RDS-TR-11-82

THE MICHIGAN MINIMOVER ROBOT INTERFACE SYSTEM

Bruce B. Gaya†
C. S. G. Lee‡

July 1982

CENTER FOR ROBOTICS AND INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109

This project was supported in part by the Society of Manufacturing Engineers Education Foundation (Grant SME 481-195) and by the Robot Systems Division of the Center for Robotics and Integrated Manufacturing (CRIM) at The University of Michigan, Ann Arbor, Michigan. Any opinions, findings, and conclusion of recommendations expressed in this publication are those of the authors and do no necessarily reflect the views of The Society of Manufacturing Engineers or The University of Michigan.

† Currently a senior in the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, Michigan.

‡ Currently an Assistant Professor in the Department of Electrical and Computer Engineering at The University of Michigan, Ann Arbor, Michigan.

Please address all correspondence to Prof. C. S. G. Lee.

TABLE OF CONTENTS

Abstract

This report discusses the Michigan Instructional Robotics (MIR) systems at The University of Michigan. The discussion focuses on the use of the hardware and software components that interface Microbot Incorporated MiniMover-5 robots to a Digital Equipment Corporation LSI-11/23 computer system.

The report shows that a robotics laboratory for education need not be expensive. The system hardware consists of an LSI-11/23 microcomputer, 8 MiniMover-5 robots, a custom-made interface, 8 CRT terminals and power supplies. The software consists of Digital Equipment Corporation's RSX-11M operating system and LSI-11 assembler, Oregon Software's Pascal Compiler, and our custom I/O drivers and Pascal utility routines. Examples are given that illustrate their use in robot arm control.

Acknowledgements

The authors would like to thank the Society of Manufacturing Engineers Education Foundation (SME), the Department of Electrical and Computer Engineering (E.C.E.), The Computer Information and Control Engineering Program (C.I.C.E.), the Department of Mechanical Engineering and Applied Mechanics (M.E.&A.M.), and the Department of Industrial and Operations Engineering (I.&O.E.) for their financial support.

1. Introduction

One of the goals of the Center for Robotics and Integrated Manufacturing is to educate professionals and prospective engineers in the basics of robotics. To further this objective, the Departments of Electrical and Computer Engineering, Industrial and Operations Engineering, Mechanical Engineering, and the Computer Information and Control Engineering Program at The University of Michigan, with additional funding provided by the Society of Manufacturing Engineers Education Foundation, acquired eight Microbot Incorporated MiniMover-5 robot arms and a Digital Equipment Corporation LSI-11/23 computer system. This equipment was named the Michigan Instructional Robotics (MIR) system. It was hoped that the MIR system would allow users to control the small and safe MiniMover-5 robots and thus become an indispensable aid in robotics education.

The MiniMover-5 robots, however, are designed to interface with a Radio-Shack TRS-80 microcomputer in a stand-alone system configuration, not the LSI-11/23 computer purchased for this task.

To remedy this problem, we have constructed a hardware and software interface between the LSI-11/23 computer and the MiniMover-5 robots. The hardware consists of a simple digital circuit that multiplexes signals from a parallel port (DRV-11) on the LSI-11/23 computer to 8 MiniMover-5 robots. The software consists of low level Input/Output drivers and Pascal callable routines that allow an LSI-11/23 user to independently control one to eight MiniMover-5 robots. We named this software package the Michigan MiniMover Control Drivers (3MCD).

The purpose of this report is to give a functional overview of the robot interface hardware on the Michigan Instructional Robotics (MIR) system, and to describe the use of the Michigan MiniMover Control Drivers (3MCD) for the control of MiniMover-5 robots[1].

[1] This document is not intended to be a technical manual. Detailed information on any component of the system can be obtained by referring to the appropriate reference listed at the end of this document.

2. Michigan Instructional Robotics System Configuration

The Michigan Instructional Robotics (MIR) system consists of a Digital Equipment Corporation LSI-11/23 microcomputer system, a specially constructed multiplexer box, eight Microbot MiniMover-5 table-top robots, and ± 12 volt at 4 amperes direct current power supplies for the robots (See Figure 1). The software on this systems includes Digital Equipment Corporation's RSX-11M real-time multi-user operating system, Oregon Software's OMSI PASCAL compiler and the University of Michigan's Michigan MiniMover Control Drivers (3MCD). When configured properly, the system allows any one LSI-11/23 user to control up to 8 robots independently and simultaneously, or allows several simultaneous LSI-11/23 users to exclusively control any subset of these 8 robots.

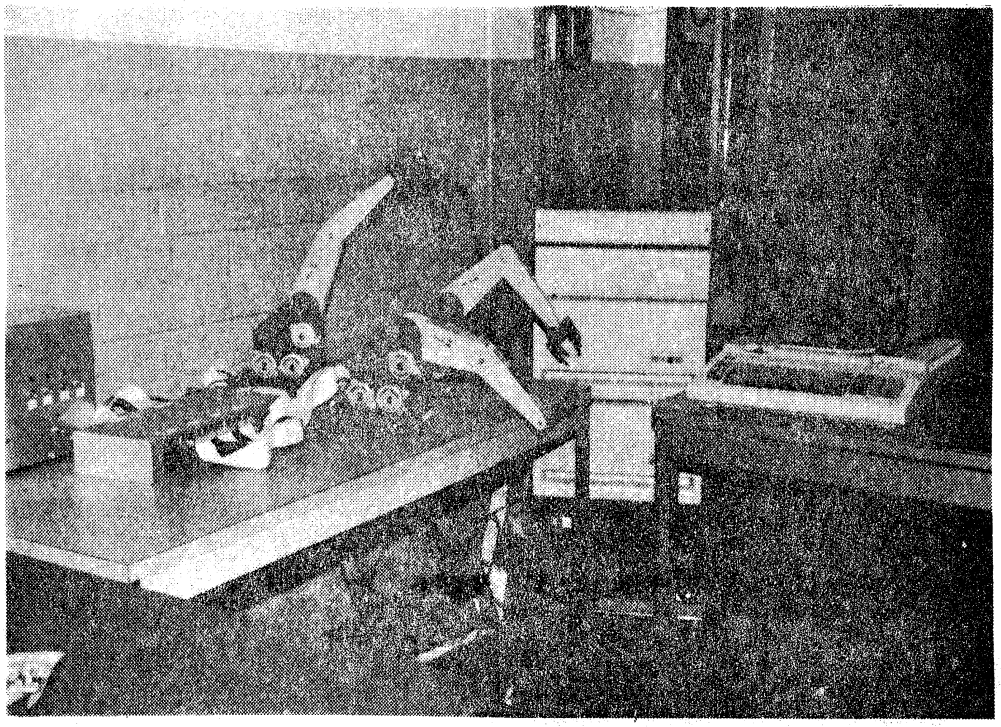


Figure 1 Michigan Instructional Robotics Systems

The discussion that follows is broken into two main segments. The first part gives an overview of the system hardware. The second part describes the Input/Output driver software available to the Pascal user for the control of the MiniMover-5 robots.

3. System Hardware

There are three main hardware components to the hardware in the system:

(1) An LSI-11/23 microcomputer system,

3.1. The LSI-11/23 Computer System

At the heart of the MIR System is a Digital Equipment Corporation LSI-11/23 microcomputer system. The LSI-11/23 computer system is equipped with two RL01 hard disk drives, one LA120 terminal, seven video terminals, and an expansion interface card cabinet.

In order to interface with the MiniMover-5 robots, the system has a DRV-11 parallel interface card installed in the backplane of the expansion cabinet. The DRV-11 provides the communication link between the LSI-11/23 computer and the MiniMover interface box.

Protruding through the back of the expansion chassis are 2 grey, 25-foot, 40-wire ribbon cables. One end of these cables leads to the DRV-11 parallel interface module inside the expansion chassis. The other end connects to the input and output connecters of the MiniMover interface box (See Section 3.3.1 for connection specifics.).

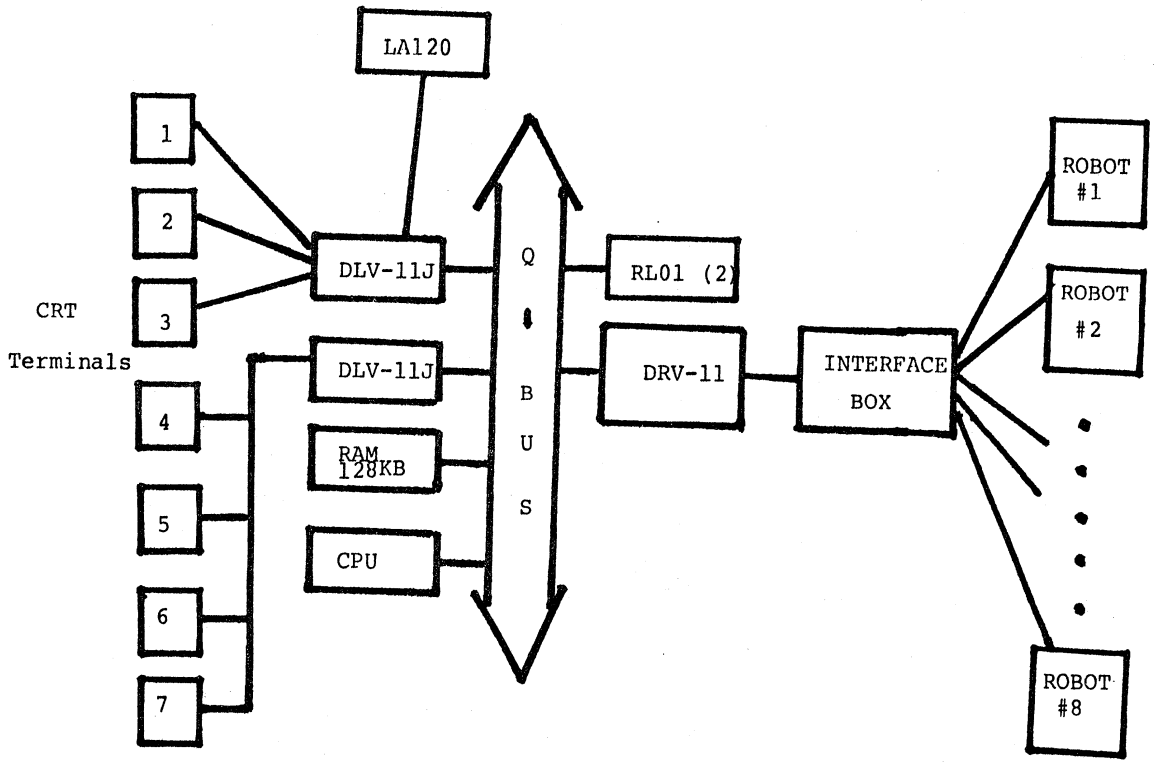


Figure 2 Instructional Robotic System Block Diagram

3.2. The Special Purpose Interface

The next stage of the MIR system hardware consists of a special purpose interface-multiplexer box[2]. The purpose of the interface is to multiplex the signals between the DRV-11 parallel interface card and eight MiniMover-5 robots. The multiplexer is housed in a 17"x6"x3" aluminum box shown in Figure 3. On one side of this box are two 40-pin dual-row connecter slots suitable for connection to 40-wire ribbon cables originating at the LSI-system's DRV-11 interface card. On the other side of the box are eight edge connecter slots suitable for connection to eight 40-wire ribbon cables originating from the MiniMover-5's. The side with the robot connector slots also has a power toggle switch, power-on indicator light and an A.C. power cord.

These edge connecters on the robot interface slot side are arranged in two rows, four connecters per row. The current version of 3MCD defines the upper right hand connecter as the port for robot #0. This slot is labeled "robot 0". The other three interface slots in the top row are defined as the ports for robots #1 through #3 starting from the slot to the right of robot #0's slot moving right to left along the row. The bottom row interface slots are similarly defined as the ports for robots #4 through #7. Although any MiniMover-5 robot can be connected to any slot, it is important to note that 3MCD defines robot numbers according to their interface box connections.

3.2.1. Making Connections to the Interface Box

The ribbon cables should always be connected to the interface box. If, however, the need arises to reconnect the ribbon cables, follow this procedure:

(1) Turn the interface box's power off and disconnect the power cord from its wall socket.

[2] A block diagram and schematic diagram of the interface box may be found in Appendix C.

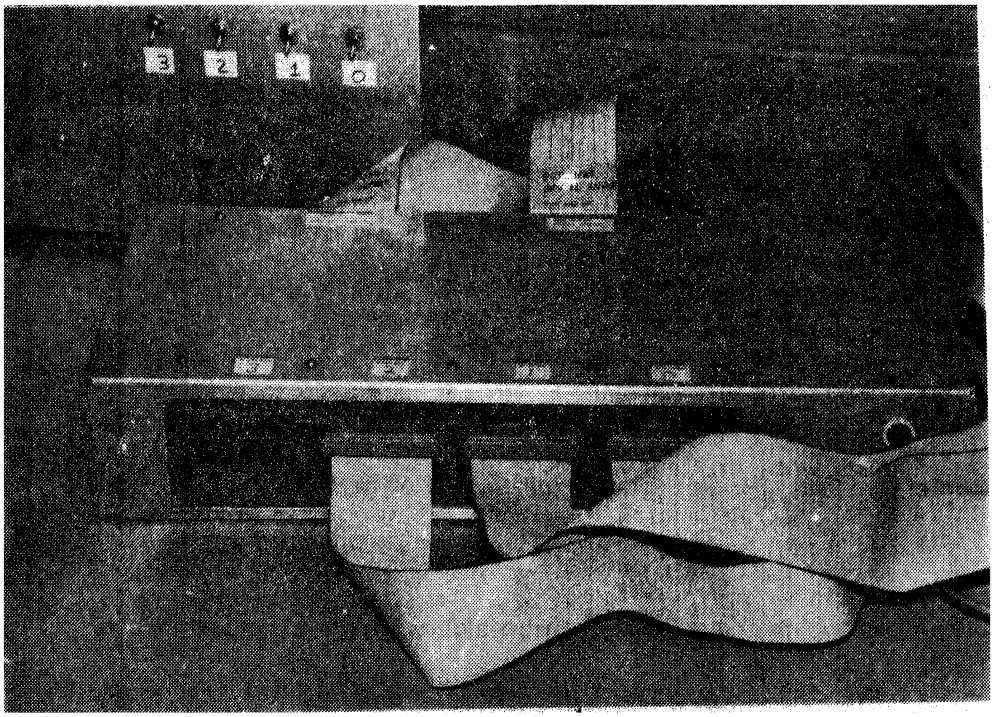


Figure 3 The MiniMover-5 Robot Interface Box

WARNING: *The MiniMover interface box has exposed 120 volts a.c. wires that remain active even when the power switch is off. It is therefore imperative to disconnect the a.c. power cord from its wall socket.*

(2) Connect the ribbon cable from the DRV-11 marked 'input' to the female connecter of the interface box marked 'input' by a) aligning the ribbon cable connector with the matching 40-pin connector on the interface box so that the ribbon cable leaves from the top of the connector; and b) pressing the ribbon cable's male connecter toward the interface box until a 'click' is heard.

