BNL-43328

A VERSATILE STEPPING MOTOR CONTROLLER FOR SYSTEMS WITH MANY MOTORS

Shuchen K. Feng, D. Peter Siddons
Brookhaven National Laboratory, Upton, NY 11973

BNL--43328

DE90 001556

ABSTRACT

A versatile system suitable for controlling beamlines or complex experimental setups is described. The system as currently configured can control up to 32 motors, with all motors capable of full speed operation concurrently. There are 2 limit switch inputs for each motor, and a further input to accept a reference position marker. The motors can be controlled via a front panel keyboard with display, or by a host computer over an IEEE-488 interface. Both methods can be used together if required. There is an "emergency stop" key on the front panel keyboard to stop the motion of all motors without losing track of the motors' position.

1. INTRODUCTION

A common difficulty experienced by researchers in setting up an X-ray or VUV synchrotron radiation beamline is that of providing a large number of remotely controlled actuators. A popular solution has been to use stepping motors controlled by a computer system at least partly dedicated to that purpose. Using commercially available motor control interfaces can result in a system which is bulky, inflexible and expensive. This is particularly true for systems not based on the CAMAC standard. Typical non-CAMAC stepper controllers provide three or four axes of motion per interface, and are more suited to machine tool control than to beamline automation. One of the highest density controllers currently available, the E500,[1] controls up to 8 motors in a single-width CAMAC module. Its high density is however only available if the user already owns a CAMAC crate and interface controller. In any case, this unit provides rather rudimentary manual control capabilities, and requires the presence of a computer to be useful. Our requirement was for a control module which could (a) control a sufficient number of axes for a typical beamline, (b) be capable of operation even without a supervisory computer, and (c) be readily interfaced to any of a variety of computer systems in use at the NSLS. In addition, it was felt essential that selected groups of axes should be capable of running simultaneously and pseudo-synchronously to facilitate the alignment of, for example, mirror tanks or motorized tables. The MC32 module described here has been developed at NSLS in an attempt to satisfy all of the above criteria. A prototype unit is currently in use on beamline X27A of the NSLS and five further units are under construction for use on various other beamlines. Fig. 1 shows a photograph of the prototype unit.

This article describes the current implementation of the module and its proposed future enhancements.

2. FEATURES AND SPECIFICATIONS

The MC32 is a high density, extremely flexible module that controls incremental motors and other pulse-driven devices. For each channel, this controller supplies a direction bit and a pulse-train output in negative true logic at TTL levels. The following is a list of the MC32 features and specifications.

1 Optional remote (IEEE-488) and manual control
2 Abort motion capability
3 Maskable Serial Request (SRQ) generation
4 3 Limit-switch inputs for each motor: upper,lower, and reference position
5 32 independent channels, one for each motor
6 Single step capability
7 Programmable start/peak/home rate (245 pps to 3125 pps).
8 Programmable acceleration/deceleration distance (4 pulses to 8160 pulses)
9 Maximum move per instruction (24 bits : 16,777,216)
10 Load/go function for different motors to move concurrently.
11 Programmable software +/- limits
12 Readout of remaining number of pulses after movement
13 Automatic completion for remaining pulses after abort
14 Error status, position readout
15 Position calibration
16 Inexpensive

Once initialized, the absolute position of each channel is maintained throughout all operations.

A move operation may be prematurely terminated by either remote control or manual control.

Maskable SRQ capability is provided for all channels. If requested, a SRQ is generated when any motor stops moving for any reason.

Inputs for three limit switches can be provided as either normally closed or normally open. The activation of a limit switch causes the selected motor to perform a decelerating stop. The status of the limit switches is readable by either the remote or manual method.

The LOAD/GO function provides the capability of moving a selected group of motors concurrently and repeatedly.

The *output status* byte reports various system errors, including the 3 limit switch states, SRQ status, motor busy flag, command error, motor enable/disable, and motion aborted state. If the command error bit is set, an error number may be read which indicates the particular error.

