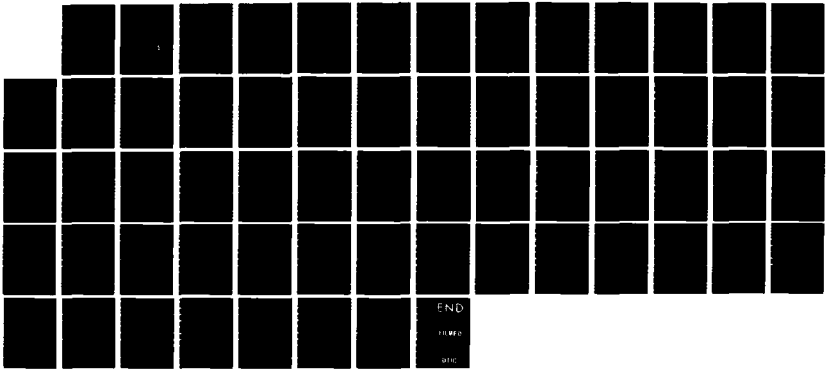
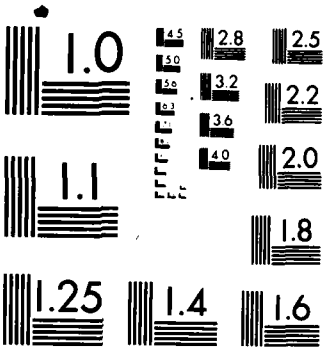
AD-A162 391 A MICROPROCESSOR SYSTEM FOR CONTROLLING THE OHIO STATE UNIVERSITY COMPACT (U) OHIO STATE UNIV COLUMBUS ELECTROSCIENCE LAB S PLEASANTS MAY 84 ESL-714190-6 1/1

UNCLASSIFIED N00014-82-K-0037 F/G 20/14 NL





(12)

OSU

The Ohio State University

A MICROPROCESSOR SYSTEM FOR CONTROLLING THE OHIO STATE UNIVERSITY COMPACT RANGE TARGET SUPPORT PEDESTAL

by

Shawn Pleasants

The Ohio State University

ElectroScience Laboratory

Department of Electrical Engineering
Columbus, Ohio 43212

Technical Report 714190-6

Contract N00014-82-K-0037

May 1984

Department of the Navy
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

AD-A162 341

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No.

4. Title and Subtitle

A MICROPROCESSOR SYSTEM FOR CONTROLLING THE OHIO STATE UNIVERSITY COMPACT RANGE TARGET SUPPORT PEDESTAL

5. Report Date

May 1984

6.

7. Author(s)

Shawn Pleasants

8. Performing Organization Rept. No

ESL 714190-6

9. Performing Organization Name and Address

The Ohio State University ElectroScience Laboratory
Department of Electrical Engineering
Columbus, Ohio 43212

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No

(C)

(G) N00014-82-K-0037

12. Sponsoring Organization Name and Address

Department of the Navy, Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

13. Type of Report & Period Covered

Technical

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

An advanced microprocessor-based stepper motor controller for the target support pedestal of the Ohio State University's compact backscatter measurement range has been constructed. The controller accurately controls target rotation and provides the target aspect angle to a host computer. Custom software for the controller implements a set of commands that provide variable rotational velocities and accelerations, as well as special stepping routines useful for the backscatter measurement process. The commands can be invoked remotely by a host computer or locally via a front panel keypad.

17. Document Analysis a. Descriptors

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement

19. Security Class (This Report) Unclassified

21. No of Pages 57

20. Security Class (This Page) Unclassified

22. Price

(See ANSI-Z39.18) See Instructions on Reverse

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

The Ohio State University ElectroScience Laboratory has developed a test object support pedestal as part of the compact radar backscatter measurement range. This project involves the development of a programmable, microprocessor based controller for the stepper motor in the pedestal assembly of the compact range. The purpose of this report is to describe the hardware and software development work done on the controller and to describe its use. The pedestal stepper motor rotates targets for radar cross section measurements. Custom software written for the controller provides programmable velocities and accelerations of the stepper motor. Radar test measurement techniques impose a limit on the velocity range of 0.1 to 9.9 degrees/second in 0.1 degree/second increments. Accelerations of 1.0 to 5.0 degrees/second^2 in 0.1 degree/second^2 increments are provided to help rotate heavy and large targets as well as to protect expensive targets from jerk when starting the motor. Specialized stepping routines were written for the radar test measurement process.

Three computer and interface circuit cards were purchased from Wintek Corporation for the construction of the controller. Advantages exist for buying manufactured boards. One is the elimination of hardware development and debug time. Another advantage is that Wintek offers a complete line of various boards and software that can be used to enhance the controller and tailor it to various applications. The three cards constitute a complete microcomputer system with a Motorola 6800 microprocessor, 4K of EPROM program memory, 1/2K of RAM memory, 40 parallel I/O lines, one serial I/O port configured for RS232 operation, a keypad for entering data and commands, 15 programmable LED's for displaying data, and one each A/D and D/A converter. The controller interfaces to a PDP 11/23 host computer through a RS232 communication link for computer control. A forty foot cable connects the controller to the pedestal. All drive circuitry for the stepper motor is contained in the pedestal itself. Four TTL level signals are sent to the motor for stepping. Located on the pedestal shaft is a shaft encoder that, through additional circuitry at the pedestal, produces 4000 pulses per shaft revolution with a direction line indicating clockwise or counter-clockwise rotation. Another signal produced by the shaft encoder gives one pulse per revolution. The 4000 pulses per revolution give 0.09 degrees resolution in shaft position. Chapter I describes in further detail the hardware and modifications and additions done to the Wintek system for this application.

As previously mentioned, the motor can step with speeds of 0.1 to 9.9 degrees/second and accelerations of 1.0 to 5.0 degrees/second^2.

INTRODUCTION

The Ohio State University ElectroScience Laboratory has developed a test object support pedestal as part of the compact radar backscatter measurement range. This project involves the development of a programmable, microprocessor based controller for the stepper motor in the pedestal assembly of the compact range. The purpose of this report is to describe the hardware and software development work done on the controller and to describe its use. The pedestal stepper motor rotates targets for radar cross section measurements. Custom software written for the controller provides programmable velocities and accelerations of the stepper motor. Radar test measurement techniques impose a limit on the velocity range of 0.1 to 9.9 degrees/second in 0.1 degree/second increments. Accelerations of 1.0 to 5.0 degrees/second^2 in 0.1 degree/second^2 increments are provided to help rotate heavy and large targets as well as to protect expensive targets from jerk when starting the motor. Specialized stepping routines were written for the radar test measurement process.

Three computer and interface circuit cards were purchased from Wintek Corporation for the construction of the controller. Advantages exist for buying manufactured boards. One is the elimination of hardware development and debug time. Another advantage is that Wintek offers a complete line of various boards and software that can be used to enhance the controller and tailor it to various applications. The three cards constitute a complete microcomputer system with a Motorola 6800 microprocessor, 4K of EPROM program memory, 1/2K of RAM memory, 40 parallel I/O lines, one serial I/O port configured for RS232 operation, a keypad for entering data and commands, 15 programmable LED's for displaying data, and one each A/D and D/A converter. The controller interfaces to a PDP 11/23 host computer through a RS232 communication link for computer control. A forty foot cable connects the controller to the pedestal. All drive circuitry for the stepper motor is contained in the pedestal itself. Four TTL level signals are sent to the motor for stepping. Located on the pedestal shaft is a shaft encoder that, through additional circuitry at the pedestal, produces 4000 pulses per shaft revolution with a direction line indicating clockwise or counter-clockwise rotation. Another signal produced by the shaft encoder gives one pulse per revolution. The 4000 pulses per revolution give 0.09 degrees resolution in shaft position. Chapter I describes in further detail the hardware and modifications and additions done to the Wintek system for this application.

As previously mentioned, the motor can step with speeds of 0.1 to 9.9 degrees/second and accelerations of 1.0 to 5.0 degrees/second^2.

Motor stepping is controlled by entering the desired speed and acceleration rates through the keypad. Software was written so that motor rotation can be predicted by the physics equations

$$v = at \text{ and } v^2 = 2as$$

where

v = velocity (degrees/second)
a = acceleration (degrees/second^2)
t = time (seconds)
s = distance (degrees)

The controller can operate in two modes: (1) local, where data and commands are entered through a 16 key keypad on the front panel and (2) remote, where a host computer transmits data and commands to the controller over an RS232 link. Twenty-one commmands are implemented on the controller. Six of them are for setting parameters for the stepping routines (speed, acceleration, position, terminal position, scan angle start, scan angle stop). Eight commands are used to choose one of four stepping routines (two commands, clockwise and counterclockwise, for each routine). The four routines are: (1) rotate to terminal angle and deaccelerate to stop, (2) jog motor one step, (3) rotate motor continuously until a stop command is entered and (4) scan from the current scan angle start to the scan angle stop. The last command steps the motor so that it is at full speed when passing through both the start and stop scan angles. The other seven commands include setting the mode for local or remote, clear entered data, start scan (used in conjunction with scan command), stop motor with deacceleration, change

LED display data to show all stepping parameters, send current position to host computer, and set a debug mode that is used when under RS232 control for helping to debug the controller programming. Chapter II discusses the software implementation of the commands while Chapter III gives more detailed information on the commands and how to enter them into the controller.

The controller has been completed and tested out on a properly equipped pedestal. All features presented in this report were tested. Controller operation in the manual mode was verified easily by entering data and commands through the keypad. Remote mode operation by programming through the RS232 interface was verified by using an Osbourne microcomputer as the host. All stepping routines control motor acceleration, constant velocity and deacceleration as specified.

