Compumotor

Model 1811 Indexer
User Guide

Compumotor Division
Parker Hannifin Corporation
p/n 88-002485-02 D

Parker

37 Resume Stopped Closed-loop Move

This command will cause the 1811 to move to the last commanded closed-loop position. It was implemented as a fix for remotely resuming closed-loop moves on the Model 3000. The Model 3000 will need D4 revision software to take adantage of this command.

COMPUMOTOR CORPORATION

Model 1810/11 Motor Controller Operator's Manual

Table of Contents

Manual 88-002485-01, 07/15/84. Release 4.0.
REV. D

DESCRIPTION

The 1810/11 Motor Controller is a member of a family of motor controllers which are built to be inserted into any Multibus*/IEEE-796 compatible computer. The Controller consists of a single printed circuit card on which are mounted an Intel 8088 microprocessor, an AMD 9513 counter/timer, a variety of interface circuity and, of course, the Multibus interface.

Although the 1810/11 has its own microprocessor and memory it still requires a host processor in the system to allow it to operate: It cannot operate as a stand-alone device. The host processor will typically take the form of another card plugged into the Multibus backplane or a Compumotor Model 3000 Indexing System.

The 1810/11 unit is designed to control a single motor by issuing pulses to attain motor motion. The **1810** Motor Controller issues its pulses open loop, that is, it does not look for any **feedback** from the motor to see if the desired position or velocity has been achieved. It is assumed that the motor being controlled has been properly designed into its motion control environment and so will not stall when executing commands issued to it by a host processor.

The **1811**, on the other hand, can control the motor using a closed position loop. By using an incremental encoder as a feedback device, the 1811 can position a motor based on encoder position. The 1811 does not require the use of an encoder, however. It can perform any of the commands that the 1810 can, i.e., the 1811 commands are a superset of the 1810 commands.

The 1810/11 relieves the host of the tasks of controlling velocity, distance and linear acceleration parameters as well as monitoring end-of-travel limit switches and trigger inputs. It will perform a sequence of moves and remember that sequence for later execution upon issuance of a single command. The host may also directly control the motion algorithm to achieve sculptured acceleration profiles for applications that require contouring. All velocity, acceleration and distance parameters are given to the Motor Controller in binary bytes representing numbers of pulses (per second, per second squared, etc...). This allows the 1810/11 to control motors of virtually **any** resolution.

Board revision level

When reading this manual, please note which revision level your 1810/11 Motor Controller is. You will find a "dash" number at the end on the board part number (61-002154-**xx**) on the right-hand side of the board (components up, gold fingers toward you). As of this writing the dash number could be **-01, -02, -03 or -04.**
The major differences in the boards will be found in the

Configuration Jumpers listing and in the Connector listing. Changes made after this writing will be noted in a sheet of errata appended to the manual.

* **Multibus is a registered trademark of Intel Corporation.**

INSPECTION

Your Model 1810/11 Motor Controller should be inspected upon receipt for obvious damage to its shipping container. Report any such damage to the shipping company as soon as possible, as Compumotor cannot be held responsible for damage incurred in shipment. The 1810/11 should then be carefully unpacked and inspected for the following items to be present and in good condition.

1. 1810/11 Printed Circuit Assembly
2. Cables: 1. Ground, P/N 71-2489-01
   2. Motor-Driver, P/N 71-2490-01
3. Jumper Clips: 10 each
4. Connector: 25 pin D, solder type with shell
5. 1810/11 Motor Controller Operator's Manual

WARRANTY INFORMATION

Your Model 1810/11 Motor Controller is warranted against manufacturing defects for two years from the date of purchase. Should you have questions about operating the 1810/11, your Compumotor representative and distributors stand ready to support your individual needs.

Should return of your Motor Controller be required to effect repairs or upgrades, do the following.

1. Get the serial number of the defective unit and a purchase order if the unit is our of warranty.
2. Call Compumotor for a Return Material Authorization Number (RMA) at (800) 358-9068 except in California.
3. Ship the unit to:

   Parker Hannifin
   Compumotor Division
   5500 Business Park Drive, Suite D
   Rohnert Park, CA 94928
   Attn: RMA # xxxxxx

Compumotor is dedicated to be a leader in digital motion control. We invite you to tell us about your application problems, questions or comments.

Since all of our indexer products are microprocessor based, significant departures from the operating parameters outlined here are possible. If you have a volume requirement for our products (over 250/yr. for example), we would like to discuss your specific needs. Call Compumotor's Applications Engineering department at (800) 358-9068.

FUNCTIONAL CHECK

The 1810/11 Motor Controller may be installed and functionally tested for proper operation very simply without instructions from the host processor card. If the 1810/11 is to control a Compumotor Motor-Driver having 25,000 steps per revolution, do the following.

1. Remove power from the Multibus computer.
2. Ensure that jumpers JU5, JU6 and JU15 are installed, and then insert the 1810/11 into an open slot in the Multibus computer.
3. Attach the Motor-Driver cable's (P/N 71-2490-01) header end (non D-end) to J4 on the 1810/11.
4. Solder the 25 pin D-connector to the cable which comes with the motor-driver listed under the section "Operation, Connecting the Motor". Attach the D-end of the Motor-Driver cable to the D-end of the cable supplied with the Compumotor Motor-Driver.
5. Set up the Compumotor Motor-Driver in accordance with the Motor-Driver Operator's manual.
6. Apply power to both the Motor-Driver and the Multibus Computer.
   Important: Ensure that the Compumotor is free to rotate a full revolution in either direction for this test.
7. The Compumotor Motor-Driver and the 1810/11 Motor Controller are now ready for self testing.

   Depress the self-test switch (switch S1). S1 is located on the upper edge of the printed circuit assembly, just off center.

The Compumotor should rotate one full revolution in the counter-clockwise direction, stop, and rotate one revolution in the clockwise direction and stop; the Board Monitor Alarm__the red LED next to S1__will come on.

If the motor does not rotate as predicted the problem will probably be found in one of the following areas.

1. The Motor Controller:
   a. The Motor Controller is not powered. Check for 5 Volts DC on J5, pin 9, the Auxiliary Connector. Ground may be found on J5, pin 16, to reference the 5 volts. If 5 volts is not present the 1810/11 is not being powered by the Multibus backplane.
   b. The limit switches are not connected and not bypassed. Ensure that jumpers JU5 and JU6 are installed for this test.

2. The Motor-Driver cable:
   Inspect the cable to ensure that it is not frayed or cut, and that it is securely fastened at each end and at its connecting point.

3. The Motor-Driver:
   a. Ensure that the motor cable is securely fastened to the Driver.
   b. Check to see if the motor has torque. It should have torque any time it is powered.

If the motor the 1810/11 is controlling is **not** a standard 25,000 steps-per-revolution Compumotor the self-test function may still be performed, but the motor will probably not turn exactly one revolution. The 1810/11 issues 25,000 pulses at a rate of 5,000 Hertz with the direction line held low (CCW) and then repeats the process with the direction line held high (CW). A 50,000 steps-per-revolution motor would turn only one-half a revolution and a 200 steps-per-revolution motor would rotate 125 revolutions under these conditions, for example. **A word of warning about a 200 steps-per-revolution motor and this test: The motor velocity will be 25 revolutions per second without the benifit of acceleration ramping and the motor might well stall.**

To verify board operation without a motor, the user may connect an oscilloscope to J4, pin 1, Step Output (ground may be found on J4, pin 2). Each motor pulse will be seen as a one to two microsecond, high going, pulse.

After the self test has been performed the board must be **RESET** before normal operation may be assumed. Reset may be accomplished by cycling the power to the 1810/11, issuing it a bus reset or by setting the reset bit in the 1810/11's Control Byte. After performing the self-test function it is wise to **remove jumper JU15** to disable the test function from being performed while the motor is under host control.

INSTALLATION

A. Configuration Jumpers

Installation of the 1810/11 Motor Controller is a straightforward process that may be quickly and easily accomplished provided the Controller is properly configured prior to attempting to operate it.

Refer to Appendix F for the location of the configuration jumpers on the 1810/11 board.

The first step in configuring the Controller is to install the configuration jumpers. These jumpers are of five basic types:

Multibus interrupts
Multibus addressing
Limit switch bypass
Trigger #1 input bypass (-01 and -02 boards only)
Test switch enable

All of the Configuration Jumper designations shown below reflect those for the -03/-04 revision boards. For other revisions see Appendix B.

1. Multibus Interrupts

Use of interrupts on the 1810/11 Motor Controller is optional. When first attempting to get the Controller up-and-running it might well be easier to leave the interrupts disabled, the way the Controller comes from the factory, until appropriate software can be written to handle the interrupts.

There are three basic interrupts:

Interrupt	Jumper	Status Bit
Message ready (MR)	JU18	3
Input ready (IR)	JU20	4
Controller failed (CF)	JU19	5

Any of these interrupts may be directed to any of the Multibus interrupt lines, INT0/-INT7/* (JU63-JU70). The three Controller interrupts are wire ORed together so that any combination of the interrupt jumpers may be installed, but the interrupt which goes active first masks any further interrupts in the Status Byte until the previous condition is cleared (see the sections "Programming, Interrupts" and "Programming, Reading and Writing" for details).

Only one of the Multibus interrupt lines (INT0/-INT7/) may be used to interrupt the host processor if interrupts are enabled

using jumpers as described above. If, for some reason, you would like to use a separate Multibus interrupt line for each of the 1810/11 interrupts a different scheme must be used. By wiring directly from any of the three 1810/11 interrupt lines to the Multibus interrupt lines, each of the three 1810/11 interrupts may have its own Multibus interrupt. To accomplish this the **1810/11 interrupt** pins (JU18-20, right side) which are to have their own **Multibus interrupts** are connected to the appropriate **Multibus interrupt** pin(s) (JU63-70, bottom row) using a wire wrap connection. For example, if you want the IR interrupt connected to Multibus interrupt INT3/, the right pin of JU20 is wired to the bottom pin of JU67. (To reference the preceding instructions the 1810/11 Motor Controller should be held with the Multibus connector oriented down.)

The Compumotor suggestion is that the **jumper** method be used to receive interrupts from an 1810/11, rather than the wire-wrap method, since the jumper method allows you the ability to mask interrupts from an 1810/11 via the "Interrupt Enable Bit" in the CONTROL BYTE (CONTROL BIT six). Generally, the only interrupt you will be interested in jumpering will be the **Message Ready interrupt.**

If you choose to use interrupts from more than one of the status bits using the jumpering method, an active status bit will mask other active status bits since all jumpered status bits are ORed into one interrupt line. Therefore, it would be advisable to use level sensitive interrupts rather than edge sensitive interrupts if more than one status bit is jumpered to the INT/ line, since a currently active jumpered status bit will mask an active going status bit. If only one status bit is jumpered to the INT/ line, either edge triggered or level triggered interrupts may be used.

CAUTION: 1810/11 interrupts wire wrapped directly to Multibus interrupts are not affected by the 1810/11's interrupt mask. This means that **there is no way available to the user to prevent an interrupt to the Multibus should an interrupt condition occur when using this technique.** (Should the host have an interrupt prioritizer this problem can be neatly handled in the host software.) See the section "Programming, Interrupts" for details on how to handle interrupts.

*** A signal name followed by a slash (/) indicates that the signal is low true.**

2. Multibus Addressing

a. Address jumpers

The 1810/11 Motor Controller requires two address locations to access its data and command registers. For this reason the least significant bit of its potential addresses is not specified. The board's address is selected by installing configuration jumpers which carry binary values. The sum of the jumper values is equal to the board's base or even address.

Jumper	Bit Number	Decimal	Hex
JU45	Address bit 1, value =	2	2h
JU46	Address bit 2, value =	4	4h
JU47	Address bit 3, value =	8	8h
JU48	Address bit 4, value =	16	10h
JU49	Address bit 5, value =	32	20h
JU50	Address bit 6, value =	64	40h
JU51	Address bit 7, value =	128	80h
JU39	Address bit 8, value =	256	100h
JU40	Address bit 9, value =	512	200h
JU41	Address bit 10, value =	1,024	400h
JU42	Address bit 11, value =	2,048	800h
JU29	Address bit 12, value =	4,096	1000h
JU30	Address bit 13, value =	8,192	2000h
JU31	Address bit 14, value =	16,384	4000h
JU32	Address bit 15, value =	32,768	8000h
JU33	Address bit 16, value =	65,536	10000h
JU34	Address bit 17, value =	131,072	20000h
JU35	Address bit 18, value =	262,144	40000h
JU36	Address bit 19, value =	524,288	80000h

If, for example, you wanted to specify an address of FFF0h (65,520 decimal) you would install address select jumpers JU29-32, 39-42 and 48-51.

b. Addressing mode jumpers

Once the board's address has been selected, select the eight, twelve or sixteen-bit addressing mode. This is done by determining the number of the most significant bit used for addressing and then assigning the next higher addressing mode number by installing the appropriate jumper(s). (See Table 1.) Should more than one addressing mode be selected, the lowest mode will be used, e.g., if the eight-bit addressing mode is selected and jumper JU37 is installed (sixteen-bit addressing or more) the 1810/11 will default to eight-bit addressing.

Do not install jumpers on address lines which are not selected by the addressing mode.

Addressing Mode	Jumpers 44*	38	43	37*
8 bit	X			
12 bit		X	X	
16 bit**			X	X
20 bit			X	X

X = Install jumper.

Table 1.

* JU37 and 44 are installed at the factory.
** To use sixteen-bit addressing **do not** install jumpers JU33 through JU36.

Referring back to the example of configuring the 1810/11 to an address of FFF0h, FFF0h requires 16 bits to decode so you would install jumpers JU39 and JU37 to give 16 bit addressing. Factory installed jumper JU44 would be removed.

c. Memory vs. I/O mapping jumpers

Having selected the 1810/11's address and addressing mode the board must be configured for either **memory-map or input/output-map addressing.**

To **memory-map** install jumpers JU58 and JU56.
To **I/O-map** install jumpers JU59 and JU57. (The 1810/11 comes I/O-mapped from the factory.)

The difference between I/O- and memory- mapping stems from your choice of whether to use the 1810/11 as an input/output port or as a memory location. The 1810/11 will work equally well either way.

3. Limit switch bypass jumpers

Limited travel mechanisms such as X-Y tables require some means for stopping the motor should it attempt to move the load beyond the motion system's end of travel. The 1810/11 deals with this problem by using limit switches. The limit switch inputs (found on pins three and five of the Auxiliary Connector, J5) use a normally-closed circuit to indicate to the motor controller that the system is operating properly. This way if one of the wires to the limit switches should be broken or cut the system

stops instead of blithely moving the load out of its range of travel.

Since not all applications require **limit switches**, the 1810/11 has two jumpers which short across the the limit switch inputs to allow the board to operate the motor when the switches are not required. These jumpers are JU5 and JU6. To ensure proper motor operation they **must be removed when limit switches are used and they must be installed when limit switches are not used.**

4. Trigger #1 input bypass jumper (-01 and -02 boards only)

JU3(-01/-02 boards), the Trigger #1 bypass should be installed if Trigger #1 is not used. Trigger #1 is found on the Auxiliary connector, J5, pin 1. This input is used to allow an external device trigger some user defined event through a single pulse applied to the line. It interrupts the 8088 microprocessor when used, so leaving this jumper off can disable the 1810/11 from performing any function aside from attempting to execute a routine related to Trigger #1 which, of course, would not exist.

5. Test-switch enable jumper

JU15, the test-switch enable, allows you to test board operation without the need for any interaction with the host processor board. This jumper should be removed unless the test function is to be executed since when the jumper is installed the 1810/11 will stop whatever it is doing to execute the test sequence, no matter what else it may have been doing at the time including executing user defined moves.

The test function causes the 1810/11 to issue 25,000 pulses at 5,000 hertz in the counter-clockwise direction and then issues the pulses again in the clockwise direction.

At the end of the test sequence, the 1810/11 tests its board monitor alarm. This means that the 1810/11 will go into a fault condition, from which it must be reset. **This is normal!** The red led mounted in about the center of side of the board away from J1 (the Multibus connector) will come on when this occurs.

B. Limit Switch Inputs

The clockwise and counter-clockwise limit switch inputs on the 1810/11 are used to cause the 1810/11 to **immediately** discontinue sending pulses to a motor-driver in an emergency situation. The response of the motor is the same as it would be to that of the Kill command (78h). The **limit switches** provide a signal ground to the limit switch inputs (J5, pins 3 and 5) under normal operating conditions and allow the inputs to float high when the switch is activated. This way, should a wire break anywhere in the limit switch circuit, the motor will immediately stop and the 1810/11 will not allow the motor to move toward that

limit switch until the open circuit condition is corrected.

In applications that **do not require limit switches,** jumpers JU5 and JU6 should be left installed as the unit comes from the factory.

A limit switch normally takes the form of a **normally closed single-pole-single-throw switch, a Hall effect proximity sensor or an optical interruptor** which is placed sufficiently far from the load's end of travel to allow the load to stop from it's maximum running velocity without damage. The switches should be able to provide the following.

1. 1 mA current sinking capability (1mA closure to ground)
2. Logic low no greater than 0.5 VDC
3. Logic high no greater than 5.5 VDC (unless open collector)
4. Minimum signal high duration of 200 microseconds

The limit switches are connected to the following pins.

Clockwise limit switch	J5, pin 3
Clockwise limit switch **return**	J5, pin 4
Counter-clockwise limit switch	J5, pin 5
Counter-clockwise limit switch **return**	J5, pin 6

The limit switch returns are the 1810/11's signal ground so care should be taken to **not connect these lines to earth ground or use them for a shield,** as this will greatly reduce the noise immunity of the 1810/11 and increase its susceptibility to damage as a result of voltage or current spikes on the power line.

The limit switch wires should be **shielded** to prevent voltages from being induced into them which could cause the limit switch inputs to be activated. An earth ground suitable for shielding purposes is found on J5, pin 20. (This shield is only effective if J6 is connected to a good earth__usually chassis__ground.)

Suitable switch types are:

Vane activated magnetic switch__Microswitch 4AV1C-T1
Vane activated optical switch__Spectronics SPX2002
Proximity sensor, Hall effect__Microswitch 205SR1A

Some limit switches require a **power source.** This is provided for in the 1810/11 on J5, pin 19. A maximum of 250 milliamps may be drawn from this connection. A word of warning about this power source: It comes straight from the Multibus five volt power supply through a one ohm, quarter watt, resistor, and so should be used with caution. Signal ground for the limit switches is found on J5, pins 4 and 6.

The limit switches attatch to J5 through a user supplied 20 pin **flat-cable connector.** Typically, the flat cable will be

routed to a panel connector which will be installed in the Multibus chassis, and another cable connector will route the wires to the motor driven assembly.

C. Home Position Sensor

Sometimes an absolute reference is required for the motion control system, so that a known position may be found upon applying power to the system or prior to attempting a very precise move. The 1810/11 allows this to be done by using a normally closed switch similar to the ones used as limit switches described above, as well as home reference marks on linear and rotary encoders.

The home position sensor is attatched to J5, pin 9, the Home Limit input, and it must have all of the electrical characteristics described above for the limit switches. The Home Limit input is ignored when it is not being used as a position reference. Signal ground for the home position sensor is the same as that of the limit switches, J5, pins 4 and 6.

D. Earth Grounding

It is recommended that an earth ground be connected to J6 using the ground cable provided in the 1810/11's ship kit. It is important that the ground used here does not connect to the signal ground used for the Multibus system or for any other electronic device. This earth ground is routed to the Multibus connectors that require a shield connection so that sensitive wires may be shielded without sacrificing noise immunity.

OPERATION

A. Introduction

Operation of the 1810/11 Motor Controller begins with its installation into a Multibus backplane, and is followed by connection of a motor-driver. The Motor Controller may then be self-tested to ensure that it is operating normally (as described in the "Installation" section above. Beyond that, all interaction with the Motor Controller **must** be done through a host processor Multibus card inserted elsewhere in the Multibus backplane.

The 1810/11 appears to the host processor as two address locations: one for CONTROL and STATUS, and one for DATA input and output. The address locations may appear anywhere in memory, or as any Input/Output (I/O) port. The Control and Status are at the even address location and Data can be found at the odd address location. Refer to the section "Installation, Configuration Jumpers" for details on how to set up addressing on the 1810/11.

The 1810/11 should be reset using the board reset bit in the Control Byte (bit 5).

The following sections describe how to connect a motor-driver assembly to the 1810/11, suggest ways to avoid problems in operating the target motion control system and how to interface the following inputs and outputs.

1. Remote power shutdown (output)
2. Trigger inputs (inputs)
3. Programmable outputs

B. Connecting the Motor

Two cables are supplied with the 1810/11 Motor Controller: a ground lead that should be connected to **earth ground**, and a cable which connects the **motor-driver** connector (J4) to a bulkhead connector. The bulkhead connector, in turn, connects the 1810/11 to a motor-driver. A 25 pin D-connector is supplied which plugs into the bulkhead end of the supplied flat cable; the motor-driver cable is soldered to it using the connector pinouts listed below.

BULKHEAD CONNECTOR LISTING

"D" connector

Pin Number	Function	Wire Color*
1	Step output	Red
2	Direction output	Green
3	CW step output**	n/a
4	CCW step output**	n/a
5	Shield	Shield
8	M-D +5 volts	**
14	Step return	Black
15	Direction return	White
16	Shutdown out	Blue
17	Shutdown out return	Brown
20	DC common	**
24	DC common	**

The connections **not** shown above are **not** used.

* The wire colors shown correspond to a standard Compumotor motor-driver.

** These outputs are used with non-Compumotor motor-drivers.

TABLE 2

A standard Compumotor motor-driver is simply wired-up using the connections in the listing above. A **non-standard** motor-driver, such as a 200/400 steps-per-revolution motor, may also be controlled. If the non-standard motor-driver uses step and direction lines it may be wired the same as a Compumotor motor-driver. If the non-standard motor-driver uses clockwise (CW) and counter-clockwise (CCW) control lines use pins 3 and 4 instead of 1 and 2 above. The CW and CCW outputs are open collector and so should be pulled-up to operate properly. Five volts (250 milliamps maximum) may be found on pin 8 for this purpose. DC common (circuit ground) is found on pins 20 and 24.

The output driver for the step, direction, clockwise and counter-clockwise control lines is a 75174 from Texas Instruments, which provides current for the standard Compumotor optically-coupled interface (typically 20 milliamps). The output voltage for this device is specified to not drop below 3 volts when high, but will not generally approach 5 volts—the standard TTL high level. Therefore a non-standard motor-driver if voltage, and not current, driven should be able to operate with a 3-5 volt high signal.

The earth **ground connection** included with the 1810/11 Motor Controller should be installed prior to operation of the motor-driver to reduce emitted radiation and to protect the Motor Controller from spurious line voltages. This line should **not** be connected to the 1810/11's signal ground.

C. Remote Power Shutdown

The Remote Power Shutdown (RPS) control line is a Compumotor feature that allows the motor to be shutdown (all torque removed) for the purpose of motor-driver cooling or to allow the user to manually position the load. Although the standard Compumotor motor-driver is rated for a 100 percent duty cycle it is sometimes advantageous to shut down the motor-driver to lessen the amount of heat dissipated into a given environment. Additionally, since stepper motors have full rated torque at zero velocity, it becomes very difficult to manually position the motor's load with power applied to the motor. Rather than remove power from the motor-driver under these conditions Compumotor has provided the Remote Power Shutdown feature.

CAUTION: Position may be lost when shutting down the motor and then powering it back up again. **When position must be maintained while the motor is in the Shutdown condition a friction lock or brake should be used to maintain shaft position.**

Remote Power Shutdown is found on pins 16 and 17 of the bulkhead connector (Pin 6 of J4). Pin 16 goes high with reference to pin 17 when RPS is activated. **RPS is controlled** with commands 12h and 13h of the command language, a Compumotor Joystick and/or the Shutdown input line on the Auxiliary connector, J5, pin 18 (active low). The motor will not be shutdown if the motor is moving; should the motor be moving when the RPS command is received the 1810/11 will ignore the command. See the section "Programming, Command Language" for more details.

D. Trigger Inputs

The trigger inputs (also called trigger "bits") consist of six inputs on the Auxiliary connector, J5. The 1810/11 may be programmed to electronically "watch" any one of these inputs and perform a predefined function when the proper signal appears. Two of these inputs (1 and 6) deliver an interrupt to the 8088 microprocessor on the 1810/11 circuit card upon a positive going trigger, providing very fast response to input signals. The other trigger inputs (2-5) are level sensitive and so are a little more flexible than 1 and 6 but are also a bit slower.

Trigger inputs 1 and 6 must make a low to high transition to be considered active. Trigger 1 requires a hold time of 500 microseconds to be recognized, and Trigger 6 requires 1 millisecond. Following this there must be a dwell period of at least two milliseconds before the input makes another positive going transition if it to be recognized as a Trigger.

Trigger inputs 2-5 may be <u>either</u> a high or low level to indicate an active condition. Selection of the level triggering is done when issuing the command. The signal must remain active for a minimum of four milliseconds for it to be recognized.

All trigger related commands allow only one trigger input to be specified for use at a given time. Thus, checking for an AND condition to occur on the trigger inputs is not possible.

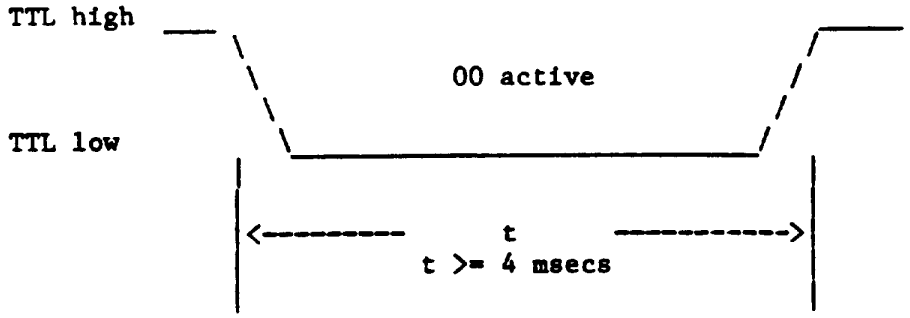
There are 11 trigger input related commands; of these, nine use only triggers 1 and 6. Command 2C, "Wait for trigger "X" to go active" is the only command which allows you to use a level sensitive input or an edge sensitive input. This command allows interruption of a sequence of moves until some external event comes true. **Refer to the section "Programming, Command Language" for more information on these commands.**

Trigger input signals must be able to drive one standard TTL load. For our purposes here this means the trigger input should be able to source 1 milliamp and sink 2 milliamps. The Auxiliary connectory may be connected to external signal sources by running a 20-pin flat cable from J5 to a D-type bulkhead connector in the Multibus housing. From there a D-connector will allow connection of shielded cabling to the motion control system.

Command (in hex)	Command description, event occurs when trigger "X" goes active.	Triggers used
2C	Wait, continue on trigger	1 thru 6
74	Stop motion	1 or 6
75	Discontinue sequence buffer	1 or 6
76	Suspend sequence buffer	1 or 6
77	Discontinue singular command	1 or 6
7C	Kill motion	1 or 6
7D	Kill the sequence buffer	1 or 6
7E	Kill singular command, wait for cont.	1 or 6
7F	Kill velocity-streaming buffer	1 or 6
AC	Interrupt	1 or 6
94	Request state of trigger inputs	2 thru 5

Table 3.

Following is an illustration of an active level sensitive trigger bit (triggers 2-5).