3. HARDWARE/FIRMWARE DESIGN

The hardware design is based on the use of a commercially available custom-programmed single chip microcontroller [2], specifically programmed to control a stepping motor. One such microcontroller is dedicated to each motor. Thus, a full system contains 32 such microcontrollers (MC's). The microcontroller was equipped with an interface compatible with common 8-bit microcomputer systems. The major design task was thus to provide a simple microcomputer system whose sole function was to direct the operation of the 32 MC's and to provide whatever input/output (I/O) facilities were needed. The I/O requirements were (a) an IEEE-488 computer interface, (b) a keyboard and display interface for stand-alone application and (c) a serial communication port (RS232) to allow the use of a remote hand-held terminal. To maximize the speed and flexibility of the system, the I/O should be event-driven as much as possible, and so a multiple interrupt capability should be provided.

Ths system is partitioned into several subsystems, each occupying a standard-size card (double Eurocard). The CPU and I/O circuits occupy one card (CPU card), and eight MC's occupy another (controller card). A future development will provide possible encoder readout facilities on a third card. The maximum capacity of the enclosure is five cards, of which one must be the CPU card. Thus, up to 32 motors (4 cards) may be controlled. The system can be configured to use fewer cards if necessary. Each card plugs into a backplane which carries the CPU data, address and control signals.

The microprocessor chosen for this system is the Rockwell 65C02. The choice was mainly based on one of the author's prior experience of hardware design using this family of devices. This choice also determined the memory and I/O architecture, since the 65C02 uses only memory-mapped I/O. The I/O functions were all implemented using standard LSI integrated circuits except for the display. The display was a commercially available liquid crystal module containing all drive electronics. The interface to the module was also compatible with the 65C02 bus. Four 28-pin JEDEC-compatible memory chip sockets are provided which can carry EPROM or RAM chips of various sizes, up to 32K bytes. The system memory map is programmed into a small PROM, so that re-configuration is simply a matter of re-programming.

The 65C02 family does not have an interrupt controller, so the Intel 8259 was used. Although this device is closely matched to the Intel 8080, it can be adapted to provide a vectored interrupt capability for the 65C02. This adaptation consists simply of providing two distinct chip-select lines to the devices, one for programming the vectors, priority scheme etc., and a second, connected to the INTA line, to read out the vectors after an interrupt has occurred under software control. The INT line from the 8259 is inverted and passed to the CPU. On receipt of an interrupt, the CPU executes a short routine which reads the vector to the appropriate service routine from the 8259, and then branches to that location. The 8088 'CALL' instruction which is also issued by the 8259 is discarded. This system only provides eight prioritized interrupts, which means that some interrupts are shared among several devices.

The least-used devices, in terms of interrupt generation, are the motor MC's themselves. Each block of eight motors therefore shares an interrupt line. The question of which MC among the eight generated an interrupt is answered by providing a read-only register which passes the current state of all eight MC interrupt lines to the CPU upon request. Therefore, the interrupt service routine for each block must first poll this register to decide which MC in the block to service.

On the front panel, there are a 4 row display unit and a keypad. The 4 row unit displays (1) Instruction : To instruct user what to do next, (2) Response Message : To respond to user's action, (3) Keyin character + Error Message : To echo user's keyin character and display error message if error occurred, and (4) Menu : A menu with a selection of functions F1, F2, F3 and F4, which are controlled by the 4 keys under the menu display. The keypad consists of these 4 keys, plus 10 keys for digits 0-9, a '-' key for a minus sign, a '.' key for a floating period, a 'DELETE' key to permit correction to keyboard input, an 'Enable/Disable Front Panel keyboard' key to prevent unauthorized front panel access, and an 'Emergency Stop' key to abort motor motion. Fig. 2 shows the front panel design.

On the back panel, there are a reset button to reboot the operating system, an IEEE-488 connector from the CPU card and 3 signal connectors from each controller card. The 3 signal connectors are (1) pulse and direction outputs for each of 8 motors. (2) CCW limit + CW limit inputs for each of 8 motors. (3) reference point + motor enable/disable control inputs for each of 8 motors. The connectors all are 20-pin, 0.1 inch pitch, ribbon cable headers, and connectors (1) and (2) are E500 compatible. Fig. 3 shows the back panel design of the MC32.