CHAPTER I

HARDWARE ADDITIONS AND MODIFICATIONS

The microprocessor based programmable pedestal controller incorporates three boards purchased from the Wintek Corporation. The heart of the controller is a control card utilizing a Motorola 6800 µp. Four kilobytes of EPROM (Intel 2732) and one half of a kilobyte of RAM (four Motorola 6810's) comprise the program and scratch pad memory, respectively. Also on this card is an asynchronous communcation interface adaptor (a Motorola 6850 ACIA) which is used with some extenal circuitry to comprise a RS232 communication link. Forty programmable I/O lines are also provided with outputs on a fifty pin connector. The other two cards are an I/O card with a sixteen key keypad and fifteen LED's and an analog interface card with one 8-bit A/D converter and one 8-bit D/A converter. A forty-four pin, five slot back plane is used to connect the cards together. A block diagram of the system appears in Figure 1.1. Complete details on these cards can be found in the Wintek Micro Modules Reference Manual. Also, the chips on these cards are Motorola standards and information on them can be found in their data books. In the following sections is information on the modifications and hardware additions done to the cards to tailor them to this application.

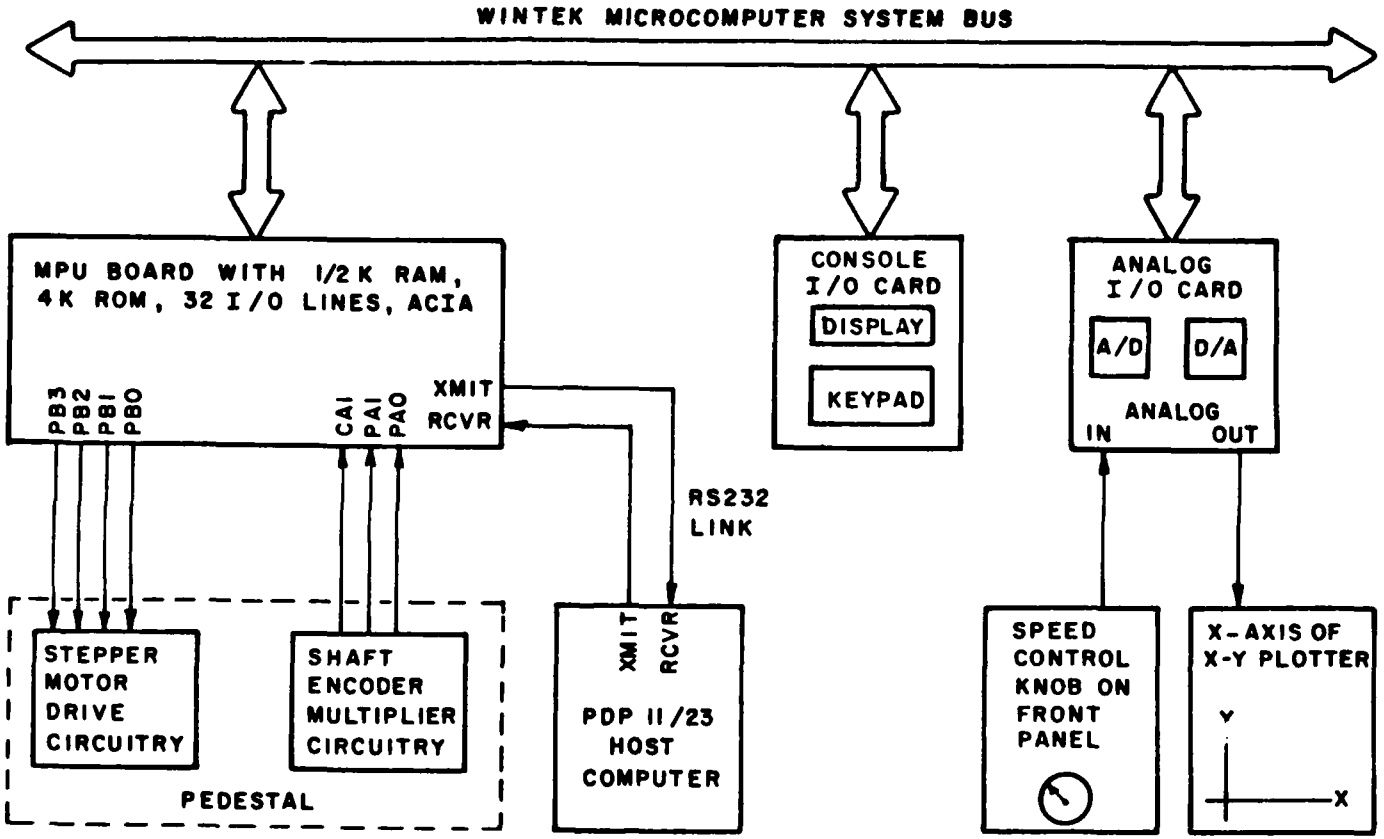


Figure 1.1. Block diagram of stepper motor controller showing host computer and pedestal interface.

A. MODIFICATIONS

Since the purpose of this project is to control a stepper motor, some sort of timing base is needed to create a constant time delay between motor steps. A problem arises here as the Wintek system has no timing base except a 1200 HZ clock that creates interrupts for refreshing the LED's on the I/O card. This doesn't provide enough resolution as the time delays needed must be accurate to the microsecond. The problem was rectified by replacing one of the Motorola 6821's that provides the parallel I/O lines with a Rockwell 6522 Versatile Interface Adaptor. This chip provides the same functions as the 6821 but has in addition two 16 bit internal timers that operate at the system clock frequency of one MHZ. Thus, time delays to the microsecond can be realized. The two chips are nearly pin for pin compatible except for differences on six pins. Therefore, a few printed circuit lines had to be cut and jumpers soldered to accomodate the 6522. Listed below are the pin differences between the two chips.

Pin	6821	6522
35	RS1	RS3
36	RS0	RS2
37	$\overline{\text{IRQA}}$	RS1
38	$\overline{\text{IRAB}}$	RS0
21	R/$\overline{\text{W}}$	$\overline{\text{IRA}}$
22	CS0	R/$\overline{\text{W}}$

Wintek addressing places the following address lines on the chip select pins of the 6821.

Pin	6821	Address Line
22	CS0	A 9
23	$\overline{CS2}$	A12
24	CS1	A 4

The following changes were made to accommodate the 6522.

1) The 6522 has two chip selects, CS1 (pin 24) and $\overline{CS2}$ (pin 23). This is the same as CS1 and $\overline{CS2}$ of the 6821. Therefore, A12 was left on $\overline{CS2}$ but A4 was cut from CS1 and A10 was jumped in its place. A4 is used for one of the register selects.
2) The 6522 has four register selects for its 16 internal registers while the 6821 only has two register selects to access its 4 internal registers. The table on the previous page reveals that RS1 (pin 35) and RS0 (pin 36) of the 6821 become RS3 and RS2 of the 6522, respectively. Thus, nothing was done to these lines. RS1 (pin 37) and RS0 (pin 38) of the 6522 correspond to $\overline{IRQA}$ and $\overline{IRQB}$ of the 6821. Therefore, these lines were cut from pins 37 and 38 and A2 and A4 were jumped in their place, respectively. This makes A2 correspond to RS1 and A4 to RS0.
3) The R/$\overline{W}$ line was removed from pin 21 of the 6821 and jumped to pin 22 of the 6522. The line removed from pin 22 in order to jumper in the R/$\overline{W}$ line is not used for anything.

4) The interrupt lines removed from pins 37 and 38 of the 6821 were jumped to pin 21 of the 6522, which was vacated due to change 3.

5) The enable signal from the processor was removed from pin 25 and the system phase two clock was jumped in its place. This was done because the timers in the 6522 need the uninterrupted system clock frequency for constant counting.

For this scheme to work, address line A9 must be a '1' when addressing the 6522 to avoid memory conflicts with the RAM. The following memory map for the 6522 results.

			$\overline{CS2}$		CS1						RS0		RS1	RS2	RS3
X	X	X	0	X	1	1	X	0	0	0	A	0	A	A	A
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

One other modification was done to the Wintek control module. The module is configured for an Intel 2708 EPROM, while we are using the Intel 2732 EPROM. Pages 8, 9 and 10 of the Wintek Micro Modules Reference Manual discuss the necessary printed circuit line cuts to be made and jumpers soldered in order to accomodate the 2732. However, this is not the real problem. Wintek permutes some of their address and data lines from the Intel standard pinout. Figure 1.2 a shows the pin differences between the two. The problem was corrected in hardware by making a small printed circuit board that reshuffles the address and data lines from Wintek to the appropriate Intel 2732 pins. A copy of the printed circuit layout appears in Figure 1.2 b. A standard 24 pin