(3) Connect the DRV-11 cable marked 'output' to the interface box connecter marked 'output' as in b above.

(4) Connect the ribbon cable of one of the MiniMover-5 robots to the unused slots of the interface box marked "0" through "7" by a) aligning its ribbon cable connecter with the interface box slot so that the red edge of the ribbon cable is th the left and the ribbon cable leaves at the bottom of the connector; and b) pressing toward the interface box firmly until the connecter stops its motion.

(5) Repeat step (4) until all MiniMover-5 robots are connected.

(6) Restore power to the interface box.

3.3. The MiniMover-5 Robots

The robots connected to the MIR system are MiniMover-5 model MM5 tabletop manipulators manufactured by Microbot Incorporated of Menlo Park, California. With a lifting capacity of 8 ounces and a reach of 17 inches when fully extended, these manipulator provide a safe educational tool.

What is presented here is a basic overview of the MiniMover-5 robots intended as a guide for a first time user. More information about these manipulators can be found in reference [7][3].

3.3.1. Detailed Description of the MiniMover-5 Robot

The MiniMover-5 manipulator is a five-jointed mechanical arm. Each joint is controlled by a stepper motor mounted on the base. The end hand can be positioned anywhere within a partial sphere which has a radius of 17.5 inches, as shown in Figure 4. The maximum speed varies from 2 to 6 inches per second depending upon the weight of the object being handled. Detailed performance characteristics of the MiniMover-5 are given in reference [7].

---

[3]Parts of Section 3.3 through 3.3.1.1.3 and Figures 4 through 9 are taken from reference[7] copyright (c) 1980 Microbot Inc. Used by permission.

The MiniMover-5 hand has a maximum opening of 3 inches. The 2-bar linkages maintain the fingers parallel during opening and closing. The hand contains a built-in grip sensor that senses the tension on the hand cable. The presence of an object as well as its size may thus be determined as the hand closes.

The major structural components of the MiniMover-5 robot are shown in Figure 5. The manipulator consists of a fixed base within which is housed the computer interface card. From the rear of the base extend the interface cable and D.C. power cord. The body swivels relative to the base on a shaft which is attached to the base. This is the base joint.

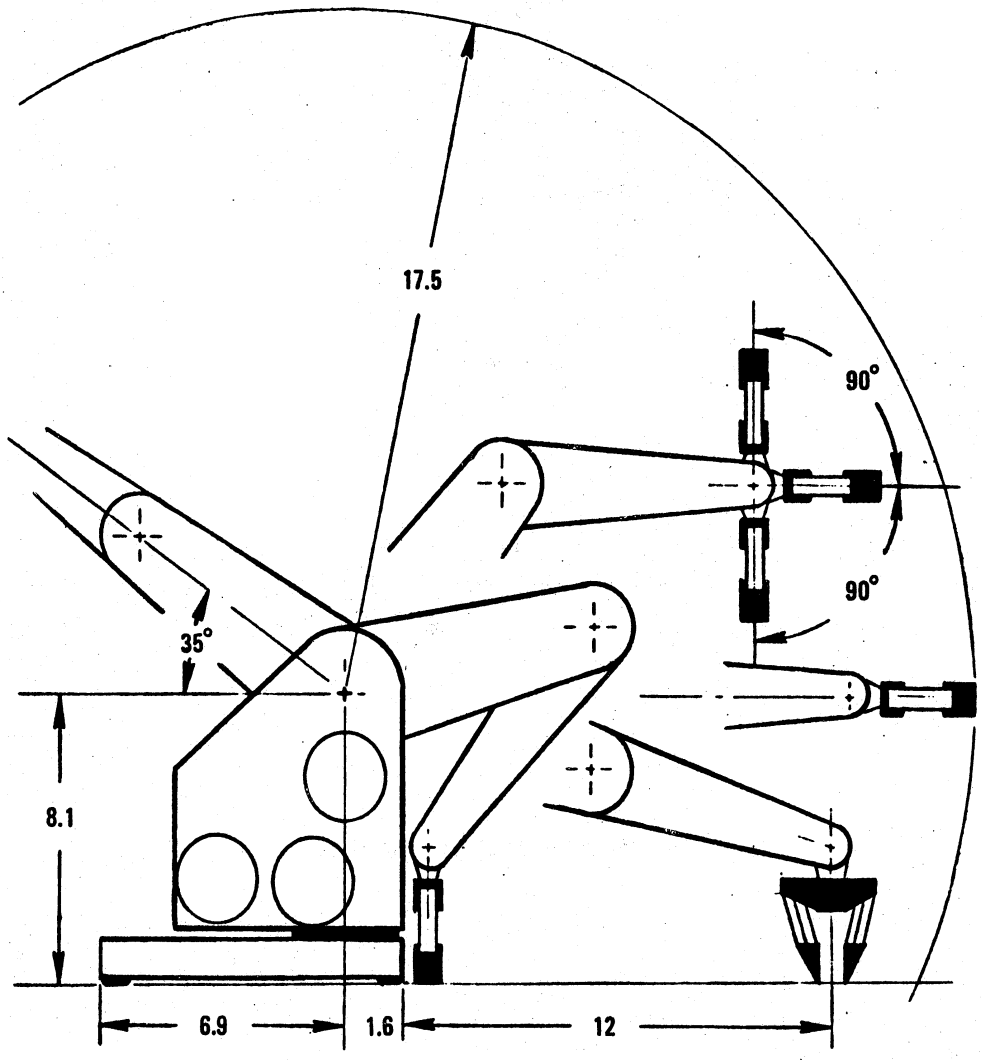
Six stepper motors with their corresponding gear reduction, are mounted at the base of the body and control each of the joints. At the upper end of the body is attached the upper arm. The upper arm rotates relative to the body on a shaft. This is the shoulder joint. Similarly, the forearm is attached to the upper arm by another shaft. This is as the elbow joint. Finally, the hand is attached to the forearm at the wrist joint.

3.3.1.1. The MiniMover-5 Robot Drive System

The motors and drive system are contained almost entirely within the body of the MiniMover-5. Each stepper motor drives a separate cable drums by means of individual gear reductions. From the drive system contained within the body, cables extend to the base, each of the robot segments, and the hand. The cable drive system is shown in Figure 6.

3.3.1.1.1. Joint Drives

The base cable (Motor 1) causes the body to rotate on the vertical base axle by driving a pulley mounted on the base.



NOTE: Dimensions in inches.

Figure 4 Operating Envelope of the MiniMover-5 Robot

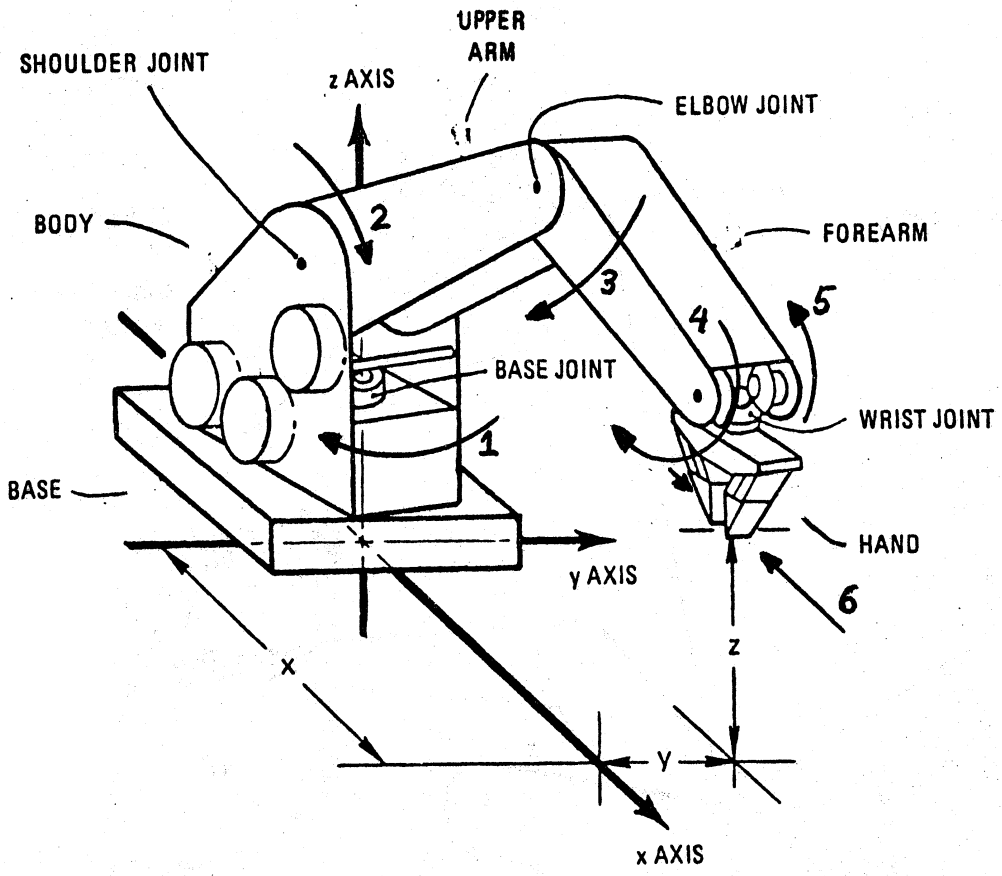


Figure 5 Major Structural Components

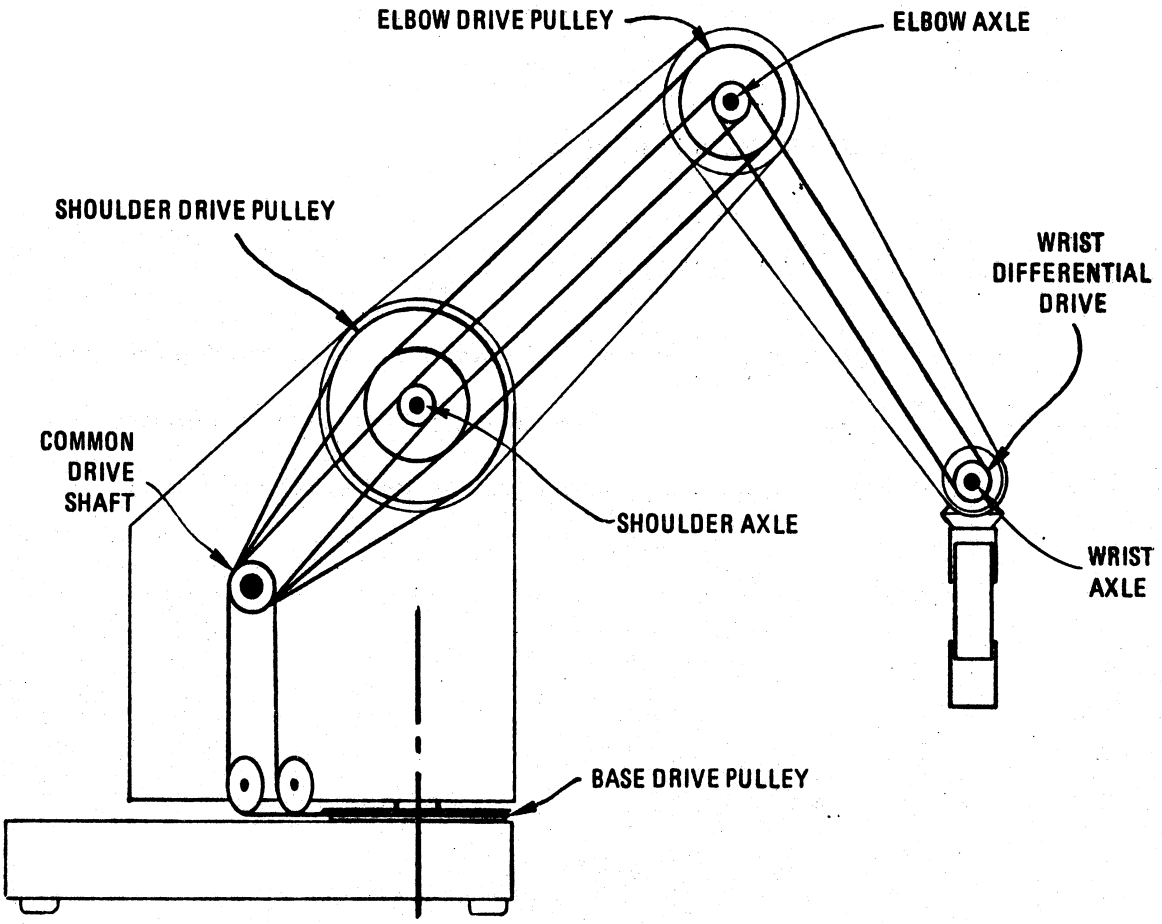


Figure 6 The MiniMover-5 Drive System

The shoulder cable (Motor 2) causes the upper arm to rotate on the horizontal shoulder axle which connects it to the body. Shoulder rotation causes equal and opposite elbow and wrist rotation so that the orientation of the hand remain unchanged.

The elbow cable (Motor 3) causes the elbow to rotate at the horizontal axle which connects the forearm to the upper arm. Elbow rotation causes equal and opposite wrist rotation, thus maintaining hand orientation.

The right and left wrist cables (Motor 4 and Motor 5) control the hand motion (see Figure 7). The two bevel gears on the wrist axle are at right angles to the wrist axle, and intersects it at the center of the wrist yoke. This configuration forms a differential gear set. Thus, if the left and right wrist motors move in the same direction, they control the pitch of the hand; if they move in opposite directions they control the roll of the hand.