4. SOFTWARE DESIGN

The ROM chips inside the MC32 contain the machine code compiled from a C program using an interface library.Since the ROM chips are Erasable/Programmable, any new feature can be added in easily as needed. The C program mainly receives the commands or data from the MC32 input devices and then displays and operates the corresponding actions through the MC32 output devices.

The code was generated using a commercial C language cross-compiler [3]. This compiler runs under a standard PC operating system, called the host system, but generates executable code for a different microprocessor, called the target system, in our case the 65C02. This code was then downloaded into a low-cost single-board in-circuit emulator over a serial link. The in-circuit emulator allowed the execution of the code using the unmodified hardware of the prototype system, but at the same time retaining control over the execution to allow debugging. When debugging was completed, the code was programmed into EPROM memories, and these memories, together with a normal microprocessor were inserted into the prototype instead of the in-circuit emulator. This procedure allowed relatively quick debugging with a modest investment in development hardware. The use of the C language for the uppermost layers of the software allowed sophisticated facilities to be provided relatively straightforwardly. The time-critical lower layers of the software (for example the I/O drivers and interrupt handlers) were coded in assembly language to maximize their efficiency. This rudimentary operating system supports standard I/O (per C specifications) to devices, but not to files, since no file oriented hardware is present. The prototype system resides in 30K bytes of ROM, and makes use of 6K bytes of RAM, mainly for I/O buffers and the motor parameter database.

In the operation of the MC32, the two user interactions are through: (1) the front panel keypads and display, and (2) the remote computer entry of the input and output.

The front panel operations are based on a menu tree. The ROOT is the main menu which contains 4 functions: (1) Setup, (2) Select Active Motor Number, (3) Motor Motion and (4) Display motor status. Fig. 4 shows the front panel menu flowchart.

Table 1 shows the current IEEE-488 command set. They are divided into four categories : (1) Parameters setup commands - to set up any parameter like flag, acceleration, etc., (2) Interrogative commands - to interrogate data like status, position, error number, etc., (3) Motor motion commands - to move motors, (4) Other execution commands like LOAD/GO and Abort motion.

5. FUTURE DEVELOPMENT

The major enhancement which will be implemented in the near future will be an interface to incremental position encoders. Facilities for quadrature inputs or step/direction inputs will be provided with a 32-bit word length per channel. This should prove adequate for most purposes. It is envisaged that a density of eight channels per card will be straightforward, with the possibility of 16 per card as a preferred goal.

6. ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy under contract DE-AC02-76CH00016 with Associated Universities, Inc.

REFERENCES

[1] *E500A Stepper Motor Controller*, available from DSP Technology Inc., Fremont, C.A., U.S.A.

[2] *Stepper Motor Control LSI : PPMC 101C/102A*, available from Sil-Walker America, Pacific Palisades, C.A., U.S.A.

[3] *Aztec CG65, Cross Development Software for 65xx-based Systems*, available from Manx Software Systems, Inc., Shrewsbury, N.J., U.S.A.

FIGURE CAPTIONS

1. A photograph of the prototype unit now in use on beamline X27A of the NSLS.
2. MC32 front panel controls and displays
3. Back panel design of the MC32
4. MC32 front panel menu flowchart