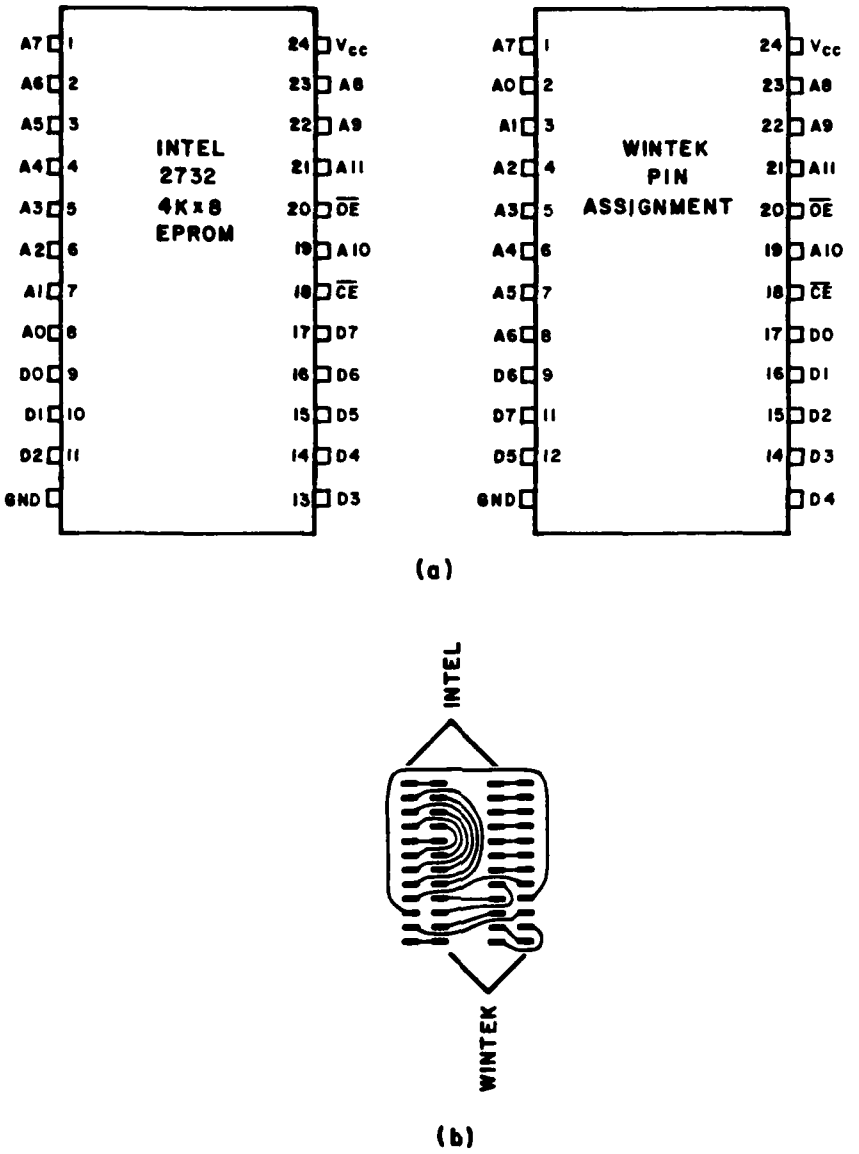


Figure 1.2. Pinout differences between Intel 2732 EPROM and Wintek assignment and printed circuit used to reshuffle appropriate pins.

solder tail socket was soldered in the pins marked Intel to accomodate the 2732. A 24 pin wire wrap socket was soldered into the pins marked Wintek and then the body was sniped off, leaving only the wire wrap pins. These pins are then plugged into the memory socket provided on the Wintek control module board. A wire wrap socket was chosen because the memory socket on the Wintek board requires long leads in order to make good contact.

B. HARDWARE ADDITIONS

An interface board was wire wrapped to provide drivers and receivers for the RS232 communication link and for the stepper motor interface cable. A block diagram showing board layout appears in Figure 1.3 and a circuit diagram is presented in Figure 1.4. A Motorola 1488 is the RS232 driver and Motorola 1489 is the receiver. Signals for these chips are supplied via a cable from a fourteen pin DIP connector on the control card that carries the necessary signals from the 6850 asynchronous communications interface adapter. Following is a list of the necessary connections to be made on the DIP socket and signal-pin correspondence.

- Connect baud clock to transmit and receive clocks (pin 2 to pins 11 and 12)
- Connect $\overline{\text{DCD}}$ and $\overline{\text{CTS}}$ to ground (pins 8 and 12 to either 3, 4 or 5)
- Transmit data (pin 9)
- Receive data (pin 13)

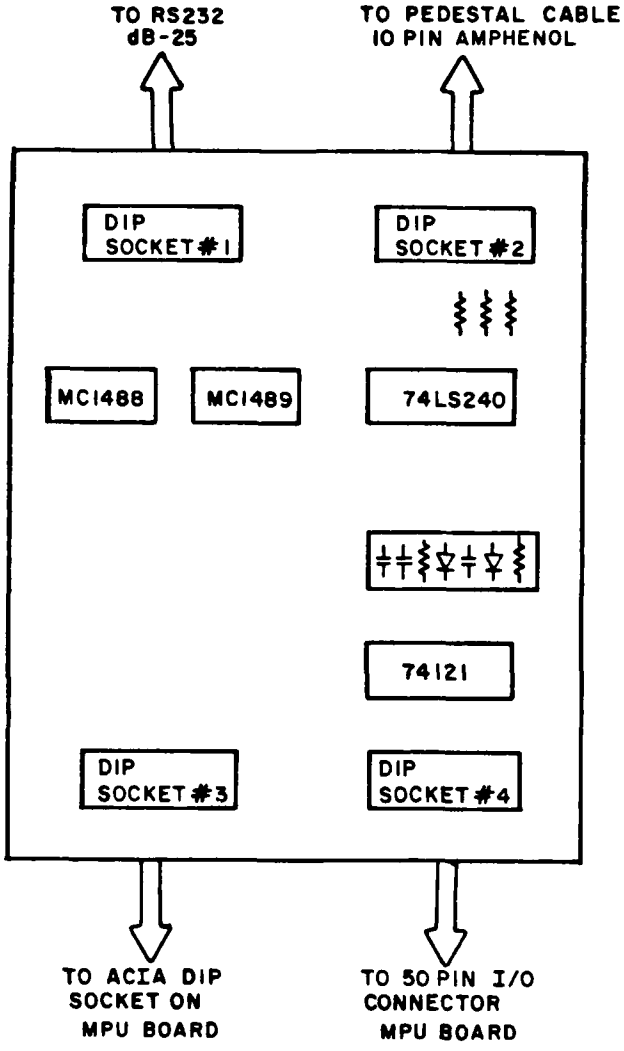
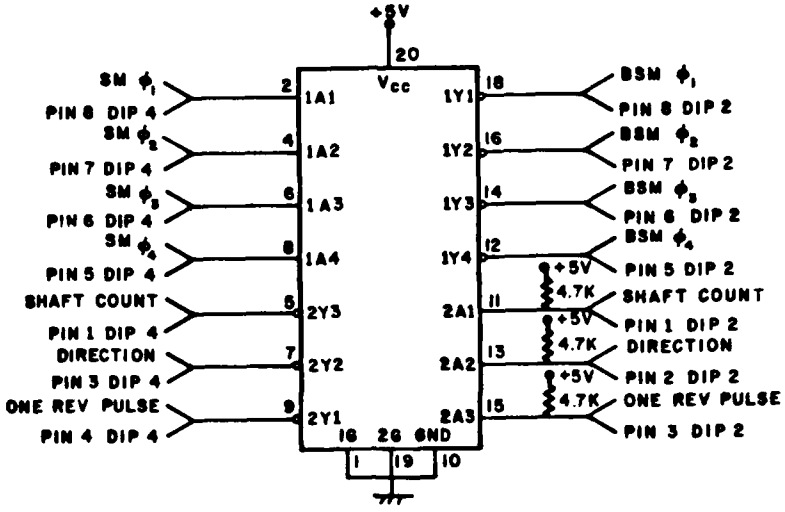
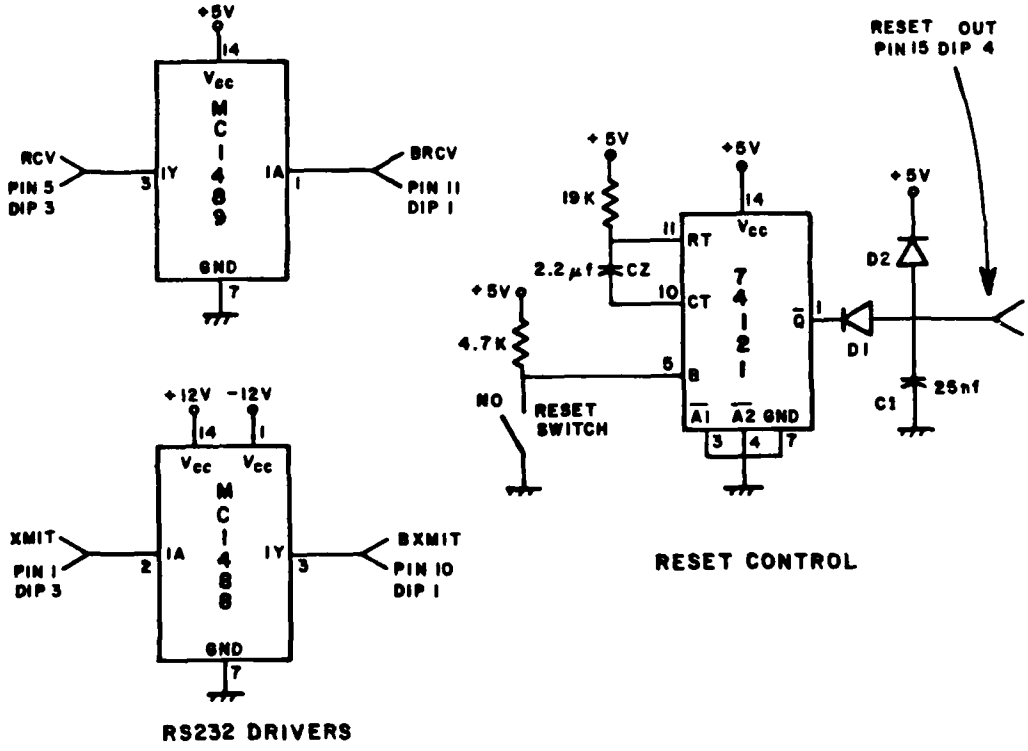


Figure 1.3. Interface board layout.



PEDESTAL CABLE LINE DRIVER/RECEIVER CIRCUIT



RS232 DRIVERS

RESET CONTROL

Figure 1.4. Schematic of circuits on interface board.

An on-board clock (555 oscillator) is used for the transmit and receive clocks of the ACIA and is set for 9600 HZ. The frequency can be adjusted by pot P1 on the control module. Data Carrier Detect ($\overline{DCD}$) and Clear To Send ($\overline{CTS}$) must be tied to ground to ensure proper operation of the ACIA. Transmit data from the ACIA is fed to the MC1488 driver and out to pin 2 of the RS232 connector. Receive data for the ACIA is acquired from pin 3 of the RS232 connector through the MC1489 receiver.