3.3.1.1.2. The Hand

The hand assembly is shown in Figure 8. The hand housing is attached to the output gear of the differential gear set. The hand housing holds two pairs of links, and each pair of links terminates in a finger. Torsion springs are located on

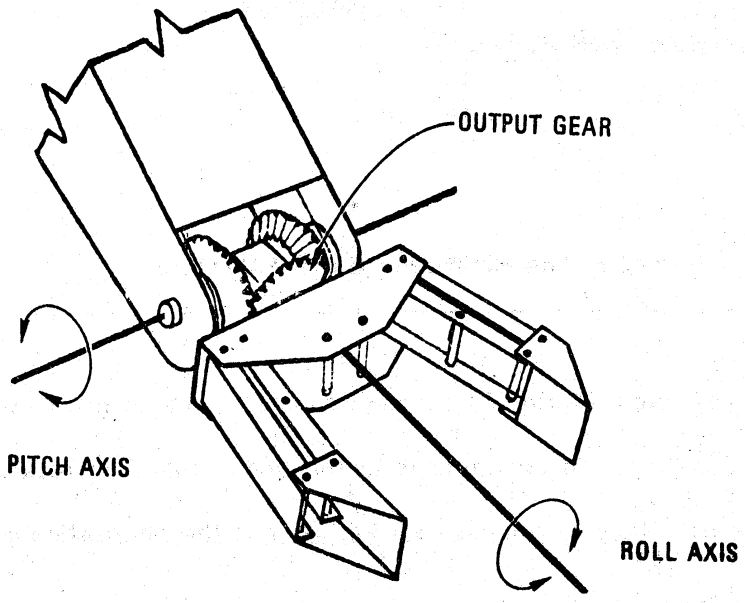


Figure 7 The MiniMover-5 Wrist Joint

the pins which attach the links to the hand housing and provide return force to the open the hand as the hand cable is slackened.

The hand cabling system is also shown in Figure 8. The hand cable passes over a pulley which is attached to a microswitch. As the hand closes upon an object to be grasped, the cable tension mounts causing the grip switch to be activated. This switch closure may be used to stop the drive motor, or to control the gripping force.

The tension spring which is mounted in series with the hand drive cable assembly permits control of the gripping force. After a position is obtained, additional cable take-up stretches the spring. This converts cable take-up into gripping force. The relationship between gripping force, hand opening, and drive

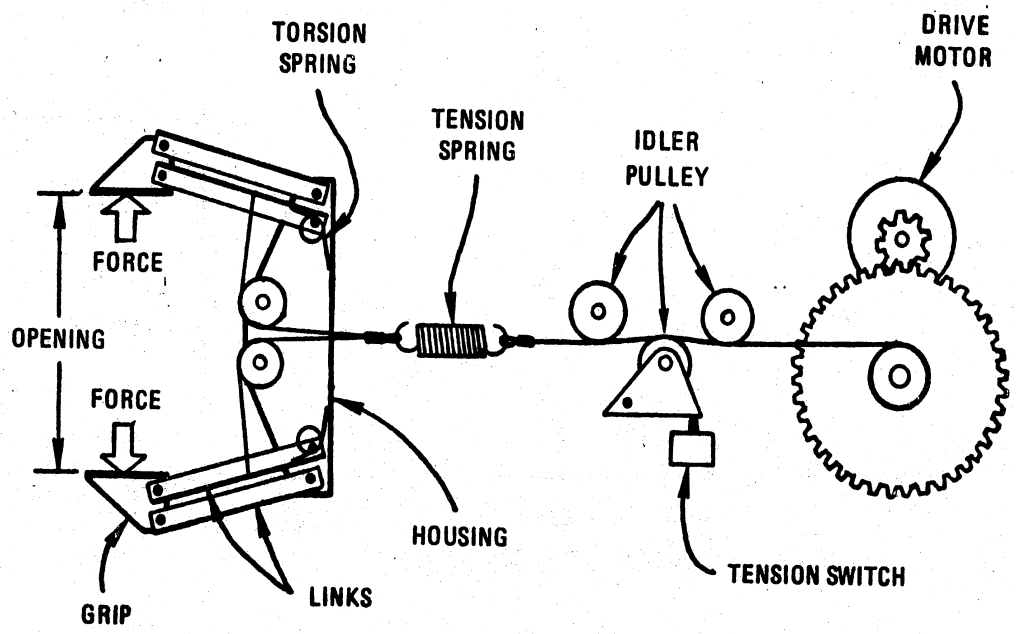


Figure 8 The MiniMover-5 Hand and Drive Mechanism

motor rotation is shown in Figure 9.

It should be noted that when the elbow is commanded to move, the grip cable becomes further wound over its elbow idler pulley. This causes the hand opening to change whenever the elbow is moved. Thus, software programs may uncouple hand opening from elbow bend by playing out the grip cable by an amount equal to the rotation of the elbow. This is necessary if the hand opening is to remain constant whenever the elbow is moved.

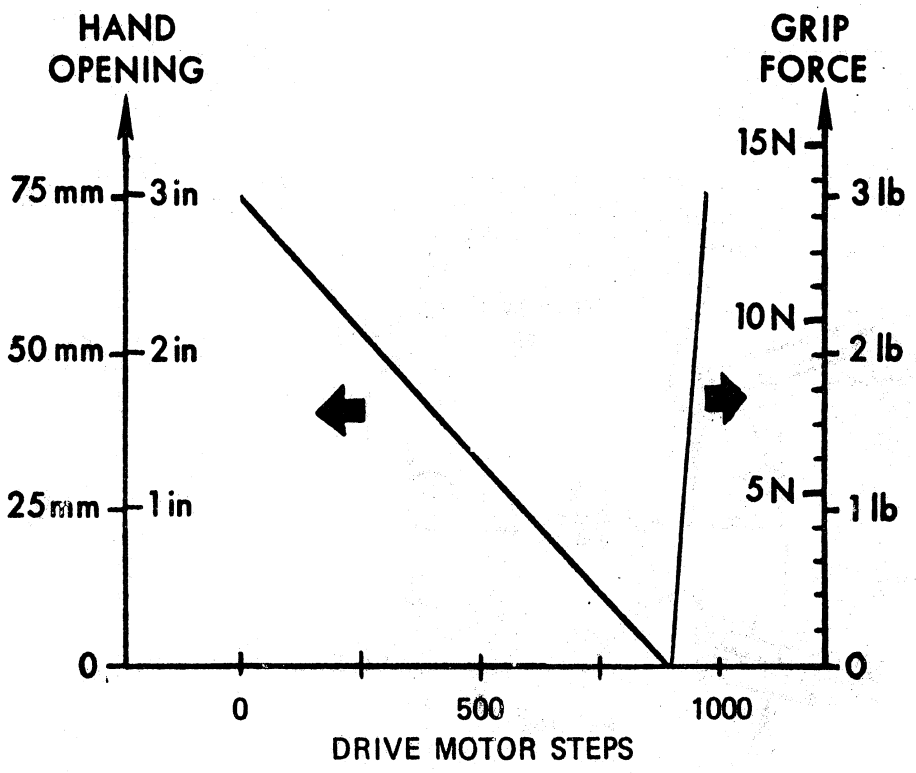


Figure 9 The MiniMover-5 Hand Opening and Grip Force vs Motor Steps

3.3.1.1.3. The MiniMover-5 Stepper Motors

Each of the drive cables of the MiniMover-5 robot is controlled by a stepper motor. The motors can be rotated forward or backward to one of 8 positions by sending out the proper phase pattern sequence to the MiniMover-5. The 3MCD software does this transparently.

Every time a stepper motor moves one step forward or backward one of the joints or the hand of the MiniMover-5 moves. The number of steps per joint revolution is given in Table 1. Hand opening and closing is proportional to the number of steps the hand motor (motor 6) moves. The constant of proportionality is:

306 steps/inch (or 12.2 steps/mm)

3.4. MiniMover Power Supplies

The final segment of the MIR system hardware consists of the MiniMover-5 robot power supplies. These supplies provide the MiniMover-5 robots with the needed 4 amperes at +/-12 volts d.c. for normal operation. The power supplies in

Motor	Joint	Steps per Revolution	Steps per Radian	Steps per Degree
1	Base	5893	937.9	16.37
2	Shoulder	5893	937.9	16.37
3	Elbow	3465	551.5	9.63
4	Right wrist	1280	203.7	3.56
5	Left wrist	1280	203.7	3.56

Table 1 Motor Step to Angle Conversion Factors

the Robotics Research Laboratory at The University of Michigan are contained in a 17"x5"x10" aluminum box shown in Figure 10. On the front side of this box are a row of 4 toggle switches and 4 light emitting diodes (L.E.D.s). These are the power switches for the 4 individual 12 volt d.c. supplies whose terminal connectors are on the right side of the box. Below these toggle switches is a single toggle switch and L.E.D.. This is the master power switch for the entire power supply. On the right side of the box is a terminal strip with 4 pairs of connectors suitable for connection to four MiniMover-5 robots. The a.c. power cord protrudes at the bottom center of this side.

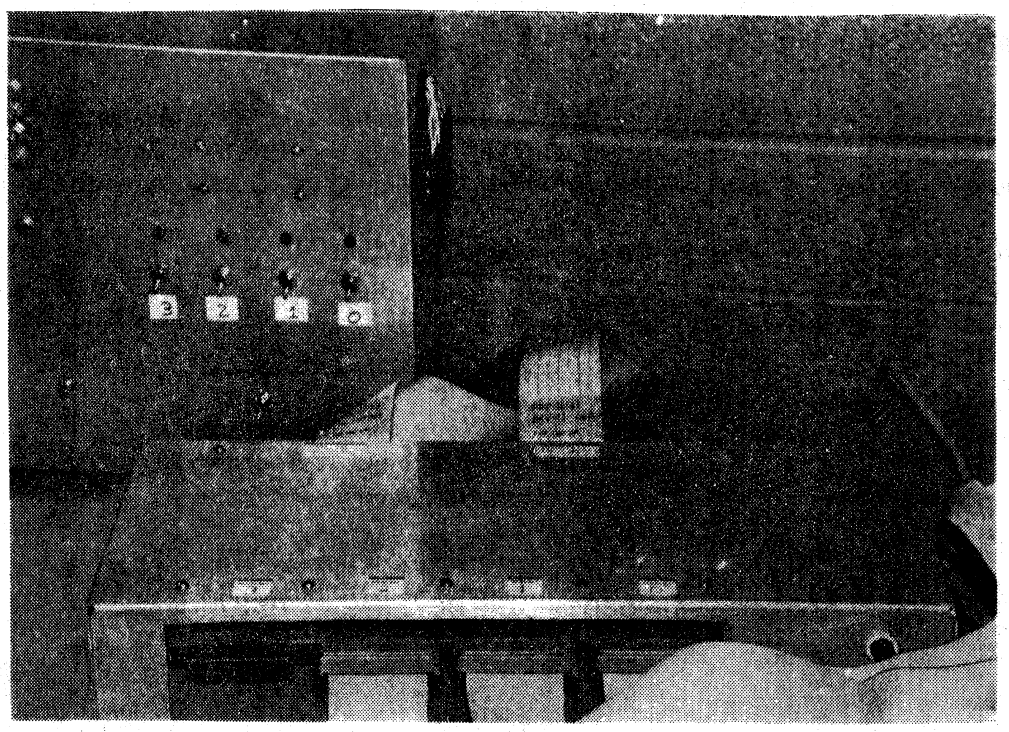


Figure 10 The MiniMover-5 Power Supply Box

3.4.1. Connecting MiniMover-5 Robots to the Power Supply

Connecting the MiniMover-5 Robots is a simple matter of matching colors. First turn the master power switch to the off position (down). Then connect the red terminal connector of a MiniMover-5 power cord to a red terminal on the power supply box and the blue terminal connector of the same MiniMover-5 power cord to a blue terminal on the power supply box. Up to 4 MiniMover-5 robots can be connected to one power supply box in this manner.

4. System Software

The software available for controlling the robots consists of the Digital Equipment Corporation RSX-11M real-time multi-user operating system, Macro-11 LSI-11 assembler, OMSI Pascal compiler, Michigan MiniMover Control Drivers (3MCD), and several Pascal robot utility programs. Programs that control the MiniMover-5 robots using the 3MCD may be written in Pascal or Macro-11.

This section assumes the reader has a basic knowledge of RSX-11M, a knowledge on Macro-11 or Pascal and has read the system hardware section of this document. The discussion centers around the use of the 3MCD software for controlling MiniMover robots.

4.1. Michigan MiniMover Control Drivers (3MCD)

The 3MCD form the low level link between a task running on the LSI-11/23 computer and a MiniMover-5 robot. The I/O driver portion of 3MCD is loadable; meaning that it becomes resident in the RSX-11M operating system upon the issuance of the MCR command LOA MB: by a privileged user. Once loaded, the driver is accessed from Macro-11 LSI assembler language by using the QIO directive. To access the 3MCD from Pascal the user calls one of the interface programs

in the file ARMLIB.OBJ[4]. This file must be linked to the user's Pascal program at task building time.

The 3MCD are setup such that each of the MiniMover-5's are configured as a separate public device. This means that all the protocols of normal I/O devices apply. Although any LSI-11/23 user may attempt control of any of the eight MiniMover-5 robots, only one user may have control over a given MiniMover robot at one time. The 3MCD commands for MiniMover control are given in Section 4.2 below.

4.1.1. Driver Services

The driver provides several important services. They allow user programs to:

(1) Allocate and exclusively use any one of the eight MiniMover-5 robots.

(2) Move any robot stepper motor one step forward or backward (directions defined in Figure 11);

(3) Read the status of a robot's grip switch; and

(4) Power down all stepper motors in any MiniMover-5 robot.

The driver also sets up 6 internal registers containing the net number of steps each motor of a particular MiniMover has moved since the motors were last powered down. This is useful when writing 'learn mode' utility programs.

4.1.2. Service Not Provided by the Michigan MiniMover Control Drivers

Beware that 3MCD **does not:**

(1) Keep track of the absolute position of the robot arm is in.

WARNING: *It is very easy to drive the robot arm to the limit of its mechani-*

[4] The name given here is the current location of the 3MCD interface files on the LSI-11/23 computer system at the University of Michigan. Check with a local authority for the location of these files on your computer system.

cal stops, possibly causing damage to the MiniMover-5 robot. If this happens, interrupt the program controlling the robot immediately!!

(2) Account for the decoupling of the MiniMover-5. As noted in the hardware documentation, certain motions of certain joints cause undesirable coupling motion in other joints. (Bending the elbow causes the hand gripper to close.) It is the applications programmer's responsibility to provide this decoupling function.