TABLE 1

ALPHABETICAL COMMAND SUMMARY

Command	Description
Annnn	setup Acceleration/deceleration
C0	Complete all motors with remaining steps from last move
C1	Complete the active motor remaining steps from last move
D	Disable local mode except emergency stop and key mask
E	Enable local mode (keypad control)
Fnnn	setup Flag status
G	Go (start executing move for load/go)
H	go Home
IA	Interrogate Acceleration/deceleration value
IE	Interrogate command Error message number
IF	Interrogate motor Flag status
IN	Interrogate active motor Number
IO	Interrogate active motor Output status
IP	Interrogate active motor current Position
IR	Interrogate active motor remaining steps
IU	Interrogate active motor start velocity
IV	Interrogate active motor peak Velocity
IX	Interrogate active motor CW (-) limit
IY	Interrogate active motor CCW (+) limit
IZ	Interrogate active motor go home velocity
K	Kill active motor motion
L	Load motor move parameters
M+/-nnnnnnnn	Move to position +/-nnnnnnnn steps
Nnnn	select active motor Number
P+/-nnnnnnnn	calibrate Position (32 bits long)
R+/-nnnnnnnn	move +/-nnnnnnnn steps (Relative move)
S1	Single move in CCW (+) direction
S0	Single move in CW (-) direction
T1	move to CCW (+) limiT
T0	move to CW (-) limiT
Unnn	setup start velocity
Vnnn	setup peak Velocity
X+/-nnnnnnnn	setup CW (-) limit
Y+/-nnnnnnnn	setup CCW (+) limit
Znnn	setup go home velocity