Wintek claims that baud rates of only 300 to 1200 can be used for ACIA operation. This is assuming the ACIA control register is set to divide the transmit and receive clock inputs by sixteen. The ACIA also can be set up for no clock frequency division, making possible higher baud rates. With this in mind, the on-board baud clock was set at 9600 HZ for a baud rate of 9600. This is the highest possible baud rate for the PDP 11/23 host computer.

A 74LS240 driver/receiver chip is used to interface the control module to the cable connecting the pedestal. This chip has high drive capabilities needed to drive the forty to fifty feet of cable. Three state outputs are permanently enabled. The drivers in the chip (a total of eight) are inverters but the receivers at the pedestal are also inverters (Schmitt triggered), nullifying the inversion. The four motor phases are outputted through port B bits 3 to 0 of the 6522 and to the 74LS240 drivers.

Three of the gates in the 74LS240 are used to receive three signals from the shaft encoder: (1) direction, (2) position pulses and (3) one revolution pulse. As mentioned before, these gates are inverters but

this time the drivers located at the pedestal (emitter follower amplifiers) do not invert. Thus, results of the inverted signals must be handled in software. The position pulse input is taken to CA1 input of the 6522, which is programmed to respond to edge transitions. One revolution pulse is input on CA2 of the 6522 and also programmed to respond to edge transitions. Direction is tied to PA7 of the 6522. When CA1 input sees an edge transition of a position pulse, it latches in the direction bit on PA7 until the microprocessor responds to the CA1 interrupt request.

Also on this interface board is a circuit to provide controller reset through a front panel push button. The circuit diagram also appears in Figure 1.4. The microprocessor responds to the reset by starting program execution at the address specified in locations FFFE H (high byte) and FFFF H (low byte) of the EPROM. A 74121 non-retriggerable one shot is used for the push button reset for two purposes: (1) to debounce the pushbutton and (2) to hold the reset line low long enough for the processor to recognize and initiate the reset operation. The 6800 requires the reset line to be low for a minimum of eight microseconds. The one shot delay of approximately 30 ms is plenty long enough to satisfy the reset requirement and ample time delay for switch debounce. The RC timing network holds the reset line low at power-on to reset the µp. Diode D1 isolates the effect of the output of the one shot on the RC network while D2 provides a short circuit to ground at power-down for discharging the capacitor.

A list of the pin signals on the four DIP sockets on the interface board and their connections is shown in Table 1.1. A ten conductor cable is used to connect the controller to the pedestal. Each conductor is a twisted pair with one of them used as a ground. In addition, each twisted pair has its own shield and the cable itself has a shield.

In addition to setting speeds through the keypad, an analog knob is present on the front panel. It has a detent position (when the knob is pointing straight up) that corresponds to zero velocity. Turning the knob clockwise rotates the pedestal clockwise and counterclockwise for the other direction. The knob is attached to a 25kΩ pot inside the controller. The center tap is taken through a shunt capacitor and series resistor to the analog input of the A/D convertor on the analog I/O board. The RC network delays transitions of pot output voltage in order to create acceleration and deacceleration. End terminals of the pot are tied to ground and five volts. The software is written to interpret zero to two volts as counterclockwise speeds and three volts to five as clockwise speeds. Voltages in the range of two to three volts are interpreted as zero velocity in order to create a symmetric dead zone around the detent position mentioned earlier.

An analog output signal to control the horizontal input of the X-Y plotter is available at the back panel through a BNC connector. It is made available from an eight bit D/A converter on the analog interface card. The voltage produced is proportional to the current position of the pedestal; zero degrees produces zero volts while 359.99 degrees corresponds to approximately 4.8 volts.

TABLE 1.1

WIRING LIST FOR DIP SOCKETS ON THE INTERFACE BOARD

DIP 1	FUNCTION	RS232 DB-25
10	B XMIT	2
11	B RCV	3
12	SIGNAL GND	7
	CHASIS GND	1

DIP 2	FUNCTION	PEDESTAL CABLE 10 PIN AMPHENOL
1	POSITION PULSE	H
2	DIRECTION	K
3	ONE REV PULSE	J
4	+12 V	A
5	BSM ϕ_4	G
6	BSM ϕ_3	F
7	BSM ϕ_2	E
8	BSM ϕ_1	D
9	SHAFT ENCODER GND	C
13	+12 V and MOTOR PHASE GND	B

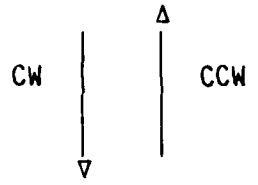
TABLE 1.1 (CONTINUED)

DIP 3	FUNCTION	ACIA CONNECTOR S2 ON MPU BOARD
1	XMIT	9
5	RCV	13
9	+5 V	1
11	GROUND	3
14	+12 V	6
15	-12 V	7

DIP 4	FUNCTION	I/O CONNECTOR S1 ON MPU BOARD
1	POSITION PULSE	10 (PIA U8 CA1)
3	DIRECTION	14 (PIA U8 PA7)
4	ONE REV PULSE	36 (PIA U8 CA2)
5	SM ϕ_4	28 (PIA U8 PB3)
6	SM ϕ_3	30 (PIA U8 PB2)
7	SM ϕ_2	32 (PIA U8 PB1)
8	SM ϕ_1	34 (PIA U8 PB0)
12	GROUND	2
13	+5 V	1
15	RESET OUT	TO PIN Z of BACKPLANE
16	RESET IN	N/O TERMINAL OF RESET SWITCH

C. PEDESTAL ELECTRONICS

The stepper motor located in the pedestal is a 200 step, four phase motor made by Slo-Syn (model # M092 FD09). In full step mode, the motor takes 200 steps per revolution, but this application uses a half step sequence that gives 400 steps per revolution, or 0.9 degrees/step. On the stepper motor output shaft is a 100:1 reduction gear box further reducing the distance per step to 0.009 degrees. The output of the gear box is attached to the pedestal shaft. The stepping sequence for the four phases is



Step	$\phi 1$	$\phi 2$	$\phi 3$	$\phi 4$
1	1	0	1	0
2	1	0	0	0
3	1	0	0	1
4	0	0	0	1
5	0	1	0	1
6	0	1	0	0
7	0	1	1	0
8	0	0	1	0
1	1	0	1	0

Mounted on the pedestal shaft is a Hewlett-Packard shaft encoder (model # HEDS-6010) that produces two phase quadrature signals of 1000 pulses per revolution. External circuitry at the pedestal multiplies the phase quadratures to produce a 4000 pulse per revolution position

signal, or 0.09 degrees/pulse. Direction is also determined by the external circuitry. Stepper motor drive circuit board layout is shown in Figure 1.5. Figure 1.6 shows the schematic of the motor drive circuitry and shaft encoder multiplier circuit. The four motor phases and three shaft encoder signals, along with power and ground, are transported on a ten conductor cable.

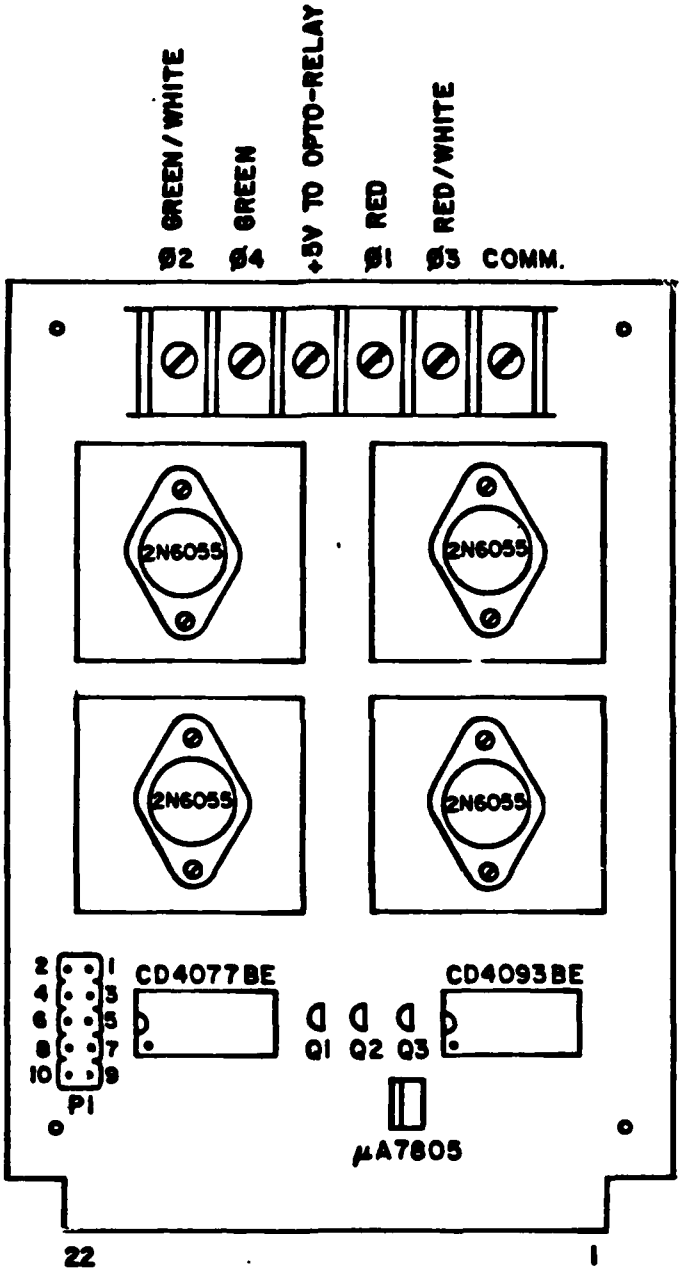


Figure 1.5. Stepper motor driver board layout.