(3) Provide variable speed. The driver only sends out signal to move a stepper motor one step. The overall speed of the robot arm is determined by how often this occurs. Moving the robot too fast with heavy payloads can cause the stepper motors to slip and result in unpredictable control. It is up to the application programmers to implement speed control in their utility programs.

4.2. Michigan MiniMover Control Driver Commands

The Michigan MiniMover Control Drivers provide seven control and status functions useful in the control of the MiniMover-5 robots. These are:

(1) **ATTCH** (attach task to logical device number) which assures the calling program exclusive control over a MiniMover-5 robot.

(2) **DETCH** (detach task to a logical device number) which relinquishes a program's control over a MiniMover-5 robot,

(3) **INITMB** (INITialize MicroBot) which powers down all the motors of a MiniMover and clears the internal motor step counter registers.

(4) **MOVBOT** (MOVe roBOT) which moves up to six motors of a MiniMover one step forward or backward,

(5) **RDGSW** (ReaD Grip SWitch) which read the state of a MiniMover's grip switch, and

(6) **RDMPOS** (ReaD MiniMover Motor POSitions) which reads the internal motor step counter registers.

Also described in this section is the **ASLUN** (ASsign Logical UNit number) directive which assigns a logical unit number to a physical device. Although this is merely a system call from Macro-11, the command is necessary for controlling the MiniMovers from Pascal and is thus included with the 3MCD Pascal interface routines.

All of the following 3MCD routines have parameters passed to and from them. In Pascal the return code for each is returned in an integer variable called *dsw*. In Macro-11, the return code appears in the device status words, $DSW or the I/O status block. If the returned value of dsw equals is.suc(+1), the subroutine succeeded. If dsw is not equal to is.suc, the subroutine call failed and dsw's ($DSW's) returned value indicates the reason for subroutine failure. The most likely dsw return codes are given in Appendix A.

The following section covers Pascal call to 3MCD, MARCO-11 calls to 3MCD perform exactly the same functions. A pull-out reference guides to 3MCD calls from MACRO-11 and Pascal can be found in Appendix A and Appendix B respectively.

4.2.1. ASLUN (ASsign Logical Unit Number)

The ASLUN directive instructs the system to assign a logical unit number (LUN) to a physical device. This routine works on any device, not just MiniMover-5 robots. A logical unit number is simply an integer which refers to a physical device. (In FORTRAN, for example, 7 is usually used to refer to the standard output device). In Pascal, LUN's can be any integer from 0 to 255. Be aware that Pascal already uses LUN 6 and 7 for standard input and output.

The device is specified by a 2 letter radix 50 integer that is the RSX-11M code for that device. The device code for the MiniMover-5 robots is "MB". The integer equivalent of radix 50 MB is 19778.

The device unit number corresponds to the interface box slot number of the robot that you wish to control. Device unit numbers therefore range from 0 to 7. The device unit number does not have to be the same as the logical device number (LUN).

The assignment of logical unit numbers can also be done at task building time.

```
aslun(lun,dev,unt,dsw);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

(2) dev = Device name (two radix 50 characters or integer).

(3) unt = Device unit number (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

Interface Declarations

```
PROCEDURE ASLUN(VAR LUN, DEV, UNT; DSW : INTEGER);
        FORTRAN;
```

4.2.2. ATTCH (ATTaCH task to logical device number)

The ATTCH directive attaches the calling task to the physical device assigned to the specified logical device number (LUN). This routine works on any device, not just MiniMover-5 robots. The device is specified by the logical unit

number assigned to it by a ASLUN call or at task building time.

Upon successful completion of an ATTCH call where the device is a MiniMover-5 robots, the calling task (program) has exclusive control over that MiniMover. A task relinquishes control when it terminates execution or successfully calls the DETCH routine. One task may attach itself to several MiniMovers.

The user should be aware that a given MiniMover may be in use by another task and cause this routine to return with dsw = ie.rsu. Also note that none of the MiniMover control and status routines will execute successfully unless the specified MiniMover is attached to the calling program.

Pascal Call

```
attch(lun,dsw);
```

Parameters Passed

(1) lun = Logical unit number of device (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

Interface Declarations

```
PROCEDURE ATTCH(VAR LUN, DSW : INTEGER);
        FORTRAN;
```

4.2.3. DETCH (DETaCH task from logical device number)

The DETCH directive detaches the calling task from the physical device currently assigned to the specified logical device number (LUN). This routine works on any task that is attached to a device, not just MiniMovers. The device

is specified by the logical unit number assigned to it by a call to ALUN or at task building time.

Upon successful completion of this routine, the calling program no longer has control over the device, and the device may be claimed by other tasks. Detaching one MiniMover has no effect on the status any other MiniMovers that may be attached to the calling program.

Note that a task automatically detaches itself from a device when it terminates execution without an explicit call to DETACH.

Pascal Call

```
detch(lun,dsw);
```

Parameters Passed

(1) lun = Logical unit number (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

Interface Declarations

```
PROCEDURE DETCH(VAR LUN, DSW : INTEGER);
        FORTRAN;
```

4.2.4. INITMB (INITialize MicroBot minimover)

The INITMB directive powers down all of the motors in the MiniMover attached to the specified LUN. The motor position counters of this MiniMover are reset to zero.

Upon successful completion of a call to INITMB, the specified MiniMover robot's stepper motors will be powered down. The MiniMover can then be moved manually by turning the large gear wheels located at the rear base of the Microbot. This may be necessary to initialize the MiniMover's position. The internal stepper motor counter registers will be reset to zero, thus allowing a new reference position to be defined.

Pascal Call

```
initmb(lun,dsw);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

Interface Declarations

```
PROCEDURE INITMB(VAR LUN, DSW : INTEGER);
        FORTRAN;
```

4.2.5. MOVBOT (MOVe roBOT)

The MOVBOT directive moves each motor in the MiniMover assigned to the specified logical device number (LUN) either one step forward, reverse, or not at all depending on the integers specified in the command matrix. The motor position counters are incremented of decremented accordingly.

Each joint of a MiniMover-5 robot is controlled by one or two of 6 stepper motors. Motor #1 controls the base movement, Motor #2 controls the shoulder

movement, Motor #3 controls the elbow movement, Motor #4 and Motor #5 control the pitch and roll of the wrist, and motor #6 controls the hand. A *forward* step is defined as one which causes controlled link of the MiniMover-5 robot to move in the direction of the arrows shown in Figure 11. A *backward* step is one which causes the controlled link to move contrary to the direction of the Figure 11 arrows.

To properly call the MOVBOT routine, the user must define a 1 by 6 integer array with indices of 1 through 6. Each element in the array should contain an integer that specifies the action to be taken at the motor corresponding to its index. An array element "k" with the value 1 will cause motor #k to move one step forward. The value -1 will cause motor #k tc

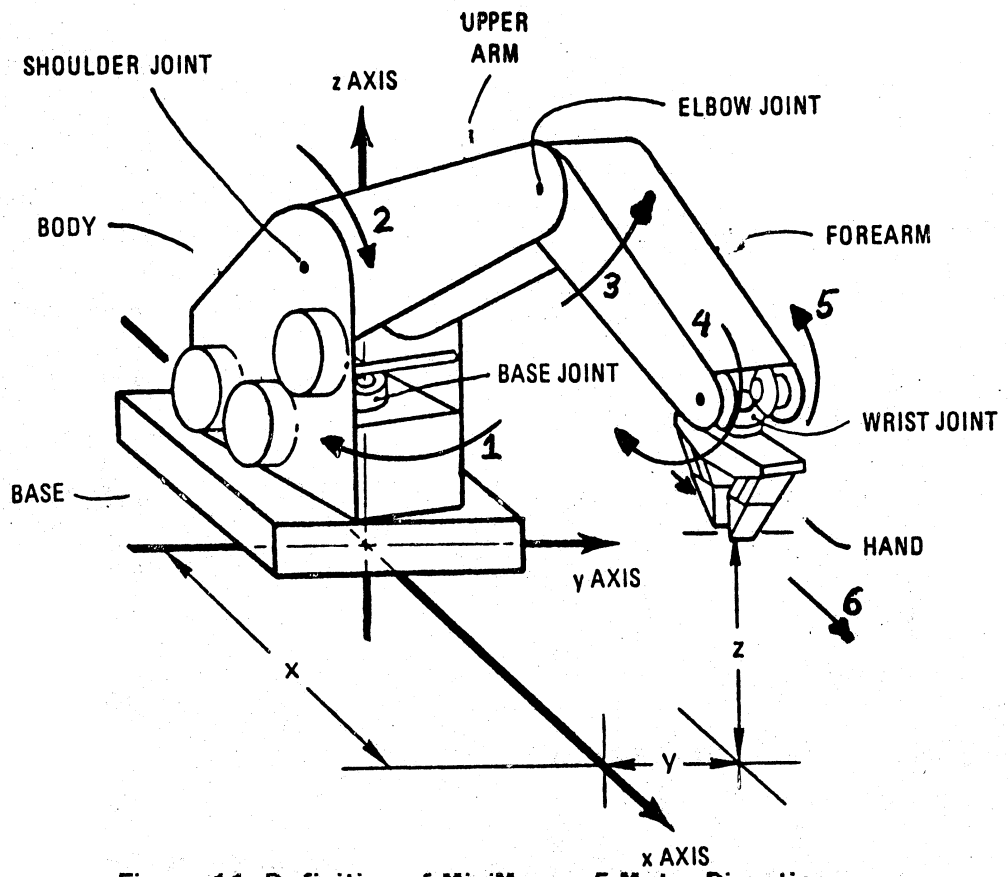


Figure 11 Definition of MiniMover-5 Motor Directions

array element has a value other than 1 or -1, the motor will retain its previous position.

As an example, if the array passed to MOVBOT is called X, and X[1] = 1, X[2] = -1, X[3] = 1, X[4] = 0, X[5] = 436, and X[6] = -1023, then motor #1 (the base) and motor #3 (the elbow) will move one step forward, motor #2 will move one step backward, and the other MiniMover motors will remain in their original positions when MOVBOT is successfully called.

Pascal Call

```
movbot(lun,cmd,dsw);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

(2) cmd = Command array (1 by 6 of integer).

Parameters Returned

(1) dsw = Directive status word (integer).

Interface Declarations

```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;
PROCEDURE MOVBOT(VAR LUN : INTEGER, VAR CMD : INTAR6, VAR DSW :
INTEGER);
        FORTRAN;
```

4.2.6. RDGSW (ReaD minimover Grip SWitch)

The RDGSW directive reads the state of the grip switch of the MiniMover attached to the specified LUN.

Upon successful completion of a call to RDGSW, the integer variable gss will contain either a 0 or a 1 in it depending on the state of the specified MiniMover hand as defined below:

value returned in gss	*MiniMover hand state*
1	open
0	closed

Pascal Call

```
rdgsw(lun,gss,dsw);
```

Parameters Passed

(1) lun = Logical unit number of MiniMover (integer).

Parameters Returned

(1) gss = Grip switch state (integer).

(2) dsw = Directive status word (integer).

Interface Declarations

```
PROCEDURE RDGSW(VAR LUN, GSS, DSW : INTEGER);
        FORTRAN;
```

4.2.7. RDMPOS (ReaD minimover Motor POSitions)

The RDMPOS directive reads the number of steps that each motor of the MiniMover attached to the specified LUN has moved since the last INITMB directive.

Upon successful completion of a call to RDMPOS, the 1 by 6 integer array passed to RDMPOS will contain the contents of the motor position counter

registers. Array element 1 will contain the net number of steps that motor #1 has moved since the last call to INITMB, array element 2 will contain the net number of steps that motor #2 has moved and so on. The values returned may be positive or negative.

Pascal Call

```
rdmpos(lun,pos,dsw);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

Parameters Returned

(1) pos = Motor position array (1 by 6 of integer).

(2) dsw = Directive status word (integer).

Interface Declarations

```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;

PROCEDURE RDMPOS(VAR LUN : INTEGER, VAR CMD : INTAR6, VAR DSW :
INTEGER);
        FORTRAN;
```

5. Summary

The purpose of this document has been to overview the Michigan Instructional Robotics system and present the details of using the Michigan MiniMover Control Drivers. These routines allow LSI-11/23 users to independently control up to 8 MiniMover-5 robots.

As you probably noted already, the 3MCD software forms only the basis for a robot control program. An applications programmer can build Pascal programs that call the 3MCD software that provide useful robot motion. Some useful Pascal robot control programs that use the 3MCD software are available from the E.C.E. Department at the University of Michigan. The practical problem of writing programs that induce the MiniMover robots to perform a useful task, however, remains the responsibility of the user.

6. References

(1) Grogono, P., *Programming in PASCAL*, Revised Edition, Addison-Wesley, Reading, 1980. New York, 1982.

(2) James, R. L., *PASCAL*, Academic Press Incorporated,

(3) Kieburtz, R. B., *Structured Programming and Problem-Solving with PASCAL*, Prentice-Hall Incorporated, Englewood Cliffs, 1978.

(4) *Memories and Peripherals*, Digital Equipment Corporation, Maynard, 1978.

(5) *Microcomputer Interfaces Handbook*, Digital Equipment Corporation, Maynard, 1981.

(6) *Microcomputer Processors*, Digital Equipment Corporation, Maynard, 1978.

(7) *MiniMover-5 User Reference and Applications Manual*, Microbot Incorporated, Menlo Park, 1980.

(8) *OMSI PASCAL-1 Version 1.2 for RSX*, Oregon Software, Portland, 1980.

(9) *RSX-11M Version 3.2 Documentation*, Vol. 1-6, Digital Equipment Corporation, Maynard, 1979.

7. Appendix A: Pascal to 3MCD Interface Routines

This appendix is intended as a reference guide for LSI-11/23 users writing Pascal programs that call the Michigan MiniMover-5 Control Drivers (3MCD). The Pascal interface source code is in three files.[5] ARMLIB.PAS contains the Pascal/MACRO-11 source code and ARMLIB.OBJ contain the linkable object code. To use any of the routines in this appendix, ARMLIB.OBJ must be linked to the calling program at task building time. The routines must also be declared as external FORTRAN program in the users main Pascal program.

[5] The names given here are the current locations of the 3MCD interface files on the LSI-11/23 system at the University of Michigan. Check with a local authority for the location of these files on your computer system.