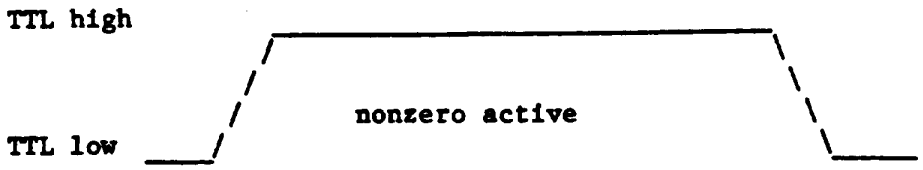


Figure 1.

The following illustration is of an active rising edge sensitive trigger (triggers 1 and 6).

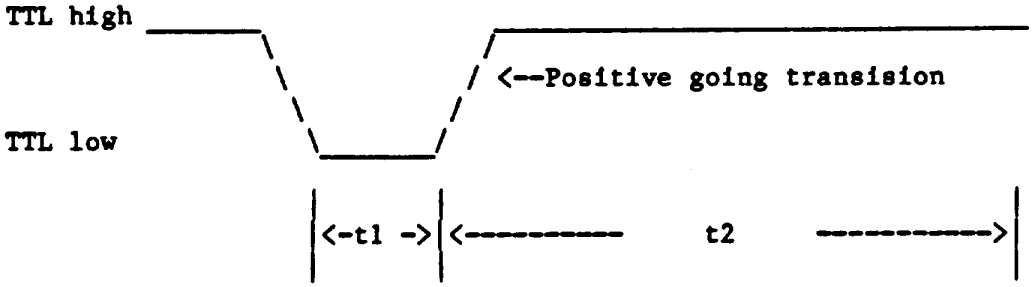


Figure 2.

Following is a list of the pin connections for the trigger inputs on J5, the Auxiliary connector. For conversion to 25 pin "D" connector format, see Appendix C.

Pin Number	Description	
7	Trigger input 1	*DC common
11	Trigger input 2	(signal ground) may
17	Trigger input 3	be found on pins
12	Trigger input 4	2, 4, 6 and 8.
10	Trigger input 5	
1	Trigger input 6	

Table 4.

Trigger inputs 3 and 4 are shared with the Drive Fault and Slip Fault functions on the Motor-Driver output connector (pins 17 and 19, respectively). Drive Fault and Slip Fault are inputs reserved for possible future explansion, and as yet have no function. Care should be taken to avoid putting signals different from Triggers 3 and 4 onto these inputs.

E. Programmable Outputs

The programmable outputs on the 1810/11 allow you to directly control two high current outputs on the Auxiliary connector of the 1810/11. Commands 0B and 0C (hex) set and reset (make high or low, respectively) these outputs.

These outputs can sink or source 60 milliamps and are short circuit protected (albeit for short periods of time). They may be used to trigger an external event during, or at the completion of, some Compumotor driven action.

The programmable output bits are found on pins 13 and 15 of J5, the Auxiliary connector. The Auxiliary connectory may be connected to external signal sources by running a 20-pin flat cable from J5 to a D-type bulkhead connector in the Multibus housing. From there a D-connector will allow connection of shielded cabling to the motion control system.

F. Joystick Interface

The Joystick interface on the 1810/11 allows you to use a Compumotor Model 852 Joystick in conjunction with the 1810/11 Motor Controller. The Model 852 is a two-and-a-half axis unit that allows full control of two axes with an X-Y spring-loaded joystick and control of a third axis via a TTL compatible quadrature output incremental encoder. Either of the two main axes of the 852 may be connected to the 1810/11 so that, when the host processor allows it, the motor may be moved under Joystick control, being monitored all the while by the 1810/11. This way a move can be made by an operator and the absolute or relative position of the move reported back for later use. The entire path the motor takes may be recorded, provided the host has enough memory to record all of the moves made.

There are only two commands in the 1810/11 which deal directly with the Joystick. These are 10 (hex) and 11 (hex), which disable and enable the Joystick, respectively. Control must be given to the Joystick via the 1810/11 Joystick enable command if moves are to be made under Joystick control. Motor position may be determined by using position requests, commands 80, 82 and 83 (hex).

Command 80 (hex) reports the position from the last place the Joystick stopped the motor. The 1810/11 has several status lines available to it from the Joystick so that it is "aware" of whether or not the motor is moving under Joystick control.

Command 82 (hex) reports absolute position with respect to the last home position found during a go-home sequence.

Command 83 (hex) reports absolute position with respect to absolute zero.

NOTE: Home position and absolute zero position are both reset upon power-up.

Connecting the 852 Joystick is accomplished by first connecting a 20 pin flat cable header to the Joystick connector, J3.

From this header, a flat cable is routed to the Multibus frame where it is attached to 15-pin D-type bulkhead connector. From there the standard cable which is provided with the 852 Joystick is connected between the bulkhead connector and the 852 Joystick. Refer to Appendix C for a complete connector listing.

G. Encoder Interface (1811 only).

The Encoder interface (J6) on the 1811 allows the 1811 to respond to an incremental encoder for positioning purposes. The interface is not designed to monitor/correct-for velocity errors, the interface allows verification of position, and motor stall detection. Appendix E lists and describes the commands which are used for postion maintenance/stall detection. The Indexer understands all of these commands as well as all of the open-loop commands listed in Appendix A.

The Encoder interface is a very powerful addition to the 1810. It allows definition of the number of pulses sent to the motor for how many pulses are expected from the encoder (command B0h). Because of this it is possible to use virtually any resolution encoder. Further, the encoder need not be mounted directly on the motor. In situations where the load could well be in a different place from that of the motor, such as when a gear train of some sort is involved, the encoder may be mounted where it will do the most good. An example of this is driving the load through a series of gears. It would be very nice to be able to ignore the backlash induced by the geartrain and get down to business. Putting the encoder on the load would cause any closed-loop move (you may choose either open- or closed-loop moves at any time) to automatically compensate for backlash and gear ratios in the gear train.

1. Electrical Considerations

A schematic of the encoder input is shown in Appendix F. Connection of the encoder to the 1811 will typically be accomplished by running a cable to a bulkhead connector in the MULTIBUS card rack, and then running a 26 pin flat cable from the bulkhead to J6 on the 1811 using a flat cable header. Connections for the encoder connector may be found in Appendix C along with a translation of flat cable to D-connector pinouts.

There are test points clearly labeled A and B next to the Encoder connector which may be used to establish whether or not the encoder input is working properly. These two test points should have a square wave signal on then (90° out of phase) any time the encoder is moved. Position tracking (closed-loop) does not have to be enable for this to occur.

A and B channel inputs are required for position tracking

and stall detection, the home input is only required if a home position is needed. The home input need not come from the encoder either. The home input may come from any appropriate TTL compatible switch as referenced in the installation section of this manual. For example, using a linear encoder on an X-Y stage you may want to utilize the home line on the encoder because that input goes active only once in the entire range of travel, is similar in electrical characteristics to the A and B channel inputs and can be wired in the same harness as the other encoder inputs.

A rotary encoder, on the other hand, which is attached directly to the motor gives its home indication once per revolution. This is of little value in most applications. To get around having the once per revolution home mark the encoder has a **home enable** input which, when inactive, causes the home input to be ignored. Thus, another switch is required that will be active only when the motor is within one revolution of the home area.

2. Encoder Resolution (lines).

The resolution of the encoder, usually given in **lines** is multiplied by four times by the 1811. This is done through a **quadrature detection** technique. Thus, if you have a 1,000 line incremental encoder, the 1811 will see it as a 4,000 line encoder. This is also known as specifying the **post-quadrature** encoder resolution. When specifying the number of motor steps to encoder steps (command BO) you must specify the encoder resolution multiplied by four.

3. Usable Encoder Resolution

The ratio of motor pulses to encoder pulses should probably not get smaller than **2:1**. The reason for this is a practical one, and is not restricted by the 1811. Simply stated, as the distance moved by a single motor step approaches the size of an encoder step it becomes increasingly difficult to position the motor accurately. Consider the following diagram.

Motor step size--> |_____|_____|_____|_____|

Encoder step size--> |_____|_____|

Motor steps equal one-half an encoder step

In this example the motor steps are half the size of the encoder steps. On paper, this might lead you to assume that there is plenty of room for the motor to find a particular encoder position. In reality, because of the elastic nature of step motors, it may be difficult for the motor to actually find the

desired position. Consider the case where the motor step size is different in each direction of motor travel. This would be true any time the motor is changing direction while driving a frictional load.

Motor step size--> ___|___|_____|____|

Encoder step size--> |_____|_____|

Motor steps are one-half an encoder step, step size is not equal.

In the above grossly exagerated example there are still four motor steps to two encoder steps, but it is not clear where each will fall. It is possible that the backlash caused by frictional loading on the motor will vary the step size enough to make it difficult for the motor to position exactly on the desired encoder position.

Motor step size--> ___|_____|_____

Encoder step size--> |_____|_____|

Motor steps are larger than encoder step

Should motor steps ever exceed the size of encoder steps, the motor would probably never find its desired position, especially if the load is subject to vibration. Even if it is not, the likelyhood of the motor positioning correctly and sitting still are small. When the motor steps are larger than encoder steps it is likely that the motor will never sit still while position maintainance is enabled (command 18h).

PROGRAMMING

A. Languages

We at Compumotor are often asked which language is compatible with the 1810/1811 Motor Controller. This is a fairly simple question but it does not, unfortunately, have a very simple answer. In one sense there are no languages compatible with it, and in another sense all of them are.

The 1810/1811 must be "spoken to" via a host card plugged into the Multibus. Therefore the data transfers on the Multibus backplane define the language used. This is not a true language: The user does not have direct access to it. A computer language can **control** what happens on that bus, but it is **not** the language itself. All in all, this means that whatever language the host processor card uses, (there has to be at least one overseer in the system) is in turn used to communicate with the 1810/1811. All communication with the 1810/1811 is done through the host processor card.

Although just about any language may be used in theory, only compiled or assembled languages will have the speed necessary to take advantage of the 1810/1811's high speed features. **BASIC,** while a popular language that is supported on virtually every type of computer made, is generally an **interpreted language** and, as such, would probably not be able to keep up with complex move profiles requiring much interaction with the host. A compiled version of BASIC might do the job. Another popular language much better suited to the task is **C.** While there is no "best language" C is a **compiled language** that runs very quickly. It has the capacity to do very complex as well as very simple jobs without wasting much memory space. Probably one of its best features for an application such as driving an 1810/1811 Motor Controller is that repeated functions may be put into library modules so that they may be called on as a separate functions by any program you write. This means you write the arduous but necessary functions, such as putting data onto the Multibus, only once.

Other languages which would do the job include, but are not limited to, **Pascal, Modula 2, Fortran** and assembly.

B. Reading and Writing

The following is an explanation of how to communicate with the 1810/1811. It covers the basic protocol involved in sending information to and taking information from the 1810/1811. The subject of how to handle interrupts is addressed in the section "Programming, Interrupts".

There are two pairs of registers which are used to communicate with the 1810/1811. Data is transferred via the **Input** and **Output** Data Buffers (IDB and ODB, respectively) and commands and status are transfered via the **Control Byte** and **Status Byte** (CB and SB, respectively). Each pair of registers has its own address or port location, depending on how the board is configured. Data and commands are transfered on the even memory/IO location and status and control are transfered on the odd memory/IO location (See "Installation, Configuration Jumpers" for more information on board configuration.)

In each pair of registers a **read** will take information from one register, and a **write** will put information into the other. It is not possible to read a write register nor write to a read register. In the 1810/1811, the read registers are the ODB (Output Data Buffer) and the SB (Status Byte); the write registers are the IDB (Input-Data-Buffer) and the CB (Control Byte).

The SB and the CB are single byte registers but the IDB and ODB are each 16 bytes deep, using FIFO (First-in, First-out) buffers so that data may be transferred as fast as possible. The 1810/1811 commands are configured to take advantage of the 16 byte-deep FIFOs: All commands are 16 bytes long, although not all of the bytes are necessarily significant. Any insignificant bytes that follow a command need not be sent to the motor controller.

Motor Controller commands are single byte numbers in the range 0 to 255 (00 to FF hex) followed by 0 to 15 parameter bytes. All commands are checked for the appropriate number of parameter bytes, and to see that those bytes are in the correct range. Commands which are missing parameter bytes or have parameter bytes which are beyond expected values are **ignored.**

Example: The command "**Perform move X**" (40 hex) is a 2 byte command.
Byte 1 is the command byte.
Byte 2 is the parameter byte telling which of the 50 possible moves is to be executed.
Bytes 3-15 are not defined and must be zeros if included.
Note that this move must have been previously defined for it to be performed.

1. Writing commands to the 1810/1811

Transferring data to the 1810/1811 is straightforward provided the following handshake procedure is followed.

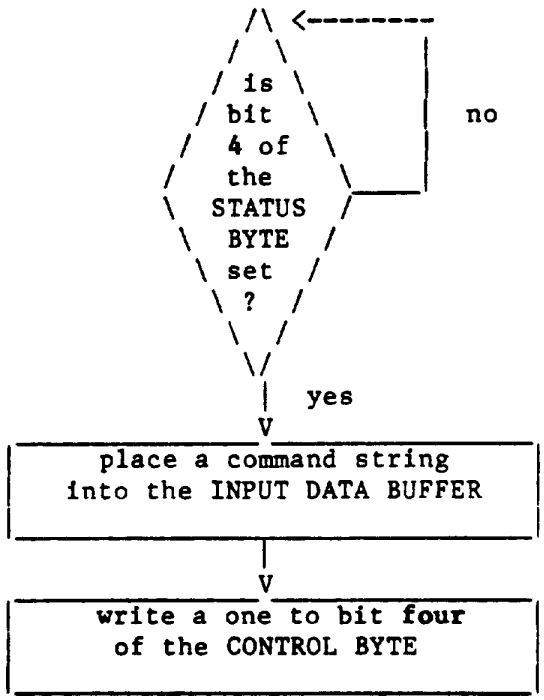
1) Read the SB and check bit four, the IDB-Ready bit. If

this bit is set go on to step two, otherwise repeat this step.

2) Write the Motor Controller command string (the command byte followed by any required parameter bytes) into the IDB.

3) Write to the CB and set bit four, the Command-Ready-In-The-IDB bit. This indicates to the Motor Controller that there is a command in the IDB.

Following step three above, the Motor Controller will go to the IDB and read the command string that the host has placed there. Commands are placed into a command buffer internal to the Motor Controller and are processed in sequential order. Following is a flow chart illustrating the above three step procedure.



2. Reading responses from the 1810/1811

Commands that require the 1810/1811 to respond via the Multibus require a six step handshaking process. The first three steps are the same as for **writing** to the Motor Controller, steps four through six result in the host **reading** the message in the ODB. Following is a flow chart illustrating the six step procedure.

1) Read the SB and check bit four, the IDB-Ready bit. If

this bit is set go on to step two, otherwise repeat this step.

2) Write the Motor Controller command string (the command byte followed by any required parameter bytes) into the IDB.

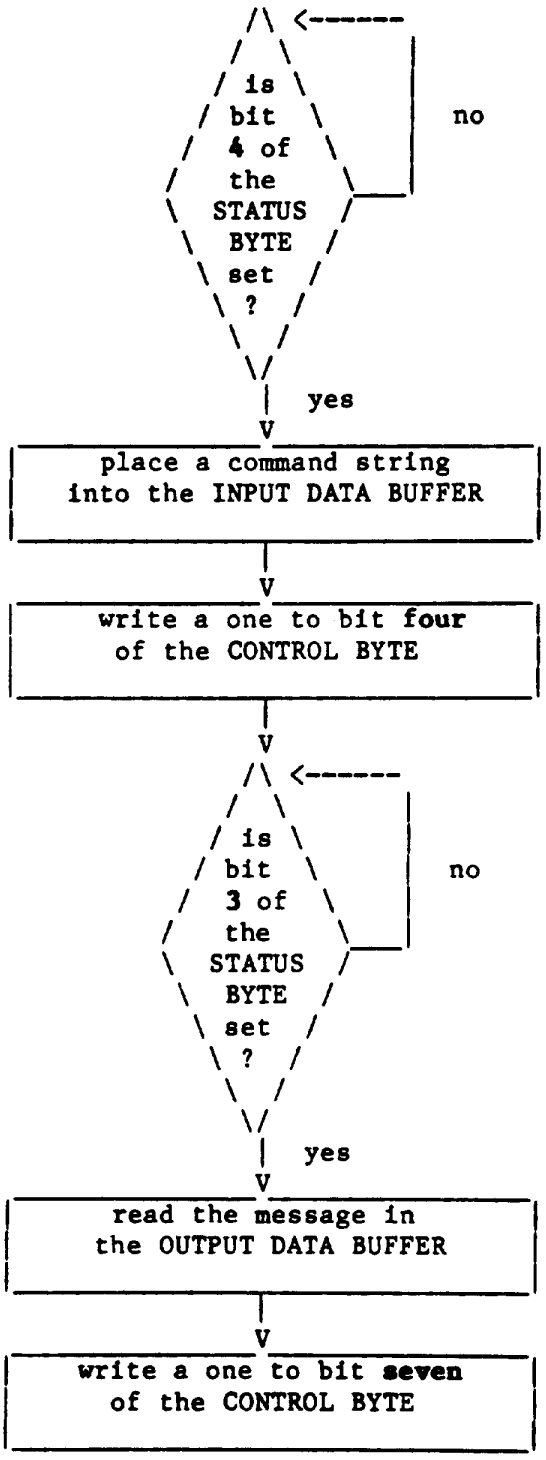
3) Write to the CB and set bit four, the Command-Ready-In-The-IDB bit. This indicates to the Motor Controller that there is a command in the IDB.

4) Read the SB and check bit three, the ODB-Ready bit. If this bit is set go on to step five, otherwise repeat this step.

5) Read the message in the ODB. There will always be sixteen bytes in the message, but only the significant bytes need be read, the extra bytes may be ignored.

6) Write to the CB and set bit seven, the Message-Accepted-From-The-ODB bit. This will indicate to the Motor Controller that the message it placed in the ODB for the host has been read. If the Motor Controller has been waiting to send additional messages to the host it will do so then.

Following is a flowchart illustrating the above handshake sequence.


is bit 4 of the STATUS BYTE set ?
no
yes
place a command string into the INPUT DATA BUFFER
write a one to bit four of the CONTROL BYTE
is bit 3 of the STATUS BYTE set ?
no
yes
read the message in the OUTPUT DATA BUFFER
write a one to bit seven of the CONTROL BYTE

3. Command responses

When the 1810/1811 sends a message to the host relative motor position, for example it does so by loading sixteen bytes into its Output Data Buffer. Of the sixteen bytes, not all of them are necessarily significant. For example, requesting the state of the Trigger bits nets a response only two bytes long, the other fourteen bytes are filled with zeroes. The extra bytes in a message to the host need not be read: setting bit 7 of the Control Byte indicates to the 1810/1811 that the message has been read, any bytes left in the Output Data Buffer will be overwritten by the 1810/1811's next response.

C. Control Byte and Status Byte

The **Control** and **Status Bytes** (CB and SB, respectively) provide the user with a window into the 1810/1811's operating conditions. The CB establishes the conditions and the SB relates the conditions.

1. The Control Byte (CB)

The information controlled by the CB is,

BIT	DEFINITION
bits 0-3	unused
bit 4	Command Ready in the IDB
bit 5	Board Reset
bit 6	Clear the Interrupt Mask
bit 7	Message accepted from the ODB

Bit 4, when set, tells the Motor Controller that a command has been put into the IDB (Input-Data-Buffer) by the host. The Motor Controller will then clear bit 4 of the SB to indicate that the IDB is unavailable, read the data in the IDB, and then set bit 4 of the SB to indicate to the host that the IDB is again ready.

Bit 5 resets the Motor Controller. Setting this bit is equivalent to cycling power to the board or activating the Multibus reset line.

Bit 6 enables interrupts from the Motor Controller when set. This bit must be set even if the interrupt jumpers are installed on the the Motor Controller board. Interrupts are disabled upon power-up.

Bit 7, when set, tells the Motor Controller that a message, which was previously placed into the ODB by the Motor Controller, has been read by the host.

2. The Status Byte (SB)

The information available to the host via the SB is,

BIT	POWER-UP* STATE	DEFINITION
bit 0	Cleared	User defined status bit number 0
bit 1	Cleared	User defined status bit number 1
bit 2	Cleared	User defined status bit number 2
bit 3	Cleared	ODB Ready
bit 4	Set	IDB Ready
bit 5	Cleared	Fail
bit 6	Set	Interrupt Mask Set
bit 7	Cleared	Interrupt

* Power-up initialization takes about one second to complete.

Bits 0-2 are set and cleared by commands 08-0A (hex), respectively. They may be used by the host to indicate a Motor Controller status not directly supported by the 1810/1811 command language. For example, it is sometimes important to know how many moves have been made out of a sequence. The moves may be loaded into the sequence buffer with the **user defined status bits** set and cleared at critical places in the sequence. Then the host may read the status byte to see which of the user bits have been set, indicating where the Motor Controller is in the sequence.

Bits 3 and 4 are set when their corresponding data buffer is ready: Bit 3 is set when the ODB contains a message for the host, signalling the host to read the information it contains; and Bit 4 is set when the IDB is ready, telling the host it may write information to the IDB.

Bit 5, when set, tells the host that the Motor Controller has detected an internal failure from which it cannot recover. The only way to clear this bit is to reset the Motor Controller. This may be done by 1.) setting the reset bit (5) in the command byte, 2.) cycling power to the Motor Controller, or 3.) activating the Multibus reset line. Motor Controller outputs are in a high impedance state when the unit fails, so no pulses will be output to the motor. Exercising the test-switch function (FF hex) will also set this bit.

Bit 6 of the SB reflects the inverse of bit 6 of the CB. If bit 6 of the CB is cleared (masking interrupts), or has not been changed since power-up, SB bit 6 will be set. When this bit

is cleared, an interrupt condition will cause a Multibus interrupt. See the sections "Installation, Configuration Jumpers" and "Programming, Interrupts" for more details.

Bit 7, when set, indicates that an interrupt has occurred. The type of interrupt may be determined by looking at bits 3-5 of the SB. See the sections "Installation, Configuration Jumpers" and "Programming, Interrupts" for more details.

D. Interrupts

The 1810/11 Motor Controller has the capacity to interrupt the host on a variety of conditions. Commands 01 and A0-AF (hex) control these conditions, along with configuration jumpers JU18-20. Any command that causes an interrupt to be sent to the host does so via the **Message ready** interrupt. Selecton of interrupt jumpers is outlined in the section "Installation, Configuration Jumpers".

There are three basic interrupts:

Interrupt	Jumper	Status Bit
Input ready (IR)	JU20	4
Message ready (MR)	JU18	3
Controller failed (CF)	JU19	5

The **Input** ready (IR) interrupt signals the host that the 1810/11 is ready to accept data into its IDB. This interrupt is activated by installing jumper JU20 and enabling interrupts by setting bit 6 of the Command Byte. Whenever these conditions are met and the IDB is empty, an interrupt will be generated on whichever Multibus line has been selected by jumpers JU63-70 (INT7/-INT0/).

This interrupt may be used whenever it is necessary to write to the Motor Controller in quick succession, such as when using the Velocity Streaming Buffer, and you don't want to constantly read the Status Byte to see if the IDB is ready. **To clear the interrupt**, you must set bit four of the CB, indicating to the 1810/11 that you have put something into its IDB. **Beware**, however, that if you haven't written to the IDB the 1810/11 will try to interpret whatever may be in the buffer as a command. This will give you unpredictable results.

The **Message** ready (MR) interrupt signals the host that a message for it has been placed in the ODB. The host will then be interrupted whenever requested information has been prepared, or when an interrupt condition, defined by commands A0-AFh, occurs. The host should then read the ODB, and interpret the interrupt.

If the condition which caused the interrupt was a response

to a request from the host__such as a request for position information or Motor Controller status__the host's operating system should check the received information to see if it is what was expected. This is done by inspecting the first byte of the returned information. This byte has the same hex value as the command byte which was sent to the Motor Controller for the purpose of causing the interrupt. For example, if command 80 (hex) (request position relative to the beginning of the current move) is given, the first byte of the position information will be 80 (hex), followed by the position data. **This interrupt is cleared by** setting bit seven of the Control Byte.

The **Fail** interrupt indicates to the host that the Motor Controller is suffering from an internally detected ailment that it cannot recover from. When failed, the Motor Controller puts all of its outputs into a high-impedance state which stops all motor motion. The interrupt is caused by the activation of a hardware circuit called a Watchdog Timer. If this device is not "fed" every so often by normal operation of the code, it will cause a hard reset of the 1810/11. The only ways to recover from this condition__and clear the interrupt__ is to set bit five of the Control Byte, reset the entire Multibus, or cycle board power, all of which cause previously set **move parameters and commands to be lost.**

Some interrupt programming tips

Interrupts are very often desirable and at the same time hard to implement! Following are some of the things Compumotor has learned in using the 1810/11 since its introduction.

First of all, concerning interrupts and the 1810/11: it is a good practice to save the state of the interrupt enable bit (CONTROL BIT six), since you cannot read this bit. The bit you read when reading bit six of the STATUS BYTE is the interrupt mask bit, which is always the negative of the current value of the interrupt enable bit, sorry about that! Thus, before you write to the CONTROL BYTE you should check the state of this saved state. Let me illustrate with an assembly language program written for the 8086/88. The few lines below will accomplish setting the "Command Ready in the INPUT DATA BUFFER" bit in the CONTROL BYTE (CONTROL BIT four). (The examples below are not meant to be efficient or elegant, but simply illustrative.)

```
            dseg
STATE:      dbs 1           ;reserve a spot for the status byte

            cseg

              .             ;here we have completed writing an
```

```
                .           ;1810/11 command string to the INPUT
                .           ;DATA BUFFER

; Now we will set the "Command Ready" bit in the CONTROL BYTE to
; let the 1810/11 know we have a command for it to read in the INPUT
; DATA BUFFER.

                mov al,#010H    ;data to set the "Command Ready" bit
                cmp STATE,#0    ;see if 1810/11 interrupts are enabled
                jz  disabled    ;jump if 1810/11 interrupts are disabled

                or  al,#040H    ;set the interrupt enable bit also

disabled:       out CONT,al     ;hit the "Command Ready" bit

; When we hit bit four of the CONTROL BYTE the 1810/11 received an
; interrupt telling it to go read the command in the INPUT DATA
; BUFFER.  Currently the "INPUT DATA BUFFER available" bit in the
; STATUS BYTE (STATUS BIT four) is zero, which means the INPUT
; DATA BUFFER is not available.  In about one millisecond the
; buffer will become available, and we may send another command
; at that time.  (Buffer availability is indicated by STATUS
; BIT four.)
```

The above illustration works on the assumption that somewhere else in the program the variable STATE is being changed to reflect the current state of the 1810/11's interrupt enable bit. At the time that STATE is changed the interrupt enable bit would also be changed. Below I'll illustrate changing the value of STATE and the state of the interrupt enable bit for both cases, enabling interrupts and disabling interrupts.

```
                dseg
STATE:          dbs 1           ;reserve a spot for the status byte

                cseg

                .           ;here we have completed writing an
                .           ;1810/11 command string to the INPUT
                .           ;DATA BUFFER

; Now we will set the "Command Ready" bit in the CONTROL BYTE to
; let the 1810/11 know we have a command for it to read in the INPUT
; DATA BUFFER.

                pushf           ;save the state of the interrupt flag
                cli             ;disable interrupts
                mov al,#010H    ;data to set the "Command Ready" bit
                cmp STATE,#0    ;see if 1810/11 interrupts are enabled
```

```
                jz  disabled    ;jump if 1810/11 interrupts are disabled

                or  al,#040H    ;set the interrupt enable bit also

disabled:       out CONT,al     ;hit the "Command Ready" bit
                popf            ;reenable interrupts


                .
                .
                .


                pushf               ;save the interrupt flag
                cli                 ;disable interrupts
                mov STATE,#0ffH     ;get ready to enable interrupts
                mov al,#040H        ;at the 1810/11
                out CONT,al         ;enable interrupts
                popf                ;enable interrupts

                .
                .
                .

                pushf               ;save the interrupt flag
                cli                 ;disable interrupts
                mov STATE,#000H     ;get ready to disable interrupts
                mov al,#000H        ;at the 1810/11
                out CONT,al         ;disable interrupts
                popf                ;enable interrupts
```

As you can see, these routines restore the prior state of the interrupt flag with the "popf" instruction and also that interrupts were disabled upon entering the routines with the "pushf", "cli" instruction sequence. On some processors the interrupt flag is not so easily manipulated. In a background routine you might choose to use the "cli", "sti" instruction sequence since the state of the interrupt flag should be known at all times there.