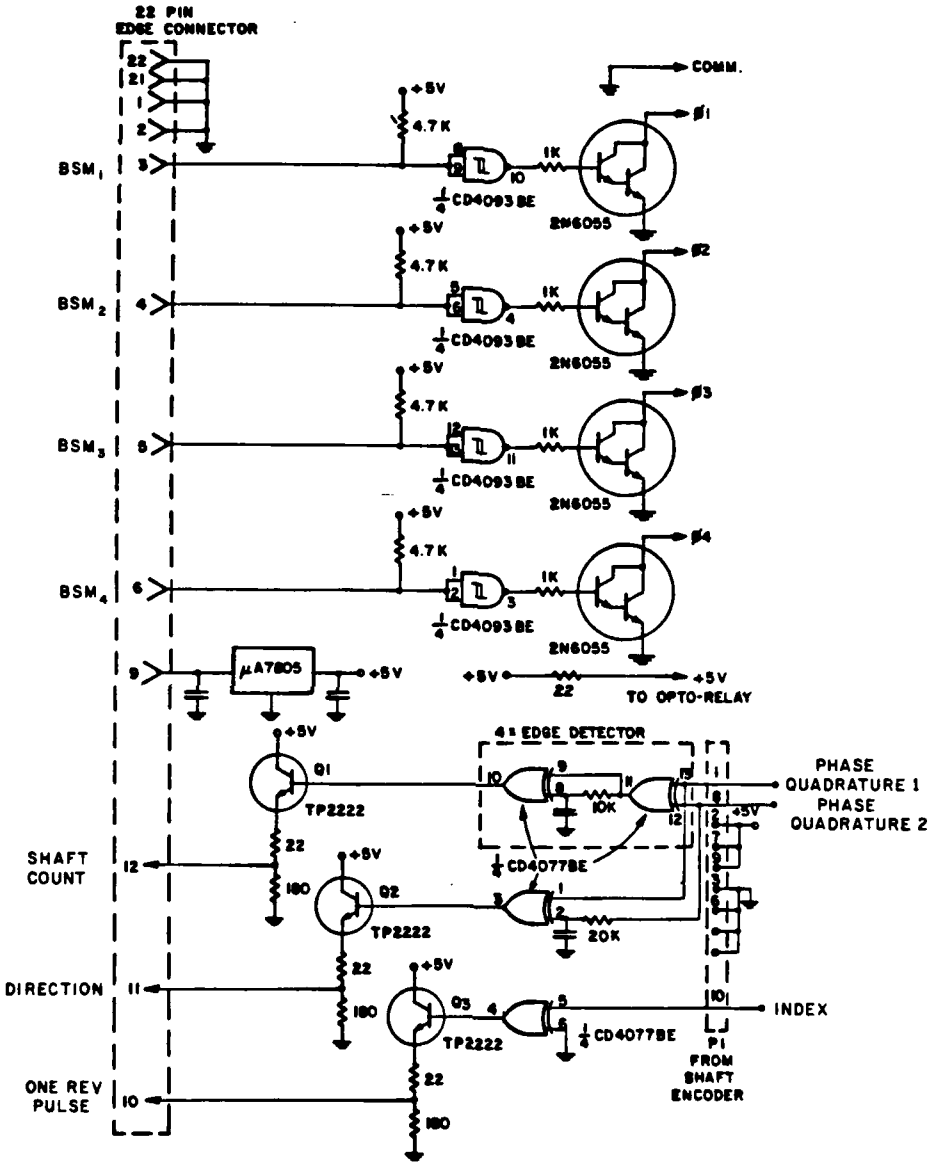


Figure 1.6. Schematic of stepper motor driver and shaft encoder multiplier ciruits.

CHAPTER II

SOFTWARE

Software development was done on a Hewlett Packard 64000 computer equipped with a 6800 microprocessor emulator. The 64000 is a general purpose computer that has an editor, assembler, linker and so on. Programs are written in the Motorola 6800 assembly language and assembled and linked into absolute code. This part of the computer is all very standard and provides no real advantage over other computers. The powerful tool of this computer is the emulator. The external part of the emulator seen by the user is a 40 pin "pod" which has the exact pinouts of the 6800 microprocessor. This pod plugs into the socket on the hardware where the microprocessor belongs, interfacing it to the 64000 computer. Programs written and resident on the 64000 can be run on the hardware as if the actual 6800 microprocessor was in the socket. The emulator also has extensive debugging capabilities for testing the programs. After being assured the programs are working correctly, an EPROM can be burned to dump the programs on the 64000. The processor and EPROM can then be plugged back into their sockets and the microprocessor system is operational.

A. ACCELERATION AND DEACCELERATION METHOD

The pedestal controller is capable of stepping the pedestal shaft at speeds from 0.1 degrees/second to 9.9 degrees/second in 0.1 degrees/second increments. The motor itself is actually stepping at 100 times these speeds due to a 100:1 gear reduction between the motor output shaft and the pedestal shaft. Time delay between motor steps is given for a few velocities in Table 2.1. The binary equivalent of the decimal values shown is the actual value loaded into the 16 bit timers used to create the time delay. The decimal time delays were calculated by the following equation.

$$\text{Time delay } \left(\frac{\text{second}}{\text{step}}\right) = \frac{1}{\left(\times \frac{\text{degrees}}{\text{second}}\right) \cdot \left(\frac{40000 \text{ steps}}{360 \text{ degrees}}\right)}$$

Accelerations of 1.0 degrees/second^2 to 5.0 degrees/second^2 were programmed to behave in accordance with the following physical laws:

(1) $v = at$

(2) $v^2 = 2as$

where

v = velocity (degrees/second)

a = acceleration (degrees/second^2)

t = time (second)

s = distance (degrees)

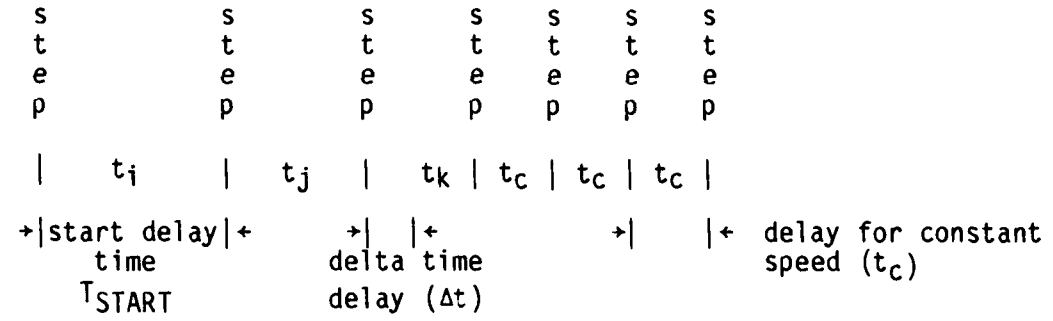
TABLE 2.1

NECESSARY TIME DELAY BETWEEN MOTOR STEPS
TO ACHIEVE CORRESPONDING VELOCITY

Velocity (degrees/second)	Time Delay Between Steps (μs/step)
0.1	90000
1.0	9000
2.0	4500
3.0	3000
4.0	2250
5.0	1800
6.0	1500
7.0	1285
8.0	1125
9.0	1000
9.9	909

Thus, with a given speed and acceleration input, the time necessary for acceleration and the distance travelled during acceleration can be calculated. The software calculates these values by solving for $t = v/a$ and $s = v^2/2a$. Then, the number of steps needed to travel the acceleration distance can be found (steps = s (degrees)/0.009 (degrees/step)).

Acceleration is accomplished by calculating a starting delay time between the first two motor steps and a constant delta time delay that is subtracted after each step from the previous step delay time until the constant velocity step delay is reached. By adding the delta time delay to the previous step delay, deacceleration is achieved. The graph below helps to relate the algorithm.



Delta time delay can be found from the following equations.

$t_i + t_j + \cdots + t_k = T$ Where T=v/a, the total time for acceleration.

but $t_i = t_c + \text{steps}\ (\Delta t)$ 'steps' is the number of steps needed to travel the acceleration distance.

$t_j = t_c + (\text{steps} - 1)\ (\Delta t)$

$t_k = t_c + \Delta t$

Then

$$[t_c + \text{steps}\ (\Delta t)] + [t_c + (\text{steps} - 1)(\Delta t)] + \cdots + (t_c + \Delta t) = T$$

$$\sum_{i=1}^{\text{steps}} t_c + \sum_{i=1}^{\text{steps}} i\ \Delta t = T$$

$$\Delta t \cdot \sum_{i=1}^{\text{steps}} i = T - \sum_{i=1}^{\text{steps}} t_c$$

The summation $\sum_{i=1}^{\text{steps}} i$ can be represented in equation form as $\frac{(\text{steps})\ (\text{steps} + 1)}{2}$. Also, it can be shown that $\sum_{i=1}^{\text{steps}} t_c$ is equal to T/2. Thus, the result is

$$\Delta t \cdot \frac{(\text{steps})\ (\text{steps} + 1)}{2} = T - T/2 = T/2$$

or

$$\Delta t = \frac{T}{(\text{steps})\ (\text{steps} + 1)}$$

This is the equation used in the software to determine Δt. T is first found from $T = v/a$ and steps is then calculated from knowledge of the distance traveled during acceleration (steps = s/0.009 where $s = v^2/2a$).

The starting delay time, T_{START}, between the first two steps is found now by the following equation.

$$T_{START} = \Delta t \cdot steps + t_c$$

The software initiates acceleration by jogging the motor one step and then loading the starting delay time, T_{START}, into internal timer 1 of the 6522 and initiates counting. While the present delay is counting down, the controller calculates the next delay value by subtracting Δt from T_{START} and loading the result into timer 1's latches. When timer 1 times out the starting delay, the 6522 is configured to transfer the delay contained in timer 1's latches into the timer and initiate counting again. Also when timer 1 times out, an interrupt is generated. The controller responds by jogging the motor one step and calculating the next delay value by subtracting Δt from the present delay value that is timing out in timer 1. The resultant value is loaded into timer 1's latches so that when the present delay times out, the next delay value is transferred from the latches to the timer and starts counting. This process continues until the controller calculates a next delay value that is less than the desired delay corresponding to the programmed velocity. At this point, the controller loads the constant velocity delay value into timer 1's latches and sets a flag indicating to the following interrupts to only jog the motor and not to calculate any new delay values.