ASLUN (ASsign Logical Unit Number)

The ASLUN directive instructs the system to assign a physical device to a logical unit number (LUN).

Pascal Call

```
ASLUN(LUN,DEV,UNT,DSW);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).
(2) dev = Device name (two radix 50 characters or integer).
(3) unt = Device unit number (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.lnl (-90) -- lun locked in use
(3) ie.ilu (-96) -- invalid LUN
(4) ie.hwr (-6) -- MiniMover driver not installed or loaded

Pascal Interface Declarations

```
PROCEDURE ASLUN(VAR LUN, DEV, UNT, DSW : INTEGER);
                FORTRAN;
```

Notes

A successful ASLUN call does not indicate that the calling task has attached itself to the device.

ASLUN Pascal Example

The following Pascal program segment assigns logical unit number (LUN) 1 to MiniMover 6.

```
PROCEDURE ASLUN(VAR LUN, DEV, UNT, DSW: INTEGER);
        FORTRAN;

PROCEDURE ASLUN_EX;

VAR       UNIT_NUMBER,    (* logical unit number of MiniMover *)
          DEV_NAME,       (* integer name of MiniMover device *)
          MICRO_NUM       (* MiniMover number *)
          RET_CODE        (* ASLUN return code *)
               :INTEGER;
          .
          .
          .

BEGIN
          .
          .
          .
  UNIT_NUMBER = 1;        (* assign unit number *)
  DEV_NAME = 19778;       (* decimal equivalent of radix 50 'MB' *)
  MICRO_NUM = 6;          (* assign MiniMover number *)

  ASLUN(UNIT_NUMBER,DEV_NAME,MICRO_NUM,RET_CODE);

  IF RET_CODE<>1
        THEN BEGIN ...   (* handle error condition *)
          .
          .
          .
 END;
```

ATTCH (ATTaCH task to logical device number)

The ATTCH directive attaches the calling task to the physical device assigned to the specified logical device number (LUN).

Pascal Call

```
ATTCH(LUN,DSW);
```

Parameters Passed

(1) lun = Logical unit number of device (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(3) ie.uln (-5) -- unassigned lun
(4) ie.ilu (-96) -- invalid LUN
(5) ie.hwr (-6) -- device driver not installed or loaded

Pascal Interface Declarations

```
PROCEDURE ATTCH(VAR LUN, DSW : INTEGER);
          FORTRAN;
```

Notes

When this directive is used for a logical device number (LUN) assigned to a MiniMover, successful completion indicates that the issuing task has exclusive control over that MiniMover.

ATTCH Pascal Example

The following Pascal program segment attaches itself to the device assigned to logical unit number (LUN) 2.

```
PROCEDURE ATTCH(VAR LUN, DSW: INTEGER);
        FORTRAN;

PROCEDURE ATTCH_EX;

VAR       UNIT_NUMBER,    (* logical unit number of MiniMover *)
          DEV_NAME,       (* integer name of MiniMover device *)
          MICRO_NUM       (* MiniMover number *)
          RET_CODE        (* ATTCH return code *)
               :INTEGER;
          .
          .
          .

BEGIN
          .
          .
          .
  UNIT_NUMBER = 1;        (* assign unit number *)

  ATTCH(UNIT_NUMBER,RET_CODE);

  IF RET_CODE<>1
        THEN BEGIN ...    (* handle error condition *)
          .
          .
          .
 END;
```

DETCH (DETaCH task from logical device number)

The DETCH directive detaches the calling task to the physical device assigned to the specified logical device number (LUN).

Pascal Call

```
DETCH(LUN,DSW);
```

Parameters Passed

(1) lun = Logical unit number (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.ilu (-96) -- invalid LUN
(6) ie.hwr (-6) -- device driver not installed or loaded

Pascal Interface Declarations

```
PROCEDURE DETCH(VAR LUN, DSW : INTEGER);
                FORTRAN;
```

Notes

When this directive is used for a logical device number (LUN) assigned to a MiniMover, successful completion indicates that the issuing task has relinquished exclusive control over that MiniMover.

DETCH Pascal Example

The following Pascal program segment detaches itself from the logical unit number assigned to (LUN) 0.

```
PROCEDURE DETCH(VAR LUN, DSW: INTEGER);
        FORTRAN;

PROCEDURE DETCH_EX;

VAR     UNIT_NUMBER,    (* logical unit number of MiniMover *)
        DEV_NAME,       (* integer name of MiniMover device *)
        MICRO_NUM       (* MiniMover number *)
        RET_CODE        (* DETCH return code *)
             :INTEGER;
        .
        .
        .

BEGIN
        .
        .
        .
  UNIT_NUMBER = 0;      (* assign unit number *)

  DETCH(UNIT_NUMBER,RET_CODE);

  IF RET_CODE<>1
       THEN BEGIN ...   (* handle error condition *)
        .
        .
        .
 END;
```

INITMB (INITialize MicroBot minimover)

The INITMB directive powers down all of the motor in the MiniMover attached to the specified LUN. The motor position counters of this MiniMover are reset to zero.

Pascal Call

INITMB(LUN,DSW);

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

Parameters Returned

(1) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Pascal Interface Declarations

```
PROCEDURE INITMB(VAR LUN, DSW : INTEGER);
                FORTRAN;
```

Notes

After an INITMB directive, the MiniMover may be forced moved.

INITMB Pascal Example

The following Pascal program segment initializes the MiniMover assigned to logical unit number (LUN) 2 and attached to the task.

```
PROCEDURE INITMB(VAR LUN, DSW: INTEGER);
        FORTRAN;

PROCEDURE INITMB_EX;

VAR       UNIT_NUMBER,    (* logical unit number of MiniMover *)
          RET_CODE        (* INITMB return code *)
               :INTEGER;
          .
          .
          .

BEGIN
          .
          .
          .
  UNIT_NUMBER = 2;        (* assign unit number *)

  INITMB(UNIT_NUMBER,RET_CODE);

  IF RET_CODE<>1
        THEN BEGIN ...   (* handle error condition *)
          .
          .
          .
 END;
```

MOVBOT (MOve roBOT)

The MOVBOT directive moves each motor in the MiniMover assigned to the specified logical device number (LUN) either one half step (one eighth turn) forward, reverse, or not at all depending on the integers specified in the command matrix. The motor position counters are incremented of decremented accordingly.

Pascal Call

```
MOVBOT(LUN,CMD,DSW);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).
(2) cmd = Command array (1 by 6 of integer).

Parameters Returned

(1) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid lun
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Pascal Interface Declarations

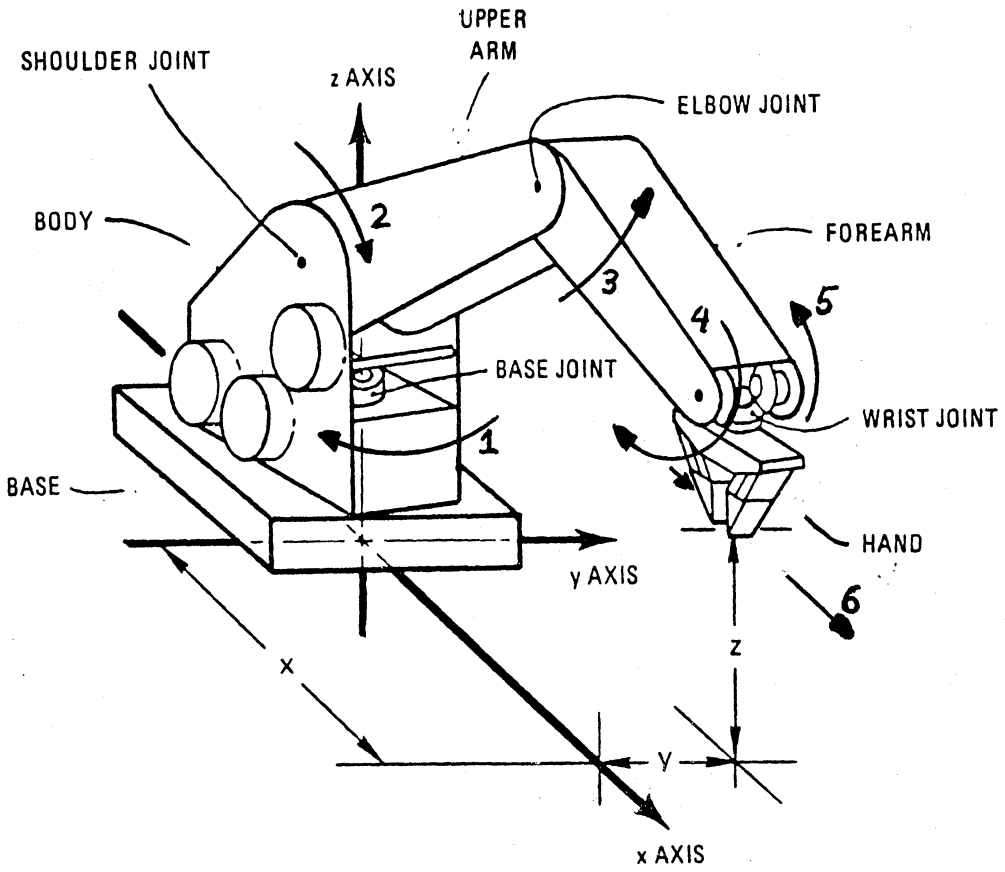
```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;

PROCEDURE MOVBOT(VAR LUN : INTEGER, VAR CMD : INTAR6, VAR DSW :
INTEGER);
                FORTRAN;
```

Notes

Each element in the array cmd corresponds to one motor in the specified MiniMover. (cmd[1] corresponds to the base motor; cmd[6] corresponds to the hand motor.) The correspondence between the values in the command array and MiniMover action is given below:

value of cmd[i]	*action in MiniMover motor[i]*
+1	move one half step forward
-1	move one half step backward
anything else	retain current motor position



Definition of MiniMover Motor Directions

Forward = with arrows Backward = against arrows

MOVBOT Pascal Example

The following Pascal program segment moves the MiniMover assigned to logical unit number (LUN) 1 and attached to the task as follows: Motor 1 (base), no movement; Motor 2 (shoulder), no movement; Motor 3 (elbow), 1 step backward; Motors 4 and 5 (wrist), 1 step forward each (this changes the pitch of the wrist by 1 step); Motor 6 (hand), 1 step forward (closing).

```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;

PROCEDURE MOVBOT(VAR LUN:INTEGER, VAR CMD:INTAR6, VAR DSW:INTEGER);
        FORTRAN;

PROCEDURE MOVBOT_EX;

VAR     UNIT_NUMBER,    (* logical unit number of MiniMover *)
        RET_CODE        (* MOVBOT return code *)
            :INTEGER;
        COMMAND         (* command array *)
            :INTAR6;
        .
        .
        .

BEGIN
        .
        .
        .
  UNIT_NUMBER = 1;        (* assign unit number *)
  COMMAND[1] = 0;         (* initialize command array *)
  COMMAND[2] = 326;
  COMMAND[3] = -1;
  COMMAND[4] = 1;
  COMMAND[5] = 1;
  COMMAND[6] = 1;

  MOVBOT(UNIT_NUMBER,COMMAND,RET_CODE);

  IF RET_CODE<>1
        THEN BEGIN ...   (* handle error condition *)
        .
        .
        .
 END;
```

RDGSW (ReaD minimover Grip SWitch)

The RDGSW directive reads the state of the grip switch of the the MiniMover attached to the specified LUN.

Pascal Call

RDGSW(LUN,GSS,DSW);

Parameters Passed

(1) lun = Logical unit number of MiniMover (integer).

Parameters Returned

(1) gss = Grip switch state (integer).
(2) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Pascal Interface Declarations

```
PROCEDURE RDGSW(VAR LUN, GSS, DSW : INTEGER);
                FORTRAN;
```

Notes

The correspondence between the state of the specified MiniMover grip switch and the value returned in gss is given below:

value returned in gss	*MiniMover hand state*
1	open
0	closed

RDGSW Pascal Example

The following Pascal program segment reads the grip switch of the MiniMover assigned to LUN 1 and attached to the task.

```
PROCEDURE RDGSW(VAR LUN, GSS, DSW: INTEGER);
        FORTRAN;

PROCEDURE RDMPOS_EX;

VAR       UNIT_NUMBER,    (* logical unit number of MiniMover *)
          GRIP_STATE,     (* grip state returned from RDGSW *)
          RET_CODE        (* RDGSW return code *)
               :INTEGER;
          .
          .
          .

BEGIN
          .
          .
          .
  UNIT_NUMBER = 1;        (* assign unit number *)

  RDGSW(UNIT_NUMBER,GRIP_STATE,RET_CODE); (* read grip switch *)

  IF RET_CODE<>1
        THEN BEGIN ...   (* handle error condition *)
          .
          .
          .
 END;
```

RDMPOS (ReaD minimover Motor POSitions)

The RDMPOS directive reads the number of steps that each motor of the MiniMover attached to the specified LUN has moved since the last INITMB directive.

Pascal Call

```
RDMPOS(LUN,CMD,DSW);
```

Parameters Passed

(1) lun = Logical device number of MiniMover (integer).

Parameters Returned

(1) pos = Motor position array (1 by 6 of integer).
(2) dsw = Directive status word (integer).

dsw Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Pascal Interface Declarations

```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;

PROCEDURE RDMPOS(VAR LUN : INTEGER, VAR CMD : INTAR6, VAR DSW :
INTEGER);
                FORTRAN;
```

RDMPOS Pascal Example

The following Pascal program segment reads the motor position registers of the MiniMover assigned to logical unit number (LUN) 1 and attached to the task.

```
TYPE INTAR6:ARRAY[1..6] OF INTEGER;

PROCEDURE RDMPOS(VAR LUN:INTEGER, VAR CMD:INTAR6, VAR DSW:INTEGER);
        FORTRAN;

PROCEDURE RDMPOS_EX;

VAR     UNIT_NUMBER,    (* logical unit number of MiniMover *)
        RET_CODE        (* RDMPOS return code *)
            :INTEGER;
        POSITION        (* position array returned from RDMPOS *)
            :INTAR6;
        .
        .
        .

BEGIN
        .
        .
        .
  UNIT_NUMBER = 1;      (* assign unit number *)

  RDMPOS(UNIT_NUMBER,POSITION,RET_CODE);

  IF RET_CODE<>1
        THEN BEGIN ...  (* handle error condition *)
        .
        .
        .
 END;
```

8. Appendix B: MACRO-11 to 3MCD Interface

This appendix is intended as a reference guide for LSI-11/23 users writing MACRO-11 assembly language programs that call the Michigan MiniMover-5 Control Drivers (3MCD). The drivers are accessed through the QIO command. When the $C and $S forms of QIO and QIOW are used the return codes for the call are in $DSW the device status word and the I/O status block (ISB). The carry flag will be reset and both the DSW and the ISB will be equal to +1 after a QIO call if the I/O attempt succeeded.

