTOR 1.
03BF 7A07          996             MOV   R2,#7H          ;
                   997                                   ;
03C1 E590          998   VLCLP1:   MOV   A,P1            ; CHECK THIS MOTOR'S
```

```
03C3 F4            999            CPL   A              ; 'SHARED RAM LOCKOUT' FLAG.
03C4 5524         1000            ANL   A,MTRLNK       ;  WAIT IN THIS LOOP UNTIL
03C6 5526         1001            ANL   A,BITLOC       ;   ACCESS IS GAINED TO
03C8 70F7         1002            JNZ   VLCLP1         ;    THIS MOTOR'S SHARED RAM.
                  1003                                 ;
03CA E3           1004            MOVX  A,@R1          ; ONCE ACCESS IS GAINED,
03CB 30E704       1005            JNB   ACC.7,VLC2S    ;  READ THE 'VELOCITY' LOCATION.
03CE 6480         1006            XRL   A,#80H         ;   IF BIT 7 IS SET, DIRECTION OF MOTION IS
03D0 F4           1007            CPL   A              ;    COUNTERCLOCKWISE, SO PUT VELOCITY IN
03D1 04           1008            INC   A              ;     2'S COMPLEMENT FORM.
                  1009                                 ;
03D2 F6           1010  VLC2S:    MOV   @R0,A          ; MOVE VELOCITY TO ITS LOCATION IN THE WRITE
03D3 08           1011            INC   R0             ; BUFFER, MOTOR 1 AT THE BOTTOM.
                  1012                                 ;
03D4 E5A0         1013            MOV   A,P2           ; COMPUTE SHARED RAM ENABLE ADDRESS
03D6 2410         1014            ADD   A,#10H         ;  FOR THE NEXT MOTOR'S SHARED RAM,
03D8 F5A0         1015            MOV   P2,A           ;   AND ENABLE IT.
                  1016                                 ;
03DA E526         1017            MOV   A,BITLOC       ; COMPUTE THE BIT LOCATION CORRESPONDING
03DC 23           1018            RL    A              ;  TO THE NEXT MOTOR'S MOTOR NUMBER.
03DD F526         1019            MOV   BITLOC,A       ;
                  1020                                 ;
03DF DAE0         1021            DJNZ  R2,VLCLP1      ; LOOP UNTIL ALL RAMS HAVE BEEN READ.
                  1022                                 ;
03E1 A62A         1023            MOV   @R0,RNOCKL     ; ABOVE ALL THE VELOCITY DATA
03E3 08           1024            INC   R0             ;  IN THE WRITE BUFFER,
03E4 A62B         1025            MOV   @R0,RNOCKH     ;   PUT THE LOW AND HIGH 'RHINO CLOCK' BYTES.
                  1026                                 ;
03E6 753909       1027            MOV   NTBUFF,#9H     ; SET 'NT BUFFER' FOR 9 BYTES OF DATA.
                  1028                                 ;
03E9 75A000       1029            MOV   P2,#IDLE       ; CLEAR LAST ENABLE ADDRESS FROM PORT 2.
                  1030                                 ;
03EC 852F3E       1031            MOV   RESPCD,VAXCMD  ; ECHO 'VRQ' CODE AS RESPONSE TO VAX.
03EF 41E8         1032            AJMP  PREWRI         ;
                  1033                                 ;
                  1034  ;.....................................
                  1035                                 ;
03F1 75A018       1036  ABSREQ:   MOV   P2,#MTRAM1     ; TO SEND ABSOLUTE POSITION DATA FROM ALL MOTORS,
03F4 752602       1037            MOV   BITLOC,#MTR1BT ;  IT IS NECESSARY TO READ THE SHARED RAM OF EACH
                  1038                                 ;   MOTOR INDIVIDUALLY.  START WITH MOTOR 1,
03F7 7840         1039            MOV   R0,#WRBOT      ;    WHOSE DATA GOES AT THE BOTTOM OF THE WRITE
03F9 7933         1040            MOV   R1,#LOBTPS     ;     BUFFER.
03FB 7A07         1041            MOV   R2,#7H         ;
                  1042                                 ;
03FD E590         1043  ABSLP1:   MOV   A,P1           ; CHECK THIS MOTOR'S
03FF F4           1044            CPL   A              ;  'SHARED RAM LOCKOUT' FLAG.
0400 5524         1045            ANL   A,MTRLNK       ;   STAY IN THIS LOOP UNTIL ACCESS
0402 5526         1046            ANL   A,BITLOC       ;    HAS BEEN GAINED TO THIS MOTOR'S
0404 70F7         1047            JNZ   ABSLP1         ;     SHARED RAM.
                  1048                                 ;
0406 E3           1049            MOVX  A,@R1          ; ONCE ACCESS IS GAINED, MOVE THE TWO BYTES
0407 F6           1050            MOV   @R0,A          ;   OF ABSOLUTE POSITION DATA INTO THEIR
0408 09           1051            INC   R1             ;    LOCATIONS IN THE WRITE BUFFER, LOW BYTE
0409 E3           1052            MOVX  A,@R1          ;     BELOW HIGH BYTE.
040A 08           1053            INC   R0             ;
040B F6           1054            MOV   @R0,A          ;
                  1055                                 ;
040C E5A0         1056            MOV   A,P2           ; COMPUTE SHARED RAM ENABLE ADDRESS
040E 2410         1057            ADD   A,#10H         ;  FOR THE NEXT MOTOR'S SHARED RAM,
0410 F5A0         1058            MOV   P2,A           ;   AND ENABLE IT.
                  1059                                 ;
0412 E526         1060            MOV   A,BITLOC       ; COMPUTE THE BIT LOCATION CORRESPONDING
0414 23           1061            RL    A              ;  TO THE NEXT MOTOR'S MOTOR NUMBER.
0415 F526         1062            MOV   BITLOC,A       ;
                  1063                                 ;
0417 08           1064            INC   R0             ; INCREMENT R0 TO NEXT WRITE BUFFER LOCATION.
0418 19           1065            DEC   R1             ; DECREMENT R1 TO LOW BYTE OF ABSOLUTE POSITION.
0419 DAE2         1066            DJNZ  R2,ABSLP1      ; LOOP UNTIL ALL MOTOR'S POSITION DATA IS READ.
                  1067                                 ;
041B A62A         1068            MOV   @R0,RNOCKL     ; ABOVE THE POSITION DATA IN THE WRITE BUFFER,
041D 08           1069            INC   R0             ;   PUT THE LOW AND HIGH BYTES OF 'RHINO CLOCK'.
041E A62B         1070            MOV   @R0,RNOCKH     ;
```

```
                        1071
                        1072            MOV   NTBUFF,#10H       ; SET 'NT BUFFER' FOR 16 BYTES OF DATA.
0420 753910             1073
                        1074            MOV   P2,#IDLE          ; CLEAR LAST ENABLE ADDRESS FROM PORT 2.
0423 75A000             1075
                        1076            MOV   RESPCD,VAXCMD     ; ECHO 'PRQ' CODE AS RESPONSE TO VAX.
0426 852F3E             1077            AJMP  PREWRI            ;
0429 41E8               1078
                        1079    ;.......................
                        1080
                        1081    SNDDAT: RET                     ; THE NT BUFFER SHOULD HAVE BEEN LOADED
042B 22                 1082                                    ;   TO 'NT' IN THE SERIAL PORT ISR.
                        1083
                        1084    ;.......................
                        1085
                        1086    RSNDVX: MOV   RESPCD,PRVXWR     ; OBTAIN CODE PREVIOUSLY WRITTEN TO VAX.
042C 853D3E             1087            MOV   PRVXWR,#1H        ;  SET THE 'PREVIOUS WRITE' LOCATION FOR
042F 753D01             1088            MOV   VAXCMD,#0H        ;   NO-OPERATION, TO AVOID INFINITE LOOP.
0432 752F00             1089            AJMP  WRITVX            ; GO WRITE PREVIOUS CODE.
0435 21F4               1090
                        1091    ;.......................
                        1092
                        1093    CR48MV: MOV   STLL48,#0H        ; CLEAR THE 'MOTOR STALLED' FLAGS.
0437 752200             1094            AJMP  PASSON            ;
043A 81D0               1095
                        1096    ;.......................
                        1097
                        1098    MOVE48: MOV   A,MTRLNK          ; IF NO MOTORS ARE CONNECTED, SYSTEM IS IN
043C E524               1099            JNZ   LDMOVS            ; DEBUG MODE, SO RESPOND TO EACH MOVE CODE
043E 7005               1100            MOV   RESPCD,#RDYNXT    ; WITH A READY FOR NEXT MOVE CODE.
0440 753E33             1101            ACALL WRITVX            ;
0443 31F4               1102
                        1103    LDMOVS: MOV   R0,#RDBOT         ; THE MOVE FOR EACH MOTOR MUST BE WRITTEN
0445 7811               1104            MOV   P2,#MTRAM1        ;  INDIVIDUALLY TO THAT MOTOR'S SHARED RAM.
0447 75A018             1105            MOV   BITLOC,#MTR1BT    ;   START WITH MOTOR 1.
044A 752602             1106            MOV   R2,#7H            ;
044D 7A07               1107
                        1108    MOVLP0: MOV   A,P1              ; CHECK THIS MOTOR'S
044F E590               1109            CPL   A                 ;  'SHARED RAM LOCKOUT' FLAG.
0451 F4                 1110            ANL   A,MTRLNK          ;   STAY IN THIS LOOP UNTIL ACCESS
0452 5524               1111            ANL   A,BITLOC          ;    TO THE SHARED RAM IS GAINED.
0454 5526               1112            JNZ   MOVLP0            ;
0456 70F7               1113
                        1114            MOV   R1,#NXMVSN        ; ONCE ACCESS IS GAINED, VARIOUS QUANTITIES
0458 7931               1115            INC   R0                ;  MUST BE DETERMINED.  FIRST, PICK OFF THE
045A 08                 1116            MOV   A,@R0             ;   SIGN OF THE MOVE FROM THE HIGH BYTE. (NOTE
045B E6                 1117            ANL   A,#7FH            ;    THAT THE MOVE DISTANCE IS POSITIVE, WITH A
045C 547F               1118            XCH   A,@R0             ;     SIGN BIT APPENDED AS THE HIGHEST BIT.)
045E C6                 1119            ANL   A,#80H            ;      WRITE THE SIGN BIT TO THE 'NEXT MOVE SIGN'
045F 5480               1120            MOVX  @R1,A             ;       LOCATION.
0461 F3                 1121
                        1122            DEC   R1                ; NEXT, WRITE THE HIGH BYTE OF THE MOVE,
0462 19                 1123            MOV   A,@R0             ;  WITH THE SIGN BIT REMOVED, TO THE 'NEXT MOVE
0463 E6                 1124            MOVX  @R1,A             ;   HIGH BYTE' LOCATION.
0464 F3                 1125
                        1126            DEC   R0                ; WRITE THE LOW BYTE OF THE MOVE
0465 18                 1127            DEC   R1                ;  TO THE 'NEXT MOVE LOW BYTE' LOCATION.
0466 19                 1128            MOV   A,@R0             ;
0467 E6                 1129            MOVX  @R1,A             ;
0468 F3                 1130
                        1131            MOV   R1,#NXMVNF        ; 'OR' TOGETHER THE HIGHEST THREE BITS OF THE
0469 792C               1132            ANL   A,#0E0H           ;   LOW BYTE AND ALL THE BITS OF THE HIGH BYTE
046B 54E0               1133            INC   R0                ;    TO OBTAIN THE 'NEXT MOVE INFINITE' FLAG.
046D 08                 1134            ORL   A,@R0             ;     BY 'INFINITE' IS MEANT THAT THE MOVE IS
046E 46                 1135            MOVX  @R1,A             ;      GREATER THAN 31 HOLES.
046F F3                 1136
                        1137            INC   R1                ; 'OR' TOGETHER BOTH BYTES OF THE MOVE
0470 09                 1138            DEC   R0                ;  TO OBTAIN THE 'NEXT MOVE RECEIVED' FLAG.
0471 18                 1139            ORL   A,@R0             ;   SO IF THE MOVE IS ZERO, NO MOVE APPEARS
0472 46                 1140            MOVX  @R1,A             ;    TO BE RECEIVED TO THE 8748 SOFTWARE.
0473 F3                 1141            JNZ   NOTRDY            ; IF THE MOVE IS ZERO, THEN CLEARLY THIS MOTOR
0474 7005               1142            MOV   A,BITLOC          ;  IS READY FOR THE NEXT MOVE, SO
0476 E526
```

```
0478 F4          1143            CPL   A               ;     CLEAR THE 'NOT READY' FLAG FOR THIS MOTOR.
0479 5221        1144            ANL   RDY48,A         ;
                 1145                                  ;
047B 08          1146  NOTRDY:   INC   R0              ; INCREMENT POINTER R0 TWICE,
047C 08          1147            INC   R0              ;         TO LOW BYTE OF NEXT MOTOR'S MOVE.
                 1148                                  ;
047D E5A0        1149            MOV   A,P2            ; COMPUTE SHARED RAM ENABLE ADDRESS FOR
047F 2410        1150            ADD   A,#10H          ;   THE NEXT MOTOR, AND ENABLE IT.
0481 F5A0        1151            MOV   P2,A            ;
                 1152                                  ;
0483 E526        1153            MOV   A,BITLOC        ; COMPUTE THE BIT LOCATION CORRESPONDING TO
0485 23          1154            RL    A               ;    THE NEXT MOTOR'S MOTOR NUMBER.
0486 F526        1155            MOV   BITLOC,A        ;
                 1156                                  ;
0488 DAC5        1157            DJNZ  R2,MOVLP0       ; CONTINUE LOOPING UNTIL MOVES HAVE BEEN
                 1158                                  ;      WRITTEN TO ALL LINKED MOTORS.
                 1159                                  ;
048A 75A000      1160            MOV   P2,#IDLE        ; CLEAR LAST ENABLE ADDRESS FROM PORT 2.
                 1161                                  ;
048D 753B07      1162            MOV   NRBUFF,#7H      ; NOW REQUEST TO RECEIVE THE VELOCITY DATA,
0490 433802      1163            ORL   XMTFLG,#2H      ;    AMOUNTING TO 7 BYTES.
0493 753E00      1164            MOV   RESPCD,#0H      ;
0496 31F4        1165            ACALL WRITVX          ;
                 1166                                  ;
0498 E53C        1167  MOVLP1:   MOV   A,NR            ; WAIT IN THIS LOOP UNTIL
049A 453B        1168            ORL   A,NRBUFF        ;       ALL 7 DATA BYTES
049C 70FA        1169            JNZ   MOVLP1          ;             HAVE BEEN RECEIVED.
                 1170                                  ;
049E 7811        1171            MOV   R0,#RDBOT       ; IT IS NECESSARY TO WRITE THE VELOCITY DATA
04A0 792E        1172            MOV   R1,#RAMVEL      ;   TO EACH MOTOR'S SHARED RAM, INDIVIDUALLY.
04A2 7A07        1173            MOV   R2,#7H          ;     START WITH MOTOR 1.
04A4 75A018      1174            MOV   P2,#MTRAM1      ;
04A7 752602      1175            MOV   BITLOC,#MTR1BT  ;
                 1176                                  ;
04AA E590        1177  MOVLP2:   MOV   A,P1            ; CHECK THIS MOTOR'S
04AC F4          1178            CPL   A               ;   'SHARED RAM LOCKOUT' FLAG.
04AD 5524        1179            ANL   A,MTRLNK        ;     STAY IN THIS LOOP UNTIL ACCESS
04AF 5526        1180            ANL   A,BITLOC        ;       TO THIS MOTOR'S SHARED RAM IS GAINED.
04B1 70F7        1181            JNZ   MOVLP2          ;
                 1182                                  ;
04B3 E6          1183            MOV   A,@R0           ; ONCE ACCESS IS GAINED, WRITE THE VELOCITY
04B4 F3          1184            MOVX  @R1,A           ;   COMMAND TO ITS LOCATION IN SHARED RAM.
                 1185                                  ;
04B5 E5A0        1186            MOV   A,P2            ; COMPUTE THE NEXT MOTOR'S
04B7 2410        1187            ADD   A,#10H          ;   SHARED RAM ENABEL ADDRESS,
04B9 F5A0        1188            MOV   P2,A            ;     AND ENABLE IT.
                 1189                                  ;
04BB E526        1190            MOV   A,BITLOC        ; COMPUTE THE   BIT LOCATION CORRESPONDING
04BD 23          1191            RL    A               ;     TO THE NEXT MOTOR'S MOTOR NUMBER.
04BE F526        1192            MOV   BITLOC,A        ;
                 1193                                  ;
04C0 08          1194            INC   R0              ; INCREMENT R0 TO NEXT READ BUFFER LOCATION.
04C1 DAE7        1195            DJNZ  R2,MOVLP2       ; LOOP UNTIL ALL VELOCITY DATA HAS BEEN WRITTEN.
                 1196                                  ;
04C3 75A000      1197            MOV   P2,#IDLE        ; CLEAR LAST ENABLE ADDRESS FROM PORT 2.
04C6 852F30      1198            MOV   GLOBL,VAXCMD    ; PASS 'MOVE' COMMAND ON TO THE 8748'S,
04C9 B16F        1199            ACALL GLOBAL          ;    USING THE 'GLOBAL WRITE' SUBROUTINE.
                 1200                                  ;
04CB 753E77      1201            MOV   RESPCD,#RCVDAT  ; LOAD 'RECEIVED DATA' AS RESPONSE TO VAX.
04CE 41E8        1202  MOVFIN:   AJMP  PREWRI          ;
                 1203                                  ;
                 1204  ;.....................................
                 1205                                  ;
04D0 852F30      1206  PASSON:   MOV   GLOBL,VAXCMD    ; PASS ON THE HALT/GO/RESET/CLEAR CODE
04D3 B16F        1207            ACALL GLOBAL          ;    TO 8748'S USING THE GLOBAL WRITE SUBROUTINE.
                 1208                                  ;
04D5 852F3E      1209            MOV   RESPCD,VAXCMD   ; ECHO ORIGINAL VAX COMMAND AS RESPONSE CODE.
04D8 41E8        1210            AJMP  PREWRI          ;
                 1211                                  ;
                 1212  ;.....................................
                 1213                                  ;
04DA 75A018      1214  SHRDRM:   MOV   P2,#MTRAM1      ; TO DO A 'SHARED RAM PEEK', IT IS NECESSARY
```

```
04DD 752602          1215              MOV   BITLOC,#MTR1BT ; TO READ EACH MOTOR'S SHARED RAM INDIVIDUALLY.
04E0 7840            1216              MOV   R0,#WRBOT     ;  START WITH MOTOR 1, WHOSE DATA WILL BE AT
04E2 A929            1217              MOV   R1,RAMADR     ;    THE BOTTOM OF THE WRITE BUFFER.
04E4 7A07            1218              MOV   R2,#7         ;      NOTE THAT ADDRESS TO BE READ IS IN R1.
                     1219                                  ;
04E6 E590            1220    RMPLP1:   MOV   A,P1          ; CHECK THIS MOTOR'S
04E8 F4              1221              CPL   A             ;   'SHARED RAM LOCKOUT' FLAG.
04E9 5524            1222              ANL   A,MTRLNK      ;      STAY IN THIS LOOP UNTIL ACCESS
04EB 5526            1223              ANL   A,BITLOC      ;        TO THE SHARED RAM IS GAINED.
04ED 70F7            1224              JNZ   RMPLP1        ;
                     1225                                  ;
04EF E3              1226              MOVX  A,@R1         ; READ DATA FROM APPROPRIATE ADDRESS IN SHARED
04F0 F6              1227              MOV   @R0,A         ;   RAM INTO THIS MOTOR'S WRITE BUFFER LOCATION.
                     1228                                  ;
04F1 E526            1229              MOV   A,BITLOC      ; COMPUTE BIT LOCATION CORRESPONDING TO
04F3 23              1230              RL    A             ;  NEXT MOTOR'S MOTOR NUMBER.
04F4 F526            1231              MOV   BITLOC,A      ;
                     1232                                  ;
04F6 E5A0            1233              MOV   A,P2          ; COMPUTE NEXT MOTOR'S
04F8 2410            1234              ADD   A,#10H        ;   SHARED RAM ENABLE ADDRESS,
04FA F5A0            1235              MOV   P2,A          ;      AND ENABLE IT.
                     1236                                  ;
04FC 08              1237              INC   R0            ; INCREMENT WRITE BUFFER POINTER.
04FD DAE7            1238              DJNZ  R2,RMPLP1     ; LOOP UNTIL ALL SHARED RAMS HAVE BEEN 'PEEKED'.
                     1239                                  ;
04FF 75A000          1240              MOV   P2,#IDLE      ; CLEAR LAST ENABLE ADDRESS FROM PORT 2.
                     1241                                  ;
0502 A629            1242              MOV   @R0,RAMADR    ; ABOVE THE PEEK DATA IN THE WRITE BUFFER,
0504 08              1243              INC   R0            ;  PUT THE ADDRESS THAT WAS PEEKED INTO,
0505 A62A            1244              MOV   @R0,RNOCKL    ;    THE LOW AND HIGH BYTES OF 'RHINO CLOCK',
0507 08              1245              INC   R0            ;      AND THE COMMAND TO WHICH THIS DATA
0508 A62B            1246              MOV   @R0,RNOCKH    ;        IS THE RESPONSE.
050A 08              1247              INC   R0            ;
050B 76DD            1248              MOV   @R0,#SHRPK    ;.
                     1249                                  ;
050D 75390B          1250              MOV   NTBUFF,#0BH   ; LOAD 'NT BUFFER' FOR 11 BYTES.
0510 753E44          1251              MOV   RESPCD,#DAT   ; LOAD 'PEEK DATA READY' CODE AS RESPONSE TO VAX.
0513 41E8            1252              AJMP  PREWRI        ;
                     1253                                  ;
                     1254    ;.....................................
                     1255                                  ;
0515 E511            1256    COEF48:   MOV   A,RDBOT       ; COEFFICIENTS ARE SENT FOR A PARTICULAR MOTOR.
0517 F525            1257              MOV   MTRNMR,A      ;  GET THE MOTOR NUMBER OF INTEREST.
0519 FA              1258              MOV   R2,A          ;    IT IS ALSO USED IN COMPUTING BIT LOCATION.
                     1259                                  ;
051A C4              1260              SWAP  A             ; COMPUTE SHARED RAM ENABLE ADDRESS
051B 4408            1261              ORL   A,#8H         ;  OF THE MOTOR TO RECEIVE THE COEFFICIENTS.
051D F5A0            1262              MOV   P2,A          ;
                     1263                                  ;
051F 7401            1264              MOV   A,#1H         ; COMPUTE THE BIT LOCATION CORRESPONDING
0521 23              1265    COELP1:   RL    A             ;  TO THE MOTOR'S MOTOR NUMBER.
0522 DAFD            1266              DJNZ  R2,COELP1     ;
0524 F526            1267              MOV   BITLOC,A      ;
                     1268                                  ;
0526 E590            1269    COELP2:   MOV   A,P1          ; CHECK THE MOTOR'S
0528 F4              1270              CPL   A             ;   'SHARED RAM LOCKOUT' FLAG.
0529 5524            1271              ANL   A,MTRLNK      ;      STAY IN THIS LOOP UNTIL ACCESS
052B 5526            1272              ANL   A,BITLOC      ;        IS GAINED TO THE SHARED RAM.
052D 70F7            1273              JNZ   COELP2        ;
                     1274                                  ;
052F 7812            1275              MOV   R0,#RDBOT+1   ; ONCE ACCESS IS GAINED,
0531 7920            1276              MOV   R1,#CF1       ;  WRITE EACH COEFFICIENT DATA BYTE
0533 7A02            1277              MOV   R2,#2         ;    FROM THE READ BUFFER TO ITS
0535 7B08            1278              MOV   R3,#8         ;      PROPER LOCATION IN SHARED RAM.
0537 E6              1279    COELP3:   MOV   A,@R0         ;
0538 F3              1280              MOVX  @R1,A         ;
0539 08              1281              INC   R0            ;
053A 09              1282              INC   R1            ;
053B DBFA            1283              DJNZ  R3,COELP3     ;
053D 7B06            1284              MOV   R3,#6H        ;
053F 7938            1285              MOV   R1,#GRIPPR    ;
0541 DAF4            1286              DJNZ  R2,COELP3     ;
```