Deacceleration is accomplished in a similar manner. The delta delay value, Δt, is added to the present delay value and the result is

stored in timer 1's latches as the next delay value. This process continues until the interrupt handler calculates a next delay value that is greater than the starting delay value T_{START}. At this point, depending on which stepping routine has been invoked, the motor is stopped completely or continues to step at the velocity corresponding to T_{START} until necessary to stop.

As an example, assume that the controller has been programmed for a velocity and acceleration of 5.0 degrees/second and 2.5 degrees/second^2, respectively. The acceleration time, T, is found as:

$$T = v/a = 5.0/2.5 = 2.0 \text{ seconds.}$$

Next, the distance travelled during acceleration is calculated.

$$v^2 = 2as \rightarrow \quad s = v^2/2a = (5.0)^2/(2 \cdot 2.5) = 5.0 \text{ degrees}$$

The value s is used in calculating steps, the number of steps necessary to travel the acceleration distance during the acceleration time. The distance travelled per motor step is 0.009 degrees. Thus,

$$\text{steps} = s/0.009 = 5.0/0.009 = 555.55\ldots \Rightarrow 555 \left(\begin{array}{l}\text{software routine}\\ \text{does integer division}\end{array}\right)$$

Now, the delta delay time Δt can be calculated.

$$\Delta t = \frac{T}{(\text{steps})(\text{steps} + 1)}$$

$$\Delta t = \frac{2.0}{(555)(556)}$$

$$\Delta t = 6.48 \Rightarrow 6\mu \text{ seconds} \left(\begin{array}{l}\text{again, software routine}\\ \text{does integer division}\end{array}\right)$$

This is the value that is either subtracted for acceleration or added for deacceleration to the present delay between motor steps to generate the next delay value between steps.

The multiply and divide routines are integer and are implemented in software. Therefore, values are scaled to represent decimal numbers. All values with units of time are represented in microseconds, with the least significant bit (LSB) being 1 μs. Time was represented in this manner because delays between motor steps for the velocity range must be resolvable to the microsecond (see Table 2.1). Also, the timer used to create the delays operates at 1 MHZ, or 1μs/clocking. Distance and position values are represented in units of degrees that are accurate to 0.01 degrees, with the LSB being equal to 0.01 degrees. This was done because the shaft encoder has a resolution of 0.09 degrees. With this in mind, for the example given, the binary representation in memory for T is 2,000,000 and s is 500. The value for Δt would be the binary value 6 since it is accurate to the microsecond.

Finally, the starting delay time is calculated as (shown using computer values)

$$T_{START} = \Delta t \cdot steps + t_c$$

$$= 6 \cdot 555 + 1800$$

$$= 5130\ \mu s$$

The acceleration and velocity ranges of the controller were chosen for several reasons. A simulation of the calculations done by the

controller to determine acceleration constants was performed on a VAX 11/780 for all combinations of velocity and acceleration in the ranges of 0.1 to 9.9 degrees/second and 0.1 to 9.9 degrees/second^2, respectively. A minimum of 1.0 degrees/second^2 was chosen because at accelerations and velocities below 1.0 degrees/second^2 and 1.0 degrees/second, the starting delay times become too large to represent in the 16 bit timer operating at 1 MHZ. Even with 1.0 degrees/second^2, velocities of 0.1 to 0.4 degrees/second still have exceedingly large starting delay times for the 16 bit timer. The controller compensates for this by dead starting the motor at these velocities irregardless of the programmed acceleration. Also, the combination of low accelerations and high velocities results in delta delay times that are less than 1μs, which is less than the accuracy of the timer. Even at 1.0 degrees/second^2 acceleration, velocities of 6.9 degrees/second and above have delta delay times of less than 1 μs. The controller is programmed to default the delta delay time to 1μs when this occurs. A ceiling of 5.0 degrees/second^2 was chosen primarily for practicality. There was no need to accelerate the motor any faster than this since slow velocities are used for taking measurements and to protect expensive targets. For instance, a velocity of 1.0 degrees/second and an acceleration of 9.9 degrees/second^2 results in an acceleration time of 0.1 seconds. This hardly provides any acceleration at all, defeating the need for acceleration completely.

The velocity maximum of 9.9 degrees/second was chosen for some of the same reasons as above. Since the primary interest is in slow velocities for data taking and to protect expensive targets, fast

velocities are not necessary. However, the faster velocities of 5 to 10 degrees/second provide fast enough rotation of the pedestal so that excessive time delays are not encountered in positioning the target for data taking. For example, a velocity of 9.9 degrees/second requires approximately 36 seconds to rotate the target 360 degrees while a velocity of 0.1 degrees/second requires one hour for one revolution of the target.

B. SOFTWARE ORGANIZATION

The main software routine, STARTMAIN, shown in flow chart form in Figure 2.1, begins with a one time initialization pass for memory locations and the I/O devices. The main body of the routine simply reads the keypad or RS232, depending on the mode. If any data are found, a subroutine is called to determine if the data represent a number or a command. If a number is found its value is stored and the program returns to reading the input device. When a command is entered, the main routine calls a subroutine that executes the desired command. Any data entered prior to the command is used, if necessary, by the command subroutine. After execution of the command subroutine, control is returned back to the main routine and polling of the input device resumes until data is entered again. This method provides flexibility of the command keys, should they desire to be changed, by simply changing the subroutine call corresponding to that key. Data values from zero to nine can be input and twenty-one commands are implemented. Chapter III discusses the commands and their syntax.

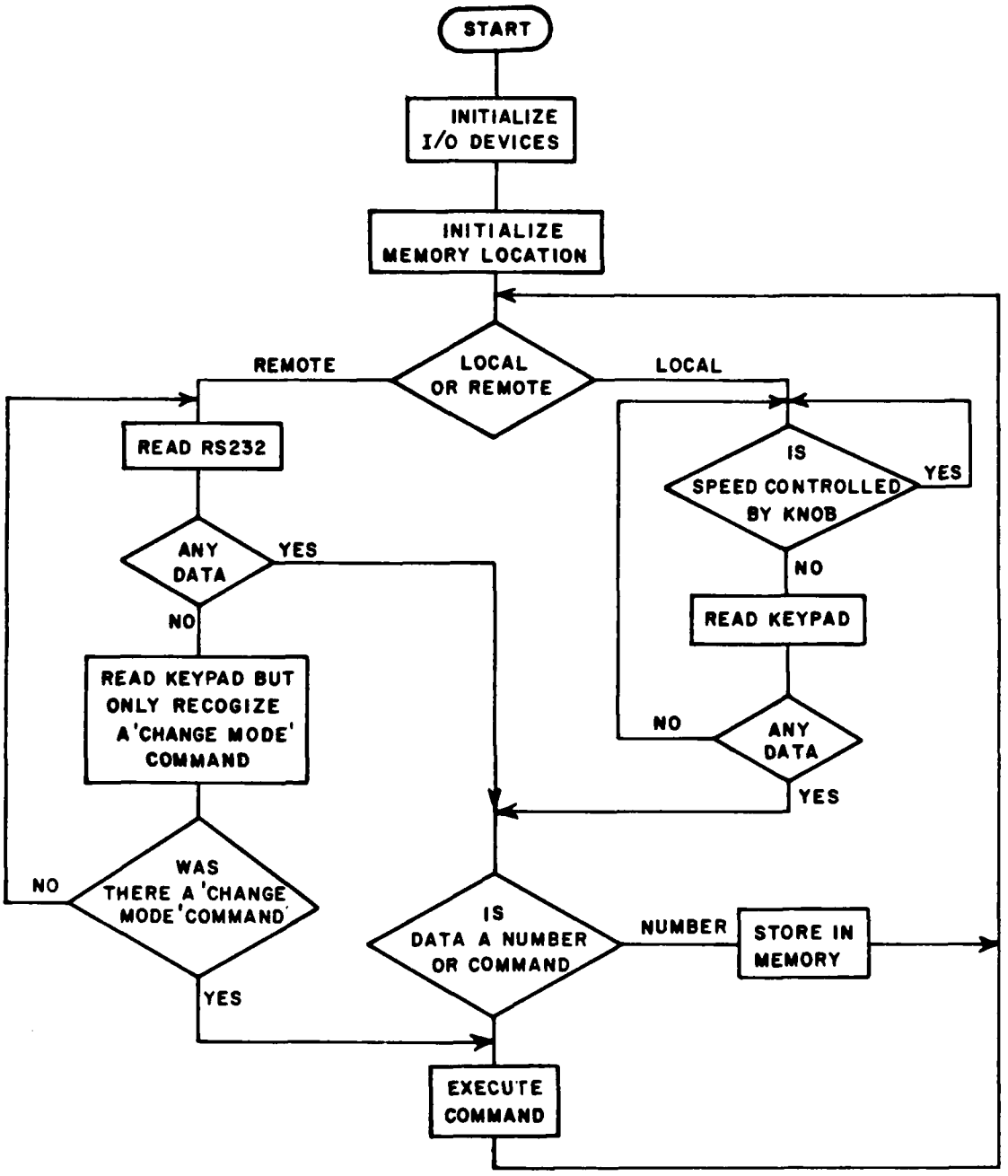


Figure 2.1. Flow chart of main routine STARTMAIN

C. SOFTWARE DEFINITION OF COMMANDS

A list of the commands and their associated keys along with the subroutine name that implements the command is found in Table 2.2. The program listings are well commented so only a brief overview is given here. The listings are not included with the report because of their length but are available at the ElectroScience Laboratory.