E. Command Modes

1. Programming Modes

The 1810/11 has two basic operating modes: **indexer** and **velocity Streaming.** Each of these modes may be further divided into two "sub-modes."

Indexer mode allows individual execution of commands such as preset-distance moves and position reportbacks, or those com-

mands may be executed sequentially by putting them into a **sequence buffer.** Individual commands have attributes (described below) which allow or disallow a given command's execution either inside or outside of a sequence buffer.

The two sub-modes for Velocity Streaming mode are Velocity-**Time** Streaming and Velocity-**Distance** Streaming. These modes give the user great flexibility in controlling motor motion by allowing him (her) to specify the motor velocity for either fixed periods of **time** or **distance.**

a. Indexer Mode

The Motor Controller is put into Indexer mode upon power up or receipt of the command **Enter open-loop indexer mode** (50h). When in this mode the Motor Controller may be instructed to execute moves of a preset distance relative to the current position or an absolute position or constant velocity, or allow operation of the Compumotor 852 Joystick (commands 10h and 11h).

In Indexer mode the host must define a move before it may be executed. A move may be defined as relative, absolute or continuous. Relative and absolute moves are of a preset distance at a preset velocity and acceleration. The absolute position reference may be difined by command 30h.

Continuous moves are executed at a preset velocity and acceleration. Once started, a continous move will continue until **killed** (78h), **stopped** (70h), or the motor encounters an end of travel limit. This contrasts relative and absolute preset moves, which will end after a given distance.

The acceleration parameter specified for any of these moves is a linear acceleration from the starting velocity to the terminal velocity. The acceleration ramp is updated every two milliseconds to provide smooth motor response.

The velocity specified for a preset move is the **maximum** velocity for that move. A **trapezoidal** move consists of three parts: an acceleration ramp, a period of constant velocity, and a deceleration ramp. The acceleration and deceleration ramps will always be the same for preset moves, but may be respecified for each change in velocity of a constant velocity move. The constant velocity **portion** of a preset move has zero time allowed for it if the average velocity (**distance** move total / **time** move total) is more than half the specified move velocity.

Preset moves are performed with no overshoot or creeping up to the end point of the move. Pulses are cut off as soon as the preset number of pulses has been issued.

All move parameters are specified in motor pulses__per unit time for velocity, per unit time squared for acceleration, and pulses of distance. This allows any motor resolution to be used without any sort of **motor selection configuration jumpers** or **commands** that would let the 1810/11 know what resolution the motor is: It does not need to know.

A command has been included to allow the setting of a start/stop velocity (D6h). This allows use of 200/400 step motors without worrying about their resonant characteristics (which could stall the motor at low velocities.) Compumotors accelerate smoothly through all allowed velocity ranges.

b. Velocity-Streaming Mode

Velocity-streaming mode, once entered, allows the host to specify absolute motor velocity with reference to user-defined units of time or distance. This way, the user may create custom acceleration ramps (such as a parabolic curve that will compensate for reduced motor torque at high velocities) or move profiles. Inter-axial coordinated motion suitable for contouring may then be accomplished, or the motor may be made to follow a sensor whose output is read and interpreted by the host.

The data loaded into the velocity streaming buffer may be used only once. It is then lost. Entering velocity streaming mode destroys any currently defined sequence buffers.

For example: Suppose you need to move a standard Compumotor one revolution in 100 milliseconds. This requires an average acceleration of 400 revolutions per second squared, and a maximum velocity of 20 revolutions per second. If a linear acceleration ramp is used, the acceleration will have to be set to what the motor can achieve at maximum velocity. This is less than what it can do at lower velocities, so the move may not be possible in the time allowed, unless the acceleration is tapered off at the high velocities. A parabolic acceleration profile will give the motor a high acceleration at low motor velocities, and then lower it as the motor's velocity increases.

The following series of steps outlines a 25,000 pulse move (approximately), using the **velocity-time** streaming mode. These steps are fundamentally the same for both velocity-time streaming and velocity-distance streaming.

Step 1. Issue the command to enter **velocity-time** streaming, along with the requisite parameters.

```
Command:  52   Enter velocity-streaming mode
(in hex)  01   base frequency = 4.3690667 Mhz
          04   divide base frequency by 4 for effective clock
          00   \
          00    \  buffer empty alarm value = 0 bytes
          00    /
          00   /
          00   \   rate = 2 milliseconds
          02   /
```

Notes: The above command tells the 1810/11 to enter velocity-time streaming mode with a base frequency of about 4 Mhz, which will be divided by four for a one Mhz effective clock frequency. The highest frequency available for output is one-half of the effective clock frequency. The buffer alarm is set to send a message to the host when the velocity-streaming buffer is completely empty (the following routine should fit easily into the velocity-streaming buffer in one piece). The rate at which the motor velocity is updated is set to 2 milliseconds, for a very smooth motion profile.

Step 2: Find out the size of velocity-streaming buffer.

```
Command:  E2   Request number of free bytes
(in hex)
```

Notes: This step causes the 1810/11 to tell the host the amount of free space in its velocity-streaming buffer, and is included here for your reference, since the following data should fit easily into the buffer, which is about 600 decimal bytes long.

Step 3: Load the pre-calculated data into the velocity-time streaming buffer (until full__see step 2).

```
Command:  E1   Command name (load the v-s buffer)
(in hex)  07, 2E, 0E, 11, 14, A9, 1A, F7, 20, F9, 26, B0, 2C, 1C
          Parameter bytes

Command:  E1   Command name
(in hex)  31, 3D, 36, 13, 3A, 9E, 3E, DE, 42, D3, 46, 7D, 49, DC
          Parameter bytes

Command:  E1   Command name
(in hex)  4C, F0, 4F, B8, 52, 36, 54, 69, 56, 51, 57, ED, 59, 3F
          Parameter bytes

Command:  E1   Command name
(in hex)  5A, 46, 5B, 01, 5B, 72, 5B, 97, 5B, 72, 5B, 01, 5A, 46
          Parameter bytes

Command:  E1   Command name
```

(in hex) 59, 3F, 57, ED, 56, 51, 54, 69, 52, 36, 4F, B8, 4C, F0
Parameter bytes

Command: E1 Command name
(in hex) 49, DC, 46, 7D, 42, D3, 3E, DE, 3A, 9E, 36, 13, 31, 3D
Parameter bytes

Command: E1 Command name
(in hex) 2C, 1C, 26, B0, 20, F9, 1A, F7, 14, A9, 0E, 11, 07, 2E
Parameter bytes

Command: E1 Command name
(in hex) 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00
Parameter bytes

Notes: The above commands load the velocity-time streaming buffer with a parabolic move profile which goes from zero to 375.632 Khz (15 RPS in a standard Compumotor) and back to zero in 100 milliseconds, and moves about 25,000 steps in so doing. There are 50 parameter byte pairs; at two milliseconds per pair, that gives us 100 milliseconds to execute the entire sequence.

Step 4: Initiate buffer execution.

Command: 42 Perform the sequence buffer
(in hex)

Step 5: Wait for the message from the 1810/11 which indicates that the buffer has been completed (actually, the buffer empty alarm). The message will begin with 52 (hex), the command which caused the message to be sent.

The preceding example illustrates how to use velocity-time streaming. You will note that the distance the motor travels while the 1810/11 is in this mode cannot be guaranteed, as the Motor Controller is more concerned about time than distance. It does, however, attempt to give realistic information ultimately resulting in a very high acceleration 25,000 pulse move. To assure a move of the proper distance, the velocity-distance streaming mode should be used. For more information concerning the commands used above, see "Summary of the Commands."

Bogus Velocity Values

The EE (hex) command, "Define command to be executed during the velocity streaming buffer," allows some commands to be executed while streaming user data. This is done by specifying a specific velocity that will correspond to a specific command. The velocity is then never executed, the specified command is

executed instead. The restrictions are that the command must have no more than two parameter bytes and must be VALID-concurrent-with-VELOCITY-STREAMING. Obviously, a velocity value of zero cannot be used because the motor could then never come to rest.

Examples of the commands which could be executed while the velocity-streaming buffer is being executed are:

08	Write to the user defined status bits
09	Set user defined status bit
0A	Clear user defined status bit
0B	Set programmable output bit
0C	Clear programmable output bit
70	Stop motion
7C	Kill motion when trigger X goes active
81	Request current position
89	Request state of the limit switches
8A	Request state of the home limit switch
8B	Request direction of travel
8C	Request whether the motor is moving or not
94	Request state of the trigger inputs
A9	Interrupt on hitting the "+" limit switch
AA	Interrupt on hitting the "-" limit switch
AC	Interrupt on trigger X active

Suppose you would like the indexer to let you know when it has finished a certain portion of the velocity streaming data you have given it. To do this you might use the velocity **FF FF** as your bogus velocity value. This means that anywhere the velocity FF FF is found in the velocity-streaming buffer the motor will **not** be told to go to that velocity, rather, the command associated with that velocity value will be executed.

For example,

EE	command name
FF	MSB of velocity
FF	LSB of velocity
09	Set user defined status bit
02	Status bit three
00	Not needed for data, but must be present

The above command would signal the 1810/11 to set user status bit number three each time the velocity FF FF is encountered in the velocity streaming buffer. The example could be easily modified to clear the user-defined status bit instead of setting it. Then the selected bit could be toggled on and off to indicate the status of the execution velocity-streaming buffer.

Up to ten special or bogus velocities may be defined.

VALID-within-a-SEQUENCE-buffer

This attribute allows a command to be performed inside a sequence buffer, but does not exclude its execution outside of a sequence buffer. Attempting to place a command **without** this attribute into a sequence buffer will cause the command to be **ignored.**

VALID-concurrent-with-SEQUENCE-buffer-execution

This attribute is a kind of "sub attribute" to **VALID-in-INDEXER-mode.** Commands having this attribute may be performed **outside** of a sequence buffer **while** a **sequence buffer** is being performed. A **command cannot have this attribute and the attribute SINGULAR, because a** SINGULAR **command cannot be performed** outside of **a sequence buffer** during **sequence buffer execution.** This attribute does not exclude a command's use inside of a sequence buffer or in velocity streaming mode.

VALID-concurrent-with-VELOCITY-STREAMING

This attribute is similar to VALID-concurrent-with-SEQUENCE-buffer-execution. It is a "sub attribute," to VALID-in-VELOCITY-STREAMING-mode. Commands having this attribute may be performed **while** the velocity streaming buffer is being performed. This attribute does not exclude a command's use in **indexer mode.**

F. Command Language Syntax

A command consists of up to sixteen bytes: a single command byte followed by up to fifteen parameter bytes. Some commands require no parameter bytes and some require all fifteen of them, with many requiring at least one. A command requiring parameter bytes must have the correct number of parameter bytes (as specified in the Command Language), and they must be in the correct range, or the command cannot be interpreted.

Unrequired parameter bytes may be filled with zeros. Using any other character will cause the command to be ignored. For example, "perform move number 5" can be written to the 1810/11 indexer (in hex numbers)

as,
40, 05

or as,
40, 05, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00.

Both strings will accomplish the same result. (In actual use, the user would follow the loading of this command by setting bit 4 in the Control Byte. See "Programming, Control and Status

Bytes" for more information.)

The following format is used to describe the 1810/11's commands.

XX Y where XX is the command name and Y is the number of required parameter bytes (in hex).

Followed by each of the required parameter bytes (if any), in hex.

AA Written here will be an explanation of each of the
BB required parameter bytes.
CC
DD
..
..
..
FF There may be as many as fifteen required parameter bytes.

G. Advanced Features

The Model 1810 Indexer has some features which make it very powerful and yet easy to use. The ability to predefine moves and then to string them together in a sequence are two examples. Move zero is another, but in a different vein.

1. Pre-defined Moves

Instead of issuing new move parameters each time a different velocity, distance or acceleration is needed, pre-defined moves allow the user who knows his/her needs in advance to prepare ahead of time. Up to 50 moves may be pre-defined, and those moves may be of a preset distance or continuous, relative to the current motor position or relative to an absolute position.

If, for example, your application always requires a one revolution move, a two revolution move and a three revolution move, these could be pre-defined. Command C8 (hex) allows definition of moves having a preset distance, velocity and acceleration. Here is an example of how it is used.

```
C8   Command name
12   Number of move (50 maximum)
00   Velocity and acceleration ranges (Compumotor standard)
00   \
04    \ Velocity = 262,144 pulses/second
00    /
00   /
00   \
```

```
04    \ Acceleration = 262,144 pulses/second
00    /
00   /
00   \
00    \ Distance = 25,000 pulses
61    /
A8   /
```

Once the move is defined it may be executed by referring to it by number. The command to do this is 40 (hex) "Perform move number 'X'". To use it,

```
40   Command name
12   Command number.
```

A move remains defined until it is redefined or until the indexer is reset. This is true even in velocity-streaming mode, even though the moves are inaccessible in that mode. They will still be there when the indexer is returned to indexer mode.

The advantage of being able to quickly specify which move is needed is an obvious advantage of having the move predefined. Another advantage is that the move gets underway in less time. When the move is pre-defined it is also pre-calculated. This is to say that all required calculations regarding the move profile are done in advance. Some indexers require as long as 30 milliseconds to calculate moves. Because the 8088 microprocessor has a divide instruction it can do it in less than half that time. But when pre-defined moves are used motor motion will begin within five milliseconds of receipt of the command. The time savings netted by using pre-defined moves can mean the difference between making the move in the required time and not making the move, especially when the time allotted for the entire move is under one-half second.

2. Sequence Buffers

Sequence buffers allow amplification of effort. They allow the listing of commands for execution from a single command. In its simplest form, as sequence buffer consists of a list of commands which are to be repeatedly performed in sequence. The commands can be move commands, of course, but also the user-defined status bits may be set or cleared, the programmable output bits may be manipulated, series of commands may be put in loops and execution of the sequence may be halted to wait for the occurrance of an external trigger.

An example of how to define a sequence buffer is,

```
D9   Begin sequence buffer definition
08   Number of the sequence (15 maximum)
```

```
40   Execute a predefined move
12   Move number 12 (hex)

0B   Set programmable output bit
01   Output bit number one

40   Execute a predefined move
05   Move number 05 (hex)

0C   Clear programmable output bit
01   Output bit number one

20   Repeat the following (this command starts a loop)
03   --- three commands
00   --- fifteen times
0F

40   Execute a predefined move (first part of the loop)
08   Move number 08 (hex)

2A   Wait (second part of the loop)
02   --- for two seconds

83   Request current absolute position (third part of loop)

D8 End sequence buffer execution
```

The above example would define a sequence to be executed upon receipt of a single command, **41** (hex) "Perform a sequence buffer". You'll notice that pre-defined moves can be used in a sequence. This saves memory space. Even though up to fifteen sequence buffers may be defined, they all have to fit into a fixed amount of available memory space. By using pre-defined moves less of that memory space is used up in defining commands. Also, any sequence can call on any pre-defined move: pre-defined moves are not limited for use with a single sequence buffer.

Sequence buffers are lost when velocity streaming mode is entered. Room can be made available for new sequences, however, by deleting old sequences with the DA (hex) command, "Delete sequence buffer X". The sequence buffer is about 600 decimal bytes long.

3. Move Zero

Move zero may be thought of as the unsung hero of the pre-defined moves. This is the move that uses the default acceleration and velocity defined in command 31 (hex). Move zero may be called using commands 33 and 34 (hex) which are the "Go to relative position X", and "Go to absolute position X",

respectively. **These are the only preset moves which may be executed while in velocity streaming mode.**

Unfortunately, move zero cannot be called by the "Perform move X" command.

SPECIFICATIONS

Physical Description

Board to Board Spacing: Center to center spacing of 0.6 inches +/- 0.02 inches (15 mm +/-.5 mm).

Board Thickness: 0.062 inches +/- 0.005 inches (1.5 mm +/-.1 mm).

Component Lead Length: 0.093 inches maximum.

Component Height: 0.420 inches (10.6 mm). Electrically conductive components are limited to be reduced to 0.400 inches (10.1 mm).

Dimensions: 12.00 inches by 6.75 inches (305 mm by 172 mm).

Weight: 0.9 lb (0.4 kg), Shipping: 3.0 lb (1.4 kg)

Environmental Data

Operating Temperature: 32° to 131°F (0° to 55°C)

Storage Temperature: -22° to 185°F (-30° to 85°C)

Humidity: 10 to 90%, non-condensing

Performance

Stepping Accuracy:	+/- 0 steps of preset distance
Velocity Accuracy:	+/- 0.01% of set rate
Velocity Repeatability:	+/- 0.01% of set rate
Velocity Range:	0.017 steps/sec to 1,092,267 steps/sec (in 6 user selectable ranges)
Acceleration Range:	0.00001 steps/sec^2 to infinity (in 3 user selectable ranges)

Electrical

Power: +5 Volts DC at 2.7 Amps nominal, 3.0 Amps maximum (Supplied by the Multibus backplane.), exclusive of any current drawn by devices attached to the 1810/11 other than a Compumotor Motor-Driver.

Inputs and Outputs

There are seven external connectors on the 1810/11 indexer, six located on the top board edge and the MULTIBUS P1 connector. The MULTIBUS P2 connector is, not used. The six top edge connectors are: Auxiliary, Joystick, Motor-Driver, Chassis ground, Encoder and Expansion. The Expansion connector

is reserved for Compumotor use and so is not listed here. For pinout listings of these connectors see Appendix C.

The electrical characteristics and functions described in this section use the term **LS TTL equivalent load.** This means an input at a TTL high level (2.0 volts to 5.5 volts) will sink 20 microamps of current, and an input at a TTL low level (0.8 volts to 0 volts) will source 0.4 milliamps of current.

The following **format** is used to describe the six top edge connectors' signals.

SIGNAL (DIRECTION WITH RESPECT TO THE 1810/11 INDEXER)
ACTIVE STATE (omitted if this is undefineable)

Paragraph describing the signal's function.

Description of the signal's electrical characteristics.

The seven connectors to be described, in the order of their description, are the AUXILIARY connector, the ENCODER connector (on 1811 boards only), the EXPANSION connector, the JOYSTICK connector, the MOTOR/DRIVER connector, the CHASIS GROUND CONNECTOR, and the MULTIBUS P1 connector.

AUXILIARY CONNECTOR

The AUXILIARY connector is a 20-pin flat-cable header. All outputs are 3.7 volts nominal, 60 milliamp maximum sink or source, short-circuit protected, unless otherwise noted. Inputs are two LS TTL equivalent loads or less with 3.3 Kohm pullups and 0.1 microfarad noise filter capacitors to ground, unless otherwise noted.

SHUTDOWN IN (INPUT)
ACTIVE LOW

Activating this input can cause the motor to shutdown. This input must first be enabled via a command to the indexer, "enable SHUTDOWN IN." If the enabling command is not issued activating SHUTDOWN IN will have no effect. This input is shared with the SHUTDOWN IN input on the JOYSTICK connector and, therefore, may not be used if a Joystick is being used. If one issues the command "turn on/off the remote power shutdown" and this input has been enabled the command will have no effect. Thus, control over the remote power shutdown output bit (SHUTDOWN on the MOTOR/DRIVER connector) belongs exclusively to either this

input or the aforementioned command, not to both at the same time.

This input is shared with the SHUTDOWN IN input on the JOYSTICK connector and may not be used if a joystick is connected to the indexer.

CW LIMIT (INPUT)
ACTIVE HIGH

Extreme mechanical limit stop. Activation of this input causes the indexer to stop any motor motion in the clockwise direction within 200 microseconds. The signal must remain active for a minimum duration of 75 microseconds.

This input may be wired to a normally-closed limit switch of suitable contact rating. The polarity, normally-closed, is "fail-safe" in that an open connection causes the motor to stop if moving clockwise. An on-board jumper is provided to bypass this input if no limit switch is connected to the input.

CCW LIMIT (INPUT)
ACTIVE HIGH

Same as CW LIMIT except it affects motion in the counterclockwise direction.

HOME LIMIT (INPUT)
ACTIVE HIGH OR LOW

This input is not used to stop motion as the CW and CCW LIMIT inputs are. It identifies a "HOME position" that may be used as a reference point. For example, exercising the command "go to the HOME limit switch" will cause the indexer to search for the HOME limit switch. This input is shared with the HOME ENABLE input on the ENCODER connector and, therefore, is not available if the board is being used in a closed-loop configuration (Closed-loop is available only on 1811 boards).

This input is not jumper defeatable.

TRIGGER #1 and #6 (INPUT)
ACTIVE HIGH EDGE

These are user defined inputs. There are several commands that can be issued to the indexer that pertain to these inputs, such as "wait for trigger 1 to go active."

These are "high-speed" inputs, response will be within 200 microseconds for #1 and 400 microseconds for #2. To assure proper operation, the input signal should be held high for 1 millisecond.

May be connected to a normally-closed limit switch like those connected to the CW and CCW LIMITs.

TRIGGER #2-#5 (INPUT)
ACTIVE HIGH OR LOW LEVEL

These are user defined inputs. Unlike TRIGGERs #1 and #2, TRIGGERs #2-#5 are not "high-speed" inputs, they are "low-speed" inputs. They may be used with the command "wait for trigger X to go active high (or low)." The worst case response to them is 3 milliseconds. The signal must be active for at least 4 milliseconds to ensure proper response. TRIGGERs #3 and #4 are in parallel with DRIVE FAULT and SLIP FAULT on the Motor-Driver connector. Since these inputs are shared with inputs on the Motor-Driver connector they will not be available as user defined inputs if the Motor-Driver enhancements are used.

The active sense of these inputs is determined by the user.

PROGRAMMABLE OUTPUTs #1 and #2 (OUTPUT)
ACTIVE LOW OR HIGH

These outputs are user defineable outputs via commands to the indexer, such as "set programmable output 1."

+5 VOLTS (OUTPUT)

Five volt supply. This supply voltage is connected to the MULTIBUS +5 volts via a one ohm resistor. It is limited to 250 milliamps.

SHIELD

Connected to the chassis ground via the two prong CHASIS GROUND CONNECTOR. Provides a SHIELD GROUND point for external cables coming inta a "D" bulkhead connector without splitting that wire out of the flat cable (ribbon cable); if that shield is not grounded elsewhere.

ENCODER

The ENCODER connector is a 20-pin flat-cable header. It is used only with the closed-loop option (available only on the 1811 board). All outputs are capable of driving twenty LS TTL equivalent loads, unless otherwise noted. Inputs are two LS TTL equivalent loads or less with 3.3 Kohm pullups and a 0.1 microfarad noise filter capacitor to ground, unless otherwise noted.

CH. A+ and B+ (INPUT)

Refers to 90° out of phase signals that are normally provided by incremental encoders.

High impedance (100k ohm nominal) input from one phase of a quadrature encoder signal. Normally a TTL level compatible signal.

CH. A- and B- (INPUT)

High impedance (100k ohm nominal) input optionally connected to a complementary output of a quadrature encoder signal. Should be left unconnected if this input is not used or not required.

CH. Z+ (INPUT)

Identical to CH. A+ (above) except that this input applies to the HOME channel of the encoder signal.

CH. Z- (INPUT)
ENCODER-15

Identical to CH. A- (above) except that this input applies to the HOME channel of the encoder signal.

HOME ENABLE (INPUT)
ACTIVE HIGH

This signal is used with systems having Channel Z connected to a rotary encoder. This input marks the encoder revolution in which the HOME bit from CH. Z is valid.

This input is usually connected to a normally-closed limit switch of suitable contact rating. This input is shared with the HOME LIMIT input on the AUXILIARY connector and therefore cannot be used if the HOME LIMIT input on the

AUXILIARY connector is being used.

STEP OUT (OUTPUT)
ACTIVE HIGH

STEP output from the quadrature detector. This output can be used to monitor encoder pulses after the quadrature detector has conditioned the quadrature input signals. The encoder circuitry is a times four detector, therefore, four pulses will be seen for every line of the encoder. Pulses are nominally 229 nanoseconds wide with a maximum repitition rate of 4.37 MHz.

DIRECTION OUT (OUTPUT)

DIRECTION output from the quadrature detector. This output can be used to track the direction of the pulses coming out of STEP OUT (above). A jumper on the 1810/11 board inverts the polarity of this signal.

+5 VOLTS (OUTPUT)

This supply voltage is connected to the MULTIBUS +5 volt signal via a one ohm resistor. It is limited to 250 milliamps of current.

SHIELD

Connected to chassis ground via user's connection to the SHIELD CONNECTOR. This line provides a shield ground point for an external cable coming into a "D" connector, if that shield is not grounded at the encoder or elsewhere.

JOYSTICK

This is a 20-pin flat cable header intended to allow convenient connection to the Compumotor two-axis Joystick, Model 852. The flat cable is normally terminated with a fifteen pin "D" bulkhead connector which connects to the standard Compumotor 2100 to Joystick cable. All outputs are 3.7 volts nominal, 60 milliamp maximum sink or source, short-circuit protected, unless otherwise noted. All inputs are two LS TTL equivalent loads or less with 3.3 Kohm pullups and have 0.1 microfarad noise capacitors to ground, unless otherwise noted.

REMOTE ENABLE (INPUT)
ACTIVE LOW

This signal is unused by the 1810/11. It is used by the Joystick to request control of the motor. It is part of a handshake used between an 852 Joystick and a 2100 indexer.

REMOTE ACKNOWLEDGE (OUTPUT)
ACTIVE HIGH

This signal is the other half of the above handshake. When the 1810/11 activates this signal it is giving control of the Motor-Driver to the Joystick. The 1810/11 will keep track of pulses to the Motor-Driver and will continue to monitor the CW and CCW limit switch inputs. (Striking a limit switch will not disable the Joystick.) Disabling the Joystick from the 1810/11 causes this signal to go inactive.

AT ZERO (INPUT)
ACTIVE HIGH

When this signal is active it signifies to the 1810/11 that there are no motor pulses coming in the STEP IN line, so the Joystick is set to zero velocity. This signal is required to warn the 1810/11 of possible directional changes. This signal must be active for at least four milliseconds before a direction change occurs (at DIRECTION IN) and for at least four milliseconds after a direction change occurs.

SHUTDOWN IN (INPUT)
ACTIVE LOW

This input can cause the REMOTE SHUTDOWN signal on the MOTOR/DRIVER connector to become active. A command to the indexer and the SHUTDOWN IN signal on the AUXILIARY connector are two other ways one can cause REMOTE SHUTDOWN to go active. This input is physically shared with the SHUTDOWN IN signal on the AUXILIARY connector and, consequently, cannot be used if that signal is being used.

If a Joystick is connected to the 1810/11 the SHUTDOWN IN signal on the AUXILIARY connector cannot be used and, vice-versa, if the SHUTDOWN IN signal on the AUXILIARY connector is being used a Joystick cannot be connected

to the JOYSTICK connector.