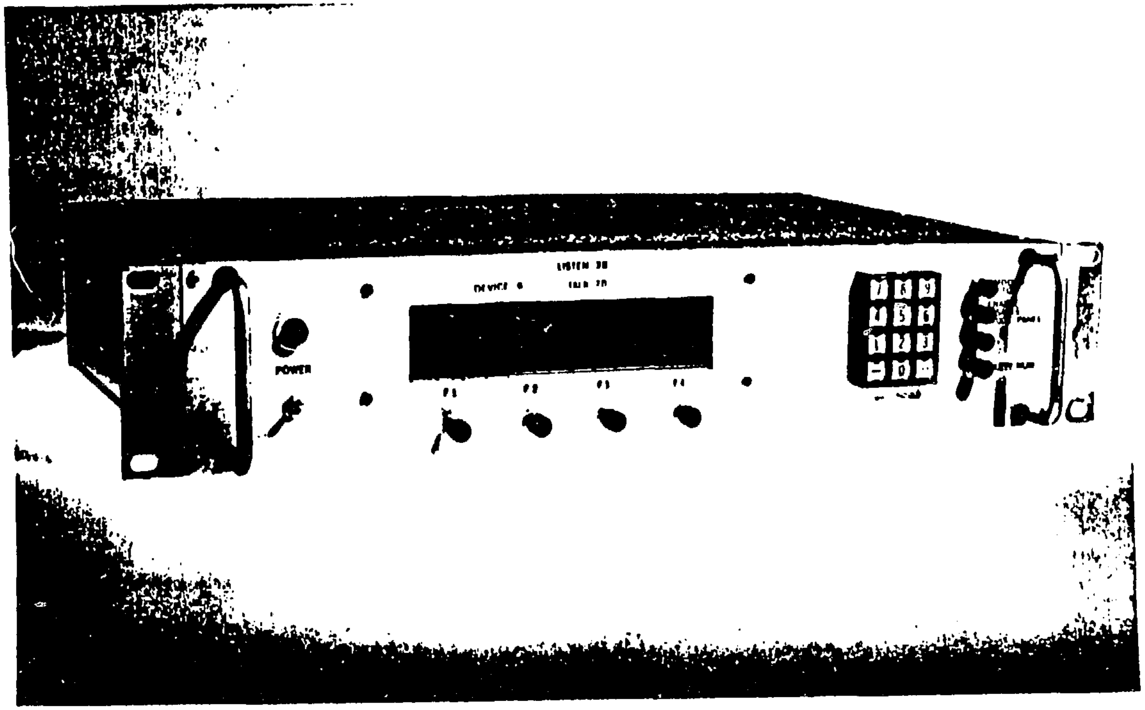


Figure 1

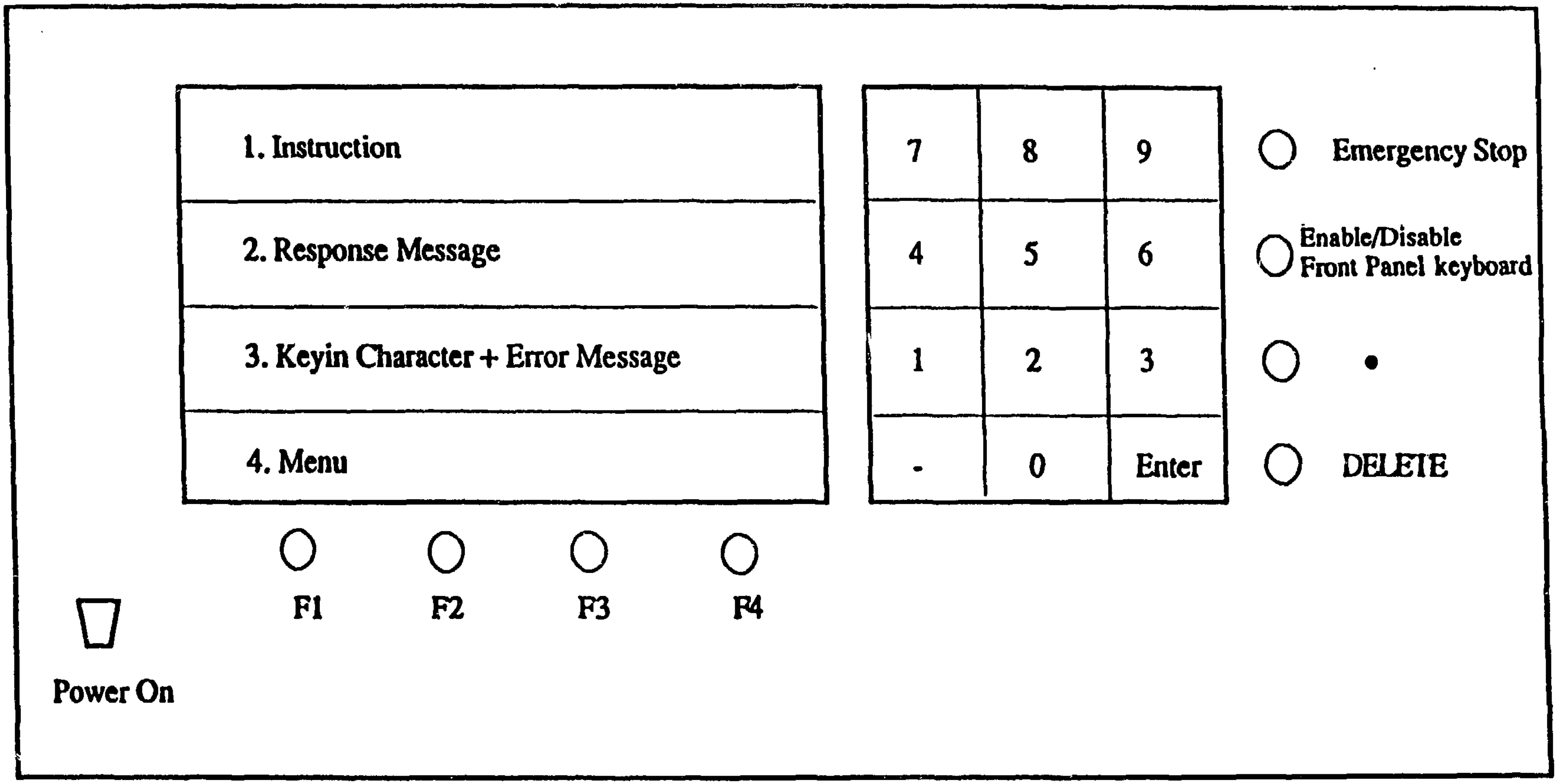
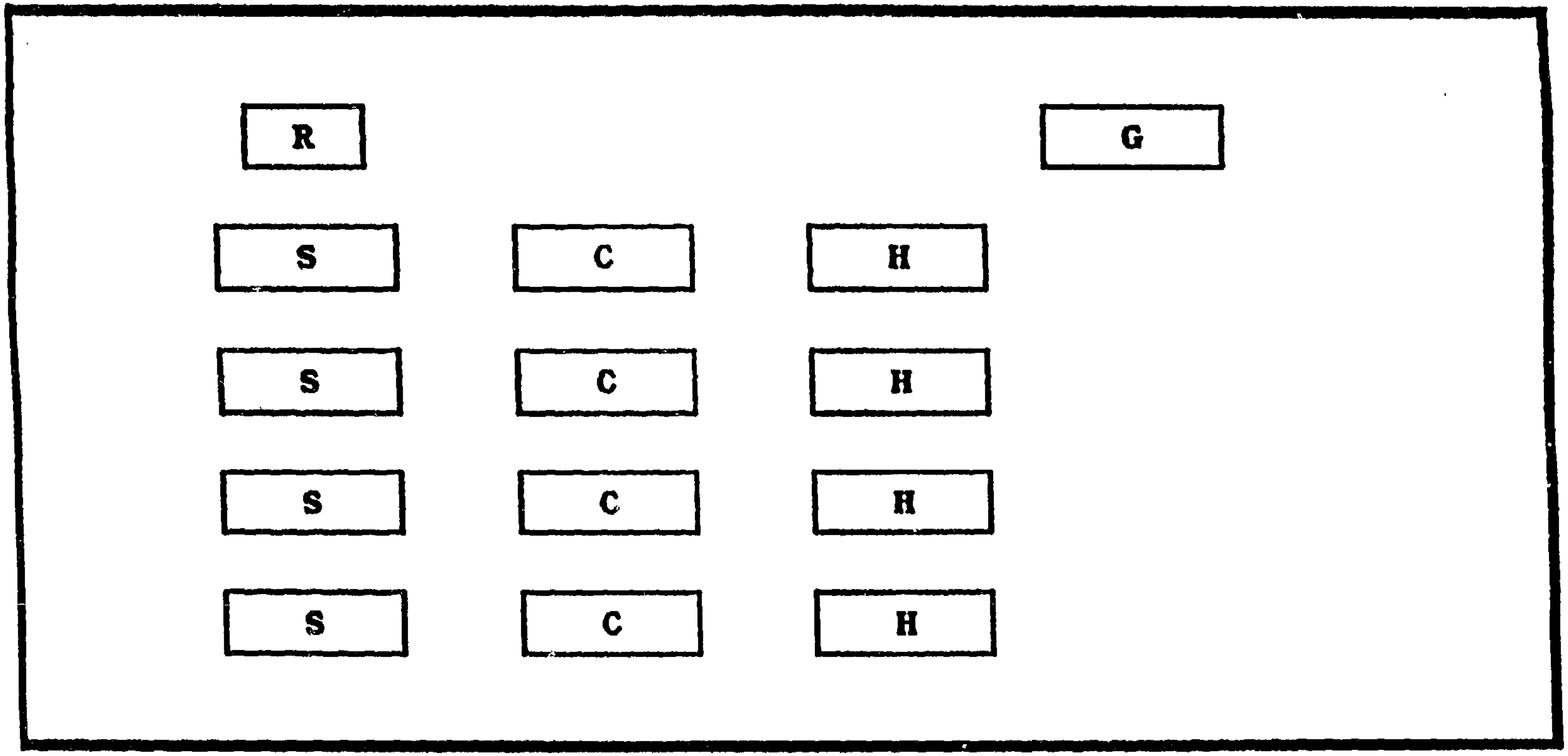
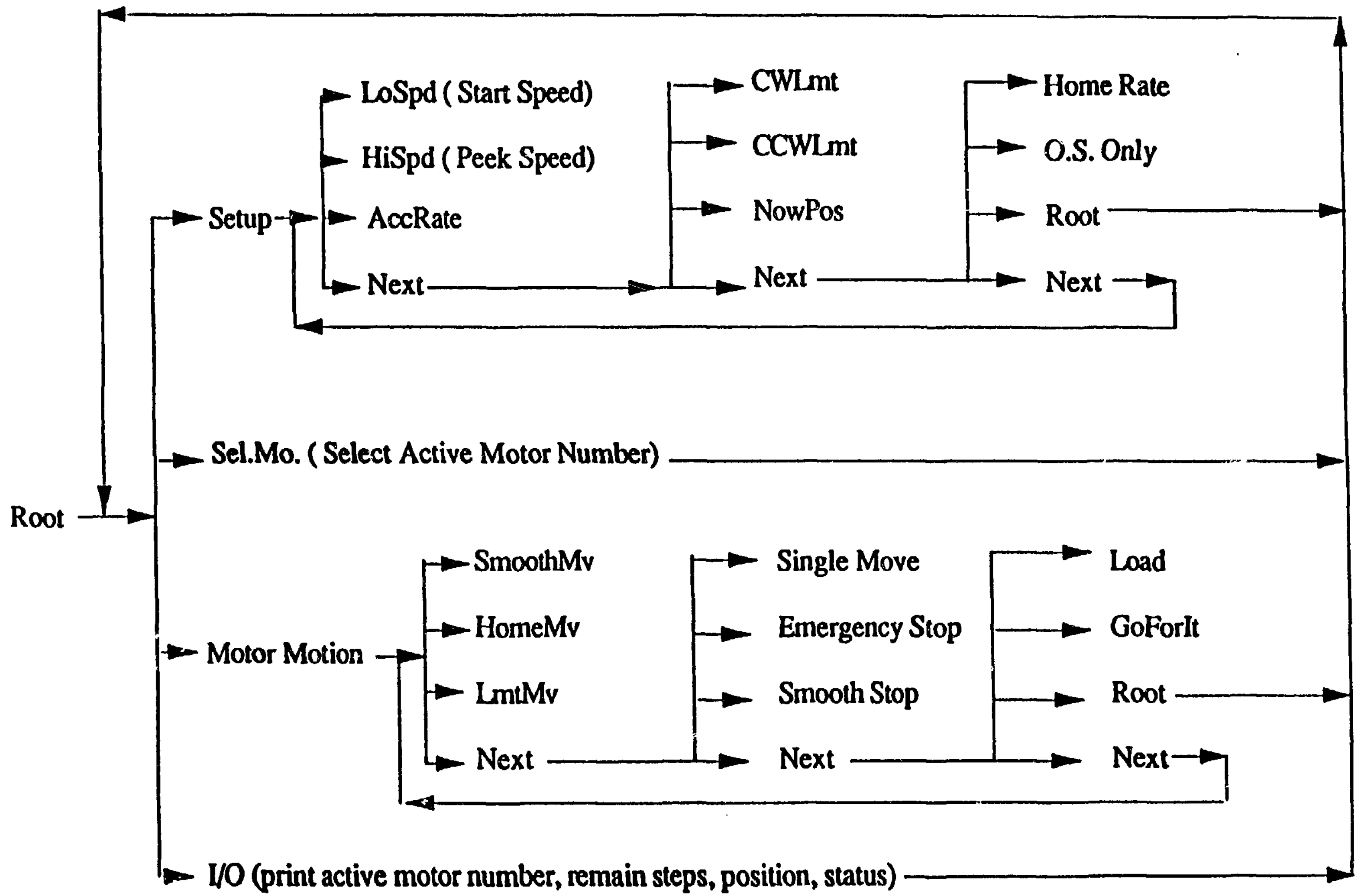

1. Instruction
2. Response Message
3. Keyin Character + Error Message
4. Menu
7
8
9
4
5
6
1
2
3
-
0
Enter
Emergency Stop
Enable/Disable Front Panel keyboard
•
DELETE
F1
F2
F3
F4
Power On

Figure 2



R: Reset button
G: GPIB interface connector
S: Step/Direction outputs connector
C: CCW/CW limit switch inputs connector
H: Home limit + Motor enable/disable inputs connector

Figure 3



Root
Setup
LoSpd (Start Speed)
HiSpd (Peek Speed)
AccRate
Next
CWLmt
CCWLmt
NowPos
Next
Home Rate
O.S. Only
Root
Next
Sel.Mo. (Select Active Motor Number)
Motor Motion
SmoothMv
HomeMv
LmtMv
Next
Single Move
Emergency Stop
Smooth Stop
Next
Load
GoForIt
Root
Next
I/O (print active motor number, remain steps, position, status)

Figure 4