```
                    1287
                    1288                ;
0543 E5A0           1289                MOV   A,P2              ; COMPUTE THE MOTOR'S
0545 6488           1290                XRL   A,#88H            ;   WRITE MAILBOX ENABLE ADDRESS,
0547 F5A0           1291                MOV   P2,A              ;     AND ENABLE IT.
                    1292                ;
0549 E590           1293    COELP4:     MOV   A,P1              ; CHECK THAT THE MOTOR'S WRITE MAILBOX
054B 5524           1294                ANL   A,MTRLNK          ;   IS NOT FULL.
054D 5526           1295                ANL   A,BITLOC          ;
054F 6013           1296                JZ    COEWRT            ;
                    1297                ;
0551 53A070         1298                ANL   P2,#70H           ; IF IT IS FULL, ENABLE THE MOTOR'S
0554 E590           1299                MOV   A,P1              ;   READ MAILBOX, AND SEE IF IT IS FULL.
0556 5524           1300                ANL   A,MTRLNK          ;
0558 5526           1301                ANL   A,BITLOC          ;
055A 6003           1302                JZ    NOREAD            ; IF IT IS NOT FULL, GO TRY WRITING AGAIN.
055C E2             1303                MOVX  A,@R0             ; IF IT IS FULL,
055D 31A2           1304                ACALL DCD48             ;  GO DECODE THE MESSAGE FROM 8748.
                    1305                ;
055F 43A080         1306    NOREAD:     ORL   P2,#80H           ; ENABLE THE MOTOR'S WRITE MAILBOX AGAIN.
0562 A149           1307                AJMP  COELP4            ; GO TRY WRITING AGAIN.
                    1308                ;
0564 74FF           1309    COEWRT:     MOV   A,#KFF            ; IF WRITE MAILBOX IS NOT FULL, THEN PUT
0566 F2             1310                MOVX  @R0,A             ;   'COEFFICIENTS' CODE INTO IT.
                    1311                ;
0567 75A000         1312                MOV   P2,#IDLE          ; CLEAR ENABLE ADDRESS FROM PORT 2.
056A 753E77         1313                MOV   RESPCD,#RCVDAT    ; LOAD 'RECEIVED DATA' AS RESPONSE TO VAX.
056D 41E8           1314                AJMP  PREWRI            ;
                    1315    ;
                    1316    ;.....................................................................
                    1317    ;.....................................................................
                    1318    ;
                    1319    ;         GLOBAL WRITE SUBROUTINE:
                    1320    ;
                    1321                                        ; PROVIDE:  CODE DESTINED FOR ALL 8748'S IN
                    1322                                        ;           LOCATION 'GLOBL'.
                    1323                                        ;
                    1324                                        ; RETURNS:  CODE IS WRITTEN TO ALL 8748
                    1325                                        ;             WRITE MAILBOXES.
                    1326                                        ;
                    1327                                        ; CHANGES:  P2,R0,R4,ACCUM,PSW
                    1328                                        ;
                    1329                                        ; CALLS:  'RESPOND TO 8748S'  ('RESPND')
                    1330                                        ;
                    1331                                        ; CALLED BY:  'DECODE 8748'  ('DCD48')
                    1332                                        ;               (VECTORS INTO CLRWR2)
                    1333                                        ;             'DECODE VAX COMMAND'  ('DCDVAX')
                    1334                                        ;             'COMPUTE VELOCITIES'  ('VELOCS')
                    1335                                        ;
056F 75A080         1336    GLOBAL:     MOV   P2,#GLBLWR        ; PUT GLOBAL MAILBOX WRITE ENABLE ON PORT 2.
0572 E590           1337                MOV   A,P1              ;   OBTAIN THE WRITE MAILBOX FULL FLAGS.
0574 5524           1338                ANL   A,MTRLNK          ; IF ALL WRITE MAILBOXES ARE EMPTY,
0576 600D           1339                JZ    CLRWRT            ;   THEN GO DO THE GLOBAL MAILBOX WRITE.
                    1340                ;
0578 75A000         1341                MOV   P2,#IDLE          ; OTHERWISE, SINCE AT LEAST ONE OF THE WRITE
057B E590           1342                MOV   A,P1              ;  MAILBOXES IS FULL, SEE IF ANY OF THE READ
057D 5524           1343                ANL   A,MTRLNK          ;   MAILBOXES ARE FULL.  IF SO, IT COULD BE THAT
057F 60EE           1344                JZ    GLOBAL            ;    SOME 8748 IS WAITING TO WRITE, AND BOTH IT
0581 3180           1345                ACALL RESPND            ;     AND THE 8751 WOULD WAIT FOREVER IF ONE OF
0583 A16F           1346                AJMP  GLOBAL            ;      THE MAILBOXES DOES NOT GET READ.
                    1347                ;
0585 E530           1348    CLRWRT:     MOV   A,GLOBL           ; IF ALL THE WRITE MAILBOXES ARE EMPTY,
0587 F2             1349    CLRWR2:     MOVX  @R0,A             ;  WRITE THE CODE TO ALL MAILBOXES.
0588 75A000         1350                MOV   P2,#IDLE          ; CLEAR THE ENABLE ADDRESS FROM PORT 2.
                    1351                ;
058B 7830           1352                MOV   R0,#PRVMTR        ; WRITE THE GLOBAL CODE TO ALL OF THE
058D 7C07           1353                MOV   R4,#7H            ;   '8748 PREVIOUS WRITE' LOCATIONS.
058F 08             1354    GLBLP1:     INC   R0                ;
0590 F6             1355                MOV   @R0,A             ;
0591 DCFC           1356                DJNZ  R4,GLBLP1         ;
                    1357                ;
0593 22             1358                RET                     ;
```

```
                        1359     ;
                        1360     ;.....................................................................
                        1361     ;.....................................................................
                        1362     ;
                        1363     ;         SERIAL PORT INTERRUPT SERVICE SUBROUTINE:
                        1364     ;
                        1365                                      ; PROVIDE:  NONE
                        1366                                      ;
                        1367                                      ; RETURNS:  PROCESSES SERIAL INPUT BYTE, OR
                        1368                                      ;             SERVICES SERIAL OUTPUT BUFFER.
                        1369                                      ;         USES VALUE OF NT (NR) TO DETERMINE
                        1370                                      ;       IF OUTPUT (INPUT) BYTE IS CODE OR
                        1371                                      ;       DATA.  RESPONDS DIRECTLY TO INPUT
                        1372                                      ;       CODES THAT WILL BE FOLLOWED BY DATA.
                        1373                                      ;
                        1374                                      ; CHANGES:  SCON.0, SCON.1, WRBUF, NT, NR,
                        1375                                      ;         TRFLAG, VAXCMD
                        1376                                      ;
                        1377                                      ; CALLS:  'SERIAL OUTPUT'  ('SEROUT')
                        1378                                      ;
                        1379                                      ; CALLED BY:  INTERRUPT ZERO FALLING EDGE
                        1380                                      ;
                        1381                                      ;
0594 C0D0               1382     SRLISR: PUSH   PSW               ; SAVE PROGRAM STATUS WORD ON STACK.
0596 C0E0               1383             PUSH   ACC               ; SAVE ACCUMULATOR CONTENTS ON STACK.
0598 D2D3               1384             SETB   PSW.3             ; SELECT REGISTER BANK 1.
                        1385                                      ;
059A 209809             1386             JB     SCON.0,RCVR       ; IF SERIAL PORT RECEIVE FLAG IS SET,
                        1387                                      ;    GO SERVICE THE SERIAL PORT RECEIVER.
                        1388                                      ;
059D 30992F             1389             JNB    SCON.1,SRLJMP     ; IF NEITHER FLAG IS SET, SPURIOUS INTERRUPT.
05A0 C299               1390             CLR    SCON.1            ;   ELSE, THE SERIAL PORT TRANSMIT FLAG IS SET,
05A2 5101               1391             ACALL  SEROUT            ;     SO GO SERVICE THE SERIAL PORT TRANSMITTER.
05A4 C131               1392             AJMP   SRLFIN            ;
                        1393                                      ;
05A6 C298               1394     RCVR:   CLR    SCON.0            ; CLEAR 'RECEIVER INTERRUPT' BIT.
05A8 E53C               1395             MOV    A,NR              ; CHECK NUMBER OF DATA BYTES LEFT TO RECEIVE.
05AA 6025               1396             JZ     SRLCOD            ; IF ZERO, INCOMING BYTE IS CODE, NOT DATA.
                        1397                                      ;
05AC 2410               1398     SRLDAT: ADD    A,#SRRDBF         ; COMPUTE CURRENT READ BUFFER LOCATION,
05AE F8                 1399             MOV    R0,A              ;  AND MOVE THE INCOMING DATA INTO IT.
05AF A699               1400             MOV    @R0,SBUF          ;
                        1401                                      ;
05B1 D53C1B             1402             DJNZ   NR,SRLJMP         ; DECREMENT RECEIVE DATA BYTE COUNT, AND IF
05B4 85102E             1403     NODATA: MOV    VAXBUF,SRRDBF     ; NO DATA IS LEFT TO RECEIVE, PUT THE ORIGINAL
05B7 751000             1404             MOV    SRRDBF,#0         ; CODE INTO THE VAX BUFFER FOR PROCESSING.
                        1405                                      ;
05BA E52E               1406             MOV    A,VAXBUF          ; SOME OF THE CODES FROM VAX ARE ACCOMPANIED
05BC 64DD               1407             XRL    A,#SHRPK          ;  BY AN ADDRESS TO BE PEEKED OR POKED.
05BE 600C               1408             JZ     ADRLCH            ;   IF THE CODE IS ONE OF THOSE, GO MOVE THAT
05C0 E52E               1409             MOV    A,VAXBUF          ;     ADDRESS FROM THE BOTTOM OF THE READ BUFFER
05C2 6444               1410             XRL    A,#PK51           ;       TO A SPECIAL LOCATION (LCHADR).
05C4 6006               1411             JZ     ADRLCH            ; IF IT IS NOT ONE OF THOSE CODES, THEN THE
05C6 E52E               1412             MOV    A,VAXBUF          ;  ROUTINE IS FINISHED.
05C8 6433               1413             XRL    A,#PK48           ;
05CA 7065               1414             JNZ    SRLFIN            ;
05CC 851129             1415     ADRLCH: MOV    RAMADR,RDBOT      ;
05CF C131               1416     SRLJMP: AJMP   SRLFIN            ;
                        1417                                      ;
                        1418                                      ;
05D1 859910             1419     SRLCOD: MOV    SRRDBF,SBUF       ; MOVE INCOMING CODE FROM SERIAL PORT RECEIVER
05D4 E510               1420             MOV    A,SRRDBF          ;  BUFFER INTO THE 'INCOMING CODE' BUFFER.
05D6 60F7               1421             JZ     SRLJMP            ;     IF CODE IS NO-OP, PROCESSING IS COMPLETE.
                        1422                                      ;
05D8 6477               1423             XRL    A,#SNDVX          ; IF INCOMING CODE IS THE 'SEND TO VAX' CODE,
05DA 700B               1424             JNZ    NOTVAX            ;  THEN LOAD THE PENDING NUMBER OF DATA BYTES
05DC 85393A             1425             MOV    NT,NTBUFF         ;    TO BE TRANSMITTED (NTBUFF) INTO 'NT', THE
05DF 753900             1426             MOV    NTBUFF,#0H        ;      NUMBER TO BE TRANSMITTED IMMEDIATELY.
05E2 751000             1427             MOV    SRRDBF,#0H        ;       'SEND TO VAX' CODE IS FULLY ANSWERED BY
05E5 C12A               1428             AJMP   SRLOUT            ;         DATA TRANSMISSION, SO CLEAR COMMAND.
                        1429                                      ;
                        1430                                      ; IF INCOMING CODE IS NOT 'SEND TO VAX':
```

```
05E7 74DD       1431    NOTVAX: MOV   A,#SHRPK      ; SOME OF THE INCOMING CODES WILL BE FOLLOWED
05E9 6510       1432            XRL   A,SRRDBF     ;  BY DATA.  DETERMINE IF IT THE CODE IS GOING
05EB 7005       1433            JNZ   NOTRMP      ;   TO BE FOLLOWED BY DATA, AND IF SO, LOAD THE
05ED 753B01     1434            MOV   NRBUFF,#1H  ;     NUMBER OF EXPECTED BYTES OF DATA INTO 'NR'.
05F0 C127       1435            AJMP  SRLXMT      ; THE 'SHARED RAM PEEK' CODE
                1436                              ;       IS FOLLOWED BY A 1-BYTE ADDRESS.
05F2 74AA       1437    NOTRMP: MOV   A,#MOVE     ;
05F4 6510       1438            XRL   A,SRRDBF    ;
05F6 7005       1439            JNZ   NOTMOV      ; THE 'MOVE' CODE IS FOLLOWED BY
05F8 753B0E     1440            MOV   NRBUFF,#0EH ;     14 BYTES OF DATA.
05FB C127       1441            AJMP  SRLXMT      ;
                1442                              ;
05FD 74FF       1443    NOTMOV: MOV   A,#KFF      ;
05FF 6510       1444            XRL   A,SRRDBF    ;
0601 7005       1445            JNZ   NOTKFF      ; THE 'COEFFICIENTS'  CODE IS
0603 753B0F     1446            MOV   NRBUFF,#0FH ;      FOLLOWED BY 15 BYTES OF DATA.
0606 C127       1447            AJMP  SRLXMT      ;
                1448                              ;
0608 7444       1449    NOTKFF: MOV   A,#PK51     ;
060A 6510       1450            XRL   A,SRRDBF    ;
060C 7005       1451            JNZ   NOT5PK      ; THE '8751 PEEK' CODE
060E 753B01     1452            MOV   NRBUFF,#1H  ;     IS FOLLOWED BY A 1-BYTE ADDRESS.
0611 C127       1453            AJMP  SRLXMT      ;
                1454                              ;
0613 7433       1455    NOT5PK: MOV   A,#PK48     ;
0615 6510       1456            XRL   A,SRRDBF    ;
0617 7005       1457            JNZ   NOT4PK      ; THE '8748 PEEK/POKE' CODE
0619 753B01     1458            MOV   NRBUFF,#1H  ;     IS FOLLOWED BY A 1-BYTE ADDRESS.
061C C127       1459            AJMP  SRLXMT      ;
                1460                              ;
061E 7411       1461    NOT4PK: MOV   A,#ECI      ;
0620 6510       1462            XRL   A,SRRDBF    ;
0622 7090       1463            JNZ   NODATA      ; THE 'INTERRUPT COMMAND ENABLE'
0624 753B01     1464            MOV   NRBUFF,#1H  ;    IS FOLLOWED BY A 1-BYTE COMMAND CODE.
                1465                              ;
0627 433802     1466    SRLXMT: ORL   XMTFLG,#2H  ; IN CASE THE CODE IS FOLLOWED BY DATA,
062A E538       1467    SRLOUT: MOV   A,XMTFLG    ;   SET THE 'ASK FOR DATA' FLAG.
062C 30E002     1468            JNB   ACC.0,SRLFIN ; IF 'TRANSMITTER IDLE' FLAG IS SET,
062F 5101       1469            ACALL SEROUT      ;    CALL THE 'SERIAL OUTPUT' SUBROUTINE.
                1470                              ;
0631 D0E0       1471    SRLFIN: POP   ACC         ; RESTORE ACCUMULATOR.
0633 D0D0       1472            POP   PSW         ; RESTORE PROGRAM STATUS WORD.
0635 32         1473            RETI              ; RETURN FROM INTERRUPT.
                1474                              ;
                1475    ;.....................................................................
                1476    ;.....................................................................
                1477                              ;
                1478                ; INTERRUPT ZERO SERVICE SUBROUTINE:
                1479                              ;
                1480                              ; PROVIDE: NOTHING.
                1481                              ;
                1482                              ; RETURNS: RHINO CLOCK IS INCREMENTED BY 1.
                1483                              ;          INTERRUPT COMMAND IS PLACED IN
                1484                              ;               INTERRUPT COMMAND BUFFER.
                1485                              ;
                1486                              ; CHANGES: RNOCKL,RNOCKH,INTCDB
                1487                              ;
                1488                              ; CALLS: NONE
                1489                              ;
                1490                              ; CALLED BY:  INTERRUPT ZERO ACTIVE
                1491                              ;
                1492                              ;
0636 C0D0       1493    INTRPO: PUSH  PSW         ; SAVE PROCESSOR STATUS WORD ON STACK.
0638 C0E0       1494            PUSH  ACC         ; SAVE ACCUMULATOR CONTENTS ON STACK.
                1495                              ;
063A 052A       1496            INC   RNOCKL      ; INCREMENT THE TWO-BYTE RHINO CLOCK.
063C E52A       1497            MOV   A,RNOCKL    ;  NOTE THAT THE HIGH BYTE IS INCREMENTED
063E 7002       1498            JNZ   HIBTOK      ;   ONLY IF THE LOW BYTE INCREMENTS TO ZERO,
0640 052B       1499            INC   RNOCKH      ;    THUS GENERATING A CARRY TO THE HIGH BYTE.
                1500                              ;
0642 E528       1501    HIBTOK: MOV   A,RAMRDY    ; IF AN 8748 RAM PEEK IS IN PROGRESS,
0644 7003       1502            JNZ   INTFIN      ;  DO NOT MOVE THE INTERRUPT COMMAND
```

```
0646 852C2D          1503              MOV   INTCDB,INTCMD ;   INTO THE INTERRUPT COMMAND BUFFER.
                     1504                                  ;
0649 D0E0            1505     INTFIN:  POP   ACC           ; RECOVER ACCUMULATOR CONTENTS FROM STACK.
064B D0D0            1506              POP   PSW           ; RECOVER PROCESSOR STATUS WORD FROM STACK.
064D 32              1507              RETI                ;
                     1508                                  ;
                     1509     ;.....................................................................
                     1510     ;.....................................................................
                     1511                                  ;
                     1512              END
```

XREF SYMBOL TABLE LISTING
---- ------ ----- -------

```
N A M E          T Y P E    V A L U E       ATTRIBUTES AND REFERENCES

ABSLP1 . .       C ADDR     03FDH    A        1043# 1047 1066
ABSREQ . .       C ADDR     03F1H    A        773 1036#
ACC. . . .       D ADDR     00E0H    A        265 639 692 865 966 1005 1383 1468 1471
                                               1494 1505
ADRLCH . .       C ADDR     05CCH    A        1408 1411 1415#
AGO. . . .         NUMB     00CCH    A        27# 474
BAUD48 . .         NUMB     00FAH    A        178# 264
BAUD96 . .         NUMB     00FDH    A        175# 266
BEG48. . .       C ADDR     0041H    A        265 267#
BEGFIN . .       C ADDR     007AH    A        297 306#
BEGINO . .       C ADDR     002CH    A        226 254#
BEGLP1 . .       C ADDR     0032H    A        259# 260
BEGLP2 . .       C ADDR     0048H    A        276# 299
BEGLP3 . .       C ADDR     0051H    A        279# 304
BITLOC . .         NUMB     0026H    A        115# 277 281 291 301 303 401 409 415 417
                                               510 514 526 528 585 597 605 609 614
                                               869 876 882 884 921 926 942 948 993 1001
                                               1017 1019 1037 1046 1060 1062 1105
                                               1111 1142 1153 1155 1175 1180 1190 1192
                                               1215 1223 1229 1231 1267 1272 1294 1300
BKCOM1 . .       C ADDR     00E7H    A        382 385# 472
CF1. . . .         NUMB     0020H    A        188# 1276
CF2. . . .         NUMB     0021H    A        189#
CF3. . . .         NUMB     0022H    A        190#
CF4. . . .         NUMB     0023H    A        191#
CHKCMD . .       C ADDR     0166H    A        466 470#
CLRWR2 . .       C ADDR     0587H    A        602 1349#
CLRWRT . .       C ADDR     0585H    A        1339 1348#
CODEOK . .       C ADDR     017EH    A        479 484#
COEF48 . .       C ADDR     0515H    A        800 1256#
COELP1 . .       C ADDR     0521H    A        1265# 1266
COELP2 . .       C ADDR     0526H    A        1269# 1273
COELP3 . .       C ADDR     0537H    A        1279# 1283 1286
COELP4 . .       C ADDR     0549H    A        1292# 1306
COEWRT . .       C ADDR     0564H    A        1295 1308#
CR48MV . .       C ADDR     0437H    A        782 1093#
CRM. . . .         NUMB     0099H    A        22#
DAT. . . .         NUMB     0044H    A        12# 438 809 1251
DCD48. . .       C ADDR     01A2H    A        522 555# 951 1303
DCDFIN . .       C ADDR     01C3H    A        555 581#
DCDOK. . .       C ADDR     024FH    A        750 753#
DCDVAX . .       C ADDR     0244H    A        383 395 745#
DCI. . . .         NUMB     0022H    A        8#
DCNX10 . .       C ADDR     028AH    A        787 789#
DCNX11 . .       C ADDR     028FH    A        790 792#
DCNX12 . .       C ADDR     0294H    A        793 795#
DCNX13 . .       C ADDR     0299H    A        796 800#
DCNXT0 . .       C ADDR     0258H    A        757 759#
DCNXT1 . .       C ADDR     025DH    A        760 762#
DCNXT2 . .       C ADDR     0262H    A        763 765#
DCNXT3 . .       C ADDR     0267H    A        766 768#
DCNXT4 . .       C ADDR     026CH    A        769 771#
```