Also included in this appendix is the documentation for a call to ALUN(Assign Logical Device Number) which is the MACRO-11 version of the 3MCD Pascal interface routines ASLUN.

ALUN (Assign Logical Unit Number)

The ALUN directive instructs the system to assign a physical device to a logical unit number (LUN).

MACRO-11 Call

```
ALUN$ lun,dev,unt
```

Parameters Passed

(1) lun = Logical device number of MiniMover
(2) dev = Device name two characters (MB for the Microbots)
(3) unt = Device unit number

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) $DSW = Directive status word

$DSW Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.lnl (-90) -- lun locked in use
(3) ie.ilu (-96) -- invalid LUN
(4) ie.hwr (-6) -- MiniMover driver not installed or loaded

Notes

A successful ALUN call does not indicate that the calling task has attached itself to the device.

ALUN MACRO-11 Example

The following MACRO-11 program segment assigns logical unit number (LUN) 1 to MiniMover 6.

```
        .MCALL  ALUN$C
        .GLOBL  $DSW
        .TITLE  ALUN_EX
        .PSECT
    TEST::
        .
        .
        .
        ALUN$C  1,MB,6              ;call ALUN
        BCS     ERR                 ;branch if carry set
                                    ;to error handling code
        .
        .
        .
        .END TEST
```

ATTCH (ATTaCH task to logical device number)

The ATTCH directive attaches the calling task to the physical device assigned to the specified logical device number (LUN).

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb
```

Parameters Passed

(1) fnc = I/O function code = 768. for ATTCH
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) $DSW = Directive status word

$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(3) ie.uln (-5) -- unassigned lun
(4) ie.ilu (-96) -- invalid LUN
(5) ie.hwr (-6) -- device driver not installed or loaded

Notes

When this directive is used for a logical device number (LUN) assigned to a MiniMover, successful completion indicates that the issuing task has exclusive control over that MiniMover.

ATTCH MACRO-11 Example

The following MACRO-11 program segment attaches itself to the device assigned to logical unit number (LUN) 2.

```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  ATTCH_EX
        .PSECT
    TEST::
        .
        .
        .
        QIOW$S  #768.,#2,#17.,,#ISB     ;ATTACH task
        BCS     ERR                     ;branch if carry set
                                        ;to error handling code
        CMP     ISB,#1.                 ;routine worked?
        BEQ     OK                      ;branch if it worked
    ERR::
        CMP     $DSW,177772             ;error handling code
                                        ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
    ISB::
        .BLKW   2                       ;I/O status block space
```

DETCH (DETaCH task from logical device number)

The DETCH directive detaches the calling task to the physical device assigned to the specified logical device number (LUN).

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb
```

Parameters Passed

(1) fnc = I/O function code = 1024. for DETCH
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) $DSW = Directive status word

$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.ilu (-96) -- invalid LUN
(6) ie.hwr (-6) -- device driver not installed or loaded

Notes

When this directive is used for a logical device number (LUN) assigned to a MiniMover, successful completion indicates that the issuing task has relinquished exclusive control over that MiniMover.

DETCH MACRO-11 Example

The following MACRO-11 program segment detaches itself from the logical unit number assigned to (LUN) 0.

```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  DETCH_EX
        .PSECT
    TEST::
        .
        .
        .
        QIOW$S  #1024.,#0,#17.,,,#ISB   ;DETACH task
        BCS     ERR                     ;branch if carry set
                                        ;to error handling code
        CMP     ISB,#1.                 ;routine worked?
        BEQ     OK                      ;branch if it worked
    ERR::
        CMP     $DSW,177772             ;error handling code
                                        ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
    ISB::
        .BLKW   2                       ;I/O status block space
```

INITMB (INITialize MicroBot minimover)

The INITMB directive powers down all of the motor in the MiniMover attached to the specified LUN. The motor position counters of this MiniMover are reset to zero.

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb
```

Parameters Passed

(1) fnc = I/O function code = 1792. for INITMB
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) $DSW = Directive status word

$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Notes

After an INITMB directive, the MiniMover may be forced moved.

INITBOT MACRO-11 Example

The following MACRO-11 program segment initializes the MiniMover assigned to logical unit number (LUN) 2 and attached to the task.

```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  INITBOT_EX
        .PSECT
TEST::
        .
        .
        .
        MOV     #2,4(R5)
        QIOW$S  #1792.,4(R5),#17.,,#ISB ;INITialize MiniMover
        BCS     ERR             ;branch if carry set
                                ;to error handling code
        CMP     ISB,#1.         ;routine worked?
        BEQ     OK              ;branch if it worked
ERR::
        CMP     $DSW,177772     ;error handling code
                                ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
ISB::
        .BLKW   2               ;I/O status block space
```

MOVBOT (MOve roBOT)

The MOVBOT directive moves each motor in the MiniMover assigned to the specified logical device number (LUN) either one half step (one eighth turn) forward, reverse, or not at all depending on the parameters of the call. The motor position counters are incremented of decremented accordingly.

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb,<cm1,cm2,cm3,cm4,cm5,cm6>
```

Parameters Passed

(1) fnc = I/O function code = 1280. for MOVBOT
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)
(4) cm1 = command for motor #1
(5) cm2 = command for motor #2
(6) cm3 = command for motor #3
(7) cm4 = command for motor #4
(8) cm5 = command for motor #5
(9) cm6 = command for motor #6

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) $DSW = Directive status word

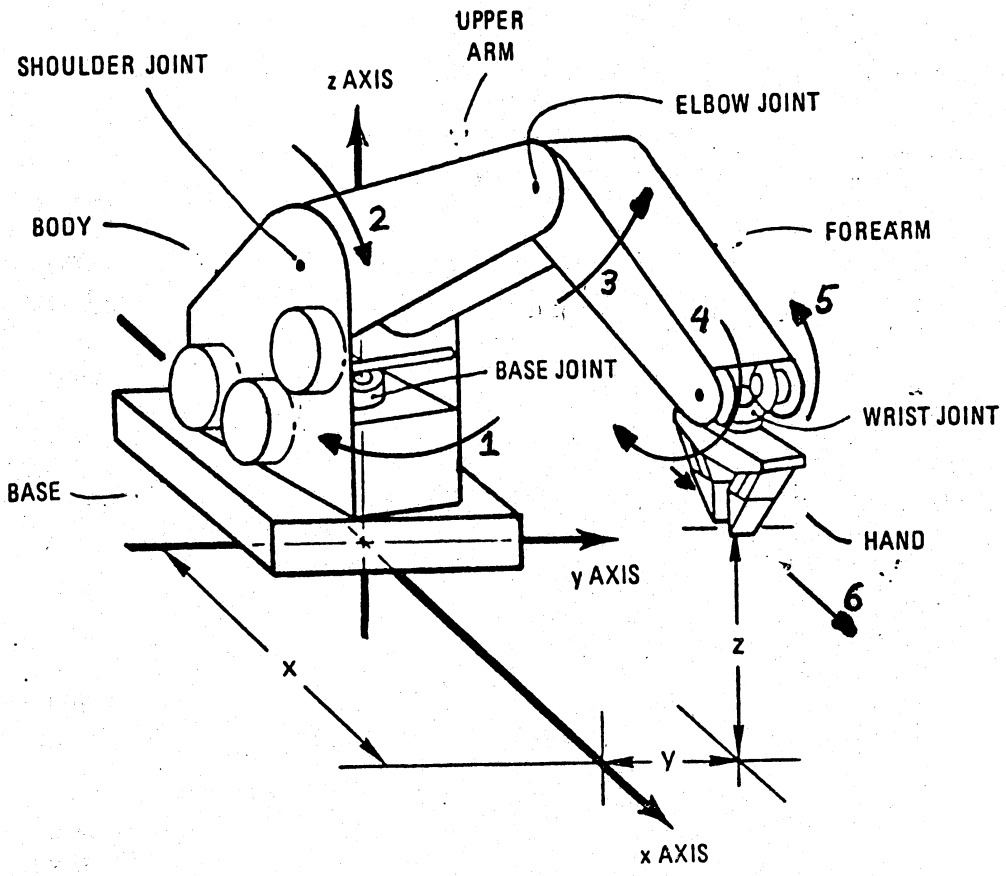
$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid lun
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Notes

Each parameter cm1 through cm6 corresponds to one motor in the specified MiniMover. (cm1 corresponds to the base motor; cm6 corresponds to the hand motor.) The correspondence between the values of these parameters and MiniMover action is given below:

value of cm[i]	*action in MiniMover motor[i]*
+1	move one half step forward
-1	move one half step backward
anything else	retain current motor position



Definition of MiniMover Motor Directions

Forward = with arrows Backward = against arrows

MOVBOT MACRO-11 Example

The following MACRO-11 program segment moves the MiniMover assigned to logical unit number (LUN) 1 and attached to the task as follows: Motor 1 (base), no movement; Motor 2 (shoulder), no movement; Motor 3 (elbow), 1 step backward; Motors 4 and 5 (wrist), 1 step forward each (this changes the pitch of the wrist by 1 step); Motor 6 (hand), 1 step forward (closing).

```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  MOVBOT_EX
        .PSECT
TEST::
        .
        .
        .
        QIOW$S  #1280.,#1,#17.,,#ISB,,<0,11,177776,1,1,1> ;MOVe MiniMover
        BCS     ERR                 ;branch if carry set
                                    ;to error handling code
        CMP     ISB,#1.             ;routine worked?
        BEQ     OK                  ;branch if it worked
ERR::
        CMP     $DSW,177772         ;error handling code
                                    ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
ISB::
        .BLKW   2                   ;I/O status block space
```

RDGSW (ReaD minimover Grip SWitch)

The RDGSW directive reads the state of the grip switch of the the MiniMover attached to the specified LUN.

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb,<buf,cnt>
```

Parameters Passed

(1) fnc = I/O function code = 1536. for RDGSW
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)
(4) cnt = byte length of buf = 2.

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) buf = 1 word buffer for hand state
(3) $DSW = Directive status word

$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

Notes

The correspondence between the state of the specified MiniMover grip switch and the value returned in buf is given below:

value returned in buf	*MiniMover hand state*
1	open
0	closed

RDGSW MACRO-11 Example

The following MACRO-11 program segment reads the grip switch of the MiniMover assigned to LUN 1 and attached to the task.

```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  RDGSW_EX
        .PSECT
TEST::
        .
        .
        .
        QIOW$S  #1536.,#1,#17.,,#ISB,,<BUF,#2> ;get grip switch status
        BCS     ERR                 ;branch if carry set
                                    ;to error handling code
        CMP     ISB,#1.             ;routine worked?
        BEQ     OK                  ;branch if it worked
ERR::
        CMP     $DSW,177772         ;error handling code
                                    ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
ISB::
        .BLKW   2                   ;I/O status block space
BUF::
        .BLKW   1                   ;grip switch status buffer
```

RDMPOS (ReaD minimover Motor POSitions)

The RDMPOS directive reads the number of steps that each motor of the MiniMover attached to the specified LUN has moved since the last INITBOT directive.

MACRO-11 Call