STEP IN (INPUT)
ACTIVE HIGH

Pulse train from a Joystick. Pulse width is limited to 250 nanoseconds and repitition rate of 2 megahertz. Pulses are routed directly to the STEP output on the MOTOR/DRIVER connector when the Joystick has been enabled by the appropriate command to the 1810/11 indexer. Therefore, one will see out of STEP (on the MOTOR/DRIVER connector) exactly what one puts into this input when the Joystick is enabled.

There is no 0.1 microfarad filter capacitor to ground on this input.

DIRECTION IN (INPUT)

Direction signal from the Joystick. When this input is high it is interpreted as a request for motion in the CW direction. When this input is low it is interpreted as a request for motion in the CCW direction. It is assumed that this signal will not change unless it is both preceded and followed by an active AT ZERO signal. AT ZERO must be active for at least four milliseconds before and four milliseconds after the DIRECTION IN signal changes.

There is no 0.1 microfarad filter capacitor to ground on this input.

JOYSTICK +5 VOLTS (INPUT)

This supply voltage is connected to the MULTIBUS +5 volt signal via a one ohm resistor. It is limited to 250 milliamps of current. The Compumotor model 852 Joystick uses this line to power opto-isolators.

SHIELD

Connected to chassis ground via user's connection to USER CHASSIS GROUND CONNECTOR. Provides a shield ground point for external cables coming into a "D" bulkhead connector without splitting that wire out of the flat cable (ribbon cable).

MOTOR/DRIVER

A 26-pin flat cable header. The flat cable may be terminated with a 9-pin "D" bulkhead connector for convenient connection to present Compumotor driver cables, or with a 25-pin "D" to accommodate future Compumotor drivers with added features. Outputs are 3.7 volts nominal, 60 milliamp maximum sink or source, short circuit protected, unless otherwise noted. Inputs are two LS TTL equivalent loads with 3.3 Kohm pullup resistors and 0.1 microfarad noise filter capacitors to ground, unless otherwise noted. Returns for the following outputs are differentially driven, they are not grounds.

STEP (OUTPUT)
ACTIVE HIGH

Step pulse to Motor-Driver. Pulse width is a function of chosen velocity range with minumum width equal to 229 microseconds at a maximum repitition rate of 2.18 megahertz. Pulse width is always one half the period of the maximum frequency out for the chosen velocity range. (Velocity ranges are a function of move definitions and of velocity-streaming mode definitions.)

DIRECTION (OUTPUT)

This signal is a TTL high when motion in the CW direction is expected and a TTL low when motion in the CCW direction is expected.

REMOTE SHUTDOWN (OUTPUT)
ACTIVE HIGH

Compumotor drives will turn off current to the motor's windings when this signal goes active.

CW STEP (OUTPUT)
ACTIVE LOW

Same as STEP output except active only for clockwise motion. This output is intended for driving non-Compumotor drives.

This output is an open-collecter driver. A maximum pullup voltage of 40 volts may be used and it will sink a maximum of 300 milliamps.

CCW STEP (OUTPUT)
ACTIVE LOW

Same as STEP output except active only for counterclockwise motion. This output is intended for driving non-Compumotor drives.

This output is an open-collector driver. A maximum pullup voltage of 40 volts may be used and it will sink a maximum of 300 milliamps.

MOTOR/DRIVER +5 VOLTS (INPUT)

This supply voltage is connected to the MULTIBUS +5 volt signal via a one ohm resistor. It is limited to 250 milliamps of current.

SHIELD

Connected to chassis ground via user's connection to USER CHASSIS GROUND CONNECTOR. Provides a shield ground point for external cables coming into a "D" bulkhead connector without splitting that wire out of the flat cable (ribbon cable).

MULTIBUS P1

This connector is the main MULTIBUS connector. All loading, drive capabilities, and timing of signals on this connector conform to the MULTIBUS specifications as outlined in the "INTEL MULTIBUS SPECIFICATION." This publication is available from:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Ask for publication number 9800683

Section three and Appendix B of the above publication describe in detail the requirements of all of the defined MULTIBUS signals on the P1 connector. The following MULTIBUS signals are used on the 1810 indexer P1 connector.

D7/ through D0/	Eight bit data bus.
ADR13/ through ADR0/	Twenty bit address bus.

INT7/ through INT0/	Eight interrupt lines.
INIT/	System reset line.
IORC/ and IOWC/ or MRC/ and MWC/	I/O read command and I/O write command or memory read command and memory write command.
XACK/	Data transfer acknowledge.
AACK/	Advanced acknowledge.
INH1/ and INH2/	Inhibit one and Inhibit two.

OPEN-LOOP COMMAND DEFINITIONS

Following is a list of the commands that can be issued to the 1810 Indexer. Commands for the 1811 Indexer are listed in Appendix E, Closed-Loop commands. Each of the commands is a single binary byte (represented here by hex numbers), giving a maximum of 256 commands. You will note that not all of the commands have been defined. Undefined commands will be ignored. Some commands require additional bytes, called parameter bytes, in order to completely specify the command.

The format of the list of commands is (horizontally),

1.) the command number (in hex),
2.) the number of parameter bytes required (in hex), and
3.) the command's definition.

00	0	Null command, does nothing
01	1	Interrupt "X"
08	1	Write "X" to the user defined status bits
09	1	Set user defined status bit number "X"
0A	1	Clear user defined status bit number "X"
0B	1	Set programmable output bit number "X"
0C	1	Clear programmable output bit number "X"
0D	1	Write "X" to the programmable output bits
0F	3	Define bit "X" to indicate state "Y"
10	0	Disable the Joystick
11	0	Enable the Joystick
12	1	Turn off/on remote power shutdown
13	1	Disable/enable the "remote shutdown in" bit
14	1	Set CW motion equal to +/-
20	3	Repeat the following "X" commands "Y" times
21	1	Repeat the following "X" commands until a CONTINUE is received
28	1	Wait for a CONTINUE
29	4	Wait "X" milliseconds
2A	2	Wait "X" seconds
2B	2	Wait "X" minutes
2C	2	Wait for TRIGGER "X" to go active
2E	4	Define the absolute open-loop postion as "X"
30	0	Define present position as the absolute zero position
31	A	Define default velocity & default acceleration
32	1	Perform the default move (trapezoidal continuous)
33	4	Go to relative position "X" at default velocity and acceleration
34	4	Go to absolute position "X" at default velocity

		and acceleration
38	4	Define HOME location and final search parameters
39	1	Go HOME at the default velocity and acceleration
40	1	Perform move number "X"
41	1	Perform sequence buffer "X"
42	0	Perform the velocity streaming buffer
48	0	CONTINUE (perform the next command)
50	0	Enter open loop indexer mode
51	6	Enter velocity-distance streaming mode
52	8	Enter velocity-time streaming mode
53	8	Enter distance-time streaming mode
70	0	STOP motion
71	0	Discontinue the sequence buffer
72	0	Suspend the sequence buffer; wait for a CONTINUE to resume
73	0	Discontinue any singular command currently being performed
74	3	STOP motion when TRIGGER "X" goes active
75	3	Discontinue the sequence buffer when TRIGGER "X" goes active
76	3	Suspend seq. buffer when TRIGGER "X" goes active; wait for CONTINUE
77	3	Discontinue any singular command when TRIGGER "X" goes active
78	0	KILL motion
79	0	KILL the sequence buffer
7A	0	KILL current sequence singular command; wait for a CONTINUE
7B	0	KILL the velocity streaming buffer
7C	3	KILL motion when TRIGGER "X" goes active
7D	3	KILL the sequence buffer when TRIGGER "X" goes active
7E	3	KILL current sequence singular command when TRIGGER "X" goes active; wait for a CONTINUE
7F	3	KILL the velocity streaming buffer when TRIGGER "X" goes active
80	0	Request position relative to the beginning of the current move
81	0	Request position relative to the end of the current move
82	0	Request position relative to the HOME limit switch

83	0	Request position relative to the absolute zero position
84	0	Request current direction
85	0	Request current velocity
86	0	Request current acceleration
88	0	Request current move status
89	0	Request state of the limit switches
8A	0	Request state of the HOME limit switch
8B	0	Request direction of travel
8C	0	Request whether motor is moving or not moving
8D	0	Request whether motor is at constant, nonzero velocity or not
8E	0	Request whether motor is or is not accelerating
8F	0	Request whether motor is or is not decelerating
90	0	Request present mode
91	1	Request move parameters for move number "X"
92	1	Request commands stored in the sequence buffer
93	0	Request state of the move definitions (complete or incomplete)
94	0	Request state of the TRIGGER inputs (2-5)
95	0	Request state of the Joystick status input
97	0	Request the state of the programmable output bits
A0	1	Interrupt at the start of the next move
A1	1	Interrupt at the start of every move
A2	1	Interrupt at constant nonzero velocity of the next move
A3	1	Interrupt at constant nonzero velocity of every move
A4	1	Interrupt at the next end of motion
A5	1	Interrupt at every end of motion
A8	1	Interrupt the next time the motor hits the + limit
A9	1	Interrupt every time the motor hits the + limit
AA	1	Interrupt the next time the motor hits the - limit
AB	1	Interrupt every time the motor hits the - limit
AC	2	Interrupt on TRIGGER "X" active
AF	0	Inhibit all of the above interrupts
C8	E	Define move "X," define it as a relative, trapezoidal move
CB	E	Define move "X," define is as an absolute, trapeziodal move
CE	B	Define move "X," define it as a continuous, trapezoidal move
D6	4	Define the start/stop velocity
D7	1	Delete move "X"
D8	0	End definiton of sequence buffer
D9	1	Begin definition of sequence buffer "X"
DA	1	Delete sequence buffer "X"

E0	C	Place data into the velocity-distance buffer
E1	E	Place data into the velocity-time buffer
E2	0	Request # of free bytes in velocity-streaming/sequence buffer
E3	F	Place the following data into the distance-time buffer
EE	5	Define command to be executed during the velocity streaming buffer. (or Define bogus velocity streaming value)
FD	0	Request software part number and revision
FE		reserved (self-test report)
FF	0	Perform the test switch function

Command Definitions

00 0 Null command, does nothing

This command does absolutely nothing. It is necessary for indicating a 'no command exists' condition. If this command is sent to the indexer nothing will happen! The only place this command has any use at all is when it is used in conjunction with the EE command, which defines commands to be executed during execution of the velocity streaming buffer.

01 1 Interrupt "X"

00 -> FF, the interrupt number, (an identifier)

This command is a simple interrupt. It will place a message into the OUTPUT DATA BUFFER to be read by the host and generate a "message ready interrupt" if this interrupt is enabled. It may be used to indicate a location within a sequence buffer or for debugging the "message ready interrupt." The parameter byte, which must be supplied, is an identifier for the user. After power up or reset interrupts are disabled.

The following list illustrates the message one would receive in the OUTPUT DATA BUFFER as a result of this command.

01	identifies the command which generated the message
00 -> FF	the parameter byte supplied with this command ("X")
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

08 1 Write "X" to the user defined status bits

00 -> 07, state one wishes to set the three user defined status bits to

The three user defined status bits (which can be read in the indexer's STATUS BYTE) will reflect the parameter byte. For example, if one sends the command string 08h, 05h user bit zero will be set, user bit one will be cleared, and user bit two will be set. Since there are three user defined status bits the parameter byte may only be 00h through 07h, any other parameter byte is considered incorrect and ignored. After power on or reset the user defined status bits are set to zero.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-within-a-SEQUENCE-buffer
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

09 1 Set user defined status bit number "X"

00 -> 02, user defined status bit number "X"

Set the specified user defined status bit. The difference between this command and the above command (08h) is this command acts on a specific bit without affecting the other two user defined status bits in the STATUS BYTE, while the above command (08h) acts on all three user defined status bits at once. The parameter byte indicates which user defined status bit is to be set. The three user defined status bits are labeled 0, 1, and 2; therefore, the parameter byte may be a 00h, a 01h, or a 02h. Any other parameter byte is considered incorrect and the command will be ignored. After power on or reset the user defined status bits are set to zero.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VELOCITY STREAMING MODE VALID
VALID WHILE PERFORMING THE VELOCITY STREAMING BUFFER

0A 1 Clear user defined status bit number "X"

00 -> 02, user defined status bit number "X"

Clear the specified user defined status bit. The difference between this command and the above command (08h) is this command acts on a specific bit without affecting the other two user defined status bits in the STATUS BYTE, while the above command (08h) acts on all three user defined status bits at once. The parameter byte indicates which user defined status bit is to be cleared. The three user defined status bits are labeled 0, 1, and 2; therefore, the parameter byte may be a 00h, a 01h, or a 02h. Any other parameter byte is considered incorrect and the command will be ignored. After power on or reset the user defined status bits are set to zero.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

0B 1 Set programmable output bit number "X"

01 or 02, programmable output bit number "X"

Set the specified programmable output bit. The parameter byte indicates which programmable output bit is to be set. The two programmable output bits are labeled 1 and 2; therefore, the parameter byte may be a 01h or a 02h. Any other parameter byte is considered incorrect and the command is ignored. After power on or reset the programmable output bits are set.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

1st part 69 pgs

0C 1 Clear programmable output bit number "X"

01 or 02, programmable output bit number "X"

Clear the specified programmable output bit. The parameter byte indicates which programmable output bit is to be cleared. The two programmable output bits are labeled 1 and 2; therefore, the parameter byte may be a 01h or a 02h. Any other parameter byte is considered incorrect and the command will be ignored. After power on or reset the programmable output bits are set.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

0D 1 Write "X" to the programmable output bits

00, 02, 04 or 06
Bit 1 corresponds to programmable output bit 1.
Bit 2 corresponds to programmable output bit 2.
All other bits of the parameter byte are undefined and are ignored.

When bit 1 or bit 2 is set, the corresponding programmable output bit will be set; when bit 1 or bit 2 is cleared, the corresponding programmable output bit will be cleared. This command allows changing the state of the programmable output bits simultaneously, using a single command.

There is no "don't care" state associated with either of the bits: using this command affects the state of **both** programmable output bits. If you wish to write to only one of the programmable output bits and not affect the state of the other programmable output bit use the 0B and 0C commands which will set and clear the programmable output bits individually.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-within-a-SEQUENCE-buffer
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

0F 3 Define bit "X" to indicate state "Y"

00 -> 04 indicates which bit will be used to indicate the desired state. Values greater than 4 are not allowed and will cause the command to be ignored. Parameter byte values are defined as follows,
- 00 corresponds to user definable bit 0,
- 01 corresponds to user definable bit 1,
- 02 corresponds to user definable bit 2,
- 03 corresponds to programmable output bit 1,
- 04 corresponds to programmable output bit 2.

00 -> FF indicates which indexer state will be displayed by the chosen definable bit (chosen by parameter byte 1 above). The code numbers indicating the desired indexer state are,
- 00 indicates the null code,
- 01 indicates the moving state,
- 02 indicates the performing-a-singular-command state,
- 03 indicates the performing-the sequence-buffer state,
- 04 indicates performing-the-velocity-streaming-buffer state,
- 05 indicates the accelerating state,
- 06 indicates the decelerating state, and
- 07 indicates the non-zero constant velocity state.

- 80 indicates the motor is within the user-definable closed-loop deadband,
- 81 indicates the state of the Joystick request input.

Codes between 00 and 7F are very accurate state indicators. Code numbers between 80 and FF are less accurate state indicators.

zero or non-zero indicates the active level of the bit. Non-zero means active high, 00 means active low.

This command is used to indicate that a given state is occurring within the 1810 Indexer. For instance, you can indicate the state of moving/not-moving through user-definable bit number one by sending the command sequence 0F 01 01 FF. This sequence means: "When the motor is moving, set user-definable bit number one, and when the motor is not moving, clear user-definable bit number one." More than one programmable bit may indicate the same state at any time, e.g., you could have the

indexer display the state of the Joystick request input by both a programmable output bit and a user-definable bit.

It is expected that the 0F command will be sent once only during initialization. It is an extremely powerful command. It can be used in place of requesting information and reading the response to a request. For instance, the 8C command is commonly used to determine whether the motor is moving or not. Using the 8C command involves waiting for the input buffer to be ready to receive a command, sending the 8C command, indicating the command has been sent, waiting for the message to return, reading the message, and indicating that you have read the message. There are quite a few steps involved here and they usually take about 5+ milliseconds to accomplish.

Using the 0F command, you can simply read the status byte, look at the previously defined bit, and know at that instant whether or not the motor is moving. In essence, it gives you a much more accurate view of the state of the machine. It also involves a much simpler and faster process for determining that state.

There is a way to "undo" previously programmed codes. You can do this by using the null code. For instance, if user bit 0 is being used to indicate the moving/not moving state, and you later want it not to indicate any state, i.e., remain constant, send a 0F 00 00 00.

This sequence says, "define user bit 0 to indicate the null code state, and leave the bit in its previous state." The last parameter byte has no effect. The first parameter byte indicates which bit to be defined, in this case, user definable bit 0. The second parameter byte, 00, means we wish to delete that bity from indicating any state. It is not expected that the null code will be used very often.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-within-a-SEQUENCE-buffer
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

10 0 Disable the Joystick

This command will disable use of the model 852 Joystick if a Joystick is currently enabled. Disabling of the Joystick occurs whether or not the motor is moving. If the Joystick has not been enabled this command has no affect. After power on or reset the Joystick is disabled.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

11 0 Enable the Joystick

This command will enable use of the model 852 Joystick. The Joystick may not be enabled if the remote power shutdown is active or if motion is occuring. When the Joystick is enabled the indexer tracks the Joystick's position, therefore, the distance the Joystick has moved the motor may be determined by interrogating the indexer. Moving the Joystick from zero velocity to a nonzero velocity and back to zero velocity is considered a move, requesting the relative position following this sequence of events results in a report of the total distance traveled during that sequence. The Joystick's pulse frequency cannot be modified by the 1810/11 indexer as can be done with the 2100 indexer, the pulse frequency out of the 1810/11 when the Joystick is enabled is identical to the pulse frequency in from the Joystick. After power on or reset the Joystick is disabled.

The indexer considers having the Joystick enabled to be equivalent to performing a move. In other words, any command that cannot be performed when the motor is moving can also not be performed when the Joystick is enabled. This is true even if the motor is not moving while the Joystick is enabled.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

12 1 Turn off/on remote power shutdown

00 or nonzero, 00 = "off" and nonzero = "on"

Turns on or off the remote power shutdown to the motor/driver according to the parameter byte passed. (Remote power shutdown is a feature of Compumotor microstep drives, enabling remote power shutdown turns off current to the motor's windings and removes the motor's holding torque.) A nonzero parameter byte will turn on remote power shutdown and a 00 parameter byte will turn off remote power shutdown. There are three means by which one may activate remote power shutdown: the model 852 Joystick, the auxiliary "remote shutdown in" bit, and this command. If the Joystick is enabled its remote power shutdown signal will have highest priority; if the "remote shutdown in" bit on the auxiliary connecter is enabled it will have priority; this command (12h) has lowest priority. This command will not be performed if the Joystick is enabled, if the remote shutdown in bit is enabled, or if the motor is moving. After remote power shutdown is turned on no moves may be performed nor may the Joystick be enabled, until remote power shutdown is turned off. After power on or reset the remote power shutdown is off.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

13 1 Disable/enable the "remote shutdown in" bit

00 or nonzero, 00 = disable and nonzero = enable

The "remote shutdown in" bit is an input on the auxiliary connector of the 1810/11 indexer, its function is to cause shutdown of the motor. (Remote power shutdown is a feature of Compumotor microstep drives, enabling remote power shutdown turns off current to the motor's windings and removes the motor's holding torque.) A nonzero parameter byte will enable the "remote shutdown in" bit and a 00 will disable the "remote shutdown in" bit on the auxiliary connector. There are three means by which one may activate remote power shutdown: the model 852 Joystick, the auxiliary "remote shutdown in" bit, and the previous command (12h). If the Joystick is enabled its remote power shutdown signal will have highest priority; if the "remote shutdown in" bit on the auxiliary connecter is enabled, via this command, it will have priority; the previous command

(12h) has lowest priority. This command is ignored if the Joystick is enabled. After remote power shutdown is turned on no moves may be performed nor may the Joystick be enabled until remote power shutdown is turned off. After power on or reset this input bit is disabled.

Note: The "remote shutdown in" bit on the auxiliary connector is shared with the Joystick's "shutdown in" bit. If one has a Joystick attached to the 1810/11 indexer this auxiliary input bit should not be used and, vice-versa, if one is using this auxiliary input bit a Joystick should not be attached to the 1810/11 indexer.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

14 1 Set CW motion equal to +/-

00 or nonzero, 00 means CW = + and nonzero means CCW = +

Set the sense of CW motion equivalent to a + direction or a - direction according to the parameter byte. A 00 will set CW motion equivalent to a + sign and a nonzero parameter byte will set CCW motion equivalent to a + sign. After power on or reset CW motion is equivalent to a + sign. This command is ignored if the motor is moving or if the Joystick is enabled. Changing the direction definition does not change the direction definition of any moves that have been defined. For example, if move number one is a + move and CW motion is equivalent to the + direction performing move number one will cause the motor to travel in the CW direction. If the direction definition is changed so that CCW motion is equivalent to the + direction performing move number one will now cause the motor to travel in the CCW direction. After the direction redefinition move number one is still a + move. Positions are always reported as either + or -, not as CW or CCW.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

20 3 Repeat the following "X" commands "Y" times

01 -> FF, number of commands to be repeated
00 -> FF, most significant byte of number of times to loop
00 -> FF, least significant byte of number of times to loop

This command can be used only in a sequence buffer, it has no meaning anywhere else. Its purpose is to create a loop in which a list of commands is repeated up to 65,535 times. A maximum of 255 commands may be within the loop. There is no nesting of loops, thus one cannot place this command within itself, if one does so the inner loop command is ignored. The first parameter byte specifies the number of commands following this command to be repeated. The second and third parameter bytes specify the number of times the loop is to be performed. If zero is specified for either "X" or "Y" the command is ignored.

SINGULAR
VALID-within-a-SEQUENCE-buffer

21 1 Repeat the following "X" commands until a CONTINUE is received

01 -> FF, number of commands to be repeated

This command can be used only in a sequence buffer, it has no meaning anywhere else. Its purpose is to create a loop that is repeated until the host computer issues a CONTINUE command (48h). Upon receipt of a CONTINUE command the current loop iteration will be completed and the next command in the sequence buffer following the loop is performed. Up to 255 commands may be placed in the loop. Loops may not be nested, if one does attempt to nest the inner loop command will be ignored.

SINGULAR
VALID-within-a-SEQUENCE-buffer

28 1 Wait for a CONTINUE

00 or nonzero, 00 = do not interrupt the host and
nonzero = interrupt the host

This command is only defined within a sequence buffer. It has no meaning outside of a sequence buffer. It causes the indexer to suspend processing of the sequence buffer until a CONTINUE command is issued by the host computer. Other commands may be issued and performed before the CONTINUE command is issued by the host, but no commands within the sequence buffer will be performed until a CONTINUE command is issued. If the parameter byte is nonzero this command will send a message to the host to indicate that this command has been encountered. If the "message ready interrupt" is enabled an interrupt will be generated on the host's bus. The message one will receive in the OUTPUT DATA BUFFER will be this command (28h) followed by fifteen zeroes.

SINGULAR
VALID-within-a-SEQUENCE-buffer

29 4 Wait "X" milliseconds

00 -> FF, most significant byte of milliseconds
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of milliseconds

This command is defined only within a sequence buffer, it has no meaning outside of a sequence buffer. It causes the indexer to perform a time delay before performing the next command in the sequence buffer. The first parameter byte is the most significant byte of the time delay and the last parameter byte is the least significant byte of the time delay. A time delay of zero is invalid and will be ignored. The time delay number one must supply whith this command is interpreted as an absolute number, not as a signed number! As is apparent, one can perform time delays of 1 to (2^32)-1 milliseconds (approximately 50 days!).

SINGULAR
VALID-within-a-SEQUENCE-buffer

2A 2 Wait "X" seconds

00 -> FF, most significant byte of seconds
00 -> FF, least significant byte of seconds

This command is defined only within a sequence buffer, it has no meaning outside of a sequence buffer. It causes the indexer to perform a time delay before performing the next command in the sequence buffer. The first parameter byte is the most significant byte of the time delay and the last parameter byte is the least significant byte of the time delay. A time delay of zero is invalid and will be ignored. The time delay number one must supply whith this command is interpreted as an absolute number, not as a signed number! As is apparent, one can perform time delays of 1 to 65,535 seconds (approximately 18 hours).

SINGULAR
VALID-within-a-SEQUENCE-buffer

2B 2 Wait "X" minutes

00 -> FF, most significant byte of minutes
00 -> FF, least significant byte of minutes

This command is defined only within a sequence buffer, it has no meaning outside of a sequence buffer. It causes the indexer to perform a time delay before performing the next command in the sequence buffer. The first parameter byte is the most significant byte of the time delay and the last parameter byte is the least significant byte of the time delay. A time delay of zero is invalid and will be ignored. The time delay number one must supply whith this command is interpreted as an absolute number, not as a signed number! As is apparent, one can perform time delays of 1 to 65,535 minutes (approximately 45 days).

SINGULAR
VALID-within-a-SEQUENCE-buffer

2C 2 Wait for TRIGGER "X" to go active

01 -> 06, the trigger bit the indexer is waiting for
00 or nonzero, the active state of the trigger bit (don't care if first parameter byte is either 01 or 06)

This command is defined only within a sequence buffer, it has no meaning outside of a sequence buffer. This command is very similar in function to the "wait for a CONTINUE" command (28h). When this command is encountered in a sequence buffer it will cause suspension of the sequence buffer until the desired condition is met (the specified TRIGGER bit becomes active). The TRIGGER bits are inputs on the 1810/11 indexer's auxiliary connector. If a TRIGGER bit is specified outside of the given range the command will be ignored. The active state of the trigger bit is specified in the second parameter byte: 00 means the TRIGGER bit is active at 0 volts (TTL low) and nonzero means it is active at +5 volts (TTL high).

SINGULAR
VALID-within-a-SEQUENCE-buffer

2E 4 Define the absolute open-loop postion as "X"

00 -> FF, most significant byte of the absolute open-loop position
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of the absolute open-loop position

This command allows re-definition of the current absolute position. It gives the ability to define the absolute position to be some number other than its current value.

If, for example, the best place for absolute position zero happens to be where a cutoff saw does its business (not a good place for a home sensor), a home sensor can be place some known distance from the saw. Then when it becomes necessary to re-verify the motor's absolute position, the indexer can move the motor so that it finds the sensor, and the absolute position is set to the known offset of the sensor.

The command's four parameter bytes indicate the absolute open loop position "X" to be defined. This command allows definition of the current static position as "X". It may not be performed while the motor is moving. The number supplied by the four parameter bytes is a signed two's complement number. It has a range of 80 00 00 00 through 7F FF FF FF.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

30 0 Define present position as the absolute zero position

This command causes the current position of the motor to be assigned the absolute zero position. The absolute zero position is the reference point for absolute moves. After power up or reset absolute zero is defined to be that position the motor is at when the power up or reset occured. The absolute zero position and the position of the HOME limit switch are two independent locations and only coincide if the user wishes them to by performing a "go to the HOME limit switch" command (39h) followed by this command, "define the present position as the absolute zero position." This command is ignored if the motor is moving or if the Joystick is enabled.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

31 A Define default velocity & default acceleration

xx don't care
XY X = velocity range and Y = acceleration range (see below)
00 -> 7F, most significant byte of the default velocity
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of the default velocity
00 -> 7F, most significant byte of the default acceleration
00 -> FF, second most significant byte

00 -> FF, third most significant byte
00 -> FF, least significant byte of the default acceleration

This command defines a default velocity and acceleration to be used by the "perform the default move (trapezoidal continuous)," "go to relative position 'X' at the default velocity and acceleration," "go to absolute position 'X' at the default velocity and acceleration," and "go to the HOME limit switch at the default velocity and acceleration" commands (32h, 33h, 34h, and 39h). If one wishes to request the default velocity and acceleration parameters they may do so by using the "request move parameters for move number 00" command (91h) (i.e., let parameter byte one of the 91h command equal 00). Move number zero is defined as a trapezoidal, continous move, (CEh).