```
DCNXT5 . .  C ADDR    0271H   A       772 774#
DCNXT6 . .  C ADDR    0276H   A       775 777#
DCNXT7 . .  C ADDR    027BH   A       778 780#
DCNXT8 . .  C ADDR    0280H   A       781 783#
DCNXT9 . .  C ADDR    0285H   A       784 786#
DCVFIN . .  C ADDR    029BH   A       746 802#
DECODE . .  C ADDR    00E1H   A       377 381# 484
DISINT . .  C ADDR    02F0H   A       761 856#
ECI. . . .    NUMB    0011H   A       41# 1461
ENINT. . .  C ADDR    02C4H   A       758 831#
FAULPi . .  C ADDR    02B8H   A       822# 824
FAULT. . .  C ADDR    02B6H   A       810 813 816 821#
GETLPi . .  C ADDR    010BH   A       406# 410 425
GETRAM . .  C ADDR    00FFH   A       356 400#
GLBLP1 . .  C ADDR    058FH   A       1354# 1356
GLBLWR . .    NUMB    0080H   A       158# 254 600 1336
GLOBAL . .  C ADDR    056FH   A       895 1199 1207 1336# 1344 1346
GLOBL. . .    NUMB    0030H   A       129# 894 1198 1206 1348
GRIPPR . .    NUMB    0038H   A       213# 1285
HDWRST . .    NUMB    00AAH   A       24# 288 326 327 449
HIBTMV . .    NUMB    0030H   A       207#
HIBTOK . .  C ADDR    0642H   A       1498 1501#
HIBTPS . .    NUMB    0034H   A       211#
HLT. . . .    NUMB    00BBH   A       26# 481 601
HWRST. . .  C ADDR    01E3H   A       575 605#
HYSTPW . .    NUMB    0039H   A       215#
IDLE . . .    NUMB    0000H   A       154# 269 344 524 958 974 1029 1074 1160
                                      1197 1240 1311 1341 1350
IE . . . .  D ADDR    00A8H   A       323 333 335
INCODE . .  C ADDR    00D3H   A       351 367 373#
INTCDB . .    NUMB    002DH   A       124# 391 393 394 857 1503
INTCMD . .    NUMB    002CH   A       123# 846 856 1503
INTFIN . .  C ADDR    0649H   A       1502 1505#
INTRPO . .  C ADDR    0636H   A       230 1493#
INV. . . .    NUMB    00FFH   A       32# 843
IP . . . .  D ADDR    00B8H   A       322
ISITGO . .  C ADDR    016CH   A       471 474#
KFF. . . .    NUMB    00FFH   A       63# 1308 1443
LCHEXT . .  C ADDR    0398H   A       966 972#
LDMOVS . .  C ADDR    0445H   A       1099 1103#
LIMLCH . .    NUMB    0088H   A       160#
LNKLCH . .    NUMB    0098H   A       161# 262
LOBTMV . .    NUMB    002FH   A       206#
LOBTPS . .    NUMB    0033H   A       210# 1040
LONGOV . .    NUMB    003BH   A       217#
MAIN . . .  C ADDR    00A1H   A       344# 374 389 392 396 443 453
MG1. . . .    NUMB    0024H   A       193#
MG2. . . .    NUMB    0025H   A       194#
MG3. . . .    NUMB    0026H   A       195#
MG4. . . .    NUMB    0027H   A       196#
MOVE . . .    NUMB    00AAH   A       54# 478 1437
MOVE48 . .  C ADDR    043CH   A       785 1098#
MOVFIN . .  C ADDR    04CEH   A       1202#
MOVLP0 . .  C ADDR    044FH   A       1108# 1112 1157
MOVLP1 . .  C ADDR    0498H   A       1167# 1169
MOVLP2 . .  C ADDR    04AAH   A       1177# 1181 1195
MRDATA . .  C ADDR    0233H   A       684 708#
MTR1BT . .    NUMB    0002H   A       181# 401 869 993 1037 1105 1175 1215
MTRAM1 . .    NUMB    0018H   A       156# 400 868 992 1036 1104 1174 1214
MTRLNK . .    NUMB    0024H   A       113# 268 271 280 329 346 350 354 370 408
                                      513 584 875 925 941 947 1000 1045 1098
                                      1110 1179 1222 1271 1293 1299 1338 1343
MTRNMR . .    NUMB    0025H   A       114# 276 284 298 509 517 530 561 912 1257
MTRSTL . .    NUMB    00DDH   A       29# 461 482
NEWRST . .  C ADDR    0140H   A       360 449#
NEWSTL . .    NUMB    0023H   A       111# 457 460 599
NOCOMM . .  C ADDR    00ACH   A       347 350#
NODATA . .  C ADDR    05B4H   A       1403# 1463
NOREAD . .  C ADDR    055FH   A       1301 1305#
NOT4PK . .  C ADDR    061EH   A       1457 1461#
NOT5PK . .  C ADDR    0613H   A       1451 1455#
NOTALW . .  C ADDR    0176H   A       475 481#
```

```
NOTFLL . .  C ADDR    037EH   A       949 953#
NOTKFF . .  C ADDR    0608H   A       1445 1449#
NOTMOV . .  C ADDR    05FDH   A       1439 1443#
NOTNOP . .  C ADDR    022CH   A       701 704#
NOTRDY . .  C ADDR    047BH   A       1141 1146#
NOTRMP . .  C ADDR    05F2H   A       1433 1437#
NOTVAX . .  C ADDR    05E7H   A       1424 1431#
NR . . . .    NUMB    003CH   A       138# 386 695 907 1167 1395 1402
NRBUFF . .    NUMB    003BH   A       137# 388 695 696 902 908 1162 1168 1434
                                      1440 1446 1452 1458 1464
NT . . . .    NUMB    003AH   A       136# 373 385 683 709 712 1425
NTBUFF . .    NUMB    0039H   A       135# 387 435 985 1027 1072 1250 1425 1426
NXMTR1 . .  C ADDR    006AH   A       282 289 296#
NXMTR2 . .  C ADDR    0073H   A       298 301#
NXMTR3 . .  C ADDR    0199H   A       515 526#
NXMVNF . .    NUMB    002CH   A       203# 1131
NXMVSN . .    NUMB    0031H   A       208# 1114
OVR. . . .    NUMB    0088H   A       20# 588 818
P1 . . . .  D ADDR    0090H   A       263 279 345 406 512 583 873 923 940 946
                                      973 998 1043 1108 1177 1220 1269 1292
                                      1298 1337 1342
P2 . . . .  D ADDR    00A0H   A       254 262 269 286 344 400 419 421 519 524
                                      556 600 868 886 888 916 936 938 945 953
                                      958 972 974 992 1013 1015 1029 1036 1056
                                      1058 1074 1104 1149 1151 1160 1174
                                      1186 1188 1197 1214 1233 1235 1240 1262
                                      1288 1290 1297 1305 1311 1336 1341 1350
PASSON . .  C ADDR    04D0H   A       788 791 797 1094 1206#
PDT. . . .    NUMB    0066H   A       16# 815
PK48 . . .    NUMB    0033H   A       44# 433 832 1413 1455
PK48RM . .    NUMB    00DDH   A       80# 894 956
PK51 . . .    NUMB    0044H   A       45# 835 983 1410 1449
PKRDY. . .  C ADDR    01EFH   A       579 614#
POKE . . .  C ADDR    032FH   A       865 902#
PREWRI . .  C ADDR    02E8H   A       819 827 844 850# 860 961 988 1032 1077
                                      1202 1210 1252 1313
PROCSS . .  C ADDR    0225H   A       687 690 699#
PRQ. . . .    NUMB    0066H   A       47# 841
PRVMTR . .    NUMB    0030H   A       130# 560 1352
PRVXWR . .    NUMB    003DH   A       140# 327 369 439 450 462 697 713 808 811
                                      814 826 851 1086 1087
PSW. . . .  D ADDR    00D0H   A       1382 1384 1472 1493 1506
RAM48. . .  C ADDR    02FBH   A       764 864#
RAM51. . .  C ADDR    038EH   A       767 965#
RAMADR . .    NUMB    0029H   A       118# 427 864 871 879 929 965 977 1217 1242
                                      1415
RAMDAT . .    NUMB    0028H   A       198# 404
RAMFIN . .  C ADDR    03A0H   A       970 976#
RAMINT . .  C ADDR    0393H   A       968#
RAMLP1 . .  C ADDR    030AH   A       873# 877 890
RAMLP2 . .  C ADDR    033AH   A       907# 909
RAMLP3 . .  C ADDR    034CH   A       919# 920
RAMLP4 . .  C ADDR    0351H   A       923# 927
RAMLP5 . .  C ADDR    0368H   A       940# 954
RAMRDY . .    NUMB    0028H   A       117# 353 436 615 892 1501
RAMVEL . .    NUMB    002EH   A       205# 1172
RAMWRT . .  C ADDR    0383H   A       943 956#
RAP. . . .    NUMB    00EEH   A       31#
RCVDAT . .    NUMB    0077H   A       18# 848 960 1201 1312
RCVDNM . .    NUMB    002DH   A       204#
RCVR . . .  C ADDR    05A6H   A       1386 1394#
RDBOT. . .    NUMB    0011H   A       104# 831 834 837 840 846 911 933 1103 1171
                                      1256 1275 1415
RDTOP. . .    NUMB    001FH   A       105#
RDY48. . .    NUMB    0021H   A       108# 329 366 370 611 1144
RDYCHK . .  C ADDR    00C4H   A       363 366#
RDYNX. . .  C ADDR    01E9H   A       577 609#
RDYNXT . .    NUMB    0033H   A       10# 368 1100
RESLP1 . .  C ADDR    0186H   A       512# 530
RESPCD . .    NUMB    003EH   A       141# 368 369 438 439 449 450 461 462 482
                                      641 688 689 818 822 826 843 848 851
                                      859 904 960 987 1031 1076 1086 1100 1164
```

```
                                                1201 1209 1251 1312
RESPND . .  C ADDR     0180H   A                348 509# 1345
RMPLP1 . .  C ADDR     04E6H   A                1220# 1224 1238
RNOCKH . .    NUMB     002BH   A                121# 431 981 1025 1070 1246 1499
RNOCKL . .    NUMB     002AH   A                120# 330 429 979 1023 1068 1244 1496 1497
RSET48 . .    NUMB     0020H   A                107# 271 293 294 296 358 451 606
RSND48 . .  C ADDR     01D1H   A                571 593#
RSNDVX . .  C ADDR     042CH   A                779 1086#
RSNRQ1 . .  C ADDR     01C4H   A                567 583# 586
RSNRQ2 . .  C ADDR     029FH   A                751 808#
SBUF . . .  D ADDR     0099H   A                326 704 711 1400 1419
SCON . . .  D ADDR     0098H   A                311 317 319 320 1386 1389 1390 1394
SEROUT . .  C ADDR     0201H   A                647 681# 1391 1469
SETFLG . .  C ADDR     0240H   A                692 716#
SHRDRM . .  C ADDR     04DAH   A                794 1214#
SHRPK. . .    NUMB     00DDH   A                59# 1248 1407 1431
SHRTOV . .    NUMB     003CH   A                218#
SND51. . .    NUMB     0011H   A                6# 694
SNDDAT . .  C ADDR     042BH   A                776 1081#
SNDHLT . .  C ADDR     01DBH   A                600# 607
SNDVX. . .    NUMB     0077H   A                49# 1423
SP . . . .  D ADDR     0081H   A                306
SPEED. . .    NUMB     002AH   A                201# 995
SRLCOD . .  C ADDR     0501H   A                1396 1419#
SRLDAT . .  C ADDR     05ACH   A                1398#
SRLFIN . .  C ADDR     0631H   A                1392 1414 1416 1468 1471#
SRLISR . .  C ADDR     0594H   A                243 1382#
SRLJMP . .  C ADDR     05CFH   A                1389 1402 1416# 1421
SRLOUT . .  C ADDR     062AH   A                1428 1467#
SRLXMT . .  C ADDR     0627H   A                1435 1441 1447 1453 1459 1466#
SRRDBF . .    NUMB     0010H   A                103# 1398 1403 1404 1419 1420 1427 1432
                                                1438 1444 1450 1456 1462
SRWRBF . .    NUMB     003FH   A                142# 686 699 702 704 705 708
STAL48 . .  C ADDR     014DH   A                364 457#
STKBAS . .    NUMB     004FH   A                147# 306
STL48. . .  C ADDR     01D4H   A                573 597#
STLCHK . .  C ADDR     00BEH   A                359 362#
STLL48 . .    NUMB     0022H   A                109# 362 598 1093
STLRSP . .  C ADDR     015CH   A                458 465#
STRTPW . .    NUMB     003AH   A                216#
TCON . . .  D ADDR     0088H   A                315 332
TH1. . . .  D ADDR     008DH   A                314
TMOD . . .  D ADDR     0089H   A                313
VALCMD . .  C ADDR     02E2H   A                833 836 839 842 846#
VAXBUF . .    NUMB     002EH   A                126# 378 379 467 468 1403 1406 1409 1412
VAXCMD . .    NUMB     002FH   A                127# 376 378 381 393 465 467 470 477 481
                                                745 749 753 802 850 859 897 987 1031
                                                1076 1088 1198 1206 1209
VDT. . . .    NUMB     0055H   A                15# 812
VLC2S. . .  C ADDR     03D2H   A                1005 1010#
VLCLP1 . .  C ADDR     03C1H   A                998# 1002 1021
VLCREQ . .  C ADDR     03B5H   A                770 992#
VRQ. . . .    NUMB     0055H   A                46# 838
WRBOT. . .    NUMB     0040H   A                144# 403 969 973 976 994 1039 1216
WRIRDY . .  C ADDR     01FEH   A                639 647#
WRITVX . .  C ADDR     01F4H   A                371 441 452 463 483 638# 642 823 852 905
                                                1089 1101 1165
WRTOP. . .    NUMB     004FH   A                145#
XMTFLG . .    NUMB     0038H   A                133# 638 681 691 693 716 903 1163 1466 1467