```
QIOW$  fnc,lun,efn,,isb,<buf,cnt>
```

Parameters Passed

(1) fnc = I/O function code = 1048. for RDMPOS
(2) lun = Logical unit number of device
(3) efn = event flag number (17. works well)
(4) cnt = byte length of buf = 6.

Parameters Returned

(1) isb = I/O status block (2 word buffer)
(2) buf = 6 word buffer for motor position counter registers
(3) $DSW = Directive status word

$DSW and I/O Status Block (ISB) Return Codes

(1) is.suc (+1) -- successful completion
(2) ie.dna (-7) -- device not attached
(3) ie.rsu (-17) -- non-sharable resource in use (another task owns device)
(4) ie.uln (-5) -- unassigned lun
(5) ie.lnl (-90) -- lun locked in use
(6) ie.ilu (-96) -- invalid LUN
(7) ie.hwr (-6) -- MiniMover driver not installed or loaded

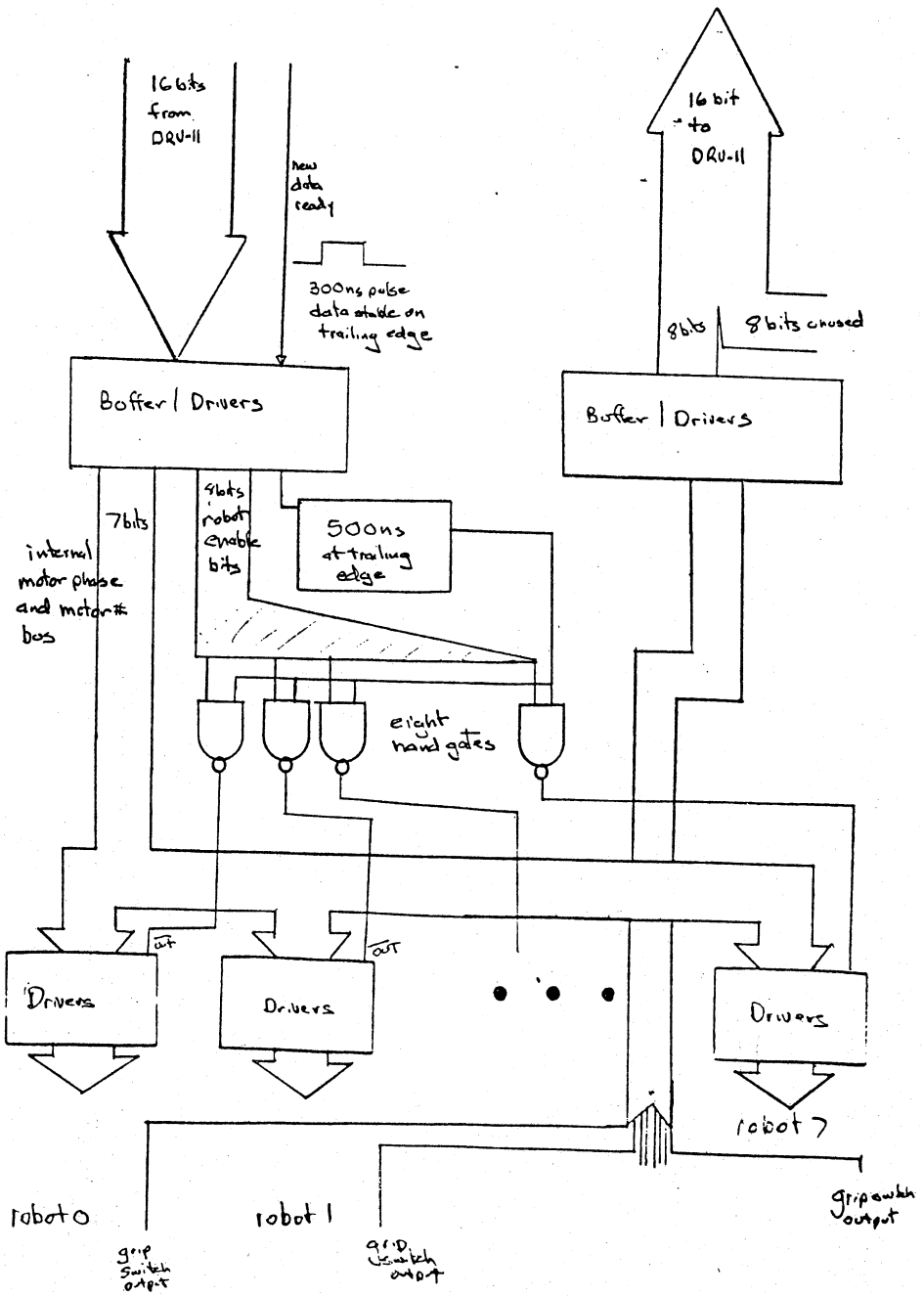
RDMPOS MACRO-11 Example

The following MACRO-11 program segment reads the motor position registers of the MiniMover assigned to logical unit number (LUN) 1 and attached to the task.

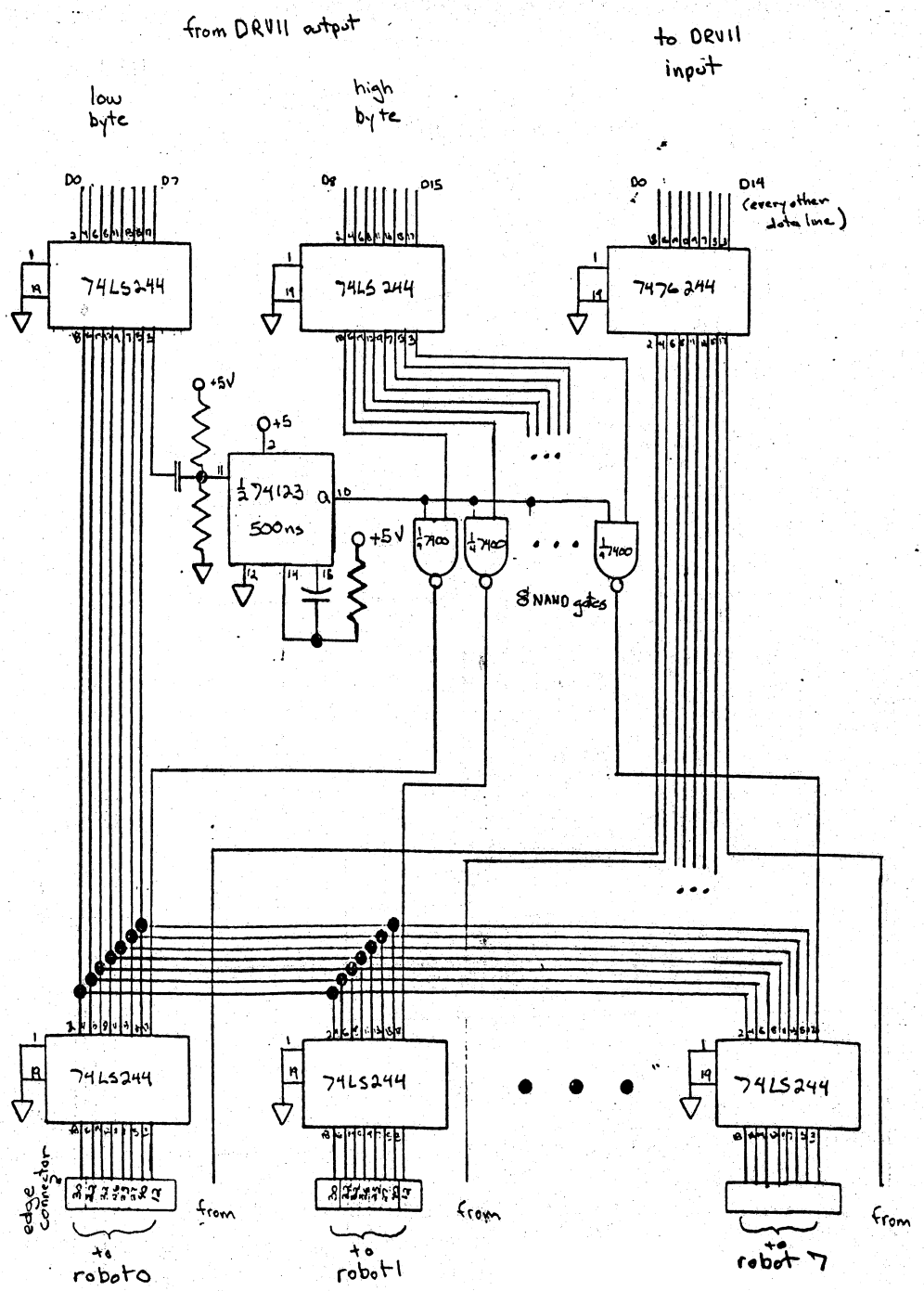
```
        .MCALL  QIOW$S
        .GLOBL  $DSW
        .TITLE  RDGSW_EX
        .PSECT
TEST::
        .
        .
        .
        QIOW$S  #1536.,#1,#17.,,#ISB,,<BUF,#6>  ;get motor position counters
        BCS     ERR                     ;branch if carry set
                                        ;to error handling code
        CMP     ISB,#1.                 ;routine worked?
        BEQ     OK                      ;branch if it worked
ERR::
        CMP     $DSW,177772             ;error handling code
                                        ;that looks at $DSW & ISB
        .
        .
        .
        .END TEST
ISB::
        .BLKW   2                       ;I/O status block space
BUF::
                                        ;motor position buffers
MOT1::
        .BLKW   1                       ;motor #1 position buffer
MOT2::
        .BLKW   1                       ;motor #2 position buffer
MOT3::
        .BLKW   1                       ;motor #3 position buffer
MOT4::
        .BLKW   1                       ;motor #4 position buffer
MOT5::
        .BLKW   1                       ;motor #5 position buffer
MOT6::
        .BLKW   1                       ;motor #6 position buffer
```

9. Appendix C: MiniMover Interface Hardware Design

MiniMover Interface Hardware Block-Diagram



MiniMover Interface Hardware Schematic



Format of Output Word to Multiplexer

Bit Number	Signal	Comment
15	ER7	enable robot 7 (1=enable)
14	ER6	enable robot 6
13	ER5	enable robot 5
12	ER4	enable robot 4
11	ER3	enable robot 3
10	ER2	enable robot 2
9	ER1	enable robot 1
8	ER0	enable robot 0
7	spare	unused
6	A2	motor number MSB
5	A1	motor number
4	A0	motor number LSB
3	D3	motor phase pattern MSB
2	D2	motor phase pattern
1	D1	motor phase pattern
0	D0	motor phase pattern LSB

Format of Input Word From Multiplexer

Bit Number	Signal	Comment
15	BZ7	unused
14	GS7	robot 7 grip switch (1=closed)
13	BZ6	unused
12	GS6	robot 6 grip switch
11	BZ5	unused
10	GS5	robot 5 grip switch
9	BZ4	unused
8	GS4	robot 4 grip switch
7	BZ3	unused
6	GS3	robot 3 grip switch
5	BZ2	unused
4	GS2	robot 2 grip switch
3	BZ1	unused
2	GS1	robot 1 grip switch
1	BZ0	unused
0	GS0	robot 0 grip switch

MiniMover-5 Cable Connections

Signal	Slot Number	Interface Box Connection
D7	20	74LS244 pin depends on robot
D6	24	no connection
D5	28	no connection
D4	18	no connection
D3	26	74LS244 pin 12
D2	32	74LS244
D1	22	74LS244 pin 16
DO	30	74LS244 pin 18
A7	36	+5 volts
A6	38	ground
A5	35	ground
A4	31	+5 volts
A3	34	+5 volts
A2	40	74LS244 pin 5
A1	27	74LS244 pin 7
AO	25	74LS244
OUT'	12	74LS244 pin 3
GND	8	ground
GND	38	ground

10. Appendix D: Excerpts from MiniMover-5 User Reference Manual

This appendix contains excerpts taken from the Minimover-5 User Reference Manual that are useful to users.

Table 2

MiniMover-5 Performance Characteristics

GENERAL

Configuration	5 revolute axes and integral hand
Drive	Electric stepper motors- Open loop control
Controller	Microcomputer provided by user
Interface	TRS-80 or 8-Bit parallel
Program language	ARMBASIC for TRS-80 (Z-80 driver available)
Power requirement	12 volts, 4 amps DC

PERFORMANCE

Resolution	0.013 in (0.3 mm) maximum on each axis
Load Capacity	8 oz (225 gm) at full extension 16 oz (450 gm) at half extension
Gripping force	3 lbs (13 N) maximum
Reach	17.5 in (444 mm)
Static load force	4 lbs (18 N) maximum

DETAILED PERFORMANCE

Motion	Range	Speed (full load)	Speed (no load)
Base	±90 deg	0.37 rad/s	0.42 rad/s
Shoulder	+144, -35 deg	0.15 rad/s	0.36 rad/s
Elbow	+0, -149 deg	0.23 rad/s	0.82 rad/s
Wrist Roll	±180 deg	1.31 rad/s	2.02 rad/s
Wrist Pitch	±90 deg	1.31 rad/s	2.02 rad/s
Hand	0-3 in (0-75 mm)	3 lb/s (13N/s)	0.80 in/s (20 mm/s)

PHYSICAL CHARACTERISTICS

Arm weight	6 lbs (3 kg)
Interface cable length	3 ft (900 mm)

The major structural components are shown in Figure 13. The manipulator consists of a fixed base within which is housed the computer

B. Speed-Torque Considerations

The torque output of the stepper motors used on the MiniMover-5 vary with their speed as shown in Figure 21. At slow speeds, maximum torque (lifting capacity) is obtained. Above a critical high speed, no torque is obtained. With intermediate speeds, intermediate torque loads can be maintained. Alternatively with any given torque load we can determine the critical speed above which the motor will not turn.

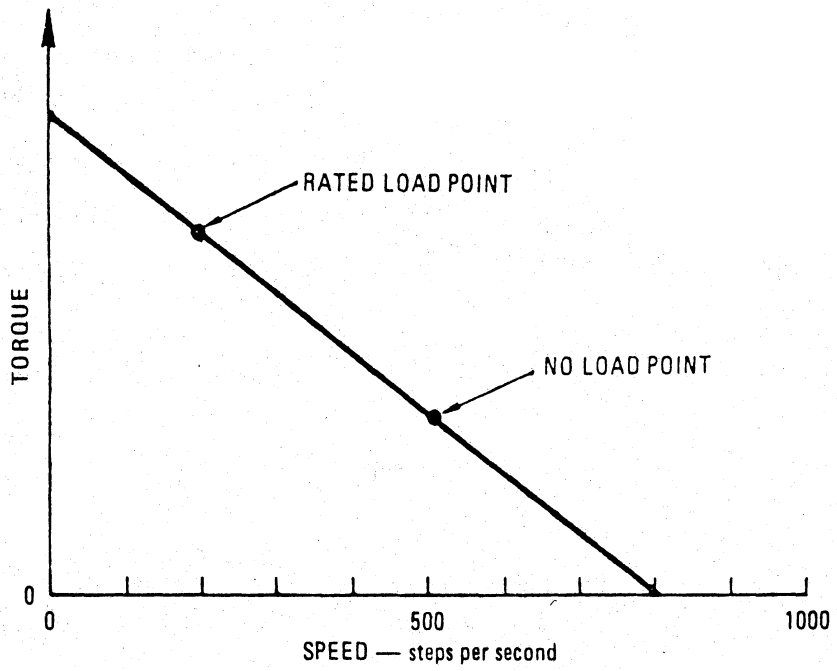


Figure 21 Stepper Motor Speed-Load Tradeoff

The torque required by the motors of the MiniMover-5 depends on the configuration of the arm and the load held in the hand. This relation is a complex trigonometric expression involving the lengths and weights of all the arm members. Instead of solving such an expression before each movement to determine the maximum speed of movement, it is simpler to program for the worst-case.

The worst case is when the members of the arm are at maximum horizontal extension, requiring the maximum motor torque. All other configurations will require less motor torque.

In the worst-case configuration the motor torque also depends on the load held in the hand. With no load, the torque on all the motors is the same (by design) and operation is at the no-load point shown in Figure 21. With the rated load (the arm lifting 8 ounces) the torque on all the motors except the base motor (which does not lift) is approximately equal, and operation is at the rated load point shown in Figure 21.