The second parameter byte determines the velocity and acceleration range for the default move, the available velocity and acceleration ranges are defined in commands C8h through D7H ("define move 'X'"). Bytes three through six specify the velocity. Bytes seven through ten specify the acceleration. The velocity and acceleration are interpreted as two's complement numbers but must be positive, two's complement numbers. A negative velocity and a negative acceleration are invalid parameters. After power up or reset the default velocity and acceleration are set to zero.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

32 1 Perform the default move (trapezoidal continuous)

00 or nonzero, move direction, 00 = + and nonzero = -

This command will perform the default move (move number 0 defined with the above command, 31h) as a trapezoidal, continuous move (same as CEh). The parameter byte specifies the direction the move is to be performed, in the + direction if the parameter byte is 00, otherwise, in the - direction. If the default move has not been defined no move will be performed.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode

33 4 Go to relative position "X" at default velocity and acceleration

00 -> FF, most significant byte of the relative position
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of the relative position

This command tells the indexer to move "X" pulses at the default velocity and acceleration defined with the above command, 31h. The relative distance specified is interpreted as a signed, two's complement number. A relative move may be performed for any distance in the signed range 80000001h through 7FFFFFFFh (-7FFFFFFFh through +7FFFFFFFh or -(2^31)-1 through +(2^31)-1). A distance of 80000000h is invalid and will be ignored. If the default move has not been defined (with command 31h) no move is performed.

This command is one of the few move commands which may be used in VELOCITY-STREAMING-mode.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

34 4 Go to absolute position "X" at default velocity and acceleration

00 -> FF, most significant byte of the absolute position
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of the absolute position

This command tells the indexer to move to absolute position "X" ("X" pulses away from the absolute zero position). The absolute distance specified as a part of this command is interpreted as a signed, two's complement number. Absolute positions may be specified only in the range C0000001h through 3FFFFFFFh (-3FFFFFFFh through +3FFFFFFFh or -(2^30)-1 through +(2^30)-1). The absolute zero position may have either sign. The indexer will move the motor at the default velocity

and acceleration defined with the above command, 31h. If the default velocity and acceleration have not been defined no move is performed.

If an absolute position is specified that is outside of the above range, or if the current absolute position of the motor is outside of the above range no move will be performed. If the motor is positioned outside of the valid absolute positioning range (-3FFFFFFFh through +3FFFFFFFh) one must perform a relative move to reposition the motor within the valid absolute positioning range before being able to perform any absolute moves.

This is one of the few move commands which can be used in the VELOCITY-STREAMING-mode.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

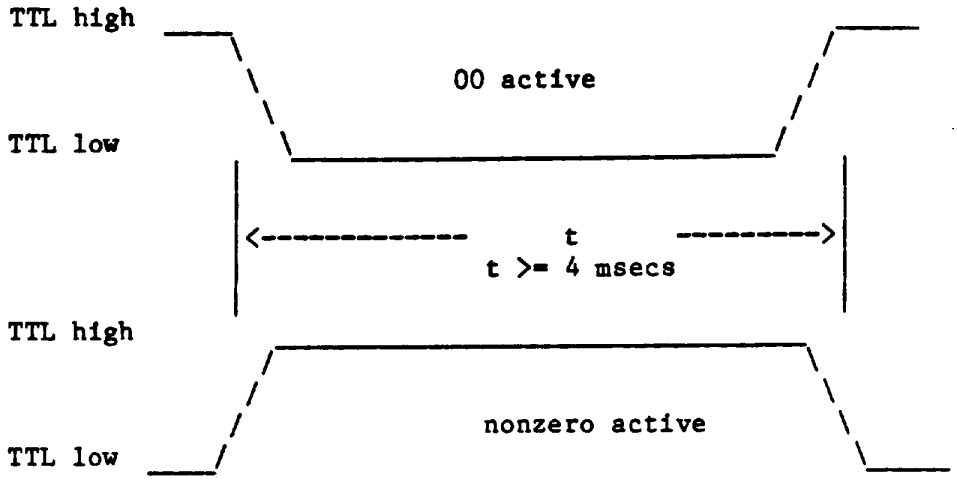
38 4 Define HOME location and final search parameters

00 or nonzero, active state of HOME input, 00 = TTL low and nonzero = TTL high
00 or nonzero, active edge of HOME input, 00 = CW edge and nonzero = CCW edge
01 -> 64 final approach velocity as % of default velocity
00 or nonzero, final approach direction, 00 = CW and nonzero = CCW

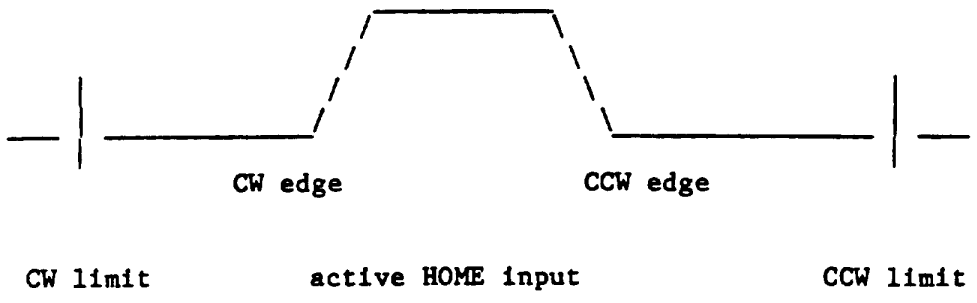
This command is meant to be used typically only after power up or system initialization. It allows the user to define the precise home position with respect to a HOME input signal of finite width and fixed position, regardless of the initial approach direction. It also defines the velocity and direction at which the final approach is performed.

The first parameter byte supplied with this command defines the active state of the HOME input. A 00 parameter byte means the HOME input is active low (TTL 0) and a nonzero parameter byte means the HOME input is active high (TTL 1). The HOME input, like TRIGGERs 2 through 5, is a level sensitive input and must be active for a minumum of four milliseconds at the default velocity, defined with the 31h command, in order for the indexer to perform properly. Following is an illustra-

tion of an active HOME input.



When searching for the HOME position the 1800 indexer does not simply search for that point at which the HOME input is active, but rather, it searches for a specific edge of the active HOME input. The edge the indexer will "home in on" is determined by the second parameter byte. If the second parameter byte is 00 the HOME position is the CW edge of the active HOME input, if the parameter byte is nonzero the HOME position is the CCW edge of the active HOME input. The following diagram illustrates the CW and CCW edges of an active high HOME input. If one starts from the CW limit, the CW edge of the HOME input is the first active going edge one will strike while travelling towards the CCW limit. The inverse is true for the CCW edge of the HOME input. Note that the edge defined as HOME is independent of the direction definition (whether CW = + or CCW = +). This is because the HOME position is defined in terms of the absolute directions CW and CCW; defining HOME in terms of these absolute directions insures repeatability of the HOME position.



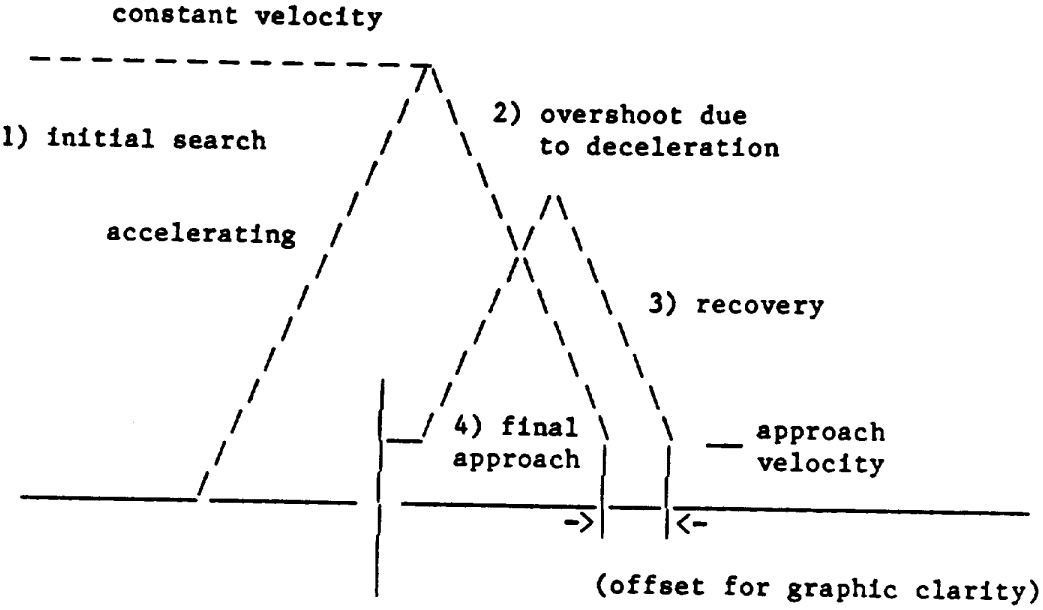
The edge approach velocity, specified by the third parameter byte, is the velocity at which the final search for the HOME position is performed. There is no acceleration to nor deceleration from this velocity, the motor must be able to step up to and down from this velocity. The repeatability of the HOME position is a function of the magnitude of the final approach velocity. The lower the final approach velocity the more repeatable, and more accurate, will be the HOME position. The final approach velocity is determined as a function of the default velocity (defined with command 31h). The third parameter byte supplied with this command expresses the final approach velocity to be a percent of the default velocity. Percents greater than 100 or equal to 0 are invalid and will cause this command to be ignored. To determine the actual final approach velocity use the following formula:

$$\text{approach velocity} = (\text{default velocity}) * \frac{\text{3rd parameter byte}}{100}$$

The fourth, and final, parameter byte supplied with this command specifies the final approach direction for finding the HOME position. This means that the HOME position will always be approached from the same absolute direction. If this parameter byte is a 00 the final approach direction is in the CW direction, towards the CW limit. If the parameter byte is nonzero the final approach direction is in the CCW direction, towards the CCW limit. Thus, if one specified the CW edge of the HOME input to be the HOME position and specified to approach in the CW direction the indexer would always approach HOME from HOME active to HOME inactive (see above diagram).

The "go HOME" move is designed to be rapid and repeatable. The initial search direction is specified with the go HOME command (39h), but if an end of limit is encountered the motor will reverse and try searching in the other direction. If the opposite end of travel limit is hit the search for HOME is abandoned. The following diagram illustrates what happens after the specified HOME edge is found once. Item number one indicates the initial search for the HOME input. When the active edge of interest is found (either at the default velocity or during acceleration to the default velocity) the indexer will begin to decelerate the motor; this begins item number two, the overshoot due to deceleration. The indexer notes the distance it has overshot and will perform a recovery move, item number three, to get back to the approximate location of the HOME position. The final approach to the HOME position will be performed at the specified percent of default velocity (parameter byte number three) in the specified absolute direction (parameter

byte number four). The diagram below does not illustrate the motions the indexer will send the motor through to acheive the final approach in the specified absolute direction.



VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-VELOCITY-STREAMING
VALID-in-VELOCITY-STREAMING-mode

39 1 Go HOME at the default velocity and acceleration

00 or nonzero, direction in which search is to begin, 00 = + and nonzero = -

This command will cause the indexer to search for the HOME limit switch at the velocity and acceleration specified with command 31h. If the HOME limit switch is not found after striking both the CW limit switch and the CCW limit switch the search is abandoned.

The parameter byte specifies the direction the indexer will begin moving the motor as it begins its search for the HOME input (as defined with command 38h above), i.e., whether

to begin searching for the HOME limit switch in the + direction or in the - direction. This command will not be performed if the above command (38h) has not been performed (the HOME position has not been defined), if the default move parameters have not been defined, if the motor is moving, or if the joystick is enabled.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

40 1 Perform move number "X"

01 -> 32, move number

This command will cause the indexer to perform previously defined move number "X." The move definition contains information specifying the distance, velocity, acceleration, direction, velocity range, acceleration range, and move type (relative, preset, or continuous). A move numbered outside of the above range is invalid and will be ignored. If a move is currently being performed this command is ignored.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode

41 1 Perform sequence buffer "X"

01 -> 0F indicates sequence buffer number

This command causes previously defined sequence buffer "X" to be performed. The parameter byte must be supplied with this command, if it is not supplied the command is ignored. If a sequence buffer has not been defined this command is ignored. To find out how much free space is left in the sequence buffer refer to command E2.

VALID-in-INDEXER-mode

42 0 Perform the streaming buffer

This command tells the indexer to perform the previously filled streaming buffer. If the velocity, distance or time-distance streaming buffer is empty no action takes place. If the streaming buffer is emptied out faster than it is refilled the motor will remain at the last velocity in the buffer. To resume streaming the buffer must be refilled and this command reissued; thus, when the streaming buffer is emptied the 1800 is no longer "performing the streaming buffer."

VALID-in-VELOCITY-STREAMING-mode

48 0 CONTINUE (perform the next command)

This command causes continuation!! If a command has been performed instructing the indexer to wait for a CONTINUE command this is the command it is waiting for. If the indexer is not waiting for a CONTINUE this command is ignored.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

50 0 Enter open loop indexer mode

This command instructs the controller to enter open loop indexer mode. In open loop indexer mode one can perform relative moves, absolute moves, and continuous moves. Operation of the joystick is also permitted. Open loop indexer mode is what one might consider the "normal" mode of the indexer. After power on or reset the indexer is in open loop indexer mode.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

51 6 Enter velocity-distance streaming mode (user profiling mode)

01 -> 09, choose base clock frequency
01 -> 10, choose divisor
00 -> FF, MS byte of buffer empty alarm
00 -> FF, second MS byte
00 -> FF, third MS byte
00 -> FF, LS byte of buffer empty alarm

Velocity-distance streaming mode requires that the user supply velocity-distance pairs to the indexer. These are supplied with the E0h command. The three parameter supplied with this command determine the "velocity range" and the point at which a "buffer empty alarm" is sent.

Entering this mode causes any previously defined sequence buffers to be lost.

The following list contains the base clock frequencies available and the parameter byte (parameter byte number one) used to choose a base frequency.

parameter byte	base frequency
01	4.3690667 MHz
02	436.90667 KHz
03	43.690667 KHz
04	4.3690667 KHz
05	436.90667 Hz
06	273.06667 KHz
07	17.066667 KHz
08	1.0666667 KHz
09	66.666667 Hz

The effective clock frequency is calculated by dividing the base clock frequency by the divisor supplied as the second parameter byte. The divisor may be any number between 01h and 10h inclusive (1 through 16). The maximum frequency (velocity) acheivable for the chosen base frequency and divisor is one half the effective clock frequency. The frequency (velocity) resolution is equal to the effective clock frequency divided by 65,536.

The "buffer empty alarm" is a message to the host indicating that the velocity streaming buffer is "nearly empty." The third parameter supplied (parameter bytes three through

six) is a comparison number the 1800 indexer uses to determine if it should send a "buffer empty alarm" to the host. If the number of data bytes in the velocity streaming buffer equals or goes below this parameter (and a certain precondition has been met) the indexer will send the host a "buffer empty alarm." A "buffer empty alarm" is accomplished by sending a message to the host. The message sent is this command followed by the parameter bytes that were supplied with this command. (Sending a message to the host will result in a "message ready" interrupt, if the interrupt has been enabled.) For example, if the third parameter was a 00 00 00 64h the indexer would send a "buffer empty alarm" to the host when the number of bytes to be processed in the velocity streaming buffer was equal or less than 100 bytes. However, the alarm would not have occured unless the buffer had been filled to more than 100 bytes; in other words, one must fill the buffer past the buffer alarm value in order to turn on the alarm function. In addition, the alarm will only be sent once, the host will not receive continuous alarms after the buffer goes below the buffer alarm value. Again, the following sequence of events must take place in order to receive a "buffer empty alarm."

1) Define the buffer alarm value with parameter three above.
2) Fill the velocity streaming buffer with more bytes than have been specified as the buffer alarm value.
3) Begin velocity streaming. When the buffer empties to a number of bytes below the value specified as the buffer empty alarm a "buffer empty alarm" message will be sent to the host.

To avoid the buffer empty alarm altogether specify a buffer alarm value that is greater than the total size of the velocity streaming buffer (see command E2h). When this is done it is impossible to turn the buffer empty alarm function on because it is impossible to fill the buffer with more bytes than the buffer alarm value specified.

One may reenter velocity-distance streaming mode while in velocity-distance streaming mode (but not while performing the velocity-streaming buffer). Reentering is the only means by which one may modify the effective clock frequency and the "buffer empty alarm." If one attempts to enter velocity-distance streaming mode while the motor is moving the request will be ignored. The base clock frequency and the effective clock frequency cannot be modified if the motor is moving.

Several move commands may be executed while in this mode. They are the "goto position at default velocity and acceleration" commands. Their command numbers are 33, 34, 35, and 36.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

52 8 Enter velocity-time streaming mode (user profiling mode)

01 -> 09, choose base clock frequency
01 -> 10, choose divisor
00 -> FF, MS byte of the buffer empty alarm value
00 -> FF, second MS byte
00 -> FF, third MS byte
00 -> FF, LS byte of the buffer empty alarm value
00 -> FF, most significant digit of the update rate, in milliseconds
00 -> FF, least significant digit of the update rate, in milliseconds

Velocity-time streaming mode requires that the user supply velocity values to be output at a predefined update rate. The first three parameter determine the "velocity range" and the point at which a "buffer empty alarm" will be sent. The update rate is supplied as the seventh and eighth parameter bytes of this command. The update rate is specified in milliseconds and can be any value between 2 and 65,535 milliseconds, inclusive; an update rate of zero or one millisecond is invalid and not allowed. Velocity values to be output by the indexer are supplied with the Elh command.

Entering this mode causes any previously defined sequence buffers to be lost.

The following list contains the base clock frequencies available and the parameter byte (parameter byte number one) used to choose a base frequency.

parameter byte	base frequency
01	4.3690667 MHz
02	436.90667 KHz
03	43.690667 KHz
04	4.3690667 KHz
05	436.90667 Hz
06	273.06667 KHz

07	17.066667 KHz
08	1.0666667 KHz
09	66.666667 Hz

The effective clock frequency is calculated by dividing the base clock frequency by the divisor supplied as the second parameter byte. The divisor may be any number between 01h and 10h inclusive (1 through 16). The maximum frequency (velocity) acheivable for the chosen base frequency and divisor is one half the effective clock frequency. The frequency (velocity) resolution is equal to the effective clock frequency divided by 65,536.

The "buffer empty alarm" is a message to the host indicating that the velocity streaming buffer is "nearly empty." The third parameter supplied (parameter bytes three through six) is a comparison number the 1810/11 indexer uses to determine if it should send a "buffer empty alarm" to the host. If the number of data bytes in the velocity streaming buffer equals or goes below this parameter (and a certain precondition has been met) the indexer will send the host a "buffer empty alarm." A "buffer empty alarm" is accomplished by sending a message to the host.

The message sent is this command followed by the parameter bytes that were supplied with this command. (Sending a message to the host will result in a "message ready" interrupt, if the interrupt has been enabled.) For example, if the third parameter was a 00 00 00 64h the indexer would send a "buffer empty alarm" to the host when the number of bytes to be processed in the velocity streaming buffer was equal or less than 100 bytes. However, the alarm would not have occured unless the buffer had been filled to more than 100 bytes; in other words, one must fill the buffer past the buffer alarm value in order to turn on the alarm function. In addition, the alarm will only be sent once, the host will not receive continuous alarms after the buffer goes below the buffer alarm value. Again, the following sequence of events must take place in order to receive a "buffer empty alarm."

1) Define the buffer alarm value with parameter three above.
2) Fill the velocity streaming buffer with more bytes than have been specified as the buffer alarm value.
3) Begin velocity streaming. When the buffer empties to a number of bytes below the value specified as the buffer empty alarm a "buffer empty alarm" message will be sent to the host.

To avoid the buffer empty alarm altogether specify a buffer alarm value that is greater than the total size of the velocity streaming buffer (see command E2h). When this is done it is impossible to turn the buffer empty alarm function on because it is impossible to fill the buffer with more bytes than the buffer alarm value specified.

One may reenter velocity-time streaming mode while in velocity-time streaming mode (but not while performing the velocity-streaming buffer). Reentering is the only means by which one may modify the effective clock frequency, the "buffer empty alarm," and the predefined update rate. If one attempts to enter velocity-time streaming mode while the motor is moving the request will be ignored. The base clock frequency and the effective clock frequency cannot be modified if the motor is moving.

Several move commands may be executed while in the VELOCITY-STREAMING-mode. These are the "go to position at the default velocity and acceleration" commands. Their command numbers are 33, 34, 35, and 36.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

53 8 Enter distance-time streaming mode (user profiling mode)

01 -> 03, choose base clock frequency
01 -> 10, choose divisor
01 -> FF, MS byte of the buffer empty alarm value
00 -> FF, third MS byte
00 -> FF, LS byte of the buffer empty alarm value
00 -> FF, most significant digit of the update rate, in milliseconds
00 -> FF, least significant digit of the update rate, in milliseconds

Distance-time streaming mode requires that the user supply velocity values to be output at a predefined update rate. The first three parameter determine the "velocity range" and the point at which a "buffer empty alarm" will be sent. The update rate is supplied as the seventh and eighth parameter bytes of this command. The update rate is specified in milliseconds and

can be any value between 2 and 65,535 milliseconds, inclusive; and update rate of zero or one millisecond is invalid and not allowed. Distance values to be output by the indexer are supplied with the E3h command.

Entering this mode causes any previously defined sequence buffers to be lost.

The following list contains the base clock frequencies available and the parameter byte (parameter byte number one) used to choose a base frequency.

parameter byte	base frequency
01	4.3690667 MHz
02	436.90667 KHz
03	43.690667 KHz

The effective clock frequency is calculated by dividing the base clock frequency by the divisor supplied as the second parameter byte. The divisor may be any number between 01h and 10h inclusive (1 through 16). The maximum frequency (velocity) achievable for the chosen base frequency and divisor is on half the effective clock frequency. The frequency (velocity) resolution is equal to the effective clock frequency divided by 65,536.

The "buffer empty alarm" is a message to the host indicating that the velocity streaming buffer is "nearly empty." The third parameter supplied (parameter bytes three through six) is a comparison number the 1810/11 indexer uses to determine if it should send a "buffer empty alarm" to the host. If the number of data bytes in the velocity streaming buffer equals or goes below this parameter (and a certain precondition has been met) the indexer will send the host a "buffer empty alarm." A "buffer empty alarm" is accomplished by sending a message to the host.

The message sent is this command followed by the parameter bytes that were supplied with this command. (Sending a message to the host will result in a "message ready" interrupt, if the interrupt has been enabled.) For example, if the third parameter was a 00 00 00 64th the indexer would send a "buffer empty alarm" to the host when the number of bytes to be processed in the velocity streaming buffer was equal or less than 100 bytes. However, the alarm would not have occurred unless the buffer had been filled to more than 100 bytes; in other words, one must fill the buffer past the buffer alarm value in order to turn on the alarm function. In addition, the alarm will only be sent once, the host will not receive continuous alarms after the buffer goes

below the buffer alarm value. Again, the following sequence of events must take place in order to receive a "buffer empty alarm."

1) Define the buffer alarm value with parameter three above.
2) Fill the velocity streaming buffer with more bytes than have been specified as the buffer alarm value.
3) Begin velocity streaming. When the buffer empties to a number of bytes below the value specified as the buffer empty alarm a "buffer empty alarm" message will be sent to the host.

To avoid the buffer empty alarm altogether specify a buffer alarm value that is greater than the total size of the velocyt streaming buffer (see command Esh). When this is done it is impossible to turn the buffer empty alarm function on because it is impossible to fill the buffer with more bytes than the buffer alarm value specified.

One may reenter distance-time streaming mode while in distance-time streaming mode (but not while performing the distance-streaming buffer). Reentering is the only means by which one may modify the effective clock frequency, the "buffer empty alarm," and the predefined update rate. If one attempts to enter distance-time streaming mode while the motor is moving the request will be ignored. The base clock frequency and the effective clock frequency cannot be modified if the motor is moving.

Several move commands may be executed while in the DISTANCE-STREAMING-mode. These are the "go to position at the default velocity and acceleration" commands. Their command numbers are 33, 34, 35 and 36.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

70 0 STOP motion

A controlled deceleration of any motion due to the performance of a move is performed. The deceleration to zero is at the most recent acceleration/deceleration rate. If the velocity streaming buffer is being performed this command is equivalent to command 78h, KILL motion.

VALID-in INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

71 0 Discontinue the sequence buffer

The command currently being performed in the sequence buffer will be completed and no more commands will be started in the sequence buffer. This command causes the indexer to leave the sequence buffer prematurely. In other words, if this command is issued while performing the sequence buffer one will not be able to restart the sequence buffer where it was discontinued. If the sequence buffer is not being performed this command has no effect. The contents of the sequence buffer are not lost when this command is executed.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

72 0 Suspend the sequence buffer; wait for a CONTINUE to resume

This command is similar to 71h. Rather than prematurely ending the sequence buffer it is temporarily suspended until a CONTINUE command (48h) is received. Upon receipt of a CONTINUE command the next command that was to be performed when the sequence buffer was suspended is performed. If the sequence buffer is not being performed when this command is executed the command is ignored.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

73 0 Discontinue any singular command currently being performed

Any singular command being performed at the time this command is executed is prematurely ended. If a move is being performed, either preset or continuous, the velocity being output at the time the command is issued is the last velocity output. In other words, the move is neither STOPPED (see command 70h), KILLED (see command 78h), nor finished, it is suspended!!