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8051

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX X. LISTING OF THE 8748 ASSEMBLY LANGUAGE PROGRAM

This program was assembled on the Intel MDS-80 Development System, using the MCS-48/UPI-41 Macro Assembler, Version 3.0. The cross-reference and no-page-numbering options were specified.

```
ASM48 :F2:P48V43.ASM XREF NOPAGING
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0                PAGE    1

LOC  OBJ          LINE        SOURCE STATEMENT
                     1 ;                 COMMUNICATION CODES FROM 8748 TO 8751:
                     2 ;                 -------------------------------------
                     3 ;
                     4 ;NOP      EQU     00H          ; NO OPERATION CODE.
                     5 ;                              ;
0088                 6 OVR       EQU     088H         ; RESEND REQUEST CODE.
0099                 7 MTRSTL    EQU     099H         ; MOTOR STALLED CODE.
00AA                 8 HDWRST    EQU     0AAH         ; HARDWARE RESET UNDERWAY CODE.
00BB                 9 RDYNXT    EQU     0BBH         ; READY FOR NEXT MOVE CODE.
00CC                10 RAMPK     EQU     0CCH         ; INTERNAL RAM PEEK COMPLETE.
                    11 ;
                    12 ;.........................
                    13 ;
                    14 ;                 COMMUNICATION CODES FROM 8751 TO 8748:
                    15 ;                 -------------------------------------
                    16 ;
                    17 ; NOP    EQU     00H          ; NO OPERATION
                    18 ; OVR    EQU     088H         ; REQUEST TO SEND PREVIOUS CODE
                    19 ; CRM    EQU     099H         ; CLEAR UNEXECUTED MOVES
                    20 ; MOV    EQU     0AAH         ; GET NEXT MOVE AND VELOCITY FROM RAM
                    21 ; HLT    EQU     0BBH         ; HALT MOTOR
                    22 ; AGO    EQU     0CCH         ; GO (EXECUTE MOVES)
                    23 ; PK48   EQU     0DDH         ; INTERNAL RAM PEEK/POKE.
                    24 ; RAP    EQU     0EEH         ; RESET ABSOLUTE POSITION
                    25 ; KFF    EQU     0FFH         ; GET CONTROL COEFFICIENTS FROM RAM
                    26 ;
                    27 ;.........................
                    28 ;
                    29 ;                 DEVICE ADDRESSES:
                    30 ;                 -----------------
                    31 ;                    UPPER 3 BITS TO 3 X 8 DECODER, THROUGH PORT 2)
                    32 ;                             ;
0000                33 IDLE      EQU     0H           ; IDLE (NO DEVICE ADDRESSED).
                    34 ;                             ;
0020                35 CLCKHI    EQU     20H          ; CLOCK HIGH BYTE ADDRESS.
0060                36 CLCKLO    EQU     60H          ; CLOCK LOW BYTE ADDRESS.
                    37 ;                             ;
0080                38 MLBXEN    EQU     80H          ; MAILBOX ENABLE ADDRESS.
00A0                39 PWLTEN    EQU     0A0H         ; PULSEWIDTH LATCH ENABLE ADDRESS.
                    40 ;                             ;
00C0                41 XRAMEN    EQU     0C0H         ; EXTERNAL RAM ENABLE.
00E0                42 BITDAT    EQU     0E0H         ; BIT DATA LATCH OUTPUT ENABLE ADDRESS.
                    43 ;
                    44 ;.........................
                    45 ;
                    46 ;            INTERNAL (INT) AND EXTERNAL (EXT) LOCATION SYMBOLS:
                    47 ;            ---------------------------------------------------
                    48 ;
0020                49 CF1       EQU     20H          ; (#32) INT/EXT COEFFICIENT FOR
                    50 ;                             ;              2-31 HOLES/SECOND (HPS).
0021                51 CF2       EQU     21H          ; (#33) INT/EXT COEFFICIENT FOR 32-63 HPS.
0022                52 CF3       EQU     22H          ; (#34) INT/EXT COEFFICIENT FOR 64-127 HPS.
0023                53 CF4       EQU     23H          ; (#35) INT/EXT COEFFICIENT FOR 128+ HPS.
                    54 ;                             ;
0024                55 MG1       EQU     24H          ; (#36) INT/EXT MAX VELOCITY ERROR FOR 2-31 HPS.
0025                56 MG2       EQU     25H          ; (#37) INT/EXT MAX VELOCITY ERROR FOR 32-63 HPS.
0026                57 MG3       EQU     26H          ; (#38) INT/EXT MAX VELOCITY ERROR FOR 64-127 HPS.
0027                58 MG4       EQU     27H          ; (#39) INT/EXT MAX VELOCITY ERROR FOR 128+ HPS.
                    59 ;                             ;
0028                60 STPCNT    EQU     28H          ; (#40) INT NUMBER OF MOTOR STOP INTERRUPTS
                    61 ;                             ;       SINCE LAST MOTION OF MOTOR.
                    62 ;                             ;
0028                63 RAMDAT    EQU     28H          ; (#40) EXT LOCATION OF DATA FOR PEEK/POKE.
0029                64 RAMADR    EQU     29H          ; (#41) INT/EXT LOC OF ADDRESS FOR PEEK/POKE.
                    65 ;                             ;
002A                66 SPEED     EQU     2AH          ; (#42) INT SPEED.
                    67 ;                             ;       EXT VELOCITY (SIGNED).
```

```
                        68
002B                    69 QHTAIL     EQU   2BH         ; (#43) INT QUARTER-HOLE TAIL LOCATION.
                        70                                ;         (NUMBER OF QUARTER-HOLES
                        71                                ;           SINCE LAST POSITION UPDATE,
                        72                                ;             RELATIVE TO COMMAND DIRECTION)
                        73                                ;
002C                    74 NXMVNF     EQU   2CH         ; (#44) INT/EXT 'NEXT MOVE INFINITE' FLAG.
002D                    75 RCVDNM     EQU   2DH         ; (#45) INT/EXT 'RECEIVED NEXT MOVE' FLAG.
                        76                                ;
002E                    77 NXTVEL     EQU   2EH         ; (#46) INT/EXT NEXT VELOCITY LOCATION.
002F                    78 NXMVLO     EQU   2FH         ; (#47) INT/EXT NEXT MOVE LOW BYTE LOCATION.
0030                    79 NXMVHI     EQU   30H         ; (#48) INT/EXT NEXT MOVE HIGH BYTE LOCATION.
0031                    80 NXMVSN     EQU   31H         ; (#49) INT/EXT NEXT MOVE SIGN LOCATION.
                        81                                ;
0032                    82 PRSVEL     EQU   32H         ; (#50) INT PRESENT VELOCITY LOCATION.
0033                    83 PRMVLO     EQU   33H         ; (#51) INT PRESENT MOVE LOW BYTE LOCATION.
                        84                                ;       EXT PRESENT ABSOLUTE POSITION LOW BYTE.
0034                    85 PRMVHI     EQU   34H         ; (#52) INT PRESENT MOVE HIGH BYTE LOCATION.
                        86                                ;       EXT PRESENT ABSOLUTE POSITION HIGH BYTE.
0035                    87 PRMVSN     EQU   35H         ; (#53) INT PRESENT MOVE SIGN LOCATION.
                        88                                ;
0036                    89 PLSWTH     EQU   36H         ; (#54) INT PULSEWIDTH LOCATION.
0037                    90 SIGNPW     EQU   37H         ; (#55) INT PULSEWIDTH SIGN.
                        91                                ;
0038                    92 GRIPPR     EQU   38H         ; (#56) INT/EXT GRIPPER (CLOSED) PULSEWIDTH
                        93                                ;
0039                    94 HYSTPW     EQU   39H         ; (#57) INT/EXT HYSTERESIS PULSEWIDTH.
003A                    95 STRTPW     EQU   3AH         ; (#58) INT/EXT STARTING PULSEWIDTH.
                        96                                ;
003B                    97 LONGOV     EQU   3BH         ; (#59) INT/EXT LONG OVERSHOOT PULSEWIDTH.
003C                    98 SHRTOV     EQU   3CH         ; (#60) INT/EXT SHORT OVERSHOOT PULSEWIDTH.
                        99                                ;
003F                   100 DELTAV     EQU   3FH         ; (#63) INT CHANGE IN VELOCITY SINCE LAST INTERRUPT.
                       101 ;
                       102 ;.........................
                       103 ;
                       104 ;               REGISTER BANK ZERO ASSIGNMENTS
                       105 ;
                       106 ;;
                       107 ;               R0 - R5  ARE WORKING REGISTERS.
                       108 ;
                       109 ;               R6 HOLDS PREVIOUS CODE WRITTEN TO 8751.
                       110 ;                       ('PREVIOUS WRITE BUFFER')
                       111 ;
                       112 ;               R7 HOLDS CODE TO BE WRITTEN TO 8751 ('WRITE BUFFER').
                       113 ;
                       114 ;.........................
                       115 ;
                       116 ;               REGISTER BANK ONE ASSIGNMENTS
                       117 ;               ----------------------------
                       118 ;
                       119 ;               R0 - R1 ARE WORKING REGISTERS.
                       120 ;
                       121 ;               R2 HOLDS CURRENT QUARTER-HOLE MOVE.
                       122 ;
                       123 ;               R3-R4 ARE WORKING REGISTERS.
                       124 ;
                       125 ;               R5 HOLDS CURRENT VELOCITY SET POINT.
                       126 ;
                       127 ;               R6 HOLDS VARIOUS FLAGS:
                       128 ;                    BIT ZERO = 'RELATIVE DIRECTION' FLAG
                       129 ;                               (DIRECTION RELATIVE TO COMMAND)
                       130 ;
                       131 ;                    BIT TWO  = 'NEW MOTION' FLAG
                       132 ;                               (MOTOR STOPPED JUST PRIOR TO THIS INTERRUPT)
                       133 ;
                       134 ;                    BIT FOUR = 'PREVIOUS ENCODER POSITION 0,0' FLAG
                       135 ;                               (MOTOR JUST LEFT 0,0 POSITION)
                       136 ;
                       137 ;                    BIT SEVEN = 'ABSOLUTE DIRECTION' FLAG
                       138 ;                               (ABSOLUTE DIRECTION OF MOTOR MOTION)
                       139 ;
```

```
                    ;               R7 HOLDS VARIOUS FLAGS:
                    ;                    BIT ZERO = 'REVERSAL SINCE LAST UPDATE' FLAG
                    ;                         (DIRECTION REVERSED AFTER LAST POSITION UPDATE)
                    ;
                    ;                    BIT ONE  = 'MOTOR STOPPED' FLAG
                    ;                         (MOTOR CURRENTLY STOPPED)
                    ;
                    ;                    BIT TWO  = RESERVED (MUST BE ZERO).
                    ;
                    ;                    BIT THREE = 'PRESENT ENCODER POSITION 0,0' FLAG
                    ;                              (VALID ONLY AFTER EXECUTION OF
                    ;                                   POSITION UPDATE SUBROUTINE)
                    ;
                    ;                    BIT FIVE = LIMIT SWITCH STATE.
                    ;                    BIT SIX = PRESENT STATE OF ENCODER BIT ONE.
                    ;                    BIT SEVEN = PRESENT STATE OF ENCODER BIT ZERO.
                    ;
                    ;.........................
                    ;       ASSIGNMENT OF MICROPROCESSOR FLAGS:
                    ;       ----------------------------------
                    ;
                    ;       FLAG ZERO (F0) = 'GO' FLAG
                    ;                              (CLEAR INDICATES MOTOR IS HALTED)
                    ;
                    ;       FLAG ONE (F1) = 'STALL' FLAG
                    ;                                  (SET INDICATES MOTOR IS STALLED)
                    ;
                    ;.........................
                    ;             MISCELLANEOUS BIT MASKS AND VALUES:
                    ;             -----------------------------------
                    ;
0080                SGNMSK  EQU   80H          ; BIT MASK TO ISOLATE SIGN BIT.
00E0                MASK32  EQU   0E0H         ; BIT MASK TO ISOLATE TOP 3 BITS.
00FF                ALL1S   EQU   0FFH         ; BIT MASK COMPOSED OF ALL ONES.
00FE                UPPER7  EQU   0FEH         ; BIT MASK WITH UPPER SEVEN BITS SET.
007F                LOWER7  EQU   7FH          ; BIT MASK WITH LOWER SEVEN BITS SET.
                    ;
                    ;.....................................................................
                    ;.....................................................................
                    ;
                    ;        INTERRUPT VECTORING:
                    ;        --------------------
                    ;
0000                        ORG   0            ; VECTOR LOCATION FOR HARDWARE RESET (POWER-UP).
                            ;
0000 0408           BEGIN:  JMP   BEGIN0       ; GO TO INITIALIZATION PROCEDURE.
0002 00                     NOP                ;
                            ;
                            ;
                            ; VECTOR LOCATION FOR EXTERNAL INTERRUPT.
                            ;
0003 8AE0                   ORL   P2,#BITDAT   ; WRITE INTERRUPT ACKNOWLEDGE TO PORT 2
                            ;                  ;    (NOTE THIS IS ALSO ENCODER DATA OUTPUT ENABLE.)
                            ;
0005 248B                   JMP   ISR          ; GO TO INTERRUPT SERVICE ROUTINE.
                            ;
                            ;
                            ; VECTOR LOCATION FOR TIMER INTERRUPT
                            ;
0007 93                     RETR               ; NO USE FOR THIS INTERRUPT.  IF IT OCCURS,
                            ;                  ; JUST RETURN TO THE POINT OF INTERRUPT.
                    ;
                    ;.....................................................................
                    ;
                    ;                  INITIALIZATION PROCEDURE:
                    ;                  -------------------------
                    ;                        (EXECUTED ONLY ON HARDWARE RESET)
                    ;
0008 9AE0           BEGIN0: ANL   P2,#BITDAT   ; ENABLE ENCODER DATA LATCH OUTPUT (#0E0H).
```

```
000A 09          212           IN    A,P1          ; OBTAIN THE ENCODER DATA.
000B D303        213           XRL   A,#3H         ; SET 'MOTOR STOPPED' FLAG, AND
000D AF          214           MOV   R7,A          ;   CLEAR 'REVERSAL SINCE LAST UPDATE' FLAG.
                 215                               ;
000E 9AC0        216           ANL   P2,#XRAMEN    ; LEAVE ADDRS OF RAM-ENABLE ON PORT 2 (#0C0H)
0010 2610        217 BEGLP1:   JNTO  BEGLP1        ; POLL RAM LOCKOUT UNTIL ACCESS
                 218                               ;                        TO SHARED RAM IS GAINED.
0012 27          219           CLR   A             ;
0013 B83F        220           MOV   R0,#3FH       ; CLEAR
0015 90          221 BEGLP2:   MOVX  @R0,A         ;    EXTERNAL RAM LOCATIONS #1 - #63 AND
0016 A0          222           MOV   @R0,A         ;      INTERNAL RAM LOCATIONS #1 - #63.
0017 E815        223           DJNZ  R0,BEGLP2     ;
                 224                               ;
0019 23AA        225           MOV   A,#HDWRST     ; LOAD CODE FOR 'HARDWARE RESET'.
001B 90          226           MOVX  @R0,A         ; TEST THE EXTERNAL RAM
001C 80          227           MOVX  A,@R0         ; BY A WRITE/READ OF LOCATION ZERO.
                 228                               ;
001D 9A80        229           ANL   P2,#MLBXEN    ; ENABLE THE MAILBOX.
001F 02          230           OUTL  BUS,A         ; WRITE THE HARDWARE RESET CODE TO MAILBOX.
0020 AE          231           MOV   R6,A          ; SAVE COPY OF CODE IN 'PREVIOUS WRITE' BUFFER.
                 232                               ;
0021 27          233           CLR   A             ;
0022 342C        234           CALL  PWWRIT        ; GO WRITE ZERO PULSEWIDTH TO PW LATCH.
                 235                               ;
                 236 ;.....................................................................
                 237 ;.....................................................................
                 238 ;
                 239 ; THIS BEGINS THE MAIN ROUTINE.  ESSENTIALLY, IT COMMUNICATES WITH
                 240 ; THE 8751, RESPONDING APPROPRIATELY TO COMMANDS, AND SENDING THE
                 241 ; NECESSARY MESSAGES DEPENDING ON INTERNAL STATES.
                 242                               ;
0024 05          243 MAIN48:   EN    I             ; STROBE INTERRUPT ENABLE TO PICK UP
0025 15          244           DIS   I             ;     THE INTERRUPT CONDITION.
                 245                               ;
0026 C5          246           SEL   RB0           ; SELECT REGISTER BANK ZERO.
                 247                               ;
0027 FF          248           MOV   A,R7          ; GET CONTENTS OF WRITE BUFFER.
0028 F269        249           JB7   WRITE         ; IF BIT 7 IS SET, GO WRITE CODE TO MAILBOX.
                 250                               ;
002A 9A00        251 RDMBX:    ANL   P2,#IDLE      ;
002C 4624        252           JNT1  MAIN48        ; IF WRITE MAILBOX IS EMPTY, RESTART MAIN.
                 253                               ;
002E 8A80        254           ORL   P2,#MLBXEN    ; IF FULL, PUT MAILBOX ENABLE ON PORT 2.
0030 08          255           INS   A,BUS         ; INPUT CODE FROM MAILBOX.
0031 C624        256           JZ    MAIN48        ; IF CODE IS ZERO, NO OPERATION.
                 257                               ;
0033 A9          258           MOV   R1,A          ; STORE INPUT CODE IN R1.
0034 47          259           SWAP  A             ; VALIDATE INPUT CODE BY
0035 D9          260           XRL   A,R1          ;   TESTING FOR NIBBLE-WISE SYMMETRY.
0036 9667        261           JNZ   RSNDRQ        ; IF CODE IS NOT VALID, GO REQUEST RESEND.
                 262                               ;
0038 F9          263           MOV   A,R1          ; OBTAIN ORIGINAL CODE AGAIN.
0039 5307        264           ANL   A,#7H         ; ISOLATE LOWEST THREE BITS.
003B C678        265           JZ    RESEND        ; IF CODE WAS 88H, WRITE PREVIOUS CODE TO MAILBOX.
                 266                               ;
003D D5          267           SEL   RB1           ; SELECT REGISTER BANK ONE.
                 268                               ;
003E 07          269           DEC   A             ;
003F C67B        270           JZ    CLRMVS        ; IF CODE WAS 99H, GO CLEAR ALL MOVE COMMANDS.
                 271                               ;
0041 07          272           DEC   A             ;
0042 C6B3        273           JZ    MOVE          ; IF CODE WAS 0AAH, A NEW MOVE IS IN SHARED RAM.
0044 07          274           DEC   A             ;
0045 C689        275           JZ    HALT          ; IF CODE WAS 0BBH, GO HALT THE MOTOR.
0047 07          276           DEC   A             ;
0048 C6AD        277           JZ    GO            ; IF CODE WAS 0CCH, BEGIN (CONTINUE) MOTION.
004A 07          278           DEC   A             ;
004B C697        279           JZ    INTRAM        ; IF CODE WAS 0DDH, GO PEEK OR POKE INTERNAL RAM.
004D 07          280           DEC   A             ;
004E C68C        281           JZ    RESET         ; IF CODE WAS 0EEH, GO RESET ABSOLUTE POSITION.
                 282                               ;
                 283                               ; OTHERWISE CODE WAS 0FFH, SO GET COEFFICIENTS.
```

```
                        ;                       (FALL THROUGH TO THAT ROUTINE.)
                        ;.......................;
                        ;
                        ; GET NEW SET OF COEFFICIENTS FOR CONTROL ALGORITHM:
                        ;
0050 8AC0               ORL   P2,#XRAMEN        ; PUT RAM ENABLE ON PORT 2.
0052 2652       COEFS:  JNTO  COEFS             ; IF RAM IS BUSY, KEEP WAITING FOR ACCESS.
                        ;
0054 B820               MOV   R0,#CF1           ; LOAD FIRST COEFFICIENT ADDRESS.
0056 BB02               MOV   R3,#2H            ; USE R3 AS OUTER LOOP COUNTER.
0058 BC08               MOV   R4,#8H            ; USE R4 AS INNER LOOP COUNTER.
                        ;
005A 80         COELP1: MOVX  A,@R0             ; MOVE IN THE COEFFICIENTS
005B A0                 MOV   @R0,A             ;     FROM EXTERNAL RAM TO
005C 18                 INC   R0                ;        CORRESPONDING LOCATIONS IN INTERNAL RAM.
005D EC5A               DJNZ  R4,COELP1         ;
005F B838               MOV   R0,#GRIPPR        ; LOAD FIRST PARAMETER ADDRESS.
0061 BC06               MOV   R4,#6H            ; USE R4 AS INNER LOOP COUNTER AGAIN.
0063 EB5A               DJNZ  R3,COELP1         ; RUN INNER LOOP A SECOND TIME FOR PARAMETERS.
                        ;
0065 0424               JMP   MAIN48            ;
                        ;
                        ;.......................;
                        ;
                        ; REQUEST A RESEND OF LAST CODE:
                        ;
0067 BF88       RSNDRQ: MOV   R7,#OVR           ; MOVE RESEND REQUEST INTO R7.
                        ;                       ;  (FALL THROUGH TO WRITE ROUTINE.)
                        ;.......................;
                        ;
                        ; WRITE CODE IN WRITE BUFFER (R7) TO MAILBOX:
                        ;
0069 27         WRITE:  CLR   A                 ;
006A 2F                 XCH   A,R7              ; CLEAR WRITE BUFFER AND
006B AE                 MOV   R6,A              ;     PUT CODE INTO PREVIOUS WRITE BUFFER.
                        ;
006C 05         WRILP1: EN    I                 ; STROBE INTERRUPT LINE IN CASE AN INTERRUPT
006D 15                 DIS   I                 ;     OCCURS WHILE WAITING TO WRITE TO MAILBOX.
006E 9A00               ANL   P2,#IDLE          ;
0070 8A80               ORL   P2,#MLBXEN        ; PUT MAILBOX ADDRESS ON PORT 2.
0072 566C               JTI   WRILP1            ; IF WRITE MAILBOX IS FULL, PREVIOUS CODE HAS NOT
                        ;                       ;     YET BEEN READ BY 8751, SO KEEP WAITING.
                        ;
0074 FE                 MOV   A,R6              ; OBTAIN CODE TO BE WRITTEN.
0075 02         NOPOLL: OUTL  BUS,A             ; WRITE IT TO MAILBOX.
0076 042A               JMP   RDMBX             ;
                        ;
                        ;.......................;
                        ;
                        ; OBTAIN PREVIOUS CODE TO EXECUTE A RESEND TO 8751:
                        ;
0078 2E         RESEND: XCH   A,R6              ; EXCHANGN NOP CODE FOR PREVIOUS CODE.
0079 0475               JMP   NOPOLL            ; WRITE TO MAILBOX WITHOUT POLLING.
                        ;
                        ;.......................;
                        ;
                        ; CLEAR ALL REMAINING MOVES AND RELATED INFORMATION:
                        ;
007B A5         CLRMVS: CLR   F1                ; CLEAR THE STALL FLAG.
                        ;
007C B92C               MOV   R1,#NXMVNF        ; LOAD ADRS OF FIRST MOVE DATA LOCATION.
007E B80B               MOV   R0,#11D           ; USE R0 AS A LOOP COUNTER.
0080 A1         CLRLP1: MOV   @R1,A             ; CLEAR ALL THE MOVE INFORMATION LOCATIONS,
0081 19                 INC   R1                ;     EXCEPT PULSEWIDTH SIGN.
0082 E880               DJNZ  R0,CLRLP1         ;
                        ;
0084 AD                 MOV   R5,A              ; CLEAR VELOCITY SET POINT.
                        ;
0085 340C               CALL  RDYMSG            ; GO PUT 'READY' MESSAGE IN BUFFER.
                        ;
0087 0424               JMP   MAIN48            ;
                        ;
```

```
                    356 ;.........................
                    357 ;
                    358             ; ENTER THE 'HALT' CONDITION:
                    359 ;
0089 85             360 HALT:   CLR   F0          ; CLEAR FLAG 0 TO INDICATE HALT CONDITION.
                    361 ;
008A 0424           362         JMP   MAIN48      ;
                    363 ;
                    364 ;.........................
                    365 ;
                    366             ; ZERO THE ABSOLUTE POSITION LOCATIONS IN EXTERNAL RAM:
                    367 ;
008C 8AC0           368 RESET:  ORL   P2,#XRAMEN   ; PUT RAM ENABLE ON PORT 2.
008E B933           369         MOV   R1,#PRMVLO  ; LOAD ADRS OF ABSOLUTE POSITION LOW BYTE.
                    370 ;                         ;     (SAME AS ADRS OF PRESENT MOVE LOW BYTE.)
0090 2690           371 RESLP1: JNT0  RESLP1      ; IF RAM IS BUSY, KEEP WAITING FOR ACCESS.
                    372 ;
0092 91             373         MOVX  @R1,A       ; CLEAR BOTH BYTES
0093 19             374         INC   R1          ;           OF ABSOLUTE POSITION.
0094 91             375         MOVX  @R1,A       ;
0095 0424           376         JMP   MAIN48      ;
                    377 ;
                    378 ;.........................
                    379 ;
0097 8AC0           380 INTRAM: ORL   P2,#XRAMEN   ; ENABLE EXTERNAL RAM.
0099 2699           381 INTLP1: JNT0  INTLP1      ; IF LOCKED OUT, KEEP WAITING.
                    382 ;
009B B829           383         MOV   R0,#RAMADR  ; LOAD LOCATION OF ADDRESS TO PEEK/POKE.
009D 80             384         MOVX  A,@R0       ; READ IN ADDRESS
009E A9             385         MOV   R1,A        ;     AND PUT IT IN ADDRESS POINTER 1.
009F C8             386         DEC   R0          ; DECR R0 TO DATA PASSING ADDRESS.
00A0 F2A9           387         JB7   POKE        ; IF BIT 7 OF ADDR IS SET, COMMAND IS A POKE.
                    388 ;                         ; ELSE, COMMAND IS A PEEK.
                    389 ;
00A2 F1             390 PEEK:   MOV   A,@R1       ; GET DATA FROM PEEK ADDRESS (R1).
00A3 90             391         MOVX  @R0,A       ; WRITE THE DATA TO DATA PASSING LOCATION.
                    392 ;
00A4 C5             393         SEL   RB0         ; SELECT REGISTER BANK 0.
00A5 BFCC           394         MOV   R7,#RAMPK   ; PUT 'RAM PEEK COMPLETE' CODE IN WRITE BUFFER.
00A7 0469           395         JMP   WRITE       ; GO WRITE CODE TO MAILBOX.
                    396 ;
00A9 80             397 POKE:   MOVX  A,@R0       ; OBTAIN DATA TO BE 'POKED'.
00AA A1             398         MOV   @R1,A       ; PUT IT IN PROPER INTERNAL RAM LOCATION.
00AB 0424           399         JMP   MAIN48      ;
                    400 ;
                    401 ;.........................
                    402 ;
                    403             ; ENTER THE 'GO' CONDITION:
                    404 ;
00AD 7624           405 GO:     JF1   MAIN48      ; IF STALLED, DO NOT START GOING.
                    406 ;
00AF 85             407         CLR   F0          ; CLEAR 'GO' FLAG
00B0 95             408         CPL   F0          ;      IN ORDER TO SET IT.
00B1 0424           409         JMP   MAIN48      ;
                    410 ;
                    411 ;.........................
                    412 ;
00B3 8AC0           413 MOVE:   ORL   P2,#XRAMEN   ; PUT EXTERNAL RAM ENABLE ON PORT 2.
00B5 B82C           414         MOV   R0,#NXMVNF  ; LOAD ADDRESS OF 'NEXT MOVE INFINITE' FLAG.
00B7 B906           415         MOV   R1,#6H      ; LOAD LOOP COUNTER FOR SIX BYTES OF DATA.
                    416 ;
00B9 26B9           417 MOVLP1: JNT0  MOVLP1      ; IF 8748 IS LOCKED OUT OF SHARED RAM,
                    418 ;                         ;                    KEEP WAITING.
00BB 80             419 MOVLP2: MOVX  A,@R0       ; TRANSFER NEXT MOVE DATA FROM
00BC A0             420         MOV   @R0,A       ;   EXTERNAL RAM LOCATIONS TO
00BD 18             421         INC   R0          ;     CORRESPONDING INTERNAL LOCATIONS.
00BE E9BB           422         DJNZ  R1,MOVLP2   ;
                    423 ;
00C0 0424           424         JMP   MAIN48      ;
                    425 ;
                    426 ;.....................................................................
                    427 ;.....................................................................
```

```
                        428
                        429          ; MOVE YET SUBROUTINE:
                        430                                  ; PROVIDE:  REGISTER BANK 1 SELECTED.
                        431                                  ;
                        432                                  ; RETURNS:  ACCUMULATOR IS ZERO IF THE
                        433                                  ;             MOTOR POSITION IS INCORRECT, AND
                        434                                  ;             NON-ZERO IF MOTOR POSITION IS
                        435                                  ;             CORRECT.  IF MOTOR POSITION IS
                        436                                  ;           INCORRECT, USABLE VALUES ARE RETURNED
                        437                                  ;         IN PULSEWIDTH AND PULSEWIDTH SIGN
                        438                                  ;       LOCATIONS, AND, IN THAT CASE, ON RETURN
                        439                                  ;       R0 = #SIGNPW AND R1 = #PLSWTH.
                        440                                  ;
                        441                                  ; CHANGES:  RB1: R0, R1, ACC, PSW, PLSWTH, SIGNPW
                        442                                  ;
                        443                                  ; CALLS:  NEW MOVE (NEWMOV)
                        444                                  ;
                        445                                  ; CALLED BY:  MOTOR STOPPED (MTRSTP)
                        446                                  ;
00C2 B934               447 MOVYET:  MOV   R1,#PRMVHI        ; LOAD ADDR OF PRESENT MOVE HIGH BYTE.
00C4 F1                 448          MOV   A,@R1             ; OBTAIN IT FOR INSPECTION.
00C5 96CC               449          JNZ   DETSGN            ; IF NOT ZERO, MORE TO MOVE.  GO DETERMINE SIGN.
                        450                                  ;
00C7 C9                 451          DEC   R1                ; IF ZERO, DECR R1 TO ADDR LOW BYTE PRESENT MOVE.
00C8 F1                 452          MOV   A,@R1             ; OBTAIN LOW BYTE FOR INSPECTION.
00C9 C6E5               453          JZ    CKQHTL            ; IF IT IS ALSO ZERO, GO CHECK FOR Q-H TAIL.
00CB 27                 454          CLR   A                 ; ELSE, MOVE IS POSITIVE NUMBER, SO CLEAR SIGN BIT.
                        455                                  ;
00CC 5380               456 DETSGN:  ANL   A,#SGNMSK         ; MASK OFF ALL BUT SIGN BIT OF MOVE.
00CE B835               457          MOV   R0,#PRMVSN        ; OBTAIN CORRECT PULSEWIDTH SIGN
00D0 D0                 458          XRL   A,@R0             ;   BY EXCLUSIVE OF WITH PRESENT MOVE SIGN,
00D1 B837               459          MOV   R0,#SIGNPW        ;     AND STORE IT IN INTERNAL RAM LOCATION.
00D3 A0                 460          MOV   @R0,A             ;
                        461                                  ;
00D4 B928               462          MOV   R1,#STPCNT        ; IF THE MOTOR STOP INTERRUPT COUNT
00D6 F1                 463          MOV   A,@R1             ;   IS EXACTLY THREE, THEN ADD IN THE VELOCITY
00D7 D303               464          XRL   A,#3H             ;   SET POINT TO THE PULSEWIDTH.  OTHERWISE,
00D9 96DC               465          JNZ   ADDGO             ;     JUST ADD IN THE STOP COUNT, ITSELF.
00DB FD                 466          MOV   A,R5              ;
                        467                                  ;
00DC B93A               468 ADDGO:   MOV   R1,#STRTPW        ; ADD IN THE VALUE OF THE STARTING
00DE 61                 469          ADD   A,@R1             ;   PULSEWIDTH TO THE PRESENT
00DF B936               470          MOV   R1,#PLSWTH        ;     PULSEWIDTH.
00E1 61                 471          ADD   A,@R1             ;
00E2 A1                 472          MOV   @R1,A             ; STORE NEW PULSEWIDTH.
00E3 27                 473          CLR   A                 ; CLEAR ACC TO INDICATE MORE TO MOVE.
00E4 83                 474          RET                     ;
                        475                                  ;
00E5 B92B               476 CKQHTL:  MOV   R1,#QHTAIL        ; LOAD ADDR OF QUARTER-HOLE TAIL.
00E7 F1                 477          MOV   A,@R1             ; OBTAIN IT FOR INSPECTION.
00E8 96CC               478          JNZ   DETSGN            ; IF NOT ZERO, GO DETERMINE SIGN OF MOVE.
                        479                                  ;
00EA 14F2               480          CALL  NEWMOV            ; ELSE, SEE IF NEW MOVE AWAITS EXECUTION.
00EC B936               481          MOV   R1,#PLSWTH        ;   IF NEW MOVE AWAITS, CLEAR PULSEWIDTH
00EE A1                 482          MOV   @R1,A             ;     AND
00EF C6CC               483          JZ    DETSGN            ;       GO DETERMINE NEW SIGN AND PW.
00F1 83                 484          RET                     ; IF NOT, RETURN WITH 'NO MOVE' INDICATED.
                        485                                  ;
                        486 ;.....................................................................
                        487 ;.....................................................................
                        488                                  ;
                        489          ; NEW MOVE SUBROUTINE:
                        490                                  ; PROVIDE:  REGISTER BANK 1 SELECTED.
                        491                                  ;
                        492                                  ; RETURNS:  IF NEXT MOVE HAS BEEN RECEIVED:
                        493                                  ;             MOVES NEXT MOVE DATA TO PRESENT
                        494                                  ;           MOVE DATA.  PUTS 'READY FOR NEXT
                        495                                  ;     MOVE' CODE (#0DDH) IN WRITE BUFFER, AND
                        496                                  ;     RESETS THE 'NEXT MOVE RECEIVED' FLAG.
                        497                                  ;     ALSO, ACC IS RETURNED ZERO ONLY IF NEXT
                        498                                  ;     MOVE HAS BEEN RECEIVED.
                        499                                  ;
```

```
                    500                             ; CHANGES:   RB1:  R0,R1,R3,R5,ACC,PSW, PRES MOVE
                    501                             ;                DATA, 'NEXT MOVE RCVD' FLAG, WRITE BUFFER
                    502                             ;
                    503                             ; CALLS:  NONE
                    504                             ;
                    505                             ; CALLED BY: MOVE YET SUBROUTINE (MOVYET)
                    506                             ;              INTERRUPT SERVICE ROUTINE ('ISR')
                    507                             ;
00F2 BB04           508 NEWMOV: MOV   R3,#4         ; USE R3 AS LOOP COUNTER FOR DATA TRANSFER, BELOW.
                    509                             ;
00F4 B82D           510         MOV   R0,#RCVDNM    ; INSPECT THE
00F6 F0             511         MOV   A,@R0         ;   'RECEIVED NEXT MOVE' FLAG.
00F7 96FB           512         JNZ   GETMOV        ; IF NOT SET, THEN NEXT MOVE NOT RECEIVED.
00F9 2410           513         JMP   NEWFIN        ;
                    514                             ;
00FB B000           515 GETMOV: MOV   @R0,#0        ; IF SET, CLEAR IT.
00FD 18             516         INC   R0            ; MOVE A COPY OF NEW VELOCITY
00FE F0             517         MOV   A,@R0         ;   COMMAND INTO VELOCITY SET POINT.
00FF AD             518         MOV   R5,A          ;
0100 B932           519         MOV   R1,#PRSVEL    ;
0102 F0             520 NEWLP1: MOV   A,@R0         ; TRANSFER THE NEXT MOVE DATA
0103 A1             521         MOV   @R1,A         ;   TO THE PRESENT MOVE DATA LOCATIONS.
0104 18             522         INC   R0            ;     (VELOCITY, MOVE, AND MOVE SIGN)
0105 19             523         INC   R1            ;
0106 EB02           524         DJNZ  R3,NEWLP1     ;
                    525                             ;
0108 B828           526         MOV   R0,#STPCNT    ;  CLEAR THE MOTOR STOP INTERRUPT COUNT.
010A B000           527         MOV   @R0,#0H       ;
                    528                             ;
010C B807           529 RDYMSG: MOV   R0,#7H        ; PUT 'READY FOR NEXT MOVE' CODE
010E B0BB           530         MOV   @R0,#RDYNXT   ;                       INTO WRITE BUFFER.
                    531                             ;
0110 FB             532 NEWFIN: MOV   A,R3          ; USE LOOP COUNTER AS 'NO NEW MOVE' FLAG.
0111 83             533         RET                 ;
                    534                             ;
                    535 ;.....................................................................
                    536 ;.....................................................................
                    537                             ;
                    538         ; UPDATE PULSEWIDTH SUBROUTINE:
                    539                             ;
                    540                             ; PROVIDE:  SELECT RB1, VELOCITY ERROR IN ACC,
                    541                             ;    COEFFICIENTS ADDRESS IN R0, CORRECT VALUES
                    542                             ;              FOR PARTICULAR ENTRY POINT.
                    543                             ;
                    544                             ; RETURNS:  PROPER PULSEWIDTH (BASED ON COEF-
                    545                             ;          FICIENTS SUPPLIED) WRITTEN TO
                    546                             ;          INTERNAL LOCATIONS AND TO
                    547                             ;          PULSEWIDTH LATCH.
                    548                             ;
                    549                             ; CHANGES:  RB1:  R0,R1,R3,ACC,PSW,PLSWTH,SIGNPW,
                    550                             ;                     PW LATCH IN PW MODULATOR.
                    551                             ;
                    552                             ; CALLS:  MULT
                    553                             ;
                    554                             ; CALLED BY:  INITIALIZATION ROUTINE
                    555                             ;                (VECTORS TO PWWRIT)
                    556                             ;             INTERRUPT SERVICE ROUTINE
                    557                             ;                (VARIOUS ENTRY POINTS)
                    558                             ;
0112 54E3           559 PWUPDT: CALL  MULT          ; CALL THE MULTIPLY ROUTINE.
0114 AB             560         MOV   R3,A          ; STORE PULSEWIDTH INCREMENT IN R3.
                    561                             ;
0115 B837           562         MOV   R0,#SIGNPW    ; DECREMENT R0 TO ADDRESS OF PW SIGN.
0117 F0             563         MOV   A,@R0         ; COMPARE PULSEWIDTH SIGN WITH THE SIGN
0118 B935           564         MOV   R1,#PRMVSN    ;  OF PRESENT MOVE REMAINING.  IF THEY ARE NOT
011A D1             565         XRL   A,@R1         ;   THE SAME, THEN PULSEWIDTH IS 'NEGATIVE' RELA-
011B F235           566         JB7   PWRVRS        ;    TIVE TO VELOCITY ERROR AND PW INCREMENT, SO
011D 19             567 JMPBK3: INC   R1            ;     GO TWO'S COMPLEMENT THE PW INCREMENT.
011E F1             568         MOV   A,@R1         ;
011F 6B             569         ADD   A,R3          ;      IF SIGNS ARE THE SAME, PW INCREMENT
0120 A1             570         MOV   @R1,A         ;       IS ADDED TO PULSEWIDTH WITH NO CHANGE.
                    571                             ;
```

```
0121 F23B          572           JB7   OVFLOW        ; IF BIT 7 IS SET, GO SEE WHY SUM OVERFLOWED.
                   573                               ;
0123 F1            574 PWHYST:   MOV   A,@R1         ; GET PRESENT PULSEWIDTH.
0124 C62B          575           JZ    PWSIGN        ; IF PW IS ZERO, DON'T ADD HYSTERESIS OFFSET.
0126 B939          576           MOV   R1,#HYSTPW    ; ELSE, ADD USER-SPECIFIED
0128 61            577           ADD   A,@R1         ;       HYSTERESIS OFFSET TO PW.
0129 F240          578           JB7   MAXOUT        ; IF THAT OVERFLOWS PW, GO SET PW TO MAXIMUM.
                   579                               ;
012B 40            580 PWSIGN:   ORL   A,@R0         ; CONCATENATE PW SIGN FOR WRITE TO PW LATCH.
                   581                               ;
012C 39            582 PWWRIT:   OUTL  P1,A          ; PUT NEW PULSEWIDTH ON PORT 1.
012D 8AA0          583           ORL   P2,#PWLTEN    ; PUT PULSEWIDTH LATCH ENABLE ON PORT 2.
012F 90            584           MOVX  @R0,A         ; GENERATE LATCH PULSE WITH WRITE TO DUMMY ADDRESS.
0130 9A00          585           ANL   P2,#IDLE      ; CLEAR ENABLE ADDRESS FROM PORT 2.
0132 89FF          586           ORL   P1,#ALLIS     ; RESTORE PORT 1 TO THE INPUT STATE.
0134 83            587           RET                 ;
                   588                               ;
0135 FB            589 PWRVRS:   MOV   A,R3          ;
0136 37            590           CPL   A             ; REVERSE THE SIGN OF THE
0137 17            591           INC   A             ;   PULSEWIDTH INCREMENT.
0138 AB            592           MOV   R3,A          ;
0139 241D          593           JMP   JMPBK3        ;
                   594                               ;
                   595                               ; DETERMINE REASON FOR OVERFLOWED SUM:
                   596                               ;
013B FB            597 OVFLOW:   MOV   A,R3          ; GET THE PULSEWIDTH INCREMENT FOR TESTING.
013C F244          598           JB7   CHGSN         ; IF NEGATIVE, THEN PULSEWIDTH SIGN CHANGED.
013E B17F          599           MOV   @R1,#LOWER7   ; IF POSITIVE, THEN PULSEWIDTH SATURATED.
                   600                               ;
0140 237F          601 MAXOUT:   MOV   A,#LOWER7     ; LOAD MAXIMUM PW VALUE (127) INTO ACC.
0142 242B          602           JMP   PWSIGN        ;
                   603                               ;
0144 F1            604 CHGSN:    MOV   A,@R1         ; OBTAIN THE
0145 37            605           CPL   A             ;    NEGATIVE PULSEWIDTH,
0146 17            606           INC   A             ;       2'S COMPLEMENT IT,
0147 A1            607           MOV   @R1,A         ;         AND REPLACE IT.
                   608                               ;
0148 F0            609           MOV   A,@R0         ; OBTAIN PW SIGN,
0149 D380          610 PWDLRV:   XRL   A,#SGNMSK     ;       REVERSE IT,
014B A0            611 PWDLAY:   MOV   @R0,A         ;         AND REPLACE IT.
014C 39            612           OUTL  P1,A          ; PUT PW SIGN BIT (WITH ZERO PW) ON PORT 1.
014D 8AA0          613           ORL   P2,#PWLTEN    ; PUT PULSEWIDTH LATCH ENABLE ADDRESS ON PORT 2.
014F 90            614           MOVX  @R0,A         ; GENERATE LATCH PULSE WITH WRITE TO DUMMY ADDRESS.
                   615                               ;
0150 BB30          616           MOV   R3,#30H       ; ALLOW TIME FOR POWER TRANSISTOR TO CUTOFF
0152 EB52          617 PWULP1:   DJNZ  R3,PWULP1     ;         BEFORE APPLYING REVERSE POWER.
                   618                               ;
0154 2423          619           JMP   PWHYST        ; GO PROCESS THE NEW PULSEWIDTH.
                   620                               ;
                   621 ;.....................................................................
                   622 ;.....................................................................
                   623                               ;
                   624           ; MOTOR STOPPED SUBROUTINE:
                   625                               ;
                   626                               ; PROVIDE:  RB1 SELECTED, CORRECT VALUES
                   627                               ;           FOR PARTICULAR ENTRY POINT.
                   628                               ;
                   629                               ; RETURNS:  STALL FLAG SET IF MOTOR STALLED.
                   630                               ;         IF MOTOR IS RECEIVING LESS THAN 50%
                   631                               ;         PULSEWIDTH BUT IS NOT YET MOVING,
                   632                               ;         PULSEWIDTH WILL BE INCREASED.
                   633                               ;           ENTRY AT 'WRPW0' SETS PULSEWIDTH
                   634                               ;         AND ASSOCIATED LOCATIONS TO ZERO.
                   635                               ;
                   636                               ; CHANGES:  RB1:  R0,R1,R7,ACCUM,PSW,F1,
                   637                               ;           P2,SIGNPW,PLSWTH,STPCNT,WRITE BUFFER
                   638                               ;
                   639                               ; CALLS:   POSITION UPDATE (PSNUPD)
                   640                               ;          PULSEWIDTH UPDATE (PWUPDT)
                   641                               ;            (ENTRY AT 'PWWRIT' OR 'PWDLAY')
                   642                               ;          MOVE YET (MOVYET)
                   643                               ;
```

```
                        644                             ; CALLED BY:  INTERRUPT SERVICE ROUTINE
                        645                             ;                     (FROM 'MOTION', 'NOPWUD',
                        646                             ;                                      AND 'OVRSHT')
                        647                             ;
0156 B928               648 STALL1: MOV   R1,#STPCNT    ; OBTAIN THE NUMBER OF
0158 F1                 649         MOV   A,@R1         ;                      MOTOR STOP INTERRUPTS.
0159 03EF               650         ADD   A,#-11H       ; SEE IF IT EXCEEDS SIXTEEN.
015B F283               651         JB7   NOSTAL        ; IF NOT, MOTOR IS NOT REALLY STALLED.
                        652                             ;
015D B5                 653         CPL   F1            ; IF SO, SET THE STALL FLAG.
015E B807               654         MOV   R0,#7H        ; PUT 'MOTOR STALLED' CODE
0160 B099               655         MOV   @R0,#MTRSTL   ;                  INTO WRITE BUFFER.
                        656                             ;
0162 9A00               657 MTRSTP: ANL   P2,#IDLE      ; CLEAR ENABLE ADDRESS FROM PORT 2.
                        658                             ;
0164 868A               659         JNI   TRNSIT        ; IF INTERRUPT STILL ACTIVE, MOTOR IS MOVING,
0166 11                 660         INC   @R1           ; ELSE, INCREMENT 'MOTOR STOPPED INTERRUPT' COUNT.
                        661                             ;
0167 FF                 662         MOV   A,R7          ; MOTOR IS NOT IN MOTION,
0168 4302               663         ORL   A,#2H         ;    SO SET 'MOTOR STOPPED' FLAG.
016A AF                 664         MOV   R7,A          ;
                        665                             ;
016B B92A               666         MOV   R1,#SPEED     ; WRITE ZERO TO
016D B100               667         MOV   @R1,#0H       ;        SPEED LOCATION.
                        668                             ;
016F 7429               669 PW0:    CALL  PSNUPD        ; CALL 'POSITION UPDATE' SUBROUTINE.
0171 B837               670         MOV   R0,#SIGNPW    ; LOAD ADDR OF PULSEWIDTH SIGN.
                        671                             ;
0173 7677               672         JF1   WRPW0         ; IF MOTOR IS STALLED, WRITE ZERO PULSEWIDTH.
0175 B67E               673         JF0   STLLCK        ; IF 'GO' FLAG SET, GO CHECK FOR STALL.
                        674                             ;
0177 27                 675 WRPW0:  CLR   A             ; WRITE ZERO
0178 B936               676         MOV   R1,#PLSWTH    ;         TO THE
017A A1                 677         MOV   @R1,A         ;                PULSEWIDTH LOCATION.
017B 342C               678         CALL  PWWRIT        ; GO WRITE THE ZERO PW TO THE PW LATCH.
017D 83                 679         RET                 ;