C. Recommended Operating Speeds

Speed control of the stepper motors can be simplified by considering only a few load conditions such as rated-, half-, and no-load as given in Table 3. With the given load, no slippage* will occur as long as these rates are not exceeded. Users can drive the arm at any speed they wish (including very slowly) but the arm will not always follow if the speed is above the maximum rates given.

Table 3

Maximum No-Slip Stepping Rates

Load	Half Steps per second	Time Delay (ARMBASIC)
None	525	20
Half (4 oz.)	370	50
Rated (8 oz.)	230	100

* Motor slippage will cause an error between where the arm is and where the computer program thinks it is, and may result in unpredictable performance.

In general we want to perform a task as fast as possible without risking slippage. The following suggestions may prove helpful:

* Lowering a load may be done at no-load speed if shoulder, elbow, and wrist all descend.
* Swiveling a load about the base joint only, may be done at no-load speed.
* Opening the hand may always be done at no-load speed.
* Closing the hand until contact may always be done at no-load speed.

Under the following load situations, special care must be exercised in selecting the proper speed:

* Raising a load with any joint.
* Developing a gripping force once contact has been detected.

X OPERATION FROM AN 8-BIT PARALLEL PORT

The MiniMover-5 may be shipped in either the TRS-80 or 8-bit parallel configuration. In either case, the printed circuit card in the base is identical, and may be converted from one to the other by following the procedure outlined in this section. Briefly, this procedure entails removing/inserting two integrated circuits and various jumpers. This discussion is written from the viewpoint of converting the TRS-80 version into the 8-bit parallel version. In order to convert back from the 8-bit parallel version, the reader need only reverse this procedure.

A. Circuit Card Modifications

Before performing this modification, the MiniMover-5 should be unplugged from the host computer and disconnected from the 12-volt power supply.

The printed circuit card is accessible by removing the four rubber feet from under the base. Once the base has been opened, the user will find the card as shown in Figure 23. Although all of the circuit card components are TTL logic and relatively immune to static charge, care must be exercised to avoid damaging any of the components.

The procedure for modifying the interface card is as follows:

* Remove IC 17-This integrated circuit which is mounted in a socket to facilitate easy removal, controls the port address to which the MiniMover-5 responds. Since, in 8-bit parallel mode, the port address is determined by the particular microcomputer system the manipulator will be attached to, the port addressing must be disabled.
* Install jumpers-Without the integrated circuit (IC 17), the decoder (IC 16) will never be enabled. Thus, jumpers must be installed in place of IC 17 to enable the decoder. The jumpers to be installed are (1) IC 17 pin 14 to pin 8, and (2) IC 17 pin 7 to pin 3 and pin 6.

* Remove IC 19-This integrated circuit is a tri-state input buffer. Since, in 8-bit parallel operation, the inputs to the computer are on a separate input port rather than the bi-directional data bus of the TRS-80, this must be removed. This IC is also installed in a socket to facilitate easy removal.

* Install Jumpers-In order to make the input signals available at the ribbon connector, jumpers must be installed in place of IC 19. The jumpers are: pin 3 to 2, pin 5 to 4, pin 7 to 6, pin 9 to 10, pin 11 to 12, and pin 13 to 14.

* Check jumper J1-Connect E1 and E2 with J1.

The completed modifications are shown in Figure 46. This procedure allows A0-A2 to select the correct output latch, and makes the input signals available on the ribbon connector.

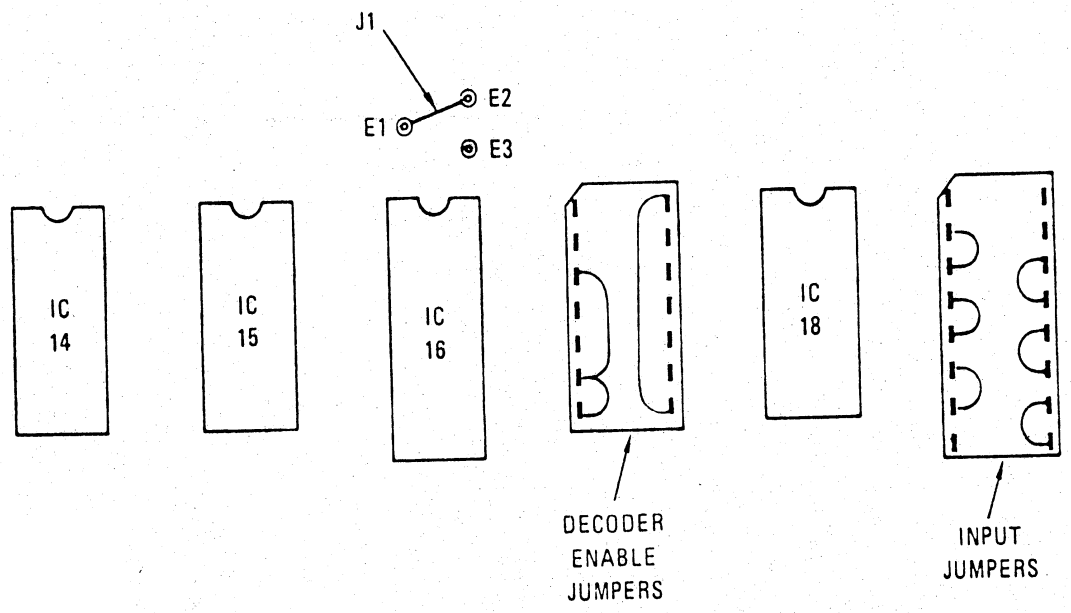


Figure 46 Jumpers Configured for Parallel Port

B. Attachment and Operation from a Parallel I/O Port

Proper connection of the MiniMover-5 to an 8-bit parallel I/O port is shown in Figure 47. Figure 48 shows the logical mapping of the bits of a typical 8-bit parallel output register to those of the MiniMover-5 interface. Figure 49 shows the MiniMover-5 ribbon connector and pin assignments.

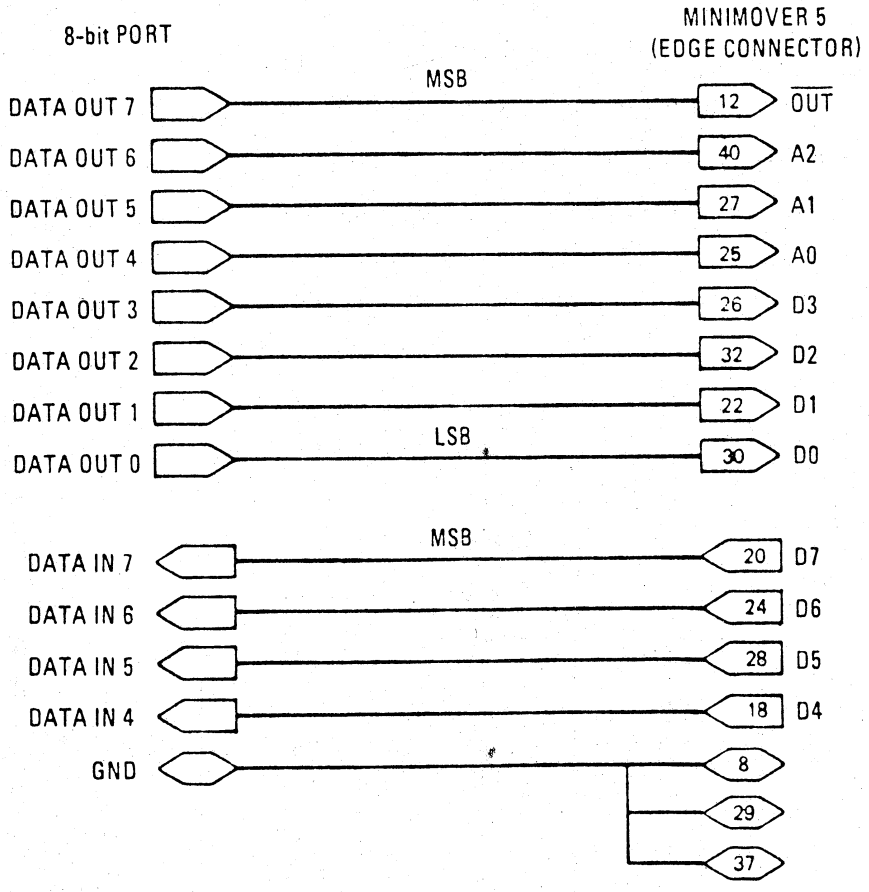


Figure 47 Connections for 8-bit Parallel I/O Port

The procedure for outputing data to a motor or port is as follows (see Figure 48):

* The driver software determines the particular 4-bit pattern which is to be output to a specific motor.
* The driver software places this pattern in bits 0, 1, 2, and 3 of a register which will eventually be output to the parallel port as shown in Figure 48.

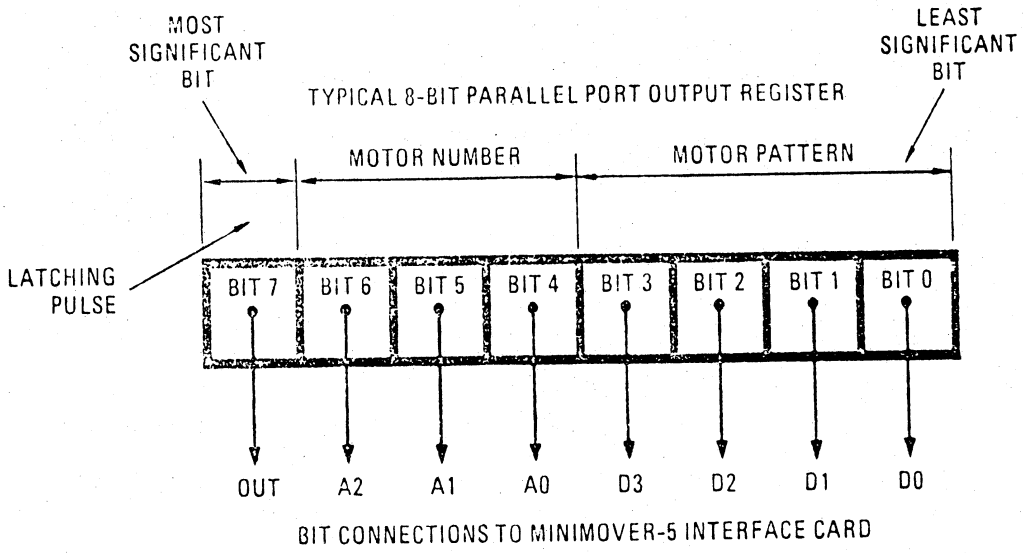
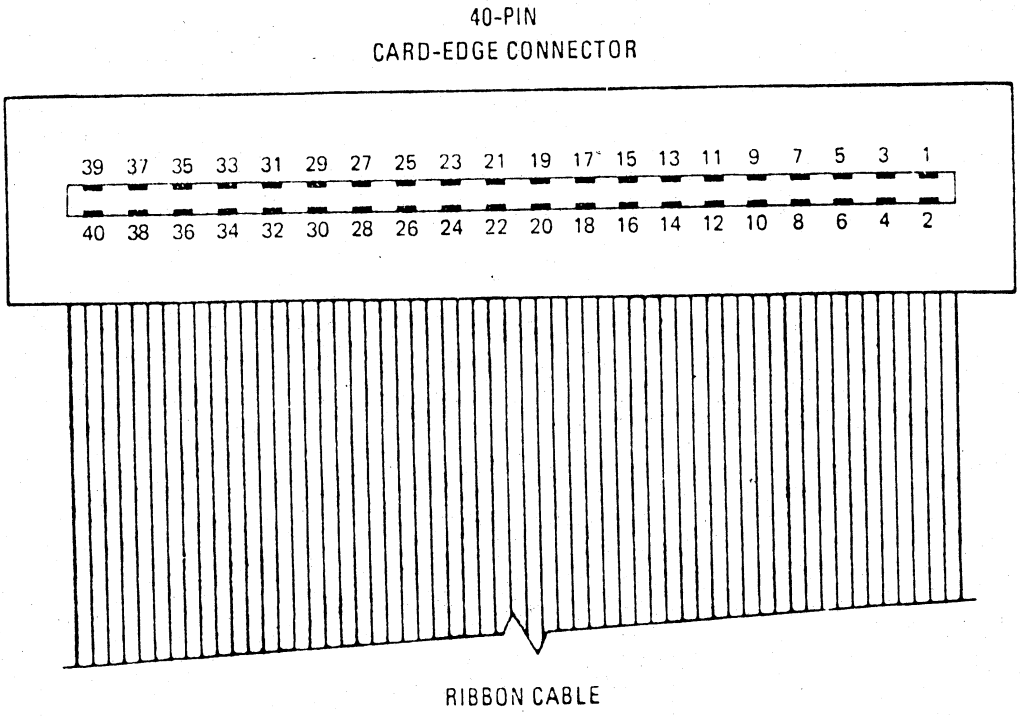


Figure 48 Logical Mapping of a Typical Parallel Output Register



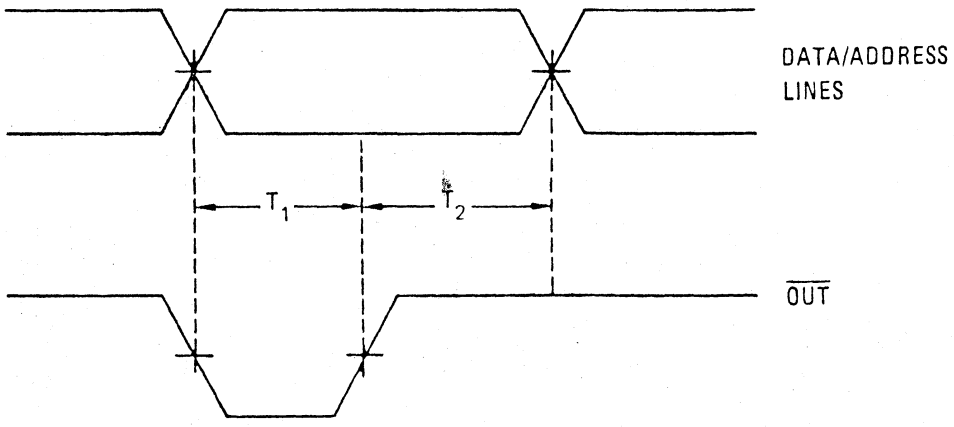
NOTE: Viewed looking into card-edge connector.

Figure 49 Pin Assignments for MiniMover-5

* The driver software clears the most significant bit of the register (latching pulse).
* The driver software places the number of the particular motor* to be driven in bits 4, 5, and 6 of the register.
* The register is output to the parallel output port**. This sets up the data bus and the address select lines.
* The most significant bit of the register should then be set and the register output again. This provides a positive going signal on the OUT line of the MiniMover-5. This will strobe the four bits of motor pattern into the latch addressed by motor number.

The procedure discussed here, and the description of the TRS-80 driver program in Section XI, are intended to aid the user in writing an assembly language motor driver program on their own computer. The driver should be written in assembly language to obtain acceptable speed. If slow speed operation is acceptable, then the driver can be written in a high level language such as BASIC.

The timing of the output signals is shown in Figure 50. Information on the data input lines may be read at any time.



Minimum Times: $T_1 > 90$ ns, $T_2 > 100$ ns

Figure 50 Output Timing Diagram

---

* Motor 1 (001) is the base and motor 6 (110) is the hand.

** Some parallel I/O ports are inverting. If this is the case, then the value of the register should always be complemented just prior to output.