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

VALID-in-VELOCITY-STREAMING-mode

74 3 STOP motion when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

The effects of this command are similar to the "STOP motion" command, (70h). A controlled deceleration of any motion due to the performance of a move occurs when the specified TRIGGER goes active. The TRIGGER is specified by the first parameter byte. The deceleration to zero is at the most recent acceleration/deceleration rate. The second parameter byte turns this function on and off. A 00 parameter byte means an active TRIGGER 1 or 6 will not STOP motion and a nonzero parameter byte means an active TRIGGER 1 or 6 will STOP motion. Following power on or reset this function is disabled. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" on the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input at a given time. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

75 3 Discontinue the sequence buffer when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

This command is very similar to 71h. Rather than discontinuing the sequence buffer immediately upon receipt of this command the sequence buffer is discontinued when the specified TRIGGER input goes active. As is the case with command 71h, the sequence buffer is prematurely ended and may not be restarted at the point of discontinuation. The second parameter byte turns this function on and off. A 00 parameter byte means an active TRIGGER 1 or 6 will not discontinue the sequence buffer and a nonzero parameter byte means an active TRIGGER 1 or 6 will discontinue the sequence buffer. Following power on or reset this function is disabled. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" on the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input at a given time. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

76 3 Suspend seq. buffer when TRIGGER "X" goes active; wait for a CONTINUE to resume

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

This command is very similar to 72h. Rather than suspending the sequence buffer when this command is received the sequence buffer will be suspended when the specified TRIGGER goes active. Upon receipt of a CONTINUE command the next command that was to be performed when the sequence buffer was suspended is performed. If the sequence buffer is not being performed when the specified TRIGGER goes active the TRIGGER is ignored. The second parameter byte turns this function on and off. A 00 parameter byte means an active TRIGGER 1 or 6 will not suspend the sequence buffer and a nonzero parameter byte means an active TRIGGER 1 or 6 will suspend the sequence buffer. Following power on or reset this function is disabled. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is non-zero the indexer will send a message to the host, which will generate a "message ready interrupt" on the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input at a given time. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

77 3 Discontinue any singular command when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

This command is nearly identical to command 73h, however, the specified TRIGGER must go active before a singular command is prematurely ended. If a move is being performed, either preset or continuous, the velocity being output at the time the specified TRIGGER bit goes active is the last velocity output. In other words, the move is neither STOPPED (see command 70h), KILLED (see command 78h), nor finished, it is suspended!! If TRIGGER 1 or 6 goes active when there is no singular command being performed it has no effect. The second parameter byte specifies whether the specified TRIGGER is to be ignored or not. If the parameter byte is 00 an active TRIGGER will not cause premature ending of a singular command, and if the parameter byte is nonzero an active TRIGGER will cause premature ending of a singular command. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" on the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input at a give time. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

78 0 KILL motion

Any motion that is occuring at the time this command is executed is immediately stopped. Pulses are stopped immediately, there is no deceleration to zero velocity. The action of this command is equivalent to hitting a limit switch.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

79 0 KILL the sequence buffer

Any singular command that is being performed in the sequence buffer is prematurely ended and the sequence buffer is prematurely ended. If the sequence buffer is not being performed when this command is executed the command is ignored.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

7A 0 KILL current sequence singular command; wait for a CONTINUE

Any singular command that is currently being performed in the sequence buffer is prematurely ended. The sequence buffer is temporarily suspended until a CONTINUE command is received. If the sequence buffer is not being performed when this command is executed the command is ignored.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

7B 0 KILL the velocity-streaming buffer

Velocity-streaming is prematurely ended at the velocity value in effect when this command is executed. This command clears the velocity-streaming buffer. To resume velocity streaming one must refill the velocity streaming buffer and reissue the "perform the velocity streaming buffer" command (42h). To bring the velocity of the motor to zero the velocity-streaming buffer must be filled with a zero velocity or the KILL motion command must be issued (78h).

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

7C 3 KILL motion when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

The effects of this comand are similar to the "KILL motion" command, (78h). Any motion that is occuring at the time the specified TRIGGER goes active is immediately stopped. Pulses are stopped immediately, there is no deceleration to zero velocity. The action of this command is equivalent to hitting a limit switch. The TRIGGER is specified by the first parameter byte. The second parameter byte turns this function on and off. A 00 parameter byte means an active TRIGGER 1 or 6 will not KILL motion and a nonzero parameter byte means an active TRIGGER 1 or 6 will KILL motion. Following power on or reset this function is disabled. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" on the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input at a given time. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

7D 3 KILL the sequence buffer when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

This command is very similar to 79h. Rather than KILLING the sequence buffer immediately upon receipt of the command the sequence buffer is KILLED when the specified TRIGGER goes active. As is the case with command 79h, the sequence buffer is prematurely ended and may not be restarted at the point of the KILL. If TRIGGER 1 or 6 goes active when there is no sequence buffer being performed it has no effect. The second parameter byte determines whether or not the specified TRIGGER is to be ignored. If the parameter byte is 00 an active TRIGGER will not cause KILLING of a sequence buffer, and if the parameter byte is nonzero an active TRIGGER will cause KILLING of a sequence buffer. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" to the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

7E 3 KILL current sequence singular command when TRIGGER "X" goes active; wait for a CONTINUE

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

If the sequence buffer is being performed any singular command being performed when the specified TRIGGER goes active is prematurely ended and the sequence buffer is suspended until a CONTINUE is received. The second parameter byte determines whether the specified TRIGGER is to be ignored or not. If the parameter byte is 00 an active TRIGGER will not cause suspension of the sequence buffer, and if the parameter byte is nonzero an active TRIGGER will cause suspension of the sequence buffer. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" to the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

7F 3 KILL the velocity-streaming buffer when TRIGGER "X" goes active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable
00 or nonzero, 00 = do not send a message, nonzero = send a message

Velocity-streaming is stopped at the current velocity

being output at the time the specified TRIGGER goes active. This command clears the velocity-streaming buffer. To resume velocity streaming one must refill the velocity-streaming buffer and send the "perform the velocity streaming buffer" command (42h). To bring the velocity of the motor to zero the velocity-streaming buffer must be filled with a zero velocity or the KILL motion command must be issued (78h). The second parameter byte determines whether the specified TRIGGER is to be ignored or not. If the parameter byte is 00 an active TRIGGER will not cause KILLING of the sequence buffer, and if the parameter byte is nonzero an active TRIGGER will cause KILLING of the sequence buffer. The third parameter byte specifies whether the indexer is to send a message to the host when the specified TRIGGER goes active. If the third parameter byte is nonzero the indexer will send a message to the host, which will generate a "message ready interrupt" to the host's bus if interrupts are enabled. The message one will receive will be this command byte followed by the first parameter byte and fourteen zeroes.

Only one function may be performed by a given TRIGGER input. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

80 0 Request position relative to the beginning of the current move

Five bytes are returned to the host indicating the position relative to the start of the current move or most recent zero velocity. If no move is in progress at the time this command is executed the most recent "excursion" from zero velocity, to nonzero velocity, and back to zero velocity is reported. The first byte returned is an 80h identifying the following four bytes to be in response to this command. The four position bytes are in two's complement notation, most significant byte first and least significant byte last. The following list illustrates the response one would receive from this command.

80	identifies the command the response is intended for
00 -> FF	most significant byte of the relative position
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the relative position
00	bytes six through
:	sixteen are always zero
00	and are always present

If one performs a sequence of continous moves that result in the motor travelling a relative distance greater than 7FFFFFFFh pulses the response from this command will be "garbage." Additionally, any other position information requested will be of questionable value. Thus, if one wishes to insure that all position information is correct and valid one must be sure the motor does not perform any relative moves greater than 7FFFFFFFh pulses.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

81 0 Request position relative to the end of the current move

Five bytes are returned to the host indicating the position relative to the end of the current preset move. If the motor is not moving but the last move performed was a preset move that relative end position is returned. If a move is occurring and it is not a preset move or if the motor is not moving and the last move that was performed was not a preset move a zero position is returned (the end position of a continous move is undefined). If an overshoot occurs (the move is discontinued and then STOPPED or KILLED after passing the desired end position) the position returned is a negative number; if an undershoot occurs (the move is STOPPED or KILLED before it is finished) the position returned is a positive number. The first byte returned is an 81h identifying the following four bytes to be in response to this command. The four position bytes are in two's complement form, most significant byte first

and least significant byte last. The following list illustrates the response one would receive from this command.

81	identifies the command the response is intended for
00 -> FF	most significant byte of the end relative position
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the end relative position
00	bytes six through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

82 0 Request position relative to the HOME limit switch

Seven bytes are returned to the host indicating the position relative to the HOME limit switch position. After power up or reset the HOME limit switch position is defined as that position the motor is currently residing at. The HOME limit switch position will only be updated if the command "go to the HOME limit switch" (39h) is given. The first byte returned is an 82h identifying the following six bytes to be in response to this command. The six position bytes are in two's complement form, most significant byte first and least significant byte last. The following list illustrates the response one would receive from this command.

82	identifies the command the response is intended for
00 -> FF	most significant byte of the HOME relative position
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	fourth most significant byte
00 -> FF	fifth most significant byte
00 -> FF	least significant byte of the HOME relative position

```
00              bytes eight through
 :              sixteen are always zero
00              and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING
```

83 0 Request position relative to the absolute zero position

Seven bytes are returned to the host indicating the position relative to the absolute zero position (which is defined with the 30h command). After power up or reset the absolute zero position is defined as that position the motor is residing at when power up or reset occurs. The first byte returned is an 83h identifying the following six bytes to be in response to this command. The six position bytes are in two's complement form, most significant byte first and least significant byte last. The following list illustrates the response one would receive from this command.

```
83              identifies the command the response is
                intended for
00 -> FF        most significant byte of the absolute
                position
00 -> FF        second most significant byte
00 -> FF        third most significant byte
00 -> FF        fourth most significant byte
00 -> FF        fifth most significant byte
00 -> FF        least significant byte of the absolute
                position

00              bytes eight through
 :              sixteen are always zero
00              and are always present
```

Exceeding the maximum expressable absolute position value (7FFFFFFFFFFFh) causes the real absolute position to be lost. So, if one is interested in insuring that all positions are valid and correct they must not exceed the maximum expressable absolute position.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

84 0 Request current direction

The current motor direction is returned. The direction returned is a 00h for travel in the + direction and an FFh for travel in the - direction. If the motor is not moving the most recent direction of travel is returned. The following list illustrates the response one would receive from this command.

84	identifies the command the response is intended for
00 or FF	direction the motor is travelling or was most recently travelling, 00 = + and FF = -
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

85 0 Request current velocity

If one is in indexer mode six bytes are returned to the host indicating the current velocity. The first byte returned is an 85h identifying the following five bytes to be in response to this command. The first of the last five bytes indicates the velocity range (see the commands to define moves, C8h through D7h). The last four bytes are the positive velocity most significant byte first and least significant byte last. The following list illustrates the response one would receive from this command if one is in indexer mode.

85	identifies the command the response is intended for

XY	X = the velocity range and Y = the acceleration range
00 -> 7F	most significant byte of the velocity
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the velocity
00	bytes seven through
:	sixteen are always zero
00	and are always present

If one is in velocity-streaming mode the response is different than the above. Only two parameter bytes are returned as a response to this command. The bytes returned have the same meaning as the velocity numbers one sends to the 1800 to fill the velocity streaming buffer (see commands E0h, E1h, 51h, and 52h). The most significant bit of the two bytes is the direction (1 = CCW and 0 = CW) and the other fifteen bits, V, represent the unsigned frequency (velocity). The frequency (velocity) can be determined from the following formula:

frequency = effective clock * (V / 65,536)

The following list illustrates the response one would receive from this command if one is in velocity-streaming mode.

85	identifies the command the response is intended for
00 -> FF	direction bit and most significant byte of the velocity number
00 -> FF	least significant byte of the velocity number
00	bytes four through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

86 0 Request current acceleration

Six bytes are returned to the host indicating the current acceleration. The first byte returned is an 86h identifying the following five bytes to be in response to this command. The first of the last five bytes returned is the velocity and acceleration range (see the commands to define moves, C8h through D7h). The last four bytes are the positive acceleration most significant byte first and least significant byte last. The following list illustrates the response one would receive from this command.

86	identifies the command the response is intended for
XY	denotes the velocity and acceleration range
00 -> 7F	most significant byte of the acceleration
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the acceleration
00	bytes seven through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

88 0 Request current move status

A two byte response indicating the move status. The first byte is an 88h indicating the command to which this response is directed. The second byte is the status of the motor. The following list defines each bit of this status byte.

88	identifies the command the response is intended for
XXXXXXXXb	eight bit number, each bit is explained below bit 0 is the least significant bit bit 7 is the most significant bit
bit 0 -> 1	the + direction limit switch is asserted
-> 0	the + direction limit switch not asserted

```
bit 1 -> 1      the - direction limit switch is asserted
      -> 0      the - direction limit switch not asserted

bit 2 -> 1      the HOME limit switch is asserted
      -> 0      the HOME limit switch is not asserted

bit 3 -> 1      the direction of motion is - direction
      -> 0      the direction of motion is + direction

bit 4 -> 1      the motor is moving
      -> 0      the motor is not moving

bit 5 -> 1      the motor is at constant, nonzero velocity
      -> 0      the motor is not at constant, nonzero
                velocity

bit 6 -> 1      the motor is accelerating
      -> 0      the motor is not accelerating

bit 7 -> 1      the motor is decelerating
      -> 0      the motor is not decelerating

00              bytes three through
 :              sixteen are always zero
00              and are always present


VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING
```

89 0 Request state of the limit switches

The state of the limit switches is returned. Two bytes are returned, the first byte indicates the command to which the response is directed and the second byte indicates which, if any, limit switch is active. The high order nibble (most significant four bits) of the second byte pertain to the + direction limit switch and the low order nibble (least significant four bits) pertain to the - direction limit switch. If a nibble is set the corresponding limit switch is active, if a nibble is 0 the limit switch is inactive. The following responses can be received from this command.

```
89              identifies the command the response is
```

	intended for
00 or 0F or FO or FF	00 means no limit switches are active 0F means only the - direction limit switch is active FO means only the + direction limit switch is active FF means both limit switches are active
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

8A 0 Request state of the HOME limit switch

The state of the HOME limit switch is returned. Two bytes are returned, the first byte indicates the command to which the response is directed and the second byte indicates whether or not the limit switch is active. If the second byte is an FFh the HOME limit switch is active, if the second byte is a 00h the limit switch is inactive. The following response would be received from this command.

8A	identifies the command the response is intended for
00 or FF	00 means the HOME limit switch is inactive FF means the HOME limit switch is active
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

8B 0 Request direction of travel

The current or most recent direction of travel is returned. This is a two byte response. The first byte identifies the command the response is intended for and the second byte identifies the most recent direction of travel. The first byte returned is an 8Bh. The second byte will be a 00h for travel in the + direction and an FFh for travel in the - direction. The response to this command is as follows.

8B	identifies the command the response is intended for
00 or FF	00 means travel in the + direction FF means travel in the - direction
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

8C 0 Request whether the motor is moving or not moving

Whether or not the motor is moving is answered. Two bytes are returned, the first byte identifies the command the response is intended for and the second byte indicates the moving/not moving status. A 00h is returned if the motor is not moving and an FFh is returned if the motor is moving.

8C	identifies the command the response is intended for
00 or FF	00 means the motor is not moving FF means the motor is moving
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

8D 0 Request whether the motor is at constant, nonzero velocity or not

Whether or not the motor is at constant, nonzero velocity is answered. Two bytes are returned, the first byte identifies the command the response is intended for and the second byte indicates the constant, nonzero velocity/not constant, nonzero velocity status. A 00h is returned if the motor is not at constant, nonzero velocity and an FFh is returned if the motor is at constant, nonzero velocity.

8D	identifies the command the response is intended for
00 or FF	00 means the motor is not at constant, nonzero velocity FF means the motor is at constant, nonzero velocity
00 : 00	bytes three through sixteen are always zero and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

8E 0 Request whether the motor is accelerating or not accelerating

Whether or not the motor is accelerating is answered. Two bytes are returned, the first byte identifies the command the response is intended for and the second byte indicates the accelerating/not accelerating status. A 00h is returned if the motor is not accelerating and an FFh is returned if the motor is accelerating.

8E	identifies the command the response is intended for
00 or FF	00 means the motor is not accelerating FF means the motor is accelerating
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode

8F 0 Request whether the motor is decelerating or not decelerating

Whether or not the motor is decelerating is answered. Two bytes are returned, the first byte identifies the command the response is intended for and the second byte indicates the decelerating/not decelerating status. A 00h is returned if the motor is not decelerating and an FFh is returned if the motor is decelerating.

8F	identifies the command the response is intended for
00 or FF	00 means the motor is not decelerating FF means the motor is decelerating
00	bytes three through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

VALID-in-VELOCITY-STREAMING-mode

90 0 Request present mode

The indexer's current mode of operation is returned. There are four possible modes of operation: indexer mode; velocity-distance streaming mode; velocity-time streaming mode; and distance-time mode. At least two bytes are returned, the first byte identifies the command the response is intended for and the second byte indicates the mode the indexer is operating in. A mode is identified by the command byte that causes the indexer to enter that mode; for example, if one wishes to enter indexer mode they must give the indexer a 50h command. A 50h would be returned if one was in indexer mode and performed this command.

If one is in any of the velocity or distance streaming modes there will be additional bytes returned, besides the mode byte (second byte returned). The third byte is the base frequency being used, the fourth byte is the frequency divisor, and the fifth through eighth bytes are the "buffer empty" alarm. If one is in velocity or distance time streaming mode the ninth and tenth bytes will contain the update rate. (See the description of commands 51h, 52h and 53h).

90	identifies the command the response is intended for
50, 51, 52 or 53h	50 means indexer mode 51 means velocity-distance streaming mode 52 means velocity-time streaming mode 53 means distance-time streaming mode
01 -> 09	base frequency (velocity or distance-streaming mode only)
01 -> 10	base frequency divisor (velocity or distance-streaming mode only)
00 -> FF	MS byte of "buffer empty" alarm (velocity or distance-streaming mode only)
00 -> FF	second MS byte
00 -> FF	third MS byte
00 -> FF	LS byte of "buffer empty" alarm
00 -> FF	MS byte of milliseconds update (velocity-time mode only)
00 -> FF	LS byte of milliseconds update (velocity-

00	bytes eight through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

91 1 Request move parameters for move number "X"

00 -> FF, move number

Returns the parameters defined for move number "X." Sixteen bytes are returned. The first byte returned is a 91h, it identifies this command as the command the response is intended for. The second through sixteenth bytes appear as the first through fifteenth bytes of a move definition. In other words, the returned bytes match the format of the bytes that were sent with the original define move command. When one is in velocity-streaming mode only the "default move" (move number 00) can be defined, therefore, in velocity-streaming mode one will only receive a response from this command if the move number is equal to 00. Below is an illustration of the response one receives from this command.

91	identifies the command the response is intended for
C8 -> D0	identifies the move type (number is move definition command)
00 -> FF	move number
XY	X = velocity range and Y = acceleration range
00 -> 7F	most significant byte of the move velocity
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the move velocity
00 -> 7F	most significant byte of the move acceleration
00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the move acceleration
00 -> FF	most significant byte of the move position, or direction

00 -> FF	second most significant byte
00 -> FF	third most significant byte
00 -> FF	least significant byte of the move position

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

92 1 Request commands stored in the sequence buffer

01, always (reserved for sequence number)

Returns the commands that have been loaded into the sequence buffer. Sixteen bytes are always returned. Only one command is returned from the sequence buffer at a time, if a command is not sixteen bytes long the command will be padded with zeroes to make it sixteen bytes long. The first byte is a 92h indicating this is the command the response is meant for. The second through sixteenth bytes contain the command. Responses are sent as long as there are commands in the sequence buffer. A null command will be sent to indicate the end of the sequence buffer, in other words, the last message sent will be a 92h followed by the null command (92h followed by fifteen zeroes). If there are any messages waiting in the message queue for "consumption" by the host (i.e., the host has made a request for some information) they will be lost. The response to this command appears as follows.

92	identifies the command the response is intended for
00 -> FF	command byte in the sequence buffer
00 -> FF	parameter one, zero if no parameter
00 -> FF	parameter two, zero if no parameter
00 -> FF	parameter three, zero if no parameter
00 -> FF	parameter four, zero if no parameter
00 -> FF	parameter five, zero if no parameter
00 -> FF	parameter six, zero if no parameter
00 -> FF	parameter seven, zero if no parameter
00 -> FF	parameter eight, zero if no parameter
00 -> FF	parameter nine, zero if no parameter
00 -> FF	parameter ten, zero if no parameter
00 -> FF	parameter eleven, zero if no parameter
00 -> FF	parameter twelve, zero if no parameter
00 -> FF	parameter thirteen, zero if no parameter

00 -> FF parameter fourteen, zero if no parameter

SINGULAR
VALID-in-INDEXER-mode

93 0 Request state of the move definitions (complete or incomplete)

The state of the move definitions is returned. This command is requesting whether or not all the moves that have been loaded into the indexer for definition have been "crunched" by the indexer. In other words, have the move definitions been converted from their raw form to a form ready for performing moves by the indexer. An FFh is returned if the moves have all been "crunched" and a 00h is returned if the moves have not all been "crunched." If a move has not been "crunched" it may still be performed by the indexer. The difference between a move being "crunched" and not being "crunched" is not the ability to perform the move but, rather, the delay time between asking a move to be performed and the move actually being executed. For all moves there are some pre-move calculations involved, however, if a move has been "crunched" the amount of pre-move calculations to be performed is minimized and the resulting delay time between asking that a move be performed and the move actually being performed is minimized. (If the indexer is asked to perform a move that has not been "crunched" the indexer will perform the "crunching" of that move and begin the move.) The following response would result from this command.

93	identifies the command the response is intended for
00 or FF	00 means not all the moves have been "crunched" FF means all the moves have been "crunched"
00 : 00	bytes three through sixteen are always zero and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

94 0 Request state of the TRIGGER inputs (2-5)

This command will return the current state of TRIGGER inputs two through five. These inputs are level sensitive inputs; TRIGGER inputs one and six are edge sensitive inputs. The information is returned in one byte; bits two through five reflect the state of the corresponding TRIGGER input, a 1 for TTL high and a 0 for TTL low. The following message would be received from this command.

94		identifies the command the response is intended for
00XXXX00b		eight bit number, each bit is explained below bit 0 is the least significant bit bit 7 is the most significant bit
bit 0 ->	0	always
	1	never
bit 1 ->	0	always
	1	never
bit 2 ->	0	TRIGGER 2 is currently a TTL low
	1	TRIGGER 2 is currently a TTL high
bit 3 ->	0	TRIGGER 3 is currently a TTL low
	1	TRIGGER 3 is currently a TTL high
bit 4 ->	0	TRIGGER 4 is currently a TTL low
	1	TRIGGER 4 is currently a TTL high
bit 5 ->	0	TRIGGER 5 is currently a TTL low
	1	TRIGGER 5 is currently a TTL high
bit 6 ->	0	always
	1	never
bit 7 ->	0	always
	1	never
00		bytes three through
:		sixteen are always zero
00		and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

95 0 Request state of the Joystick status input

This command is used to determine whether the Joystick is requesting control of the motor. When the Compumotor Model 852 Joystick's Enable switch is in the Enable or the Shutdown position it actively requests control of the motor. When the Enable switch is in the Disable position, or no Joystick is attached to the 1810/11's Joystick input, or power is off to the Joystick there will be no Joystick request.

The 1810/11 Indexer takes no action based on the state of this input bit from the Joystick. So, switching the Joystick Enable switch to the Enable position is not recognized by the 1810/11. This is different from Compumotor's Model 2100 which does recognize Joystick requests. In the case of the 1810/11 Indexer, the Enable the Joystick command (11 hex) must be issued to move the motor via the Joystick.

Thus, this command is used to find out if the Joystick is requesting control of the motor, which would necessitate sending an 11 command. So, it's possible to tell if a Joystick is present, and if it wants control of the motor. Sending a 10 command disables the Joystick regardless of the state of the Joystick request input.

A response looks like, 95 00 or FF, followed by bytes 3-16 which are always zero and are always present. 00 indicates the input is low, FF indicates the input is high.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

97 0 Request the state of the programmable output bits

This command is used to determine the state of the programmable output bits. In the response byte, bit 1 corresponds to the state of programmable output bit 1; bit 2 corresponds to the state of programmable output bit 2 (bit 0 is not used). The state returned is the programmed state of the programmable output bits. The response will always accurately indicate the state of the output circuit unless an external signal is overpowering the outputs, or the output circuity is defective.

A response looks like, 97 00 or 02 or 04 or 06, followed by bytes 3-16 which are always zero and are always present. Bit 1 of the first parameter byte indicates the state of programmable output bit 1 and bit 2 indicates the state of programmable output bit 2.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

A0 1 Interrupt at the start of the next move

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Interrupt the host bus when the next move begins. The interrupt occurs only once, at the beginning of the next move. Subsequent moves will not cause an interrupt. The interrupt one will receive from the 1810/11 indexer is a "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (A0h) followed by fifteen zeroes. If a move is occurring at the time this command is issued the next move to start will cause an interrupt, the current move will not cause an interrupt as a result of this command. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A1 1 Interrupt at the start of every move

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Same as A0h except an interrupt will be issued every time a move begins. (Unless the command is issued when a move is in progress, then the move in progress will not cause an interrupt.) The message in the OUTPUT DATA BUFFER will be an A1h followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A2 1 Interrupt at constant, nonzero velocity of the next move

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Interrupt the host bus when the next move reaches constant, nonzero velocity. The interrupt occurs only once, at constant nonzero velocity of a next move. The interrupt one will receive from the 1810/11 indexer is the "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (A2h) followed by fifteen zeroes. Subsequent constant, nonzero velocities of any moves will not cause an interrupt. If a move is occuring at the time this command is issued the next move started will cause an interrupt, the current move will not cause an interrupt as a result of this command. If constant, nonzero velocity is not reached (the move is prematurely stopped or the performance of a preset move results in a triangular profile) no interrupt will be received by the host. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A3 1 Interrupt at constant, nonzero velocity of every move

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Same as A2h except an interrupt will be issued every time any move reaches constant, nonzero velocity. (Unless the command is issued when a move is in progress, then the move in progress will not cause an interrupt.) The message one will receive will be an A3h followed by fifteen zeroes. If constant, nonzero velocity is not reached (the move is prematurely stopped or the performance of a preset move results in a triangular profile) no interrupt will be received by the host. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A4 1 Interrupt at the next end of motion

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Interrupt the host bus when the next end of motion occurs. The interrupt occurs only once, when the motor arrives at zero velocity. Subsequent excursions to zero velocity will not cause an interrupt. The interrupt one will receive from the 1810/11 indexer is a "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (A4h) followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A5 1 Interrupt at every end of motion

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Same as A4h except an interrupt will be issued every time end of motion occurs is reached. The message one will receive in the DATA BUFFER will be an A5h followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

A8 1 Interrupt the next time the motor hits the + limit

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Interrupt the host bus when the + direction limit switch is struck. The interrupt occurs only once, the first time the + limit switch is struck. Subsequent active + direction limits will not cause an interrupt. The interrupt one will receive from the 1810/11 indexer is a "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (A8h) followed by fifteen zeroes. If the + direction limit is active at the time this command is issued the next active + direction limit will cause an interrupt, the current active + limit will not cause an interrupt as a result of this command.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

A9 1 Interrupt every time the motor hits the + limit switch

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Same as A8h except an interrupt will be issued every time the + direction limit is struck. (Unless the command is issued when the + limit is active, then the next active + limit will cause an interrupt.) The message one would receive in the OUTPUT DATA BUFFER is an A9h followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

AA 1 Interrupt the next time the motor hits the - limit

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Interrupt the host bus when the - direction limit switch is struck. The interrupt occurs only once, the first time the - limit switch is struck. Subsequent active - direction limits will not cause an interrupt. The interrupt one will receive from the 1810/11 indexer is a "message ready" interrupt, the message in the DATA BUFFER is this command (AAh) followed by fifteen zeroes. If the - direction limit is active at the time this command is issued the next active - direction limit will cause an interrupt, the current active - limit will not cause an interrupt as a result of this command. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

AB 1 Interrupt every time the motor hits the - limit switch

00 or nonzero, 00 = disable the interrupt, nonzero = enable the interrupt

Same as AAh except that an interrupt will be issued every time the - direction limit is struck. (Unless the command is issued when the - limit is active, then the next active - limit will cause an interrupt.) The message one will receive in the OUTPUT DATA BUFFER will be an ABh followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

AC 2 Interrupt on TRIGGER "X" active

01 or 06, 01 = TRIGGER 1 and 06 = TRIGGER 6
00 or nonzero, 00 = disable this function, nonzero = enable

Interrupt the host bus when when the specified TRIGGER goes active. The interrupt one will receive from the 1810/11 indexer is a "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (ACh) followed by the parameter byte (a 01 or a 06) and fourteen zeroes.