                        680                             ;
017E B936               681 STLLCK: MOV   R1,#PLSWTH    ; INSPECT
0180 F1                 682         MOV   A,@R1         ;          PRESENT PULSEWIDTH.
0181 D256               683         JB6   STALL1        ; IF PW EXCEEDS 63, THERE MAY BE A STALL.
                        684                             ;
0183 14C2               685 NOSTAL: CALL  MOVYET        ; SEE IF A MOVE REMAINS TO EXECUTE.
0185 9677               686         JNZ   WRPW0         ; IF NO MOVE IS INDICATED, GO WRITE ZERO PW.
                        687                             ;
0187 F0                 688         MOV   A,@R0         ; IF THERE IS A MOVE, RECALL PULSEWIDTH SIGN.
0188 344B               689         CALL  PWDLAY        ; GO WRITE PULSEWIDTH WITH DELAY.
018A 83                 690 TRNSIT: RET                 ;
                        691                             ;
                        692 ;.....................................................................
                        693 ;.....................................................................
                        694                             ;
                        695       ; THIS IS THE INTERRUPT SERVICE ROUTINE.  IT WILL BE ENTERED
                        696       ; WHENEVER THERE IS AN ENCODER BIT TRANSITION, OR IF 1/8 SECOND OF
                        697       ; TIME PASSES WITHOUT AN ENCODER BIT TRANSITION.  THE TRANSITION
                        698       ; INTERRUPT REMAINS ACTIVE FOR 700 MICROSECONDS.
                        699       ; THE ONE-SECOND INTERRUPT REMAINS ACTIVE UNTIL ACKNOWLEDGED.
                        700                             ;
                        701                             ; NOTE THAT THE INTERRUPT ACKNOWLEDGE SENT AT THE
                        702                             ; INTERRUPT VECTOR POINT (PC=3) IS ALSO THE
                        703                             ; ENCODER DATA LATCH OUTPUT ENABLE.
                        704                             ;
018B D5                 705 ISR:    SEL   RB1           ; SELECT REGISTER BANK ONE FOR THE ISR.
018C B928               706         MOV   R1,#STPCNT    ; LOAD ADDR OF MOTOR STOP INTERRUPT COUNT.
018E 09                 707         IN    A,P1          ; INPUT THE ENCODER DATA.
018F DF                 708         XRL   A,R7          ; SEE IF PRESENT POSITION AND PREVIOUS POSITION
0190 53C0               709         ANL   A,#0C0H       ; ARE THE SAME (BY EXCLUSIVE OR OF THE TWO BITS).
0192 9697               710         JNZ   MOTION        ; IF NOT ZERO, THE MOTOR IS IN MOTION.
0194 3462               711         CALL  MTRSTP        ; IF SO, THEN CALL THE MOTOR STOPPED SUBROUTINE.
0196 93                 712         RETR                ; RETURN TO MAIN PROGRAM, RESTORING PSW.
                        713                             ;
                        714                             ; IF MOTOR IS MOVING, ISR BRANCHES TO HERE:
                        715                             ;
```

```
0197 B100          716 MOTION: MOV   @R1,#0H       ; CLEAR MOTOR STOPPED INTERRUPT COUNT.
0199 09            717         IN    A,P1         ; INPUT THE ENCODER DATA AGAIN
019A 9A20          718         ANL   P2,#CLCKHI   ; PUT ADDRESS OF HIGH CLOCK BYTE ON PORT 2.
019C 2F            719         XCH   A,R7         ; EXCHANGE FOR PREVIOUS ENCODER BITS.
019D 12A0          720         JB0   DIRDET       ; PRESERVE STATE OF
019F CF            721         DEC   R7           ;    'REVERSAL SINCE LAST UPDATE' FLAG.
                   722                            ;   (NOTE ENCODER DATA BIT 0 IS HARDWARE SET HIGH)
                   723 ;.........................
                   724                            ;
                   725         ; DIRECTION DETERMINATION:  BOTH ABSOLUTE AND RELATIVE-TO-MOVE-
                   726         ; SIGN.  ALSO POS ERROR IS INCREMENTED/DECREMENTED ACCORDINGLY.
                   727                            ;
01A0 F7            728 DIRDET: RLC   A            ; MOVE B1(K-1) TO THE B0 POSITION (BIT 7).
01A1 DF            729         XRL   A,R7         ; XCLSV-OR B1(K-1) WITH B0(K)
                   730                            ;        TO GET 'ABSOLUTE DIRECTION' FLAG.
01A2 53FE          731         ANL   A,#UPPER7    ; CLEAR THE 'RELATIVE DIRECTION' FLAG.
                   732                            ;
01A4 2E            733         XCH   A,R6         ; EXCHANGE FOR PREVIOUS FLAG VALUES OF R6.
01A5 DE            734         XRL   A,R6         ; EXCLUSIVE-OR PREVIOUS 'ABSOLUTE DIRECTION'
                   735                            ;    AND PRESENT 'ABSOLUTE DIRECTION' FLAGS
                   736                            ;    TO SEE IF DIRECTION HAS REVERSED.
01A6 F2AA          737         JB7   DIRRVS       ; IF THEY ARE DIFFERENT, DIRECTION HAS REVERSED.
01A8 24B4          738         JMP   DIRCMP       ; OTHERWISE GO COMPARE ABSOLUTE DIRECTION
                   739                            ;                  TO COMMANDED DIRECTION.
01AA FE            740 DIRRVS: MOV   A,R6         ; SINCE MOTION JUST REVERSED, MOTION IS JUST
01AB 4304          741         ORL   A,#4H        ; STARTING, SO SET, 'NEW MOTION' FLAG.
01AD AE            742         MOV   R6,A         ;
                   743                            ;
01AE 92B4          744         JB4   DIRCMP       ; IF 'PREVIOUS POSITION 0,0' FLAG IS SET,
                   745                            ; THEN REVERSAL DOES NOT AFFECT POSITION UPDATE.
01B0 FF            746         MOV   A,R7         ; OTHERWISE, IT DOES AFFECT IT, SO SET
01B1 D301          747         XRL   A,#1H        ;      THE 'REVERSAL SINCE LAST UPDATE' FLAG.
01B3 AF            748         MOV   R7,A         ;
                   749                            ;
01B4 B835          750 DIRCMP: MOV   R0,#PRMVSN   ; COMPARE THE PRESENT MOVE SIGN
01B6 F0            751         MOV   A,@R0        ;      TO THE 'ABSOLUTE DIRECTION' OF MOTION
01B7 DE            752         XRL   A,R6         ;          BY EXCLUSIVE OR'ING THEM.
01B8 F2BC          753         JB7   DIROPP       ; IF DIRRERENT, GO SET 'RELATIVE DIRECTION' FLAG.
01BA 24BD          754         JMP   VELDET       ; OTHERWISE, LEAVE IT CLEARED (DONE ABOVE).
                   755                            ;
01BC 1E            756 DIROPP: INC   R6           ; SET 'RELATIVE DIRECTION' FLAG.
                   757                            ;
                   758 ;.........................
                   759
                   760                ; VELOCITY DETERMINATION
                   761                            ;
01BD B901          762 VELDET: MOV   R1,#1H       ; LOAD LEAST SHIFT.
01BF 52CD          763         JB2   VELNEW       ; IF 'NEW MOTION' FLAG IS SET, GO SET SPEED
                   764                            ;          TO ITS SMALLEST NON-ZERO VALUE (1).
                   765                            ;
01C1 09            766         IN    A,P1         ; INPUT THE HIGH BYTE OF THE CLOCK.
01C2 8A60          767         ORL   P2,#CLCKLO   ; PUT ADDRESS OF LOW CLOCK BYTE ON PORT 2.
01C4 96D0          768         JNZ   VELSLO       ; IF HIGH BYTE IS NOT ZERO, VEL < 128.
                   769                            ; ELSE, VEL >= 128.
01C6 09            770         IN    A,P1         ; INPUT THE LOW CLOCK BYTE.
                   771                            ;
01C7 F2E3          772         JB7   LOOKUP       ; IF HIGH BIT IS SET, 128 <= VEL < 256
                   773                            ;
01C9 23FE          774         MOV   A,#0FEH      ; VELOCITY IS MAXIMUM.  SET IT TO 254.
01CB 24E4          775         JMP   SHIFT        ;
                   776                            ;
01CD F9            777 VELNEW: MOV   A,R1         ; NEW MOTION, SO SET VEL TO 1 HOLE PER SECOND
01CE 24E4          778         JMP   SHIFT        ;                    (THE MINIMUM).
                   779                            ;
01D0 AB            780 VELSLO: MOV   R3,A         ; STORE HIGH CLOCK BYTE TEMPORARILY.
01D1 09            781         IN    A,P1         ; OBTAIN LOW CLOCK BYTE.
01D2 19            782         INC   R1           ; LOOK-UP TABLE SHIFT COUNT IS AT LEAST 2
                   783                            ;     FOR QUARTER-HOLE VELOCITY LESS THAN 128.
                   784                            ;
01D3 2B            785 VELLP1: XCH   A,R3         ; EXCHANGE LOW CLOCK BYTE, RECOVERING THE HIGH.
01D4 97            786         CLR   C            ; CLEAR THE CARRY BIT.
01D5 67            787         RRC   A            ; ROTATE THE HIGH CLOCK BYTE RIGHT.
```

```
01D6 C6E1          788           JZ    VELSHF       ; IF BITS LEFT ARE ALL ZERO, GO GET TABLE VALUE.
01D8 2B            789           XCH   A,R3         ; EXCHANGE HIGH CLOCK BYTE, RECOVERING THE LOW.
01D9 67            790           RRC   A            ; ROTATE IN THE BIT FROM THE HIGH CLOCK BYTE.
01DA 19            791           INC   R1           ; INCREMENT THE LOOK-UP TABLE SHIFT COUNT.
01DB 24D3          792           JMP   VELLP1       ;
                   793                              ;
01DD 97            794 SHFRT:    CLR   C            ; DIVIDE THE VELOCITY BY TWO AGAIN.
01DE 67            795           RRC   A            ;
01DF 24E4          796           JMP   SHIFT        ;
                   797                              ;
01E1 2B            798 VELSHF:   XCH   A,R3         ; RECOVER THE 7 BITS IN THE LOW CLOCK BYTE.
01E2 67            799           RRC   A            ; ROTATE IN THE EIGHTH BIT FROM CARRY.
01E3 E3            800 LOOKUP:   MOVP3 A,@A         ; OBTAIN LOOK-UP TABLE VELOCITY VALUE.
                   801                              ;
01E4 E9DD          802 SHIFT:    DJNZ  R1,SHFRT     ; KEEP SHIFTING UNTIL COUNTER IS ZERO,
01E6 AA            803           MOV   R2,A         ;  TO OBTAIN THE VELOCITY.
01E7 97            804           CLR   C            ;    SHIFT A FINAL TIME TO FIT INTO SEVEN BITS.
01E8 67            805           RRC   A            ;
                   806                              ;
01E9 1300          807           ADDC  A,#0         ; ROUND UPWARD, USING THE BIT LAST SHIFTED OUT.
                   808                              ;
                   809 ;.....................
                   810
                   811                 ; VELOCITY IS USED TO DETERMINE VELOCITY ERROR.
                   812                 ; PULSEWIDTH IS UPDATED.  POSITION IS UPDATED.
                   813                              ;
01EB 9A00          814 VELCTY:   ANL   P2,#IDLE     ; CLEAR ENABLE ADDRESS FROM PORT 2.
01ED B82A          815           MOV   R0,#SPEED    ; DETERMINE THE VALUE
01EF 20            816           XCH   A,@R0        ;  OF THE CHANGE IN SPEED SINCE LAST
01F0 37            817           CPL   A            ;   INTERRUPT, AND STORE IT IN
01F1 17            818           INC   A            ;    THE LOCATION CALLED 'DELTA V'.
01F2 60            819           ADD   A,@R0        ;
01F3 B93F          820           MOV   R1,#DELTAV   ;
01F5 A1            821           MOV   @R1,A        ;
                   822                              ;
01F6 FA            823           MOV   A,R2         ; ISOLATE THE LOW BIT ONLY OF THE
01F7 5301          824           ANL   A,#1H        ;   FULL VELOCITY MEASURE.
01F9 AA            825           MOV   R2,A         ;
01FA 00            826           NOP                ; ADJUST FOR PAGE BOUNDARY.
01FB 00            827           NOP                ;
                   828                              ;
01FC B927          829           MOV   R1,#MG4      ; DETERMINE THE ADDRESS OF THE
01FE FD            830           MOV   A,R5         ;  MAXIMUM MAGNITUDE FOR
01FF D208          831           JB6   LIMIT        ;   THE RANGE IN WHICH THE
0201 C9            832           DEC   R1           ;    VELOCITY SET POINT
0202 B208          833           JB5   LIMIT        ;     FALLS.
0204 C9            834           DEC   R1           ;
0205 9208          835           JB4   LIMIT        ;
0207 C9            836           DEC   R1           ;
                   837                              ;
0208 FE            838 LIMIT:    MOV   A,R6         ; GET REGISTER 6 FLAGS.
0209 67            839           RRC   A            ; ROTATE 'RELATIVE DIRECTION' FLAG INTO CARRY.
020A F0            840           MOV   A,@R0        ; RECOVER PRESENT VELOCITY.
020B F60F          841           JC    MTNWRG       ; IF CARRY IS SET, MOTION IS IN WRONG DIRECTION.
                   842                              ;
020D 37            843           CPL   A            ; OTHERWISE, PRESENT VELOCITY MUST BE
020E 17            844           INC   A            ;   SUBTRACTED FROM THE VELOCITY SET POINT.
                   845                              ;
020F 6D            846 MTNWRG:   ADD   A,R5         ; ADD VELOCITY SET PNT TO GET VELOCITY ERROR.
0210 AC            847           MOV   R4,A         ; STORE PRESENT VELOCITY ERROR IN REGISTER 4.
                   848                              ;
0211 97            849           CLR   C            ; RESTORE FULL VELOCITY ERROR.
0212 67            850           RRC   A            ;
0213 6A            851           ADD   A,R2         ;
                   852                              ;
0214 2C            853           XCH   A,R4         ; SEE THAT ERROR DID NOT OVERFLOW.
0215 DC            854           XRL   A,R4         ;
0216 F223          855           JB7   MAXERR       ;
                   856                              ;
0218 FC            857           MOV   A,R4         ; OBTAIN VELOCITY ERROR FOR
0219 F21D          858           JB7   NEGERR       ;   COMPARISON TO MAXIMUM ALLOWABLE ERROR.
021B 37            859           CPL   A            ;
```

```
021C 17          860           INC   A
                 861
021D 61          862 NEGERR:   ADD   A,@R1        ; GET DIFFERENCE:  (MAXMAG) - (VEL ERROR).
021E F222        863           JB7   PREMAX       ;   IF NEGATIVE THEN ERROR EXCEEDS MAXIMUM.
0220 442B        864           JMP   CHKDLV
                 865
0222 37          866 PREMAX:   CPL   A            ; DETERMINE SIGN OF MAXIMUM ERROR.
0223 DC          867 MAXERR:   XRL   A,R4         ;   AND PUT SIGN IN CARRY.
0224 F7          868           RLC   A
0225 F1          869           MOV   A,@R1        ; MOVE MAXIMUM MAGNITUDE INTO ACCUMULATOR,
0226 E62A        870           JNC   MAXPOS       ;   AND DETERMINE PROPER SIGN FOR IT.
0228 37          871           CPL   A
0229 17          872           INC   A
022A AC          873 MAXPOS:   MOV   R4,A
                 874
022B FC          875 CHKDLV:   MOV   A,R4         ; IF THE SIGN OF DELTA V IS
022C C635        876           JZ    UPDATE       ;  THE SAME AS THE SIGN OF THE VELOCITY ERROR
022E B83F        877           MOV   R0,#DELTAV   ;   THE THE PRESENT PULSEWIDTH IS ALREADY
0230 D0          878           XRL   A,@R0        ;    BRINGING THE VELOCITY CLOSER TO THE SET
0231 F235        879           JB7   UPDATE       ;      POINT, SO LOAD ZERO AS THE VELOCITY
0233 BC00        880           MOV   R4,#0H       ;        ERROR (NO PULSEWIDTH INCREMENT).
                 881
0235 B63B        882 UPDATE:   JF0   UPDPW        ; IF 'GO' FLAG SET, GO CHECK 'STALL' FLAG.
0237 346F        883 NOPWUD:   CALL  PW0          ; ELSE, SET PULSEWIDTH TO ZERO.
0239 44D3        884           JMP   FNPLP1
                 885
023B 7637        886 UPDPW:    JF1   NOPWUD       ; IF MOTOR IS STALLED, NO PULSEWIDTH UPDATE.
                 887
023D F9          888           MOV   A,R1         ; OBTAIN THE ADDRESS OF THE VELOCITY ERROR
023E 03FC        889           ADD   A,#-4H       ;  COEFFICIENT BY SUBTRACTING FOUR FROM
0240 A8          890           MOV   R0,A         ;   THE ADDRESS OF THE MAXIMUM MAGNITUDE.
                 891
0241 FC          892           MOV   A,R4         ; RECALL THE VELOCITY ERROR,
0242 3412        893           CALL  PWUPDT       ;   AND CALL PULSEWIDTH UPDATE SUBROUTINE.
                 894
0244 7429        895           CALL  PSNUPD       ; CALL THE POSITION UPDATE SUBROUTINE.
                 896
                 897 ;.........................
                 898
                 899                ; PREPARE FOR NEXT INTERRUPT BY ADJUSTING VELOCITY SET POINT
                 900                ;         AND ADJUSTING PRESENT PULSEWIDTH, IF NECESSARY.
                 901
0246 B934        902 PREPAR:   MOV   R1,#PRMVHI   ; CONSIDER HIGH BYTE OF PRESENT MOVE
0248 F1          903           MOV   A,@R1
0249 F2A0        904           JB7   OVRSH1       ; IF IT IS NEGATIVE, OVERSHOOT IS OCCURRING.
024B 96BC        905           JNZ   INFMOV       ; IF IT NOT ZERO, LONG MOVE REMAINS.
                 906
024D C9          907           DEC   R1           ; INSPECT THE LOW BYTE
024E F1          908           MOV   A,@R1        ;          OF REMAINING PRESENT MOVE.
024F F2BC        909           JB7   INFMOV       ; IF IT EXCEEDS 127, THEN MOVE IS VERY LONG.
0251 AA          910           MOV   R2,A
0252 D27B        911           JB6   DCLSTP       ; IF REMAINING MOVE EXCEEDS 32, DONT LOOK
0254 B27B        912           JB5   DCLSTP       ;    FOR NEXT MOVE, JUST GO DECELERATE.
0256 965F        913           JNZ   PREDCL       ; IF IT IS NOT ZERO, GO CHECK THE NEXT MOVE.
                 914
0258 B92B        915           MOV   R1,#QHTAIL   ; IF IT IS ZERO, CHECK THE QUARTER-HOLE
025A F1          916           MOV   A,@R1        ;  POSITION ERROR.
025B F2A5        917           JB7   OVRSH2       ; IF IT IS NEGATIVE, SHORT OVERSHOOT IS OCCURRING.
025D C68C        918           JZ    NOMOVE       ; IF IT IS ZERO, THEN MOVE IS COMPLETE.
                 919
                 920 ;.........................
                 921
                 922                ; SMALL MOVE REMAINING.
                 923
025F B82D        924 PREDCL:   MOV   R0,#RCVDNM   ; INSPECT THE
0261 F0          925           MOV   A,@R0        ;      'NEXT MOVE RECEIVED' FLAG.
0262 C67B        926           JZ    DCLSTP       ; IF IT IS NOT SET, GO DECELERATE TO STOP.
                 927
0264 B931        928           MOV   R1,#NXMVSN   ; IF IT IS SET, THERE IS A NEXT MOVE TO EXECUTE.
0266 F1          929           MOV   A,@R1        ; INSPECT THE SIGN OF THE NEXT MOVE
0267 B935        930           MOV   R1,#PRMVSN   ;     TO SEE IF IT IS IN THE SAME DIRECTION
0269 D1          931           XRL   A,@R1        ;          AS THE PRESENT MOVE.
```

```
026A 967B        932          JNZ   DCLSTP       ; IF NOT, GO DECELERATE TO STOP.
                 933                               ;
026C C8          934          DEC   R0           ; IF NEXT MOVE IS IN SAME DIRECTION, THEN
026D F0          935          MOV   A,@R0        ;    INSPECT THE 'INFINITE MOVE' FLAG.
026E 96BC        936          JNZ   INFMOV       ;       IF SET, THEN NEXT MOVE IS A VIRTUAL
                 937                               ;          'INFINITE EXTENSION' OF PRESENT MOVE.
                 938                               ;
                 939                               ; OTHERWISE, NEXT MOVE IS A
                 940                               ;    SMALL EXTENSION OF THE PRESENT MOVE:
                 941                               ;
0270 B833        942          MOV   R0,#PRMVLO   ; ADD THE SMALL EXTENSION (I.E., SMALL NEXT MOVE)
0272 F0          943          MOV   A,@R0        ;          TO THE PRESENT MOVE,
0273 B82F        944          MOV   R0,#NXMVLO   ;             STORING THE SUM AS THE
0275 60          945          ADD   A,@R0        ;                NEW NEXT MOVE.
0276 B2BC        946          JB5   INFMOV       ;
0278 A0          947          MOV   @R0,A        ;
0279 14F2        948          CALL  NEWMOV       ; CALL 'NEW MOVE' TO LOAD SUM AS PRESENT MOVE.
                 949                               ;
                 950 ;.....................
                 951                               ;
                 952                    ; DECELERATE TO A STOP.
                 953                               ;
027B FD          954 DCLSTP:  MOV   A,R5         ; INSPECT VELOCITY SET POINT.
027C C6BF        955          JZ    UORSHT       ; IF IT IS ZERO, THEN UNDERSHOOT IS OCCURRING.
027E FA          956          MOV   A,R2         ; OTHERWISE,
027F 0310        957          ADD   A,#10H       ;    THE VELOCITY SHOULD NOT EXCEED THE
0281 F2D3        958          JB7   FNPLP1       ;      POSITION ERROR PLUS SIXTEEN.
0283 AA          959          MOV   R2,A         ;
0284 37          960          CPL   A            ;
0285 6D          961          ADD   A,R5         ;
                 962                               ;
0286 F2D3        963          JB7   FNPLP1       ; (VELOCITY SET POINT) < (MOVE + 10) IS OK.
0288 FA          964          MOV   A,R2         ; OTHERWISE, REPLACE THE VELOCITY SET POINT
0289 AD          965          MOV   R5,A         ;                BY THE AMOUNT OF THE MOVE.
028A 44D3        966          JMP   FNPLP1       ;
                 967                               ;
                 968 ;.....................
                 969                               ;
                 970                    ; NO MOVE REMAINS, BUT MOTOR IS IN MOTION.
                 971                               ;
028C B832        972 NOMOVE:  MOV   R0,#PRSVEL   ; CLEAR THE PRESENT VELOCITY LOCATION
028E A0          973          MOV   @R0,A        ;    IN INTERNAL RAM AND ALSO
028F AD          974          MOV   R5,A         ;          CLEAR THE VELOCITY SET POINT.
0290 14F2        975          CALL  NEWMOV       ; SEE IF A NEW MOVE IS WAITING TO EXECUTE.
0292 C646        976          JZ    PREPAR       ; IF THERE IS A NEW MOVE, RESTART PREPARATIONS.
                 977                               ;
0294 B92A        978          MOV   R1,#SPEED    ; OTHERWISE, USE PRESENT MOTOR SPEED
0296 F1          979          MOV   A,@R1        ;    TO DETERMINE THE PROPER
0297 07          980          DEC   A            ;       BRAKING PULSEWIDTH.  NOTE THAT IF
0298 B936        981          MOV   R1,#PLSWTH   ;          SPEED = 1, THEN NO BRAKING PW IS
029A A1          982          MOV   @R1,A        ;            APPLIED (PREVENTS JITTER ON STOP).
                 983                               ;
029B FE          984          MOV   A,R6         ; GET REGISTER 6 FLAGS.
029C 5380        985          ANL   A,#SGNMSK    ; ISOLATE 'ABSOLUTE DIRECTION' FLAG.
029E 44B6        986          JMP   WRPW02       ; (JUMP TO LOCATION IN OVERSHOOT ROUTINE, BELOW)
                 987                               ;
                 988 ;.....................
                 989                               ;
                 990                    ; OVERSHOOT IS OCCURRING.
                 991                               ;
02A0 B83B        992 OVRSH1:  MOV   R0,#LONGOV   ; LOAD THE 'LONG OVERSHOOT'
02A2 F0          993          MOV   A,@R0        ;          PULSEWIDTH INTO ACCUMULATOR.
02A3 44A8        994          JMP   WRPW01       ;
                 995                               ;
02A5 B83C        996 OVRSH2:  MOV   R0,#SHRTOV   ; LOAD THE 'SHORT OVERSHOOT'
02A7 F0          997          MOV   A,@R0        ;          PULSEWIDTH INTO ACCUMULATOR.
                 998                               ;
02A8 B936        999 WRPW01:  MOV   R1,#PLSWTH   ; WRITE THE PULSEWIDTH
02AA A1         1000          MOV   @R1,A        ;    TO ITS INTERNAL RAM LOCATION.
                1001                               ;
02AB FE         1002          MOV   A,R6         ; GET REGISTER 6 FLAGS.
02AC 12B3       1003          JB0   RITDIR       ; MOTION SHOULD BE OPPOSITE COMMANDED.
```

```
02AE B82A        1004           MOV   R0,#SPEED     ;    IF NOT, THEN ADD THE
02B0 F0          1005           MOV   A,@R0         ;       PRESENT SPEED TO THE OVERSHOOT
02B1 61          1006           ADD   A,@R1         ;          PULSEWIDTH FOR ADDITIONAL BRAKING.
02B2 A1          1007           MOV   @R1,A         ;
                 1008                               ;
02B3 B835        1009 RITDIR:   MOV   R0,#PRMVSN    ; LOAD PRESENT MOVE SIGN
02B5 F0          1010           MOV   A,@R0         ;             INTO THE ACCUMULATOR.
                 1011                               ;
02B6 B837        1012 WRPW02:   MOV   R0,#SIGNPW    ; LOAD ADDRS OF PULSEWIDTH SIGN.
02B8 3449        1013           CALL  PWDLRV        ; GO WRITE PULSEWIDTH WITH REVERSAL OF SIGN.
02BA 44D3        1014           JMP   FNPLP1        ; PREPARATIONS ARE FINISHED.
                 1015                               ;
                 1016 ;.............................;
                 1017
                 1018                     ; INFINITE MOVE (I.E., >31) REMAINING.
                 1019                               ;
02BC FD          1020 INFMOV:   MOV   A,R5          ; INSPECT VELOCITY SET POINT,
02BD 96D3        1021           JNZ   FNPLP1        ; IF NOT ZERO, THEN MOVE IS BEING EXECUTED.
                 1022                               ;
                 1023                               ; ELSE, MOTOR IS OUT OF POSITION
                 1024                               ;          IN UNDERSHOOT DIRECTION.
02BF B933        1025 UDRSHT:   MOV   R1,#PRMVLO    ; INSPECT THE LOW BYTE
02C1 F1          1026           MOV   A,@R1         ;   OF THE UNDERSHOOT.
02C2 B93B        1027           MOV   R1,#LONGOV    ;       IF IT IS ZERO, ASSUME
02C4 96C7        1028           JNZ   LNGUND        ;         SHORT AMOUNT OF UNDERSHOOT,
02C6 19          1029           INC   R1            ;           ELSE, ASSUME LONG AMOUNT OF
02C7 F1          1030 LNGUND:   MOV   A,@R1         ;             UNDERSHOOT.   LOAD PULSEWIDTH
02C8 B936        1031           MOV   R1,#PLSWTH    ;               ACCORDINGLY.
02CA A1          1032           MOV   @R1,A         ;
                 1033                               ;
02CB B835        1034           MOV   R0,#PRMVSN    ; LOAD PRESENT MOVE SIGN
02CD F0          1035           MOV   A,@R0         ;    AS THE PROPER SIGN FOR
02CE B837        1036           MOV   R0,#SIGNPW    ;          THE PULSEWIDTH.
02D0 A0          1037           MOV   @R0,A         ;
                 1038                               ;
02D1 344B        1039           CALL  PWDLAY        ; GO WRITE TO PW LATCH, WITH A DELAY.
                 1040                               ;
02D3 86D3        1041 FNPLP1:   JNI   FNPLP1        ; WAIT UNTIL INTERRUPT IS NO LONGER ACTIVE.
02D5 93          1042           RETR                ; RETURN, RESTORING PSW.
                 1043                               ;
                 1044 ;.....................................................................
                 1045 ;.....................................................................
                 1046                               ;
                 1047          ; MULTIPLY SUBROUTINE:
                 1048                               ; PROVIDE:  COEFFICIENT ADDRESS IN R0,
                 1049                               ;           MULTIPLICAND IN ACCUMULATOR.
                 1050                               ;
                 1051                               ; RETURNS:  PRODUCT OF COEFFICIENT AND
                 1052                               ;         MULTIPLICAND, DIVIDED BY SIXTEEN,
                 1053                               ;           IN ACCUMULATOR.
                 1054                               ;
                 1055                               ; CHANGES:  ACCUM,FLAGS,
                 1056                               ;           RB0: R0,R1,R2,R3,R4
                 1057                               ;           RB1: R0,R3
                 1058                               ;
                 1059                               ; CALLS:  NONE
                 1060                               ;
                 1061                               ; CALLED BY:  UPDATE PULSEWIDTH (UPDTPW)
                 1062                               ;
02DF             1063           ORG   2DFH          ; START SUBROUTINES SO PAGE BOUNDARY IS GOOD.
                 1064                               ;
02DF 37          1065 MLTNEG:   CPL   A             ; TWO'S COMPLEMENT
02E0 17          1066           INC   A             ;      THE MULTIPLICAND.
02E1 44E6        1067           JMP   MULT2         ; JUMP BACK INTO MULTIPLY ROUTINE.
                 1068                               ;
02E3 AB          1069 MULT:     MOV   R3,A          ; STORE ORIGINAL MULTIPLICAND.
02E4 F2DF        1070           JB7   MLTNEG        ; IF MULTIPLICAND IS NEGATIVE, SPECIAL CASE.
02E6 C5          1071 MULT2:    SEL   RB0           ; SELECT REGISTER BANK ZERO.
                 1072                               ;
02E7 A9          1073           MOV   R1,A          ; STORE MULTIPLICAND TIMES 1.00 IN R1.
                 1074                               ;
02E8 E7          1075           RL    A             ; MULTIPLY THE MULTIPLICAND BY TWO.
```

```
02E9 A8          1076          MOV   R0,A          ; STORE MULTIPLICAND TIMES 2.00 IN R0.
                 1077                               ;
02EA F9          1078          MOV   A,R1          ; RECOVER MULTIPLICAND.
02EB 17          1079          INC   A             ; INCREMENT IT TO EFFECT ROUNDING UPWARD.
02EC 97          1080          CLR   C             ; CLEAR CARRY FOR NEXT DIVIDE BY TWO.
02ED 67          1081          RRC   A             ; DIVIDE IT BY TWO.
02EE AA          1082          MOV   R2,A          ; STORE MULTIPLICAND TIMES 1/2 IN R2.
                 1083                               ;                 (NOTE ROUNDING UPWARD)
02EF 97          1084          CLR   C             ;
02F0 67          1085          RRC   A             ;
02F1 AB          1086          MOV   R3,A          ; STORE MULTIPLICAND TIMES 1/4 IN R3.
                 1087                               ;
02F2 97          1088          CLR   C             ;
02F3 67          1089          RRC   A             ;
02F4 AC          1090          MOV   R4,A          ; MOVE MULTIPLICAND TIMES 1/8 INTO R4.
                 1091                               ;
02F5 97          1092          CLR   C             ;
02F6 67          1093          RRC   A             ;
02F7 AD          1094          MOV   R5,A          ; MOVE MULTIPLICAND TIMES 1/16 INTO R5.
                 1095                               ;
02F8 D5          1096          SEL   RB1           ;
02F9 F0          1097          MOV   A,@R0         ; OBTAIN COEFFICIENT TO USE AS MULTIPLIER.
02FA C5          1098          SEL   RB0           ;
                 1099                               ;
02FB B2FF        1100          JB5   ADDX2         ; IF BIT 5 IS SET, MLPLCD TIMES 2 IS ADDED IN.
02FD B800        1101          MOV   R0,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1102                               ;
02FF 9203        1103 ADDX2:   JB4   ADDX1         ; IF BIT 4 IS SET, MLPLCD TIMES 1 IS ADDED IN.
0301 B900        1104          MOV   R1,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1105                               ;
0303 7207        1106 ADDX1:   JB3   ADDHLF        ; IF BIT 3 IS SET, MLPLCD TIMES 1/2 IS ADDED IN.
0305 BA00        1107          MOV   R2,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1108                               ;
0307 520B        1109 ADDHLF:  JB2   ADD4TH        ; IF BIT 2 IS SET, MLPLCD TIMES 1/4 IS ADDED IN.
0309 BB00        1110          MOV   R3,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1111                               ;
030B 320F        1112 ADD4TH:  JB1   ADD8TH        ; IF BIT 1 IS SET, MLPLCD TIMES 1/8 IS ADDED IN.
030D BC00        1113          MOV   R4,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1114                               ;
030F 1213        1115 ADD8TH:  JB0   ADDUP         ; IF BIT 0 IS SET, MLPLCD TIMES 1/16 IS ADDED IN.
0311 BD00        1116          MOV   R5,#0H        ; OTHERWISE CLEAR THAT REGISTER.
                 1117                               ;
0313 FD          1118 ADDUP:   MOV   A,R5          ; OBTAIN FIRST PARTIAL SUM (TIMES 1/16).
0314 6C          1119          ADD   A,R4          ; ADD IN NEXT PARTIAL SUM (TIMES 1/8).
0315 6B          1120          ADD   A,R3          ; ADD IN NEXT PARTIAL SUM (TIMES 1/4).
0316 6A          1121          ADD   A,R2          ; ADD IN NEXT PARTIAL SUM (TIMES 1/2).
0317 69          1122          ADD   A,R1          ; ADD IN NEXT PARTIAL SUM (TIMES 1).
0318 68          1123          ADD   A,R0          ; ADD IN NEXT PARTIAL SUM (TIMES 2).
                 1124                               ;
0319 D5          1125          SEL   RB1           ; SELECT REGISTER BANK ONE.
                 1126                               ;
031A F221        1127          JB7   MAXPRD        ; IF BIT 7 IS SET, PRODUCT HAS OVERFLOWED.
031C 2B          1128 MULT3:   XCH   A,R3          ; RECOVER ORIGINAL MULTIPLICAND.
031D F225        1129          JB7   NEGPRD        ; IF IT IS NEGATIVE, GO NEGATE THE PRODUCT.
031F FB          1130          MOV   A,R3          ; OTHERWISE, RECOVER THE POSITIVE PRODUCT.
0320 83          1131          RET                 ;
                 1132                               ;
0321 237F        1133 MAXPRD:  MOV   A,#7FH        ; OBTAIN SIGN OF PRODUCT.
0323 641C        1134          JMP   MULT3         ; IF POSITIVE GO LOAD MAXIMUM POSITIVE VALUE.
                 1135                               ;
0325 FB          1136 NEGPRD:  MOV   A,R3          ; RECOVER PRODUCT.
0326 37          1137          CPL   A             ; TWO'S COMPLEMENT
0327 17          1138          INC   A             ;       THE PRODUCT.
0328 83          1139          RET                 ;
                 1140                               ;
                 1141 ;.....................................................................
                 1142 ;.....................................................................
                 1143                               ;
                 1144          ; UPDATE POSITION INFO SUBROUTINE:
                 1145                               ;
                 1146                               ; PROVIDE:  RB1 SELECTED, PROPER VALUES IN
                 1147                               ;           REGISTERS 6 AND 7.
```

```
                        1148
                        1149                    ;
                        1150                    ; RETURNS:  UPDATED VELOCITY IN EXTERNAL RAM.
                        1151                    ;           IF ENCODER POSITION IS 0,0 THE VALUES
                        1152                    ;         OF REMAINING MOVE, ABSOLUTE POSITIN, AND
                        1153                    ;        'ENCODER POSITION 0,0' FLAG ARE UPDATED.
                        1154                    ;
                        1155                    ; CHANGES:  R0,R1,R3,R7,QHTAIL,LO&HI PRSMOV
                        1156                    ;           EXTERNAL RAM LOCS LO&HI ABS POS.
                        1157                    ;
                        1158                    ; CALLS:  NONE
                        1159                    ;
                        1160                    ; CALLED BY:  INTERRUPT SERVICE ROUTINE
                        1161                    ;
0329 FE                 1161 PSNUPD:  MOV   A,R6        ; OBTAIN REGISTER 6 FLAGS.
032A 5380               1162          ANL   A,#SGNMSK   ; ISOLATE THE 'ABSOLUTE DIRECTION' FLAG.
032C B82A               1163          MOV   R0,#SPEED   ; CONCATENATE THE DIRECTION BIT
032E 40                 1164          ORL   A,@R0       ;    WITH THE SPEED AND WRITE IT
032F 8AC0               1165          ORL   P2,#XRAMEN  ;      TO THE VELOCITY LOCATION IN EXTERNAL RAM.
0331 2631               1166 PSNLP1:  JNT0  PSNLP1      ;
0333 90                 1167          MOVX  @R0,A       ;
                        1168                    ;
0334 F0                 1169          MOV   A,@R0       ; INSPECT THE SPEED.
0335 C67C               1170          JZ    PSNFIN      ; IF SPEED IS ZERO, POSITION HAS NOT CHANGED.
                        1171                    ;
0337 18                 1172          INC   R0          ; INCR TO ADDR OF QUARTER-HOLE TAIL.
0338 FE                 1173          MOV   A,R6        ; INSPECT THE 'RELATIVE DIRECTION' FLAG.
0339 123F               1174          JB0   MTNOP1      ;  IF MOTION IS OPPOSITE COMMANDED,
033B F0                 1175          MOV   A,@R0       ;    THEN QUARTER-HOLE TAIL MUST BY INCREMENTED.
033C 07                 1176          DEC   A           ;         IF MOTION IS IN THE COMMANDED DIRECTION,
033D 07                 1177          DEC   A           ;            THEN QUARTER-HOLE TAIL MUST BE
033E A0                 1178          MOV   @R0,A       ;                 DECREMENTED.
033F 10                 1179 MTNOP1:  INC   @R0         ;
                        1180                    ;
0340 FF                 1181          MOV   A,R7        ; OBTAIN REGISTER 7 FLAGS AND ENCODER BIT DATA.
0341 F27C               1182          JB7   PSNFIN      ; IF BIT 7 SET, ENCODER NOT ON 0,0.
0343 D27C               1183          JB6   PSNFIN      ; IF BIT 6 SET, ENCODER NOT ON 0,0.
0345 B000               1184          MOV   @R0,#0      ; OTHERWISE, POSITION IS 0,0 SO CLEAR Q-H TAIL.
                        1185                    ;   AND UPDATE PRESENT MOVE AND ABSOLUTE POSITION:
                        1186                    ;
0347 B833               1187          MOV   R0,#PRMVLO  ; SET ADDRESS POINTERS TO
0349 B934               1188          MOV   R1,#PRMVHI  ;  LOW AND HIGH BYTES OF MOVE COMMAND.
                        1189                    ;    (SAME ADDRESSES FOR ABSOLUTE POSITION BYTES.)
                        1190                    ;
034B 4308               1191          ORL   A,#8H       ; SET THE 'ENCODER POSITION 0,0' FLAG.
034D AF                 1192          MOV   R7,A        ;
                        1193                    ;
034E 125E               1194          JB0   PSNSAM      ; IF 'REVERSAL SINCE LAST UPDATE' FLAG IS SET,
                        1195                    ;    POSITION IS THE SAME AS IT WAS AT LAST UPDATE.
                        1196                    ;
0350 FE                 1197          MOV   A,R6        ; OBTAIN REGISTER 6 FLAGS.
0351 1261               1198          JB0   MTNOP2      ; IF 'RELATIVE DIRECTION' FLAG IS SET, MOTION
                        1199                    ;   IS OPPOSITE COMMANDED, SO MOVE IS INCREMENTED.
                        1200                    ;
0353 F0                 1201          MOV   A,@R0       ; OTHERWISE, MOTION IS IN COMMANDED DIRECTION,
0354 07                 1202          DEC   A           ;    SO MOVE IS DECREMENTED.
0355 A0                 1203          MOV   @R0,A       ;
0356 17                 1204          INC   A           ; (NOTE THAT HIGH BYTE MUST BE DECREMENTED
0357 9666               1205          JNZ   PSNSET      ;    ONLY IF LOW BYTE WAS EQUAL TO ZERO)
0359 F1                 1206          MOV   A,@R1       ;
035A 07                 1207          DEC   A           ;
035B A1                 1208          MOV   @R1,A       ;
035C 6466               1209          JMP   PSNSET      ;
                        1210                    ;
                        1211                    ;
035E CF                 1212 PSNSAM:  DEC   R7          ; PC BRANCES HERE WHEN 'REVERSAL SINCE LAST
035F 647C               1213          JMP   PSNFIN      ;    UPDATE' FLAG IS SET.   CLEAR THAT FLAG.
                        1214                    ;       (POSITION IS THE SAME AS AT LAST UPDATE.)
                        1215                    ;
0361 10                 1216 MTNOP2:  INC   @R0         ; SINCE MOTION IN OPPOSITE COMMANDED,
0362 F0                 1217          MOV   A,@R0       ;    MOVE MUST BE INCREMENTED.
0363 9666               1218          JNZ   PSNSET      ; (NOTE HIGH BYTE IS ONLY INCREMENTED IN
0365 11                 1219          INC   @R1         ;   CASE LOW BYTE BECOMES ZERO ON INCREMENT.)
```

```
                        1220
                        1221 PSNSET: MOV    A,R6          ; OBTAIN REGISTER 6 FLAGS.
                        1222
0367 F273               1223         JB7    MTNOP3        ; IF 'ABSOLUTE DIRECTION' FLAG IS SET, THEN ABSOLUTE
0369 80                 1224         MOVX   A,@R0         ;  POSITION IS DECREMENTED (BELOW) SINCE MOVE IS
036A 17                 1225         INC    A             ;   COUNTERCLOCKWISE.  OTHERWISE, IF 'ABSOLUTE POS-
036B 90                 1226         MOVX   @R0,A         ;    ITION' FLAG IS CLEAR, THEN ABSOLUTE POSITION
036C 967C               1227         JNZ    PSNFIN        ;      MUST BE INCREMENTED, SINCE MOVE IS CLOCKWISE.
036E 81                 1228         MOVX   A,@R1         ;
036F 17                 1229         INC    A             ; (NOTE HIGH BYTE IS ONLY INCREMENTED
0370 91                 1230         MOVX   @R1,A         ;       IF LOW BYTE INCREMENTS TO ZERO.)
0371 647C               1231         JMP    PSNFIN        ;
                        1232                              ;
0373 80                 1233 MTNOP3: MOVX   A,@R0         ; MOVE IS COUNTER-CLOCKWISE,
0374 07                 1234         DEC    A             ; SO DECREMENT ABSOLUTE POSITION.
0375 90                 1235         MOVX   @R0,A         ;
0376 17                 1236         INC    A             ; (NOTE HIGH BYTE IS ONLY DECREMENTED IF
0377 967C               1237         JNZ    PSNFIN        ;      LOW BYTE WAS PREVIOUSLY ZERO.)
0379 81                 1238         MOVX   A,@R1         ;
037A 07                 1239         DEC    A             ;
037B 91                 1240         MOVX   @R1,A         ;
                        1241                              ;
037C 9A00               1242 PSNFIN: ANL    P2,#IDLE      ; CLEAR ENABLE ADDRESS FROM PORT 2.
037E 83                 1243         RET                  ;
                        1244                              ;
                        1245 ;.....................................................................
                        1246 ;.....................................................................
                        1247                              ;
                        1248                              ;
0380                    1249         ORG    380H          ; START VELOCITY LOOKUP TABLE HERE
                        1250                              ;       (SECOND HALF OF PAGE THREE).
                        1251                                                                  ;
                        1252                                                                  ;
0380 FE                 1253         DB       0FEH, 0FEH, 0FCH, 0FAH, 0F8H, 0F6H, 0F5H, 0F3H
0381 FE
0382 FC
0383 FA
0384 F8
0385 F6
0386 F5
0387 F3
                        1254                                                                  ;
0388 F1                 1255         DB       0F1H, 0EFH, 0EDH, 0ECH, 0EAH, 0E8H, 0E7H, 0E5H
0389 EF
038A ED
038B EC
038C EA
038D E8
038E E7
038F E5
                        1256                                                                  ;
0390 E4                 1257         DB       0E4H, 0E2H, 0E0H, 0DFH, 0DDH, 0DCH, 0DAH, 0D9H
0391 E2
0392 E0
0393 DF
0394 DD
0395 DC
0396 DA
0397 D9
                        1258                                                                  ;
0398 D8                 1259         DB       0D8H, 0D6H, 0D5H, 0D3H, 0D2H, 0D1H, 0CFH, 0CEH
0399 D6
039A D5
039B D3
039C D2
039D D1
039E CF
039F CE
                        1260                                                                  ;
03A0 CD                 1261         DB       0CDH, 0CCH, 0CAH, 0C9H, 0C8H, 0C7H, 0C5H, 0C4H
03A1 CC
03A2 CA
```

```
03A3 C9
03A4 C8
03A5 C7
03A6 C5
03A7 C4
                    1262
03A8 C3             1263          DB      0C3H, 0C2H, 0C1H, 0C0H, 0BFH, 0BDH, 0BCH, 0B8H
03A9 C2
03AA C1
03AB C0
03AC BF
03AD BD
03AE BC
03AF BB
                    1264
03B0 BA             1265          DB      0BAH, 0B9H, 0B8H, 0B7H, 0B6H, 0B5H, 0B4H, 0B3H
03B1 B9
03B2 B8
03B3 B7
03B4 B6
03B5 B5
03B6 B4
03B7 B3
                    1266
03B8 B2             1267          DB      0B2H, 0B1H, 0B0H, 0AFH, 0AEH, 0ADH, 0ACH, 0ACH
03B9 B1
03BA B0
03BB AF
03BC AE
03BD AD
03BE AC
03BF AC
                    1268
03C0 AB             1269          DB      0ABH, 0AAH, 0A9H, 0A8H, 0A7H, 0A6H, 0A5H, 0A5H
03C1 AA
03C2 A9
03C3 A8
03C4 A7
03C5 A6
03C6 A5
03C7 A5
                    1270
03C8 A4             1271          DB      0A4H, 0A3H, 0A2H, 0A1H, 0A1H, 0A0H,  9FH,  9EH
03C9 A3
03CA A2
03CB A1
03CC A1
03CD A0
03CE 9F
03CF 9E
                    1272
03D0 9E             1273          DB       9EH,  9DH,  9CH,  9BH,  9BH,  9AH,  99H,  98H
03D1 9D
03D2 9C
03D3 9B
03D4 9B
03D5 9A
03D6 99
03D7 98
                    1274
03D8 98             1275          DB       98H,  97H,  96H,  96H,  95H,  94H,  94H,  93H
03D9 97
03DA 96
03DB 96
03DC 95
03DD 94
03DE 94
03DF 93
                    1276
03E0 92             1277          DB       92H,  92H,  91H,  90H,  90H,  8FH,  8EH,  8EH
03E1 92
03E2 91
```

```
03E3 90
03E4 90
03E5 8F
03E6 8E
03E7 8E
                        1278                                                    ;
03E8 8D                 1279            DB      8DH,  8DH,  8CH,  8BH,  8BH,  8AH,  8AH,  89H
03E9 8D
03EA 8C
03EB 8B
03EC 8B
03ED 8A
03EE 8A
03EF 89
                        1280                                                    ;
03F0 89                 1281            DB      89H,  88H,  87H,  87H,  86H,  86H,  85H,  85H
03F1 88
03F2 87
03F3 87
03F4 86
03F5 86
03F6 85
03F7 85
                        1282                                                    ;
03F8 84                 1283            DB      84H,  84H,  83H,  83H,  82H,  82H,  81H,  81H
03F9 84
03FA 83
03FB 83
03FC 82
03FD 82
03FE 81
03FF 81
                        1284                                                    ;
                        1285                                                    ;
                        1286            END                                     ;

USER SYMBOLS
ADD4TH 030B    ADD8TH 030F    ADDGO  00DC    ADDHLF 0307    ADDUP  0313    ADDX1  0303    ADDX2  02FF    ALL1S  00FF
BEGIN  0000    BEGINO 0008    BEGLP1 0010    BEGLP2 0015    BITDAT 00E0    CF1    0020    CF2    0021    CF3    0022
CF4    0023    CHGSN  0144    CHKDLV 022B    CKQHTL 00E5    CLCKHI 0020    CLCKLO 0060    CLRLP1 0080    CLRMVS 007B
COEFS  0052    COELP1 005A    DCLSTP 027B    DELTAV 003F    DETSGN 00CC    DIRCMP 01B4    DIRDET 01A0    DIROPP 01BC
DIRRVS 01AA    FNPLP1 02D3    GETMOV 00F8    GO     00AD    GRIPPR 0038    HALT   0089    HDWRST 00AA    HYSTPW 0039
IDLE   0000    INFMOV 02BC    INTLP1 0099    INTRAM 0097    ISR    018B    JMPBK3 011D    LIMIT  0208    LNGUND 02C7
LONGOV 003B    LOOKUP 01E3    LOWER7 007F    MAIN48 0024    MASK32 00E0    MAXERR 0223    MAXOUT 0140    MAXPOS 022A
MAXPRD 0321    MG1    0024    MG2    0025    MG3    0026    MG4    0027    MLBXEN 0080    MLTNEG 02DF    MOTION 0197
MOVE   00B3    MOVLP1 00B9    MOVLP2 00BB    MOVYET 00C2    MTNOP1 033F    MTNOP2 0361    MTNOP3 0373    MTNWRG 020F
MTRSTL 0099    MTRSTP 0162    MULT   02E3    MULT2  02E6    MULT3  031C    NEGERR 021D    NEGPRD 0325    NEWFIN 0110
NEWLP1 0102    NEWMOV 00F2    NOMOVE 028C    NOPOLL 0075    NOPWUD 0237    NOSTAL 0183    NXMVHI 0030    NXMVLO 002F
NXMVNF 002C    NXMVSN 0031    NXTVEL 002E    OVFLOW 013B    OVR    0088    OVRSH1 02A0    OVRSH2 02A5    PEEK   00A2
PLSWTH 0036    POKE   00A9    PREDCL 025F    PREMAX 0222    PREPAR 0246    PRMVHI 0034    PRMVLO 0033    PRMVSN 0035
PRSVEL 0032    PSNFIN 037C    PSNLP1 0331    PSNSAM 035E    PSNSET 0366    PSNUPD 0329    PWO    016F    PWDLAY 014B
PWDLRV 0149    PWHYST 0123    PWLTEN 00A0    PWRVRS 0135    PWSIGN 012B    PWULP1 0152    PWUPDT 0112    PWWRIT 012C
QHTAIL 002B    RAMADR 0029    RAMDAT 0028    RAMPK  00CC    RCVDNM 002D    RDMBX  002A    RDYMSG 010C    RDYNXT 00BB
RESEND 0078    RESET  008C    RESLP1 0090    RITDIR 02B3    RSNDRQ 0067    SGNMSK 0080    SHFRT  01DD    SHIFT  01E4
SHRTOV 003C    SIGNPW 0037    SPEED  002A    STALL1 0156    STLLCK 017E    STPCNT 0028    STRTPW 003A    TRNSIT 018A
UDRSHT 02BF    UPDATE 0235    UPDPW  023B    UPPER7 00FE    VELCTY 01EB    VELDET 01BD    VELLP1 01D3    VELNEW 01CD
VELSHF 01E1    VELSLO 01D0    WRILP1 006C    WRITE  0069    WRPW0  0177    WRPW01 02A8    WRPW02 02B6    XRAMEN 00C0

ASSEMBLY COMPLETE,   NO ERRORS
1


ISIS-II ASSEMBLER SYMBOL CROSS REFERENCE, V2.1                    PAGE   1

ADD4TH 1109   1112#
ADD8TH 1112   1115#
ADDGO   465    468#
ADDHLF 1106   1109#
ADDUP  1115   1118#
ADDX1  1103   1106#
```