Six commands are used for setting parameters used by the stepping routines. The commands are SET SPEED, SET ACCELERATION, SET POSITION, SET TERMINAL POSITION, SET SCAN ANGLE START, and SET SCAN ANGLE STOP. These commands use data that were entered prior to the execution of the command. If any of these commands are invoked inadvertantly without any data having been entered, the subroutine will return without effecting the present value. SET SPEED and SET ACCELERATION store the entered data in binary at ten times the actual value. That is, 0.1 degrees/second is stored as a binary 1 and 9.9 degrees/second as binary 99 (same for acceleration). Commands for setting angles result in a binary number equivalent to 100 times the entered value. Thus, angles are accurate done to 0.01 degrees. This was done because the shaft encoder has a resolution of 0.09 degrees.

The five following commands require no data input. In fact, for the first two commands listed, if data are mistakenly entered before the command is invoked, the subroutine will return without execution of the command. The commands are SET MODE, SET DEBUG, STOP, CLEAR and CHANGE DISPLAYS. SET MODE toggles the controller between local or remote operation. SET DEBUG toggles the RS232 debug mode on and off. Details

TABLE 2.2

LIST OF THE COMMANDS RECOGNIZED BY THE CONTROLLER, THEIR ASSOCIATED KEYS, AND SOFTWARE ROUTINES THAT IMPLEMENT THE COMMANDS

KEY	COMMAND	SUBROUTINE NAME
0-9	data 0-9	---
A	set angle	SET_ANGLE
B	set terminal angle	SET_TANGLE
C	clear data	CHANGE_PNT
D	set mode	SET_MODE
E	set speed	SET_SPEED
F	shift	SHIFT

TABLE 2.2 (CONTINUED)

KEY	COMMAND	SUBROUTINE NAME
Pressing the shift key first will redefine the keys for the next key stroke only as follows:		
0	go to terminal angle CW	TANGLE_CW
1	go to terminal angle CCW	TANGLE_CCW
2	scan CW	SCAN_CW
3	scan CCW	SCAN_CCW
4	go continuously CW	CONT_CW
5	go continuously CCW	CONT_CCW
6	jog one step CW	JOG_CW
7	jog one step CCW	JOG_CCW
8	start scan	STARTSCAN
9	send position over RS232	SEND_POS
A	stop with deacceleration	STOPMOTOR
B	set debug mode	SET_DEBUG
C	set scan angle start	SCAN_START
D	set scan angle stop	SCAN_STOP
E	set acceleration	SET_ACCL
F	change display data	CHANGE_DIS

on the debug mode are given in Section F. STOP calls a subroutine that stops the motor with deacceleration. If the motor is not rotating, execution of the command does nothing. When invalid data is entered, CLEAR can be executed to cancel the data. It clears the display of entered data and returns back to the regular display information. Since all stepping parameters cannot be displayed on the fifteen LED's at once, two sets of led display data are kept in memory. CHANGE DISPLAYS toggles between the two sets of display information.

D. STEPPING ROUTINES

Four different stepping routines are programmed into the controller. They are: (1) rotate continuously until a STOP command is entered, (2) rotate to the current terminal angle and deaccelerate to a stop, (3) jog motor one step and (4) scan pedestal from the current scan angle start to the scan angle stop. This last command will rotate the motor so that it is at full speed when traveling through the scan start and stop angles. Eight commands are associated with these four routines, one each for clockwise and counterclockwise rotation. A nineth command, STARTSCAN, is used in conjunction with the scan routines. When the scan command is given, the motor will rotate from its current position to a second position and stop that will give the motor enough distance of travel during acceleration in order to be at

full speed when it passes through the scan angle start. The STARTSCAN command starts the motor from this position for scanning.

All of these routines, with the exception of JOG, start the motor with acceleration and stop it with deacceleration, except for speeds of 0.1 to 0.4 degrees/second, which are programmed to always start and stop with no acceleration. Before the motor is started, subroutines DET_DIST and DELTIME are called to determine the stepping parameters of starting delay time T_{START} and delta delay time Δt. Subroutine STARTMOTOR is then called which loads the starting delay time into the timer 1 counter of the 6522 and initiates counting. The first delta delay time is subtracted from the starting delay time and the result is loaded into timer 1 latches. When timer 1 times out the present delay, it loads the value from its latches into the counter and starts counting down again. From this point, motor stepping is interrupt driven. When the timer times out, an interrupt is generated. The controller responds to the interrupt by jogging the motor one step and then calculating the next delay value by subtracting the delta delay time from the present delay value and loading the result into timer 1's latches. This process continues until the next delay time calculated by the interrupt handler is less than the delay for the programmed constant velocity. When this occurs, the delay for constant velocity is loaded into timer 1's latches and all interrupts after this just step the motor and return.

Since the stepping routines are interrupt driven, the subroutines that initiate stepping are free to do other things. While the motor is

rotating, the three routines, excluding JOG, monitor the keypad or RS232 port (depending on the mode) for data input but the only commands responded to are STOP and CHANGE DISPLAYS. All others are ignored. When a STOP command is found, the motor is deaccelerated to a stop and the subroutine returns back to the main program. This is the only responsibility of the rotate continuous subroutine. The terminal angle and scan routines have one other important responsibility. Since these routines must stop the motor at a specific terminal angle, a deacceleration position must be known where the motor starts deaccelerating so that it will stop at the terminal angle. This position is calculated by subroutine DET_DEAPOS before the motor is started. While the motor is rotating, the subroutines check the current position to see if it equals the deacceleration position. When this condition is found, the routines set a flag for the interrupt handler to start deacceleration. After this, while the motor is stepping at the velocity corresponding to T_{START}, the routines will check the terminal position against the current position and when found equal stop the motor completely and return to the main program.

E. POSITION ENCODER

A Hewlett Packard shaft encoder is mounted on the pedestal shaft near the stop of the pedestal. It produces two phase quadrature signals of 1000 pulses per revolution. External circuitry multiplies the two phase quadratures to get 4000 pulses per revolution, or 0.09 degrees/pulse, and a direction bit. This information is what the

controller receives. The position clock is connected to CA1 of the 6522 and the direction bit to PA7. CA1 is set to produce an interrupt on a negative edge transition. The interrupt handler responds to the position clock interrupt by calling subroutine READ_POSITION which reads the direction bit to determine whether to increment or decrement the position value in memory. If the direction bit is high, INCPOS is called to increment the position value by 9 (LSB equals 0.01 degrees) or if the direction bit is low, position is decremented by 9 by routine DECPOS.

The D/A converter on the analog I/O module that drives the position axis of an X-Y plotter is updated at this time. It is an 8 bit D/A converter but the position is represented with 16 bits (again, LSB = 0.01 degrees, so 359.99 degrees is stored as 35999 binary, or 8C9F hex). Thus, the 16 bit position has to be scaled to 8 bits. It is done in the following manner. The 8 bi. converter has 256 possible codes to represent 360 degrees, or 1.41 degrees per code. Therefore, the 16 bit position value is divided by 141 (scaled by 100 since LSB = 0.01 degrees) and the integer result is used as the D/A code.

Also when a position interrupt occurs, the new position value is sent to the host computer via the RS232, no matter the controller mode. A total of four 8 bit bytes are sent. The first two bytes are the same, A0 hex, and indicate that the following two bytes are the position. The third byte is the most significant and the last is the least significant byte of the position data.

F. PROGRAMMING THROUGH THE RS232

To program the controller through the RS232, the host computer must send data in the exact sequence as if programmed through the front panel keypad. For example, suppose it is desired to set the velocity to 2.5 degrees/second. The programming sequence would be:

1) Send binary equivalent of 2 in first byte.
2) Send binary equivalent of 5 in second byte.
3) Send SET VELOCITY command (binary 14, or hex E, since key corresponding to SET VELOCITY is E).

After each of the above data transmissions, the controller will return a two byte code indicating that the data was accepted, processed properly, and the host computer can send more data. This code is the same for both bytes and is DO hex. Should the command that is sent invoke a stepping routine, the controller does not send this two byte code until it has completed the stepping routine and the motor has come to a complete stop. This is done so that the host computer knows when the controller has completed the stepping routine and returns to polling the RS232 channel. At this time more data can be sent. During the stepping routine all data received through the RS232 is ignored except for the STOP command or CHANGE DISPLAYS command. This allows the host computer to stop the motor or change the displayed stepping parameters at any time.

If an error is detected or invalid data is received by the controller during transmission, it will return a one byte error code to

the host computer corresponding to the error. Following is a list of errors and their associated error codes.

Error	Code (hex)
overrun error	E0
framing error	E1
invalid data	E2
parity error	E3

In the event of an error, the host computer must decide what action to take. The controller does nothing other than sending a code to identify the error. Valid data received from previous transmissions before the error is not lost.

A debug mode is programmed into the controller to aid in debugging it when programming. The command SET_DEBUG invokes this mode. Executing the command again turns off the debug mode (i.e., SET_DEBUG toggles the debug mode). If an overrun, framing, or parity error is detected, the same error code as sent back to the host computer is displayed on LED's 1 and 2. In the event that invalid data (data greater than 0F hex) is received, the value will be stored directly on LED 2 as a seven segment code, along with an E on LED 1. The seven segment code displayed on LED 2 then corresponds to the invalid data received, thus identifying it. When the debug mode is off, no error codes or invalid data are shown on the LED's but error codes are still sent to the host computer.

G. AVAILABLE MEMORY AND MEMORY MAPPED I/O

Four kilobytes of program memory (Intel 2732) is available on the control module. All but approximately one hundred bytes have been used. Scratch pad memory consisting of 512 bytes of Motorola 6810 RAM is also available with less than half being used. The stack also occupies this memory starting at location FF hex and working down. All I/O devices in the system (keypad, LED's, A/D and D/A converters, parallel I/O) are memory mapped through Motorola 6821's. Memory maps of the RAM, ROM, and I/O devices can be found in the Wintek Reference manuals, with the exception of PIA U8, which was replaced by the 6522. The new memory map used for the 6522 was discussed in Section A of Chapter I. All variables used in the programs are listed in table 2.3 along with the program name where the variable is defined and its associated address in RAM.