Only one function may be performed by a given TRIGGER input. For example, if TRIGGER 1 has already been initialized to STOP motion on active TRIGGER 1 no other function may be specified for active TRIGGER 1 until the STOP motion function is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

AF 0 Inhibit all of the interrupts

Turns off all interrupts turned on by commands A0h through AEh. If none of the above interrupts are enabled this command has no effect.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

C8 E define move "X," define it as a relative, trapezoidal move

01 -> FF,	move number
XY	X = velocity range and Y = acceleration range (see below)
00 -> 7F,	most significant byte of the velocity
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the velocity
00 -> 7F,	most significant byte of the acceleration
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the acceleration
00 -> FF,	most significant byte of the relative distance (pulses)
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the relative distance

This command will define move "X" as a relative, trapezoidal move to be performed by the "perform move number 'X'" command (40h). The first parameter byte specifies the move number, a move number of 00h is illegal and will be ignored. The most significant nibble of the second parameter byte denotes the velocity range of the move and the least significant nibble denotes the acceleration range. The velocity range determines the units of the velocity number sent, the maximum velocity acheivable, and the velocity resolution. The acceleration range determines the units of the acceleration number with respect to the velocity number's units. The last twelve bytes specify the velocity, acceleration, and distance (in

pulses); all three numbers are interpreted in two's complement form. The velocity and acceleration numbers must be specified as positive numbers. The distance specified is a relative distance! A distance of 80000000h is illegal and may not be specified. All of the bytes must be present in order for a relative move to be defined. Redefinition of a move is allowed by reissuing a complete define a move command.

Only one move number "X" may be defined at any time; for example, one can only define one move number 9, whether move 9 is a relative move, an absolute move, or a continuous move is of no relevance, there may be only one move 9.

There are six velocity ranges defined with three acceleration ranges per velocity range. If one fails to specify a defined range the move definition is invalid. The velocity and acceleration ranges are as follows.

0 velocity number is pulses/second (pps)
maximum velocity is 546,133.3 Hertz
velocity resolution is 16.667 Hertz

1 velocity number is pulses/ten seconds (0.1 pps)
maximum velocity is 54,613.33 Hertz
velocity resolution is 1.6667 Hertz

2 velocity number is pulses/hundred seconds (0.01 pps)
maximum velocity is 5,461.333 Hertz
velocity resolution is 0.16667 Hertz

3 velocity number is pulses/thousand seconds (0.001 pps)
maximum velocity is 546.1333 Hertz
velocity resolution is 0.016667 Hertz

4 velocity number is pulses/sixty seconds (ppm)
maximum velocity is 9,102.222 Hertz
velocity resolution is 0.27778 Hertz

5 velocity number is pulses/second (pps)
maximum velocity is 1,092,266.67 Hertz
velocity resolution is 33.333 Hertz

0 acceleration number is velocity units/second
1 acceleration number is velocity units/ten seconds
2 acceleration number is velocity units/hundred seconds

A relative, preset move is defined to be an excursion from zero velocity to a nonzero velocity and back to zero velocity. The goal of a preset move is to get from the current position to a new position. Therefore, if the indexer is performing a preset move it considers the action of "performing a move" to be synonymous with the "motor moveing." Thus, a relative, preset move can only be performed if the motor is not moving. It is a good idea not to mix continuous moves with preset moves.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

CB E **Define move "X," define it as an absolute, trapezoidal move**

01 -> FF,	move number
XY	X = velocity range and Y = acceleration range (see below)
00 -> 7F,	most significant byte of the velocity
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the velocity
00 -> 7F,	most significant byte of the acceleration
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the acceleration
00 -> FF,	most significant byte of the absolute position (pulses)
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the absolute position

This command will define move "X" as an absolute, trapezoidal move to be performed by the "perform move number 'X'" command (40h). The first parameter byte specifies the move number, a move number of 00h is illegal and will be ignored. The most significant nibble of the second parameter byte denotes the velocity range of the move and the least significant nibble denotes the acceleration range. The velocity range determines the units of the velocity number sent, the maximum velocity acheivable, and the velocity resolution. The acceleration range determines the units of the acceleration number with respect to the velocity number's units. The last

twelve bytes specify the velocity, acceleration, and distance (in pulses); all three numbers are interpreted in two's complement form. The velocity and acceleration must be specified as positive numbers. The distance specified is an absolute distance! All of the bytes must be present in order for an absolute move to be defined. Redefinition of a move is allowed by reissuing a complete define a move command.

Absolute positions can only be defined within the range C0000001h through 3FFFFFFFh (-3FFFFFFFh through +3FFFFFFFh or -(2^30)-1 through +(2^30)-1). An absolute position specified outside of this range is invalid. If the motor is situated outside of this range one must perform a relative distance move to return to the inside of the range before an absolute move can be performed.

Only one move number "X" may be defined at any time; for example, one can only define one move number 9, whether move 9 is a relative move, an absolute move, or a continuous move is of no relevance, there may be only one move 9.

There are six velocity ranges defined with three acceleration ranges per velocity range. If one fails to specify a defined range the move definition is invalid. The velocity and acceleration ranges are as follows.

0 velocity number is pulses/second (pps)
maximum velocity is 546,133.3 Hertz
velocity resolution is 16.667 Hertz

1 velocity number is pulses/ten seconds (0.1 pps)
maximum velocity is 54,613.33 Hertz
velocity resolution is 1.6667 Hertz

2 velocity number is pulses/hundred seconds (0.01 pps)
maximum velocity is 5,461.333 Hertz
velocity resolution is 0.16667 Hertz

3 velocity number is pulses/thousand seconds (0.001 pps)
maximum velocity is 546.1333 Hertz
velocity resolution is 0.016667 Hertz

4 velocity number is pulses/sixty seconds (ppm)
maximum velocity is 9,102.222 Hertz
velocity resolution is 0.27778 Hertz

5 velocity number is pulses/second (pps)
maximum velocity is 1,092,266.67 Hertz

74 pgs

velocity resolution is 33.333 Hertz

0	acceleration number is velocity units/second
1	acceleration number is velocity units/ten seconds
2	acceleration number is velocity units/hundred seconds

An absolute, preset move is defined to be an excursion from zero velocity to a nonzero velocity and back to zero velocity. The goal of a preset move is to get from the current position to a new position. Therefore, if the indexer is performing a preset move it considers the action of "performing a move" to be synonymous with the "motor moving." Thus, an absolute, preset move can only be performed if the motor is not moving. It is a good idea not to mix continuous moves with preset moves.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

CE B define move "X," define it as a continuous, trapezoidal move

01 -> FF,	move number
XY	X = velocity range and Y = acceleration range (see below)
00 -> 7F,	most significant byte of the continuous velocity
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the continuous velocity
00 -> 7F,	most significant byte of the acceleration
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the acceleration
00 or nonzero,	direction of the move, 00 = + and nonzero = -

This command will define move "X" as a continuous, trapezoidal move to be performed by the "perform move number 'X'" command (40h). The first parameter byte specifies the move number, a move number of 00h is illegal and will be ignored. The most significant nibble of the second parameter byte denotes

the velocity range of the move, the least significant nibble denotes the acceleration range. The velocity range determines the units of the velocity number, the maximum acheivable velocity, and the velocity resolution. The acceleration range determines the units of the acceleration number with respect to the velocity number's units. The velocity and acceleration of the move are interpreted as two's complement numbers, only positive numbers are allowed. The eleventh parameter byte specifies the direction of travel: 0 for travel in the + direction and nonzero for travel in the - direction. All of the bytes must be present in order for a continuous move to be defined. Redefinition of a move is allowed by reissuing a complete define a move command.

Only one move number "X" may be defined at any time; for example, one can only define one move number 9, whether move 9 is a relative move, an absolute move, or a continuous move is of no relevance, there may be only one move 9.

There are six velocity ranges defined with three acceleration ranges per velocity range. If one fails to specify a defined range the move definition is invalid. The velocity and acceleration ranges are as follows.

0 velocity number is pulses/second (pps)
maximum velocity is 546,133.3 Hertz
velocity resolution is 16.667 Hertz

1 velocity number is pulses/ten seconds (0.1 pps)
maximum velocity is 54,613.33 Hertz
velocity resolution is 1.6667 Hertz

2 velocity number is pulses/hundred seconds (0.01 pps)
maximum velocity is 5,461.333 Hertz
velocity resolution is 0.16667 Hertz

3 velocity number is pulses/thousand seconds (0.001 pps)
maximum velocity is 546.1333 Hertz
velocity resolution is 0.016667 Hertz

4 velocity number is pulses/sixty seconds (ppm)
maximum velocity is 9,102.222 Hertz
velocity resolution is 0.27778 Hertz

5 velocity number is pulses/second (pps)
maximum velocity is 1,092,266.67 Hertz
velocity resolution is 33.333 Hertz

0 acceleration number is velocity units/second
1 acceleration number is velocity units/ten seconds
2 acceleration number is velocity units/hundred seconds

A continuous move is defined to be an excursion from one velocity to another velocity. The goal of a continuous move is to get from the current velocity to a new velocity. Therefore, if the indexer is performing a continuous move the action of "performing a move" is NOT synonymous with the "motor moving," it is synonymous with either the act of accelerating or decelerating. Thus, a continuous move can only be performed if the motor is not accelerating or decelerating AND a preset move is not being performed. It is a good idea not to mix continuous moves with preset moves.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

D6 4 Define the start/stop velocity

00 -> 7F, most signficant byte of the start/stop velocity
00 -> FF, second most significant byte
00 -> FF, third most significant byte
00 -> FF, least significant byte of the start/stop velocity

This command will define the start/stop velocity. The start/stop velocity is a part of all moves performed. If a move whose velocity is less than the start/stop velocity is performed no acceleration is performed to that velocity, instead the indexer will jump to the move's defined velocity. If a move whose velocity is greater than the start/stop velocity is performed the acceleration to the move's velocity is begun at the start/stop velocity and deceleration ends at the start/stop velocity. The start/stop velocity is always specified in units of pulses per second (pps). After power on or reset the start/stop velocity is defined as zero.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

D7 1 Delete move "X"

01 -> FF, move number

Deletes move number "X" from the library of defined moves if move "X" has been previously defined. Otherwise, this command does nothing.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

D8 0 End definition of sequence buffer

Instructs the indexer that the last command to be stored in the sequence buffer has been transmitted. Commands sent following this command will be interpreted literally, and performed. This command ends definition of whichever sequence buffer is currently being defined.

VALID-in-INDEXER-mode

D9 1 Begin definition of sequence buffer "X"

01 -> 0F indicates which sequence buffer is being defined

Instructs the indexer to begin filling the specified sequence buffer. Every command sent after this command is considered a candidate for the sequence buffer except "end definition of sequence buffer" and "perform the test switch function" (commands D8h and FFh). In other words, no commands are performed after this command has been sent to the indexer, they are considered for insertion into a sequence buffer; if they are valid sequence buffer commands they are placed into the sequence buffer, if they are not valid sequence

buffer commands they are ignored.

Thus, no commands can be performed (except the above exceptions!) while a sequence buffer is being filled.

There is a limited amount of space allocated for all sequence buffers (about 600 decimal bytes). Even though 15 sequence buffers may be defined, if the available buffer space becomes filled by some number of sequence buffers less than 15, you may not define any more sequence buffers until some space is made by deleting another sequence buffer (command D7). To find out how much space remains in the sequence buffer refer to command E2.

SINGULAR
VALID-in-INDEXER-mode

DA 1 Delete sequence buffer "X"

01 -> 0F indicates which sequence buffer is to be deleted

This command deletes one of the 15 sequence buffers available in the 1810/11. All sequence buffers in the 1810/11 share a common memory, so if one or more of the sequence buffers is very large less than 15 buffers will fit into the memory available. Deleting a sequence buffer frees memory making it available for use by other sequence buffers. The amount of space remaining in the sequence buffer common memory may be requested via the E2 command.

SINGULAR
VALID-in-INDEXER-mode

E0 C Place the following data into the velocity-distance buffer

00 -> FF, most significant byte of velocity A
00 -> FF, least significant byte of velocity A
00 -> FF, most significant byte of relative distance A
00 -> FF, least significant byte of relative distance A
00 -> FF, most significant byte of velocity B
00 -> FF, least significant byte of velocity B
00 -> FF, most significant byte of relative distance B

00 -> FF, least significant byte of relative distance B
00 -> FF, most significant byte of velocity C
00 -> FF, least significant byte of velocity C
00 -> FF, most significant byte of relative distance C
00 -> FF, least significant byte of relative distance C

The data will be loaded into the velocity-distance streaming buffer if the 1810/11 is in the velocity-distance streaming mode, otherwise the command is ignored. The parameter bytes are three velocity-distance pairs. All three pairs must be supplied if the command is to be executed. The velocity number is a fifteen byte number; the most significant bit is the sign bit, 1 = CCW motion and 0 = CW motion. The velocity number, V, has an unsigned range of 0000h through 7FFFh. The following formula can be used to determine what the velocity number means in units of the "effective clock frequency."

velocity = effective clock frequency * (V / 65,536)

The "effective clock frequency" is determined when velocity-distance streaming mode is entered (see command 51h). The relative distance is in units of pulses. When supplying velocity-distance pairs one must be sure that the distance is such that it will take a minimum of two milliseconds to cover at its associated velocity. In other words, one should be sure the following equation is satisfied, or an incorrect profile can result.

(relative distance) / (velocity) >= 0.002 seconds

If a distance of zero is supplied the velocity-distance pair is ignored; if a velocity of zero is specified the distance is irrelevant (and may be zero), but must be supplied!

One cannot fill the velocity streaming buffer with more bytes than it will hold! To determine how many bytes are available in the velocity streaming buffer use the E2h command below, the E2h command will report the number of available bytes in the velocity streaming buffer. There must be at least fifteen free bytes in the velocity streaming buffer to use this command, if there are fourteen or fewer the data supplied with this command will be lost, it will not be placed into the velocity streaming buffer.

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

E1 E Place the following data into the velocity-time buffer

00 -> FF, most significant byte of velocity number A
00 -> FF, least significant byte of velocity number A
00 -> FF, most significant byte of velocity number B
00 -> FF, least significant byte of velocity number B
00 -> FF, most significant byte of velocity number C
00 -> FF, least significant byte of velocity number C
00 -> FF, most significant byte of velocity number D
00 -> FF, least significant byte of velocity number D
00 -> FF, most significant byte of velocity number E
00 -> FF, least significant byte of velocity number E
00 -> FF, most significant byte of velocity number F
00 -> FF, least significant byte of velocity number F
00 -> FF, most significant byte of velocity number G
00 -> FF, least significant byte of velocity number G

The data will be loaded into the velocity-time streaming buffer if the 1810/11 is in velocity-time streaming mode. Otherwise, the command is ignored. As is apparent, one must load seven velocity values at a time into the velocity-time streaming buffer. (The time is not loaded since the indexer automatically updates the velocity at a prespecified rate.) If one is finishing a move and does not have a full seven velocities to place into the buffer the excess velocity values should be equal to the final velocity value. For example, if one was performing a velocity streaming move to a velocity of 1000h and the sequence of velocities one desired to send to the indexer was 100h, 200h, 300h, 400h, 500h, 600h, 700h, 800h, 900h, and 1000h the following commands would be sent to the 1810/11 to fill the velocity-streaming buffer:

E1,01,00,02,00,03,00,04,00,05,00,06,00
E1,07,00,08,00,09,00,10,00,10,00,10,00

Every time this command is used seven velocity numbers must be supplied. If one was to send either of the following commands in place of the second command above the motor would not end up at 1000h. In the first case the 1810/11 would ignore the command entirely so the motor would remain at a velocity of 900h. In the second case the 1810/11 would go to 1000h and then go directly to zero on the next update.

E1,07,00,08,00,09,00,10,00
E1,07,00,08,00,09,00,10,00,00,00,00,00

The velocity numbers have the same meaning as they do for the above command (E0h). The velocity number is a fif-

teen byte unsigned number; the most significant bit is the sign bit, 1 = CCW motion and 0 = CW motion. The velocity number, V, has an unsigned range of 0000h through 7FFFh. The following formula can be used to determine what the velocity number means in units of the "effective clock frequency."

velocity = effective clock frequency * (V / 65,536)

The "effective clock frequency" is determined when velocity-time streaming mode is entered (see command 52h).

One cannot fill the velocity streaming buffer with more bytes than it will hold! To determine how many bytes are available in the velocity streaming buffer use the E2h command below, the E2h command will report the number of available bytes in the velocity streaming buffer. There must be at least fifteen free bytes in the velocity streaming buffer to use this command, if there are fourteen or fewer the data supplied with this command will be lost, it will not be placed into the velocity streaming buffer.

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

E2h 0 Request # of free bytes in velocity-streaming/sequence buffer

This command reports back to the host the number of bytes available in the sequence buffer if the 1810/11 indexer is in indexer mode (see 50h command) and the number of bytes available in the velocity streaming buffer if the 1810/11 indexer is in velocity streaming mode (see commands 51h and 52h). Five bytes are returned to the host, this command followed by the number of free bytes. The number of free bytes returned is a four byte number. The following illustrates the response one would receive from this command.

E2	identifies the command the response is intended for
00	MS byte of the number of free bytes, always zero
00	second MS byte, always zero
00 -> FF	third MS byte
00 -> FF	LS byte of the number of free bytes

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

E3 F Place the following data into the distance-time buffer

01 -> 07, number of significant distnce values
00 -> FF, most significant byte of distance number A
00 -> FF, least significant byte of distance number A
00 -> FF, most significant byte of distance number B
00 -> FF, least significant byte of distance number B
00 -> FF, most significant byte of distance number C
00 -> FF, least significant byte of distance number C
00 -> FF, most significant byte of distance number D
00 -> FF, least significant byte of distance number D
00 -> FF, most significant byte of distance number E
00 -> FF, least significant byte of distance number E
00 -> FF, most significant byte of distance number F
00 -> FF, least significant byte of distance number F
00 -> FF, most significant byte of distance number G
00 -> FF, least significant byte of distance number G

The data will be loaded into the distance-time streaming buffer if the 1810/11 is in distance-time streaming mode. Otherwise, the command is ignored. All fifteen bytes must be sent, but the 1810/1811 will place into the buffer only the number of distance parts specified by the first parameter of this command. (The time is not loaded since the indexer automatically updates the velocity at a prespecified rate.) value.

The distance number is a sixteen bit signed number; that is, the direction of the incremental distance is contained in the two's complement value. The distance number, D, has a signed range of 8000h through 7FFFh. The value of 8001h is taken as a special signal to the 1810. It carries a distance value of zero, but also tells the 1810 to quit executing the distance-time buffers and to go to zero velocity. This point must terminate a

distance-time buffer to insure the 1810 travels a distance which is the sum of all the distances in the buffer, and comes to zero velocity. The following formula can be used to determine at what each (distance included) will be traveled.

velocity = D/(update rate *1 msec)

The "update rate" is determined when distance-time streaming mode is entered (see command 53h). Care must be taken to insure that the resultant velocity does not exceed the maximum specified by the clock frequency and divisor specified in the 53 command.

One cannot fill the distance streaming buffer with more bytes than it will hold! To determine how many bytes are available in the streaming buffer use the E2h command below, the E2h command will report the number of available bytes in the streaming buffer. There must be at least fifteen free bytes in the velocity streaming buffer to use this command, if there are fourteen or fewer the data supplied with this command will be lost, it will not be placed into the streaming buffer.

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

EE 5 Define command to be executed during the streaming buffer. (or Define bogus streaming value)

00	->	FF	the most significant byte of special value
00	->	FF	the least significant byte of special value
00	->	FF	command to be executed
00	->	FF	first parameter byte of command to be executed
00	->	FF	second parameter byte of command to be executed

This command defines a command which will be executed upon the indexer finding a special, or bogus, velocity value when executing the velocity streaming buffer. **The special velocity values can be any number except zero.** In distance-time streaming, the value 8000h is also exclued. The commands which can be executed upon reaching a special velocity value in a velocity streaming buffer must be VALID-in-VELOCITY-STREAMING-mode and they must be VALID-concurrent-with-VELOCITY-STREAMING. In addition, they are limited to 1, 2 and 3 byte commands--that

is, commands which have 0, 1, or 2 parameter bytes. The null command (00 hex) is a special command for the EE function and is used to delete a velocity marker, or undefine a velocity marker.

The way this command is used is to say, for instance, 7F FF would be defined as a special velocity. One might define it to set a programmable output bit. So the command sequence sent to accomplish this would be EE 7F FF 0B 01 00. That sequence would accomplish setting programmable output bit 1 whenever 7F FF is found in the velocity streaming buffer. Notice that the third parameter byte is used even though the 0B command does not require it, the EE command requires all five parameter bytes to be filled whether they are used or not.

10 special velocity markers may be defined, so you might have a corresponding special velocity value of 7F FE that would be set up to clear programmable output bit one whenever a velocity value of 7F FF is encountered. This way you can mark points within the buffer. If two or more special velocity markers are encountered in the buffer in a row, only the last one will be executed. In other words, one cannot have consecutive special velocity values. Enough time must be allowed for the command associated with a special velocity to be executed before another special velocity marker is be encountered. If, for instance, you have a special velocity marker in the buffer, a real velocity, then another special velocity marker in the buffer, the second special velocity marker would not be executed if the command associated with the first special velocity marker was still in the process of being executed. Therefore, one should allow enough time for commands associated with special velocity markers to be executed.

Generally, most of the commands associated with special velocity markers take approximately two to four milliseconds to execute. So, if you assure that there will be about a four millisecond time frame between special velocity markers in a buffer, there should be no problem. If you find that a command associated with a special velocity marker is not being executed, it may be due to its being too close to another special velocity marker.

If you wish to undefine a special velocity as a marker, you would send, as the parameter bytes, the special velocity value and three zeros, i.e., the null command referred to above. The null command will erase that special velocity marker from the table.

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

FD 0 Request software part number and revision

This command enables the user to determine the part number and revision letter or the software in the 1830 Indexer, without removing the Indexer from the host. The information is returned in a twelve or thirteen character string, with each byte being one ASCII character. The following message would be received from this command:

FD	identifies command for which the response is intended
39	ASCII character "9"
34	ASCII character "4"
2D	ASCII character "-"
XX...XX	Six ASCII characters representing the port number
2D	ASCII character "-"
XX,XX	two ASCII characters representing revision number
XX	ASCII character representing revision letter
(XX)	ASCII character representing revision letter modifier. This character will only be present on indexers containing temporary, or intermediate, software.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

FF 0 Perform the test switch function

This command is supplied primarily as a debugging tool. If a standard Compumotor (25,000 pulses per revolution) is properly attached to the 1810/11 indexer and has power applied this command will result in performance of the test switch function. The test switch is a small mechanical switch on the edge of the board, when the switch is pushed a standard Compumotor (25,000 pulses per revolution) will turn one complete revolution in the CCW direction and one complete revolution in the CW direction. The pulse frequency during each of the above moves is 5000 Hertz, which results in a 0.2 rps move for a standard Compumotor. Following this move the "board monitor alarm" will be "set off," which results in the small LED on the edge of the board lighting up. After the test switch function has been exercised the board MUST be reset, either by a bus reset, cycling the power, or by setting the reset bit in the 1810/11's CONTROL BYTE.

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

APPENDIX B

Following is a list of the **configuration jumpers** on the 1810/11 Motor Controller. Most jumpers are either explicitly labelled on the board or are located between others so labelled. The jumpers are ordered roughly as you would read a page. Jumpers that are mutually exclusive are so noted below, except where it is physically impossible to install two jumpers because they share a pin.

Caution: Before inspecting the configuration jumpers, verify the revision level of the printed circuit board you are using and follow the appropriate column below. The revision level may be found on the right side of the component side of the board (MULTIBUS connector toward you). The number is 61-002154-XX, where XX is the revision level of the board.

On revision **-03 and -04** boards, Jumpers JU5, 1, 3, 5, 6, 8, 10, 12, 15, 23, 28, 44, 57 and 59 are installed at the factory.

On revision **-01 and -02** boards, Jumpers JU1, 2, 12, 25, 32, 41, 42, 43, 48, 50 and 68 are installed at the factory.

Jumper installed produces stated condition.

JUMPER -04	JUMPER -02 -03	DESCRIPTION P.C Board Revision	
---	JU3	Bypass Trigger bit one.	
---	JU6	2732 EPROM used.	
---	JU7	27128 EPROM used.	
---	JU9	27128 EPROM used.	
JU1	---	User Programmable bit 1 **or**,	
JU2	---	Motor-driver Gearshift function.	
JU3	---	User Programmable bit 2 **or**,	
JU4	---	Motor-driver Overdrive function.	
JU5	JU2	Bypass CW Limit.	Note 1.
JU6	JU1	Bypass CCW Limit.	Note 1.
JU7	---	Shutdown driven differentially **or**,	
JU8	---	Shutdown driven single-ended.	
JU9	---	Motor-driver Fault **or**,	
JU10	---	Trigger bit three.	
JU11	---	Motor-driver Slip Fault **or**, (see next page)	

JUMPER -04 -02 -03	DESCRIPTION P.C Board Revision	
JU12 ---	Trigger bit four.	
JU13 JU69	Select 75174 driver IC or,	
JU14 JU70	Select 75172 driver IC.	
JU15 JU68	Enable test-switch. Remove to disable test-switch.	
JU16 JU4	Invert encoder Home bit.	1811 only.
JU17 JU5	Invert encoder Direction bit.	1811 only.
	Enable Interrupt from:	
JU18 JU41	"OUTPUT DATA BUFFER READY" Status	
JU19 JU42	"FAIL" Status.	
JU20 JU40	"INPUT DATA BUFFER READY" Status	
JU21 JU66	Enable Waitstate generator or,	
JU22 JU67	Disable Waitstate generator.	
JU23 JU43	Enable board monitor Alarm.	
	MULTIBUS XACK/ Delay:	
JU24 JU16	Delay 0.23 - 0.46 usecs	Note 3.
JU25 JU15	Delay 0.46 - 0.69 usecs	Note 3.
JU26 JU14	Delay 0.69 - 0.92 usecs	Note 3.
JU27 JU13	Delay 0.92 - 1.15 usecs	Note 3.
JU28 JU12	Delay 1.15 - 1.38 usecs	Note 3.
	MULTIBUS Board Address:	
JU29 JU24	Bit 12 weight = 4,096	
JU30 JU23	Bit 13 weight = 8,192	
JU31 JU22	Bit 14 weight = 16,384	
JU32 JU21	Bit 15 weight = 32,768	
JU33 JU20	Bit 16 weight = 65,536	
JU34 JU19	Bit 17 weight = 131,072	
JU35 JU18	Bit 18 weight = 262,144	
JU36 JU17	Bit 19 weight = 524,288	
	MULTIBUS Address Mode:	
JU37 JU25	16 bit addressing or more	
JU38 JU26	12 bit addressing or less	
	MULTIBUS Board Address:	
JU39 JU30	Bit 8 weight = 256	
JU40 JU29	Bit 9 weight = 512	
JU41 JU28	Bit 10 weight = 1,024	
JU42 JU27	Bit 11 weight = 2,048	

MULTIBUS Address Mode:

JU43 JU31 12 bit addressing or more
JU44 JU32 8 bit addressing

JUMPER -03	JUMPER -02	DESCRIPTION P.C Board Revision
		MULTIBUS Board Address:
JU45	JU39	Bit 1 weight = 2
JU46	JU38	Bit 2 weight = 4
JU47	JU37	Bit 3 weight = 8
JU48	JU36	Bit 4 weight = 16
JU49	JU35	Bit 5 weight = 32
JU50	JU34	Bit 6 weight = 64
JU51	JU33	Bit 7 weight = 128
JU52	JU10	8k by 8 RAM used.
JU53	JU11	2K by 8 RAM used.
JU54	JU8	27256 EPROM not used.
JU55	---	27256 EPROM used.
JU56	JU49	MULTIBUS MWTC/ Install if memory-mapped.
JU57	JU50	MULTIBUS IOWC/ Install if I/O mapped.
JU58	JU47	MULTIBUS MRDC/ Install if memory-mapped.
JU59	JU48	MULTIBUS IORC/ Install if I/O mapped.
JU60	JU51	MULTIBUS INH1/ Enable Note 4.
JU61	JU53	MULTIBUS AACK/ Enable Note 5.
JU62	JU52	MULTIBUS INH2/ Enable Note 4.
JU63	JU54	MULTIBUS INT7/ from 1810/11 Interrupt Note 6.
JU64	JU55	MULTIBUS INT6/ from 1810/11 Interrupt Note 6.
JU65	JU56	MULTIBUS INT5/ from 1810/11 Interrupt Note 6.
JU66	JU57	MULTIBUS INT4/ from 1810/11 Interrupt Note 6.
JU67	JU58	MULTIBUS INT3/ from 1810/11 Interrupt Note 6.
JU68	JU59	MULTIBUS INT2/ from 1810/11 Interrupt Note 6.
JU69	JU60	MULTIBUS INT1/ from 1810/11 Interrupt Note 6.
JU70	JU61	MULTIBUS INT0/ from 1810/11 Interrupt Note 6.