ADDX2 1100 1103#
ALL1S 176# 586
BEGIN 188#
BEGIN0 188 211#
BEGLP1 217# 217
BEGLP2 221# 223
BITDAT 42# 194 211
CF1 49# 292
CF2 51#
CF3 52#
CF4 53#
CHGSN 598 604#
CHKDLV 864 875#
CKQHTL 453 476#
CLCKHI 35# 718
CLCKLO 36# 767
CLRLP1 346# 348
CLRMVS 270 342#
COEFS 290# 290
COELP1 296# 299 302
DCLSTP 911 912 926 932 954#
DELTAV 100# 820 877
DETSGN 449 456# 478 483
DIRCMP 738 744 750#
DIRDET 720 728#
DIROPP 753 756#
DIRRVS 737 740#
FNPLP1 884 958 963 966 1014 1021 1041# 1041
GETMOV 512 515#
GO 277 405#
GRIPPR 92# 300
HALT 275 360#
HDWRST 8# 225
HYSTPW 94# 576
IDLE 33# 251 322 585 657 814 1242
INFMOV 905 909 936 946 1020#
INTLP1 381# 381
INTRAM 279 380#
ISR 197 705#
JMPBK3 567# 593
LIMIT 831 833 835 838#
LNGUND 1028 1030# 1027
LONGOV 97# 992
LOOKUP 772 800#
LOWER7 178# 599 601
MAIN48 243# 252 256 304 354 362 376 399 405 409 424
MASK32 175#
MAXERR 855 867#
MAXOUT 578 601#
MAXPOS 870 873#
MAXPRD 1127 1133#
MG1 55#
MG2 56#
MG3 57#
MG4 58# 829
MLBXEN 38# 229 254 323
MLTNEG 1065# 1070
MOTION 710 716#
MOVE 273 413#
MOVLP1 417# 417
MOVLP2 419# 422
MOVYET 447# 685
MTNOP1 1174 1179#
MTNOP2 1198 1216#
MTNOP3 1223 1233#
MTNWRG 841 846#
MTRSTL 7# 655
MTRSTP 657# 711
MULT 559 1069#
MULT2 1067 1071#
MULT3 1128# 1134
NEGERR 858 862#