TABLE 2.3

ALL VARIABLES DEFINED IN THE PROGRAMS AND THEIR CORRESPONDING MEMORY LOCATION IN RAM

ADD RESS (HEX)	LABEL	ROUTINE
00	KEYCODE	READKEY
01-1F	BUFDATA	LED_DATAO
10	BUFENDDATA	LED_DATAO
11-1F	BUFENTRY	LED_ENTRY
20	BUFENDENTRY	LED_ENTRY
21-22	BUFPNT	DISPLAY
23-24	BUFSTART	DISPLAY
25-26	BUFEND	DISPLAY
27	MODE	SET_MODE
28-2A	KEY	DECIPHER
2B	KEYEND	DECIPHER
2C-2D	KEYPNT	DECIPHER
2E	KEYCNT	DECIPHER
2F	SHIFT	SET_SHIFT
30	INTRCNT	INTRHAND
31-32	POINTER	INDEX
33	OFFSET	INDEX
34-36	POSITION	SET_ANGLE
37-39	T_POSITION	SET_TANGLE
3A-3C	SCPOSTART	SCAN_START
3D-3F	SCPOSTOP	SCAN_STOP

TABLE 2.3 (CONTINUED)

ADD RESS (HEX)	LABEL	ROUTINE
40	DIRECTION	JOG
41-42	STEPPNT	JOG
43	DELAY_H	SET_SPEED
44	DELAY_L	SET_SPEED
45	COUNT	SHOW
46	LEDPOS	SHOW
47	LEDOFFSET	SHOW
48-49	WHICH_BUF	SHOW
4A-58	BUFDATA1	LED_DATA1
59	ENDDATA1	LED_DATA1
5A	WHICH_DIS	CHANGE_PNT
5B	STEP_CNT	INTRHAND
5C-5D	MINUEND	SUBT_BIN
5E-5F	ADDER	ADD_BIN
60-61	SCRATCH	READ_POSITION
62-63	POS_BIN	POS_BIN
64-65	TPOS_BIN	TPOS_BIN
66-67	SCST_BIN	SCAN_START
68-69	SCSP_BIN	SCAN_STOP
6A-6B	RESULT	BCDTOBIN
6C	LOAD_CNT	DIS_POS
6D	ACCORDEA	STARTMOTOR
6E	STEPAGAIN	STARTMOTOR

TABLE 2.3 (CONTINUED)

ADD RESS (HEX)	LABEL	ROUTINE
6F	LASTKEY	READ
72-73	DEAPOS_BIN	DET_DEAPOS
74-75	COMPARED	CMP_BIN
76-77	SWITCH	CMP_BIN
78	OKTOSCAN	SCAN_CW
79-7A	DIFF	STARTSCAN
7B	NUMREV	STARTSCAN
7E-7F	SAVETPOS	SCAN_CW
82-85	DIVIDEND	DIVIDE
86-89	REMAIN	DIVIDE
8A-8D	DIVISOR	DIVIDE
8E	DIVIDE_CNT	DIVIDE
8F	DIG_COUNT	DIS_POS
90	VEL1	SPEEDKNOB
91	VEL0	SPEEDKNOB
92	SPEED1S	SHOWSPEED
93	SPEED10THS	SHOWSPEED
94-95	DELTA_POS	DET_DEAPOS
96	OKTOACCL	ACLORNOACL
97-98	TESTDIFF	ACLORNOACL
99	ACCELERATION	SET_ACCL
9A-9B	DISTANCE	DET_DIST
9C	MULT_CNT	MULTIPLY

TABLE 2.3 (CONTINUED)

ADD RESS (HEX)	LABEL	ROUTINE
9D-9E	MULTIPLIER	MULTIPLY
9F-A0	LOW_ANS	MULTIPLY
A1-A2	MULTIPLICAND	MULTIPLY
A3-A4	STEPS	DELTIME
A5-A6	DELTA_TIME	DELTIME
A7-A8	START_TIME	DELTIME
A9-AA	COUNT_TIME	DELTIME
AB	DEBUG	SET_DEBUG
AC-AD	SAVE	DET_DEAPOS
AE-AF	STOP_TIME	DEL_TIME
B0-B1	POS_BUF	READ_POSITION
B2	INDEX_POS	INTRHAND

CHAPTER III

COMMAND DESCRIPTIONS

Twenty-one commands are implemented on the controller. Clockwise rotation is defined when looking straight down on the pedestal from above. Following is a description of each command and how to enter it.

Data 0-9: Keys marked 0-9 represent that decimal number. When a number is pushed, present data on the screen will be cleared and the new data will appear on the blank screen. A total of 3 numbers can be entered at once. Any more numbers entered after the first three will be ignored. Values for acceleration and velocity are entered without the decimal point. That is, a value of 1.0 degrees/second is entered as 10 (no decimal point). Angles are entered as whole numbers only from 0-359.

A - SET POSITION: Sets the current position to the entered data. If no data has been entered the command returns without changing the current position value. Care must be taken as values greater than 360 can be entered.

B - SET TERMINAL POSITION: Sets the terminal angle with the entered data. If no data is entered, the command returns without changing the current terminal angle. Care must be taken as values greater than 360 can be entered.

C - CLEAR: Clears any data entered and resets the display back to the original display information.

D - SET MODE: Toggles the mode of the controller back and forth between remote and local. The new mode is reflected on the first LED; 0 for local and 1 for remote. While in remote this key is still functional in order to set mode back to local. All other keys are disabled.

E - SET SPEED: Sets the speed to the entered data. If an invalid velocity, or no velocity is entered, the command returns without execution or effecting the current speed value. Speeds of 0.1 to 9.9 degrees/second are possible in 0.1 degrees/second increments. To enter 0.1, a 0 is entered first followed by a 1. There is no decimal point. A velocity of 9.9 degrees/second would be entered as 99.

F - SHIFT: This key relabels the 16 keys with different commands on the next key stroke only.

F 0 - ROTATE TO TERMINAL ANGLE CLOCKWISE: Rotates motor from the present position to the current terminal angle in the clockwise direction. If the current position equals exactly the terminal angle, the motor will still rotate one full revolution back to the terminal angle. If the position is within the tolerance of ± 0.04 degrees of the terminal angle, but not equal to it, the pedestal will not rotate and the command will return.

F 1 - ROTATE TO TERMINAL ANGLE COUNTERCLOCKWISE: Same as rotate to terminal angle CW but direction is CCW.

F 2 - SCAN CLOCKWISE: This command sets up the motor for the actual scan. It does this by rotating the motor from the current position to a terminal position that will allow the motor, when given the start scan command, to accelerate to full speed before it passes through the scan start angle. The motor will rotate to this terminal position in the direction that provides the least distance. After the motor has reached the position and stops, the direction is set for CW. The STARTSCAN command must then be given to start the scan.

F 3 - SCAN COUNTERCLOCKWISE: Same as SCAN CLOCKWISE but direction is counterclockwise. Also, the controller uses the SCAN ANGLE STOP for the scan starting angle and SCAN ANGLE START for the scan stopping angle.

F 4 - ROTATE MOTOR CONTINUOUSLY CLOCKWISE: Starts the motor with acceleration and rotates it continuously in the clockwise direction until a STOP command is entered.

F 5 - ROTATE MOTOR CONTINUOUSLY COUNTERCLOCKWISE: Same as rotate continuously CW but direction is counterclockwise.

F 6 - JOG CLOCKWISE: Jogs motor one step clockwise.

F 7 - JOG COUNTERCLOCKWISE: Jogs motor one step counterclockwise.

F 8 - STARTSCAN: This command starts the motor with acceleration from the current position and direction set up by the commands SCAN CW or SCAN CCW. The motor will be at full speed when passing through both the SCAN ANGLE START and SCAN ANGLE STOP, at which time it will deaccelerate to a stop. If SCAN CCW had been previously invoked, the motor actually rotates from SCAN ANGLE STOP to SCAN ANGLE START.

F 9 - SEND POSITION: Sends current pedestal shaft position to the host computer. Four bytes are sent; the first two (A0 hex) signify the following data is the position and the last two are the position.

F A - STOP MOTOR: Stops motor with deacceleration from current speed. This command can still be entered from the keypad while in remote mode.

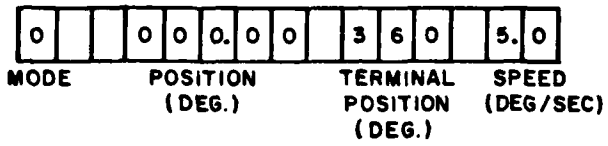
F B - SET DEBUG MODE: Toggles the debug mode on and off for RS232 operation. For a description of the debugger, see Section H of Chapter II.

F C - SET SCAN ANGLE START: Sets the scan angle start with the current entered data. If no data has been entered, the command returns without effecting the current value. Care must be taken as angles greater than 360 can be entered.

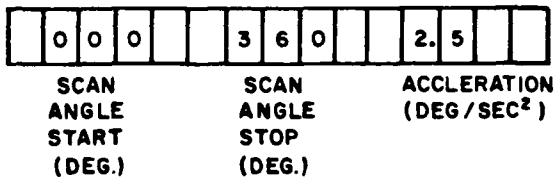
F D - SET SCAN ANGLE STOP: Sets the scan angle stop with the current entered data. If no data has been entered, the command returns without effecting the current value. Care must be taken as angles greater than 360 can be entered.

F E - SET ACCELERATION: Sets acceleration with the current entered data. If an invalid acceleration or no acceleration is entered, the command returns without effecting the current acceleration. Values from 1.0 degrees/second^2 to 5.0 degrees/second^2 can be entered in increments of 0.1 degrees/second^2. No decimal point is provided so a value of 2.5 degrees/second^2 must be entered as 25.

F F - CHANGE DISPLAY DATA: Toggles back and forth between two sets of data displayed on the LED's. The data displayed at power-on is shown below.



The other set of data that can be displayed with this command is:



These are power-on initialization values.

END

FILMED

1-86

DTIC