NEGPRD 1129 1136#
NEWFIN 513 532#
NEWLP1 520# 524
NEWMOV 480 508# 948 975
NOMOVE 918 972#
NOPOLL 328# 336
NOPWUD 883# 886
NOSTAL 651 685#
NXMVHI 79#
NXMVLO 78# 944
NXMVNF 74# 344 414
NXMVSN 80# 928
NXTVEL 77#
OVFLOW 572 597#
OVR 6# 310
OVRSH1 904 992#
OVRSH2 917 996#
PEEK 390#
PLSWTH 89# 470 481 676 681 981 999 1031
POKE 387 397#
PREDCL 913 924#
PREMAX 863 866#
PREPAR 902# 976
PRMVHI 85# 447 902 1188
PRMVLO 83# 369 942 1025 1187
PRMVSN 87# 457 564 750 930 1009 1034
PRSVEL 82# 519 972
PSNFIN 1170 1182 1183 1213 1227 1231 1237 1242#
PSNLP1 1166# 1166
PSNSAM 1194 1212#
PSNSET 1205 1209 1218 1221#
PSNUPD 669 895 1161#
PWO 669# 883
PWDLAY 611# 689 1039
PWDLRV 610# 1013
PWHYST 574# 619
PWLTEN 39# 583 613
PWRVRS 566 589#
PWSIGN 575 580# 602
PWULP1 617# 617
PWUPDT 559# 893
PWWRIT 234 582# 678
QHTAIL 69# 476 915
RAMADR 64# 383
RAMDAT 63#
RAMPK 10# 394
RCVDNM 75# 510 924
RDMBX 251# 329
RDYMSG 352 529#
RDYNXT 9# 530
RESEND 265 335#
RESET 281 368#
RESLP1 371# 371
RITDIR 1003 1009#
RSNDRQ 261 310#
SGNMSK 174# 456 610 985 1162
SHFRT 794# 802
SHIFT 775 778 796 802#
SHRTOV 98# 996
SIGNPW 90# 459 562 670 1012 1036
SPEED 66# 666 815 978 1004 1163
STALL1 648# 683
STLLCK 673 681#
STPCNT 60# 462 526 648 706
STRTPW 95# 468
TRNSIT 659 690#
UDRSHT 955 1025#
UPDATE 876 879 882#
UPDPW 882 886#
UPPER7 177# 731
VELCTY 814#
VELDET 754 762#

```
VELLP1   785#    792
VELNEW   763     777#
VELSHF   788     798#
VELSLO   768     780#
WRILP1   320#    324
WRITE    249     316#    395
WRPW0    672     675#    686
WRPW01   994     999#
WRPW02   986    1012#
XRAMEN    41#    216     289     368     380     413     1165

CROSS REFERENCE COMPLETE
```

APPENDIX Y. LISTINGS OF THE INPUT/OUTPUT PROCEDURES ON VAX

The VAX program calls these external procedures to communicate in 8-bit format with the 8751 serial port. They are external to the main program because they are written in FORTRAN, while the main program is in PASCAL.

At this writing, the main VAX input/output system call was SYS$QIO. These programs call that routine, and make use of some other system calls and constants. It seems at this point that there is not much reason to explain the functioning of these routines because they will all be defunct within two months of this writing, when a new operating system is to be installed on VAX. However, the routines are included here for the sake of completeness.

```
0001
0002    c        This subroutine derives from the READ_TWO subroutine
0003    c         written by Dr. Roger Ehrich of the Virginia Tech Computer
0004    c         Science Department.
0005
0006    c
0007    c       RHINORD (device,keybuff,rhinobuff,term,flag1,chan1)
0008    c
0009    c         RHINORD  attempts to read a character from two terminals
0010    c                  simultaneously and returns the character that is read first.
0011    c         device    character*(*), the name of the rhino communication line.
0012    c         keybuff   byte, code from the keyboard is returned in this buffer.
0013    c         rhinobuff byte, code from the rhino is returned in this buffer.
0014    c         term      byte, identifies the source of the character:
0015    c                   0 means the login terminal and 1 means the rhino.
0016    c         flag1     the event flag used for communication with the rhino
0017    c         chan1     the channel used for communication with the rhino
0018    c
0019    c       The logical device can be given in upper or lower case, with or without
0020    c       trailing blanks
0021    c
0022            subroutine rhinord (device,keybuff,rhinobuff,term,flag1,chan1)
0023
0024            CHARACTER*(*) device
0025            CHARACTER*63 updevice
0026            BYTE keybuff,rhinobuff,term,call_flag /0/
0027            INTEGER*4 chan0,chan1,terminators(2) /0,0/
0028            INTEGER*4 flag0,flag1,mask,op0,op1,state
0029            EXTERNAL io$_readvblk,io$m_noecho,io$m_nofiltr
0030            EXTERNAL io$_ttyreadall
0031
0032    c      The first time this subroutine is called from the rhinocom
0033    c       program, this part of it gets event flags and channels
0034    c       assigned for the rhino line and the login terminal.  After
0035    c       the first call, the call_flag is never zero again, so this
0036    c       part of the subroutine is skipped over.
0037
0038            if (call_flag.eq.0) then
0039              j=index (device//' ',' ')-1
0040              call str$upcase (updevice,device)
0041              call sys$assign ('TT',chan0,,)
0042              call sys$assign (updevice(1:j),chan1,,)
0043
0044              n=62
0045              do while (lib$reserve_ef(n).ne.1)
0046                n=n-1
0047              end do
0048              flag0=n
0049              do while (lib$reserve_ef(n).ne.1)
0050                n=n-1
0051              end do
```

```
0052              flag1=n
0053
0054              mask=2**(flag0-32)+2**(flag1-32)
0055
0056              op0=ior (%loc (io$_readvblk),%loc (io$m_nofiltr))
0057
0058              op1=ior (%loc (io$_ttyreadall),%loc (io$m_noecho))
0059
0060              call_flag=1
0061            end if
0062
0063      c     Makes two simultaneous read requests
0064      c                 using the system i/o routine:
0065
0066            call sys$qio (%val (flag0),%val (chan0),%val (op0),,,,keybuff,%val(1)
0067            1,,terminators,,)
0068            call sys$qio (%val (flag1),%val (chan1),%val (op1),,,,rhinobuff,%val(1)
0069            1,,terminators,,)
0070
0071
0072      c     Wait for the logical or of the event flags to become set,
0073      c      indicating that one of the read requests has been filled:
0074
0075      10     call sys$wflor (%val (flag0),%val (mask))
0076
0077
0078      c     Get the system event flags:
0079
0080            call sys$readef (%val (flag0),state)
0081
0082
0083      c     If the flag assigned to the login terminal is set,
0084      c      then clear the 'term' flag and cancel the read request
0085      c      of the rhino channel:
0086
0087            if (iand (state,2**(flag0-32)).ne.0) then
0088              term=0
0089              call sys$cancel (%val (chan1))
0090              go to 20
0091            end if
0092
0093
0094      c     If the flag assigned to the rhino is set, then set
0095      c      the 'term' flag and cancel the read request
0096      c      of the login terminal channel:
0097
0098            if (iand (state,2**(flag1-32)).ne.0) then
0099              term=1
0100              call sys$cancel (%val (chan0))
0101              go to 20
0102            end if
```

```
0103
0104     c      If neither event flag is set, go wait for the
0105     c       logical-or of them to be high:
0106
0107              go to 10
0108
0109     20       return
0110
0111              end
```

```
PROGRAM SECTIONS
    Name                                    Bytes   Attributes
  0 $CODE                                     418   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                      5   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                    300   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG
    Total Space Allocated                     723
ENTRY POINTS
    Address  Type  Name
  0-00000000       RHINORD
VARIABLES
    Address  Type  Name              Address  Type  Name              Address  Type  Name
Name
  2-00000047  L*1  CALL_FLAG       2-00000048  I*4  CHAN0          AP-00000018@ I*4  CHAN1
DEVICE
  2-0000004C  I*4  FLAG0          AP-00000014@ I*4  FLAG1           2-00000060  I*4  J
KEYBUFF
  2-00000050  I*4  MASK            2-00000064  I*4  N               2-00000054  I*4  OPO
 AP-0000000C@ L*1  RHINOBUFF       2-0000005C  I*4  STATE          AP-00000010@ L*1  TERM
UPDEVICE
ARRAYS
    Address  Type  Name                Bytes  Dimensions
  2-00000000  I*4  TERMINATORS             8  (2)
LABELS
    Address   Label       Address   Label
  0-0000012E  10        0-000001A1  20
FUNCTIONS AND SUBROUTINES REFERENCED
  Type  Name                                Type  Name                                Type  Name
        IO$M_NOECHO                               IO$M_NOFILTR                              IO$_READVBLK
        IO$_TTYREADALL                      I*4   LIB$INDEX                           I*4   LIB$RESERVE_EF
        STR$UPCASE                                SYS$ASSIGN                                SYS$CANCEL
        SYS$QIO                                   SYS$READEF                                SYS$WFLOR
COMMAND QUALIFIERS
  FORTRAN /LIS RHINORD
  /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
  /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
```

```
  /F77  /NOG_FLOATING  /I4  /OPTIMIZE  /WARNINGS  /NOD_LINES  /NOCROSS_REFERENCE  /NOMACHINE_CODE
COMPILATION STATISTICS
  Run Time:            1.70 seconds
  Elapsed Time:        2.92 seconds
  Page Faults:         385
  Dynamic Memory:      122 pages
```

```
0001    c     This subroutine derives from the TT_WRITE subroutine, written by
0002    c      Dr. Roger Ehrich of the Virginia Tech Computer Science Department.
0003
0004
0005    c       RHINOWR (efn,channel,buffer)
0006    c
0007    c         RHINOWR   writes one byte to the specified channel and returns
0008    c                   when the information transfer is complete.
0009    c                   RHINOWR is designed to write 8-bit bytes to any
0010    c                   allocated terminal without physical I/O privileges.
0011    c         efn       integer*4, the event flag to be used
0012    c         channel   integer*4, the channel assigned to the communication
0013    c         buffer    byte, the data to be sent
0014    c
0015    c       All 256 8 bit bytes can be written
0016    c
0017
0018            subroutine rhinowr (efn,channel,buffer)
0019
0020            BYTE buffer
0021            INTEGER*4 efn,channel,status,msgvec(3)
0022            INTEGER*4 sys$qio,sys$qiow,operation
0023            DATA msgvec /'F0001'x,0,0/
0024            EXTERNAL io$_writevblk,ss$_normal,io$m_noformat
0025
0026
0027    c     Define the particular output operation to be performed
0028    c      by the vax i/o system call:
0029
0030             operation=ior (%loc (io$_writevblk),%loc (io$m_noformat))
0031
0032
0033    c     Make the system output call with a wait for execution
0034    c      and with a status code returned indicating whether
0035    c      execution was normal:
0036
0037             status=sys$qiow (%val (efn),%val (channel),%val (operation),
0038             1  ,,,buffer,%val (1),,,,)
0039
0040
0041    c     If the status code indicates abnormal execution,
0042    c      stop program execution and print a message:
0043
0044             if (status.ne.%loc (ss$_normal)) then
0045               msgvec(2)=status
0046               call sys$putmsg (msgvec,,'SS')
0047               stop 'STOP from RHINOWR'
0048             end if
0049
0050             return
0051             end
```

```
PROGRAM SECTIONS
    Name                                    Bytes   Attributes
  0 $CODE                                     94    PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                    20    PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                   108    PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG
    Total Space Allocated                    222
ENTRY POINTS
    Address  Type  Name
  0-00000000       RHINOWR
VARIABLES
    Address  Type  Name              Address  Type  Name              Address  Type  Name
Name
 AP-0000000C@ L*1  BUFFER          AP-00000008@ I*4  CHANNEL         AP-00000004@ I*4  EFN
OPERATION
  2-0000000C  I*4  STATUS           2-00000010  I*4  SYS$QIO
ARRAYS
    Address  Type  Name                 Bytes  Dimensions
  2-00000000  I*4  MSGVEC                 12  (3)
FUNCTIONS AND SUBROUTINES REFERENCED
  Type  Name              Type  Name              Type  Name              Type  Name
        IO$M_NOFORMAT           IO$_WRITEVBLK           SS$_NORMAL              SYS$PUTMSG
COMMAND QUALIFIERS
  FORTRAN /LIS RHINOWR
  /CHECK=(NOBOUNDS,OVERFLOW,NOUNDERFLOW)
  /DEBUG=(NOSYMBOLS,TRACEBACK)
  /STANDARD=(NOSYNTAX,NOSOURCE_FORM)
  /SHOW=(NOPREPROCESSOR,NOINCLUDE,MAP)
  /F77  /NOG_FLOATING  /I4  /OPTIMIZE  /WARNINGS  /NOD_LINES  /NOCROSS_REFERENCE  /NOMACHINE_CODE
COMPILATION STATISTICS
  Run Time:           0.71 seconds
  Elapsed Time:       1.71 seconds
  Page Faults:        354
  Dynamic Memory:     117 pages
```

BIBLIOGRAPHY

Deshpande, Pradeep B., and Ash, Raymond H. Elements of Computer Process Control, Research Triangle Park, North Carolina, 1981: Instrument Society of America.

Digital Equipment Corporation. VAX-11 FORTRAN Language Reference Manual, Order number AA-D034C-TE, Maynard, Massachusetts, 1982: Digital Equipment Corporation.

Digital Equipment Corporation. VAX-11 PASCAL Language Reference Manual, Order number AA-H484C-TE, Maynard, Massachusetts, 1982: Digital Equipment Corporation.

Digital Equipment Corporation. VAX-11 Run-Time Library Reference Manual, Order number AA-D036C-TE, Maynard, Massachusetts, 1982: Digital Equipment Corporation.

Digital Equipment Corporation. VAX-11 Run-Time Library User's Guide, Order number AA-1824A-TE, Maynard, Massachusetts, 1982: Digital Equipment Corporation.

Digital Equipment Corporation. VAX/VMS I/O User's Guide (Volume 1), Order number AA-M540B-TE, Maynard, Massachusetts, 1983: Digital Equipment Corporation.

Digital Equipment Corporation. VAX/VMS I/O User's Guide (Volume 2), Order number AA-M541B-TE, Maynard, Massachusetts, 1983: Digital Equipment Corporation.

Digital Equipment Corporation. VAX/VMS System Services Reference Manual, Order number AA-D018C-TE, Maynard, Massachusetts, 1982: Digital Equipment Corporation.

Engineering Staff of Texas Instruments, Incorporated. The Optoelectronics Data Book for Design Engineers, First Edition, Dallas, Texas, undated: Texas Instruments, Incorporated.

Engineering Staff of Texas Instruments, Incorporated. The Power Semiconductor Data Book for Design Engineers, First Edition, Dallas, Texas, undated: Texas Instruments, Incorporated.

Engineering Staff of Texas Instruments, Incorporated. The TTL Data Book for Design Engineers, Second Edition, Dallas, Texas, 1981: Texas Instruments, Incorporated.

Kuo, Benjamin. Digital Control Systems, New York, 1980: Holt, Rinehart, and Winston.

Intel Corporation. Intel Component Data Catalog, Santa Clara, California, 1979: Intel Corporation.

Intel Corporation. ISIS-II CREDIT CRT-Based Text Editor User's Guide, Manual Order Number 9800902 Rev. B, Santa Clara, CFlifornia, 1980: Intel Corporation.

Intel Corporation. ISIS-II User's Guide, Manual Order Number 9800306 Rev. F, Santa Clara, California, 1980: Intel Corporation.

Intel Corporation. IUP-200/201 Universal Programmer Users Guide, Manual Order Number 162613-001, Santa Clara, California, 1982: Intel Corporation.

Intel Corporation. MCS-48 and UPI-41 Assembly Language Manual , Manual Order Number 9800255D, Santa Clara, California, 1978: Intel Corporation.

Intel Corporation. MCS-48 Family of Single Chip Microcomputers User's Manual, Manual Order Number 9800270D, Santa Clara, California, 1978: Intel Corporation.

Intel Corporation. MCS-51 Macro Assembler User's Guide , Manual Order Number 9800937-02, Santa Clara, California, 1981: Intel Corporation.

Intel Corporation. MCS-51 Family of Single Chip Microcomputers User's Manual, Part Number 121517-001, Santa Clara, California, 1981: Intel Corporation.

Intel Corporation. Microcontroller Handbook, Manual Order Number 210918-002, Santa Clara, California, 1984: Intel Corporation.

Motorola,Incorporated. Motorola Linear and Interface Integrated Circuits, Series D, Phoenix, Arizona, 1983: Motorola, Incorporated.

National Semiconductor. Linear Applications Handbook, Santa Clara, California, 1982: National Semiconductor.

The vita has been removed from the scanned document