Note 1. Install if there is no external limit.
Note 2. Install if input not used.
Note 3. Only one of JU24 thru JU28 may be installed.
Note 4. Install if 1810/11 "shadows" system RAM.
Note 5. Install to increase bus rate.
Note 6. Only one of JU63 thru JU70 may be installed.

APPENDIX C

Following is a list of the connectors found on the top edge of the 1810/11 Motor Controller.

USER CHASSIS GROUND CONNECTOR

J1 on -03/04 Board
(2-pin Header)

Pinout for -03/-04 boards
J1

Pin #	Signal
1	Chassis ground
2	Chassis ground

J6 on -01/-02 Board
(2-pin Header)

Pinout for -01/-02 boards
J6

Pin #	Signal
1	Chassis ground
2	Chassis ground

AUXILLIARY FUNCTIONS CONNECTOR

J2 on -03/04 Board
(20-pin header)

Pinout for -03/-04 boards
J2

Pin #	Signal
1	Trigger 6
2	Trigger 6 return
3	CW limit
4	CW limit return
5	CCW limit
6	CCW limit return
7	Trigger 1
8	Trigger 1 return
9	Home limit
10	Trigger 5
11	Trigger 2
12	Trigger 4
13	Programmable output 2
14	programmable output 2 return
15	programmable output 1
16	programmable output 1 return
17	Trigger 3
18	Shutdown in
19	Auxilliary connector +5volts
20	Shield

J5 on -01/-02 Board
(20-pin header)

Pinout for -01/-02 boards
J5

Pin #	Signal
1	Trigger 6
2	Trigger 6 return
3	CW limit
4	CW limit return
5	CCW limit
6	CCW limit return
7	Trigger 1
8	Trigger 1 return
9	Home limit
10	Trigger 5
11	Trigger 2
12	Trigger 4
13	Programmable output 2
14	programmable output 2 return
15	programmable output 1
16	programmable output 1 return
17	Trigger 3
18	Shutdown in
19	Auxilliary connector +5volts
20	Shield

MOTOR/DRIVER CONNECTOR

J3 on -03/-04 Board
(26-pin header)

Pinout for -03/-04 boards
J3

Pin #	Signal
1	Step
2	Step return
3	Direction
4	Direction return
5	CW step
6	Remote shutdown
7	CCW step
8	Remote shutdown return
9	Shield
10	Gearshift return
11	Gearshift
12	Fault reset return
13	Fault reset
14	DC common
15	Motor/driver +5 volts
16	Drive fault return
17	Drive fault
18	Slip fault return
19	Slip fault
20	Overdrive return
21	Overdrive
22	DC common
23	Reserved
24	(n.c.)
25	(n.c.)
26	(n.c.)

J4 on -01/-02 Board
(26-pin header)

Pinout for -01/-02 boards
J4

Pin #	Signal
1	Step
2	Step return
3	Direction
4	Direction return
5	CW step
6	Remote shutdown
7	CCW step
8	Remote shutdown return
9	Shield
10	DC common
11	Reserved
12	Boost return
13	Boost
14	DC common
15	Motor/driver +5 volts
16	Boost shutdown return
17	Boost shutdown
18	Driver shutdown return
19	Driver shutdown
20	Alarm return
21	Alarm
22	Drive error return
23	Drive error
24	(n.c.)
25	(n.c.)
26	(n.c.)

JOYSTICK CONNECTOR

J4 on -03/-04 Board
(20-pin header on -03)
(16-pin header on -04)

J3 on -01/-02 Board
(20-pin header)

Pinout for -03/-04 boards

J4

Pin #	Signal
1	Remote enable
2	Shield
3	Direction in
4	DC common
5	At zero
6	Joystick +5 volts
7	Step in
8	(n.c.)
9	(n.c.)
10	(n.c.)
11	Remote acknowledge
12	(n.c.)
13	Shutdown in
14	(n.c.)
15	(n.c.)
16-20	(n.c.)

Pinout for -01/-02 boards

J3

Pin #	Signal
1	Remote enable
2	Shield
3	Direction in
4	DC common
5	At zero
6	Joystick +5 volts
7	Step in
8	(n.c.)
9	(n.c.)
10	(n.c.)
11	Remote acknowledge
12	(n.c.)
13	Shutdown in
14	(n.c.)
15	(n.c.)
16-20	(n.c.)

ENCODER CONNECTOR

J6 on -03/-04 Board
(26-pin header)

Pinout for -04 boards

J6 Pin #	Signal
1	Ch.A(+)
2	DC common
3	Ch.A(-)
4	DC common
5	Ch.B(+)
6	DC common
7	Ch.B(-)
8	DC common
9	Ch.Z(+)
10	DC common
11	Ch.Z(-)
12	DC common
13	n.c.
14	DC common
15	Shield
16	n.c.
17	n.c.
18	n.c.
19	Step out
20	+5 volts out
21	Direction out
22	+5 volts out
23	n.c.
24	+5 volts out
25	Home enable
26	n.c.

J1 on -01/-02 Board
(20-pin header)

Pinout for -01/-02/-03 boards

J1 Pin #	Signal
1	Step out
2	Step out return
3	Direction out
4	Direction out return
5	Home enable
6	Home enable return
7	Shield
8	DC common
9	+12 volts
10	DC common
11	-12 volts
12	DC common
13	Encoder +5 volts
14	DC common
15	Ch.Z(-)
16	Ch.Z(+)
17	Ch.B(+)
18	Ch.A(+)
19	Ch.A(-)
20	Ch.B(-)

Following is a conversion chart that lists the pin number the 1810/11 PC card header for any number of pins up to 25, and also the pin number a 'D' connector would have for each number of pins listed. This list should be useful for those who need to know which pin a given signal is on at the bulkhead connector normally used to get the signal off of the MULTIBUS cards and out of the MULTIBUS computer.

PC Pin #	9-pin 'D' Pin #	15-pin 'D' Pin	25-pin 'D' Pin #
1	1	1	1
2	6	9	14
3	2	2	2
4	7	10	15
5	3	3	3
6	8	11	16
7	4	4	4
8	9	12	17
9	5	5	5
10	-	13	18
11	-	6	6
12	-	14	19
13	-	7	7
14	-	15	20
15	-	8	8
16	-		21
17	-		9
18	-		22
19	-		10
20	-		23
21	-		11
22	-		24
23	-		12
24	-		25
25	-		13
26	-		-

APPENDIX D

Presented here are two of programs written in the C programming language which might be used by a host computer to read from and write to the 1810/11 Indexer. They were written with the idea of presenting a means by which the user could cut development time when using a Compumotor 1810/11 Indexer; and they have been used in a variety of forms with great success, but they have not been run in the form you see here. For this reason you may want to use the programs as structuring examples rather than cutting and pasting them into your present system.

Program names	Description
1810U.C	Utility program written in C
FIFO.C	Read/Write program written in C

Introduction:

The following functions are written two different ways, one assumes the 181X board is I/O mapped and the other assumes the 181X board is memory mapped. The I/O mapped version of the function will be written first followed by the memory mapped version of the function. For I/O mapping the function will work with either eight bit or twelve bit I/O mapping (assuming the functions "in", "out", "i_fifo", and "o_fifo" are written to accomodate either number of address bits). Memory mapping is a very strong function of your particular MULTIBUS CPU card and its associated processor. The memory mapping functions may need to be modified depending upon your computer's ability to address the MULTIBUS's total one megabyte memory address space. No assumption is made as to the number of address bits involved when using the 181X in the memory mapped space.

These functions have been written to be explanatory, not efficient or elegant. There is no guarantee that they will work exactly as written here, however, versions of these have been used.

Program #1: 1810U.C

```
#define COMMAND 0x10    /* if set in CONTROL, command in FIFO          */
#define RESET   0x20    /* if set in CONTROL, reset 1810               */
#define ENB_INT 0x40    /* if set in CONTROL, enables interrupts       */
#define MSG_RED 0x80    /* if set in CONTROL, message was read         */

#define MSG_RDY 0x08    /* if set in STATUS, message is ready          */
#define BFR_RDY 0x10    /* if set in STATUS, buffer is ready           */
#define FAILED  0x20    /* if set in STATUS, 181X has failed           */
#define INT_MSK 0x40    /* if set in STATUS, interrupts are masked     */
```

```
#define INTRUPT 0x80    /* if set in STATUS, 181X has interrupted us    */

#define STAT_ADDR address+1     /* STATUS BYTE address                  */
#define CTRL_ADDR address+1     /* CONTROL BYTE address                 */
#define DATA_ADDR address+0     /* DATA BUFFER address                  */

#define PASS    0       /* zero means passed                            */
#define FAIL    -1      /* nonzero means failed                         */

/*****************************************************************************/
/*                                                                           */
/*  SNDCMD - General function to write commands to the 181X's FIFO.          */
/*                                                                           */
/*   This function will send a variable length message to the 181X's         */
/* INPUT DATA BUFFER.  Included in this function are handshakes to test      */
/* that the INPUT DATA BUFFER is ready and to signal to the 181X that        */
/* there is a command ready in it's INPUT DATA BUFFER.                       */
/*                                                                           */
/* Name and Form of the function:                                            */
/*                                                                           */
/*      sndcmd (address, command array, number of bytes)                     */
/*      char *address;                                                       */
/*      char *command array;                                                 */
/*      int number of bytes;                                                 */
/*                                                                           */
/*                                                                           */
/* Input to the function:                                                    */
/*                                                                           */
/*      Array containing the commands to be sent, and a number               */
/*      indicating the number of bytes of the array to be sent.              */
/*                                                                           */
/*                                                                           */
/* Output from the function:                                                 */
/*                                                                           */
/*      A PASS/FAIL flag is returned to the caller.                          */
/*                                                                           */
/*                                                                           */
/* Functions called:                                                         */
/*                                                                           */
/*      stat_wait  o_fifo  out                                               */
/*                                                                           */
/*****************************************************************************/
```

```
/* I/O mapped version, SNDCMD                                          */

sndcmd (address,cmd_arr,nbr_bytes)
char *address;                  /* base address of the 181X            */
char *cmd_arr;                  /* array containing one 181X command   */
int nbr_bytes;                  /* number of characters in the command */
{

  if (stat_wait (STAT_ADDR,BFR_RDY) != BFR_RDY)
        return (FAIL);          /* indicates timeout without getting   */
                                /* buffer ready signal                 */

  o_fifo (DATA_ADDR,nbr_bytes,cmd_arr); /* put a command into the FIFO */
  out (CTRL_ADDR,COMMAND);              /* tell 181X there is a command */
  return (PASS);
 }
```

```
/* Memory mapped version, SNDCMD                                       */

sndcmd (address,cmd_arr,nbr_bytes)
char *address;                  /* base address of the 181X            */
char *cmd_arr;                  /* array containing one 181X command   */
int nbr_bytes;                  /* number of characters in the command */
{

  if (stat_wait (STAT_ADDR,BFR_RDY) != BFR_RDY)
        return (FAIL);          /* indicates timeout without getting   */
                                /* buffer ready signal                 */

  for (;  nbr_bytes;  nbr_bytes--){     /* put a command into the FIFO */
        *DATA_ADDR = *cmd_arr;
        cmd_arr++;
  }

  *CTRL_ADDR = COMMAND;                 /* tell 181X there is a command */
  return (PASS);
 }
```

```
/*************************************************************************/
/*                                                                       */
/*  RCVCMD - General function for reading the 181X's FIFO.               */
/*                                                                       */
/*   This function will read a variable length message from the 181X's   */
/* FIFO.  Included in this function are handshakes to test that the      */
/* OUTPUT DATA BUFFER has a message ready to be read and to signal to    */
/* the 181X that the OUTPUT DATA BUFFER has been read.                   */
/*                                                                       */
/*                                                                       */
/* Name and Form of the function:                                        */
/*                                                                       */
/*      rcvcmd (address, command array, number of bytes)                 */
/*      char *address;                                                   */
/*      char *command array;                                             */
/*      int number of bytes;                                             */
/*                                                                       */
/*                                                                       */
/* Input to the function:                                                */
/*                                                                       */
/*      Array to receive the commands from the FIFO and a number         */
/*      indicating the number of bytes to be read from the FIFO.         */
/*                                                                       */
/*                                                                       */
/* Output from the function:                                             */
/*                                                                       */
/*      A PASS/FAIL flag is returned to the caller.                      */
/*                                                                       */
/*                                                                       */
/* Functions called:                                                     */
/*                                                                       */
/*      stat_wait  i_fifo  out                                           */
/*                                                                       */
/*************************************************************************/
```

```
/* I/O mapped version, RCVCMD                                          */

rcvcmd (address,cmd_arr,nbr_bytes)
char *address;                  /* base address of the 181X            */
char *cmd_arr;                  /* where we will put the 181X response */
int nbr_bytes;                  /* number of bytes to be received      */
{

  if (stat_wait (STAT_ADDR,MSG_RDY) != MSG_RDY)
        return (FAIL);          /* indicates timeout without getting   */
                                /* message ready signal                */

  i_fifo (DATA_ADDR,nbr_bytes,cmd_arr);        /* read FIFO message.   */
  out (CTRL_ADDR,MSG_RED);              /* tell 181X the message is read*/
  return (PASS);
 }


/* Memory mapped version, RCVCMD                                       */

rcvcmd (address,cmd_arr,nbr_bytes)
char *address;                  /* base address of the 181X            */
char *cmd_arr;                  /* where we will put the 181X response */
int nbr_bytes;                  /* number of bytes to be received      */
{

  if (stat_wait (STAT_ADDR,MSG_RDY) != MSG_RDY)
        return (FAIL);          /* indicates timeout without getting   */
                                /* message ready signal                */

  for (;  nbr_bytes;  nbr_bytes--){     /* read a message from the FIFO */
        *cmd_arr = *DATA_ADDR;
        cmd_arr++;
  }

  *CTRL_ADDR = MSG_RED;         /* tell 181X we read the message       */
  return (PASS);
 }


/*************************************************************************/
/*                                                                       */
/*  STAT_WAIT - Function to wait for a status bit from the 181X.         */
/*                                                                       */
/*   This function will read the specified port until its contents       */
/* match the mask.  If the bit does not get set after a maximum number  */
/* of reads a zero is returned.  If the function is successful the mask  */
/* is returned.                                                          */
/*                                                                       */
```

```
/*                                                                      */
/* Name and Form of the function:                                       */
/*                                                                      */
/*      char stat_wait (address,mask)                                   */
/*      char *address;                                                  */
/*      int mask;                                                       */
/*                                                                      */
/*                                                                      */
/* Input to the function:                                               */
/*                                                                      */
/*      A pointer to the byte to be read and the mask to be compared    */
/*      with.                                                           */
/*                                                                      */
/*                                                                      */
/* Output from the function:                                            */
/*                                                                      */
/*      Zero if the function is unsuccessful and the mask if the        */
/*      function is successful.                                         */
/*                                                                      */
/*                                                                      */
/* Functions called:                                                    */
/*                                                                      */
/*      in                                                              */
/*                                                                      */
/************************************************************************/
```

```
/* I/O mapped version, STAT_WAIT

#define MAX_READS 10000          /* max # of reads before "time out"     */

char stat_wait (address,mask)
char *address;  /* address of the byte we are reading to check status   */
int mask;       /* the mask we will be ANDing reads with                */
{

int i;                           /* just a count variable                */

for (i = 0;  i < MAX_READS;  i++)
        if ((in (address) & mask)  ==  mask)    /* check state of bits  */
                return (mask);                  /* of interest          */

return (0);                      /* timed out, send error message        */
}


/* Memory mapped version, STAT_WAIT                                      */

#undef  MAX_READS                /* define new MAX_READS                 */
#define MAX_READS 32767          /* max # of reads before "time out"     */

char stat_wait (address,mask)
char *address;  /* address of the byte we are reading to check status   */
int mask;       /* the mask we will be ANDing reads with                */
{

int i;                           /* just a count variable                */

for (i = 0;  i < MAX_READS;  i++)
        if ((*address & mask)  ==  mask)        /* check state of bits  */
                return (mask);                  /* of interest          */

return (0);                      /* timed out, send error message        */
}
```

Program #2: FIFO.C

```
/*************************************************************************/
/*                                                                       */
/*  I_FIFO - Receives a character string from the 181X's DATA BUFFER.    */
/*                                                                       */
/*   This function will receive a variable length message from the       */
/* 181X's INPUT DATA BUFFER.                                             */
/*                                                                       */
/* Name and Form of the function:                                        */
/*                                                                       */
/*      i_fifo (address, number of bytes, command array)                 */
/*      char *address;                                                   */
/*      int number of bytes;                                             */
/*      char *command array;                                             */
/*                                                                       */
/* Input to the function:                                                */
/*                                                                       */
/*      Address of array where messages will be stored and a number      */
/*      indicating the number of bytes to be read.                       */
/*                                                                       */
/* Output from the function:                                             */
/*                                                                       */
/*      none                                                             */
/*                                                                       */
/* Functions called:                                                     */
/*                                                                       */
/*      in                                                               */
/*                                                                       */
/*************************************************************************/

/* I/O mapped version, I_FIFO                                            */

i_fifo (address,nbr_bytes,cmd_arr)
char *address;                  /* address of the 181X's DATA BUFFER     */
int nbr_bytes;                  /* number of characters to be read       */
char *cmd_arr;                  /* array to be filled                    */
{

  while (n != 0){
        *cmd_arr = in (address);        /* read the DATA BUFFER          */
        cmd_arr++;                      /* point to next slot in array   */
        n--;                            /* decrement # of reads          */
  }
 }
```

```
/* Memory mapped version, I_FIFO                                    */

i_fifo (address,nbr_bytes,cmd_arr)
char *address;                  /* address of the 181X's DATA BUFFER    */
int nbr_bytes;                  /* number of characters to be read      */
char *cmd_arr;                  /* array to be filled                   */
{

  while (n != 0){
        *cmd_arr = *address;            /* read the DATA BUFFER         */
        cmd_arr++;                      /* point to next slot in array  */
        n--;                            /* decrement # of reads         */
  }
 }
```

```
/*************************************************************************/
/*                                                                       */
/*  O_FIFO - Sends a character string to the 181X's DATA BUFFER.         */
/*                                                                       */
/*   This function will send a variable length message to the 181X's     */
/* INPUT DATA BUFFER.                                                    */
/*                                                                       */
/* Name and Form of the function:                                        */
/*                                                                       */
/*      o_fifo (address, number of bytes, command array)                 */
/*      char *address;                                                   */
/*      int number of bytes;                                             */
/*      char *command array;                                             */
/*                                                                       */
/*                                                                       */
/* Input to the function:                                                */
/*                                                                       */
/*      Array containing the bytes to be sent and a number               */
/*      indicating the number of bytes of the array to be sent.          */
/*                                                                       */
/*                                                                       */
/* Output from the function:                                             */
/*                                                                       */
/*      none                                                             */
/*                                                                       */
/*                                                                       */
/* Functions called:                                                     */
/*                                                                       */
/*      out                                                              */
/*                                                                       */
/*************************************************************************/

/* I/O mapped version, O_FIFO                                            */

o_fifo (address,nbr_bytes,cmd_arr)
char *address;                  /* address of the 181X's DATA BUFFER     */
int nbr_bytes;                  /* number of characters to be written    */
char *cmd_arr;                  /* array to be written                   */
{

  while (n != 0){
        out (address,*cmd_arr);         /* write to the DATA BUFFER      */
        cmd_arr++;                      /* point to next slot in array   */
        n--;                            /* decrement # of writes         */
  }
 }
```

```
/* Memory mapped version, O_FIFO                                    */

o_fifo (address,nbr_bytes,cmd_arr)
char *address;                  /* address of the 181X's DATA BUFFER    */
int nbr_bytes;                  /* number of characters to be written   */
char *cmd_arr;                  /* array to be written                  */
{

  while (n != 0){
        *address = *cmd_arr;            /* write to the DATA BUFFER     */
        cmd_arr++;                      /* point to next slot in array  */
        n--;                            /* decrement # of writes        */
  }
 }
```

ADDENDUM TO 1810/11 INSTRUCTION MANUAL FOR POSITION TRACKING

INTRODUCTION

The position tracking option for the 1810 Motor Controller (making it the 1811) is a hardware modification to the 1810 which allows it to monitor an incremental quadrature-output encoder for the purpose of maintaining position and/or motor stall detection.

DESCRIPTION

The 1811 may be differentiated from the 1810 primarily by its ENCODER connector. The software used in the 1810 is identical with that in the 1811. You will note, however, that the 1811 has a few more parts than the 1810. Primarily, U16, U19, U20, U21 and U54 will be found only on the 1811.

To identify which revision level pertains to your 1811, look at the right hand side of the component side of the board (with the MULTIBUS connector down). A number is etched there which specifies the Compumotor part number for the board (72-2155-???). The revision level will be inscribed with an indellible marker (-04 as of this writing).

The pin connections for the ENCODER connector may be found in Appendix C. A description of each pin may be found in the Specifications section.

CHANGES

The commands described below are current as of June, 1984. The software revision level as of this writing is 92-3657-01A/11A. Several commands have been added with this revision of software, namely 0F, 96 and B5.

The only software revision prior to this one changed the software from 92-2514-01A/11A to 92-2514-01B/11B and added the 2F command.

Please refer to the open-loop command description for changes in that software.

NOTE: All commands and parameter bytes are hexadecimal numbers.

COMMAND LIST

The following is a list of the special commands created for software part number 94-002514-01. Each of the commands is a single binary byte. Some commands require additional bytes, called parameter bytes, in order to completely specify the command. The following list of commands gives:

1.) the command number (in hex),
2.) the number of parameter bytes required (in hex) and,
3.) the command's "name."

0F	3	Define bit "X" to indicate state "Y"
18	1	Turn off/on post-move position maintenance
19	1	Turn off/on move termination on stall detect
35	4	Go to relative encoder position "X"
36	4	Go to absolute encoder position "X"
3A	1	Go to encoder HOME at the default velocity and acceleration
2F	4	Define absolute closed-loop position
96	0	Request the state of the Channel Z home input
98	0	Request relative encoder count
99	0	Request relative error from desired closed-loop position
9A	0	Request absolute encoder count
9B	0	Request slip detect status
9C	0	Request motor pulse to encoder pulse ratio
9D	0	Request motor resolution
9E	0	Request backlash sigma (motor steps)
9F	0	Request position maintenance alg., const., and max. velocity
A6	1	Interrupt on next stall detect
A7	1	Interrupt on every stall detect
B0	4	Define motor pulse to encoder pulse ratio
B1	4	Define motor resolution
B2	4	Define backlash sigma (motor steps)
B3	7	Define position maintenance alg., const., and max. velocity
B4	4	Define the number of rotor teeth
B5	4	Define the deadband region in encoder pulses
D4	E	Define move "X," define it as a relative, closed-loop move
D5	E	Define move "X," define it as an absolute, closed-loop move

COMMAND DESCRIPTION

NOTE: For a complete description of the command syntax and command attributes refer to the Programming section, parts E and F.

OF 3 Define bit "X" to indicate state "Y"

00 -> 04 indicates which bit will be used to indicate the desired state. Values greater than 4 are not allowed and will cause the command to be ignored. Parameter byte values are defined as follows,
- 00 corresponds to user definable bit 0,
- 01 corresponds to user definable bit 1,
- 02 corresponds to user definable bit 2,
- 03 corresponds to programmable output bit 1,
- 04 corresponds to programmable output bit 2.

00 -> FF indicates which indexer state will be displayed by the chosen definable bit (chosen by parameter byte 1 above). The code numbers indicating the desired indexer state are,
- 00 indicates the null code,
- 01 indicates the moving state,
- 02 indicates the performing-a-singular-command state,
- 03 indicates the performing-the sequence-buffer state,
- 04 indicates performing-the-velocity-streaming-buffer state,
- 05 indicates the accelerating state,
- 06 indicates the decelerating state, and
- 07 indicates the non-zero constant velocity state.
- 80 indicates the motor is within the user-definable closed-loop deadband,
- 81 indicates the state of the joystick request input.

Codes between 00 and 7F are very accurate state indicators. Code numbers between 80 and FF are less accurate state indicators.

zero or non-zero
indicates the active level of the bit. Non-zero means active high, 00 means active low.

This command is used to indicate that a given state is occurring within the 1810 Indexer. For instance, you can

indicate the state of moving/not-moving through user-definable bit number one by sending the command sequence 0F 01 01 FF. This sequence means: "When the motor is moving, set user-definable bit number one, and when the motor is not moving, clear user-definable bit number one." More than one programmable bit may indicate the same state at any time, e.g., you could have the indexer display the state of the Joystick request input by both a programmable output bit and a user-definable bit.

It is expected that the 0F command will be sent once only during initialization. It is an extremely powerful command. It can be used in place of requesting information and reading the response to a request. For instance, the 8C command is commonly used to determine whether the motor is moving or not. Using the 8C command involves waiting for the input buffer to be ready to receive a command, sending the 8C command, indicating the command has been sent, waiting for the message to return, reading the message, and indicating that you have read the message. There are quite a few steps involved here and they usually take about 5+ milliseconds to accomplish.

Using the 0F command, you can simply read the status byte, look at the previously defined bit, and know at that instant whether or not the motor is moving. In essence, it gives you a much more accurate view of the state of the machine. It also involves a much simpler and faster process for determining that state.

There is a way to "undo" previously programmed codes. You can do this by using the null code. For instance, if user bit 0 is being used to indicate the moving/not moving state, and you later want it not to indicate any state, i.e., remain constant, send a 0F 00 00 00.

This sequence says, "define user bit 0 to indicate the null code state, and leave the bit in its previous state." The last parameter byte has no effect. The first parameter byte indicates which bit to be defined, in this case, user definable bit 0. The second parameter byte, 00, means we wish to delete that bity from indicating any state. It is not expected that the null code will be used very often.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-within-a-SEQUENCE-buffer
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

18 1 Turn off/on post-move position maintenance

00 or nonzero, 00 = "off" and nonzero = "on"

Turns on or off the post-move position maintenance function according to the parameter byte passed. A nonzero parameter byte turns post-move position maintenance on and a 00 parameter byte will turn the post-move position maintenance feature off. After power up or reset this function is off.

Position maintenance is the primary closed-loop function. When position maintenance is enabled, and the indexer is not performing a move, the indexer attempts to maintain the "desired encoder position." After power up or reset the desired encoder position is that position the encoder was at when power up or reset occured (and is defined as zero). From then on the desired encoder position is determined by the sum of closed-loop moves that have occured, or, if this function is turned off, the encoder position the motor was at after completion of the most recent open-loop move. For example, if, following reset or zeroing of the absolute position (see the 30h command), one performed a closed-loop move of distance +30 relative encoder pulses and a second closed-loop move of distance -10 relative encoder pulses the desired encoder position would be the absolute encoder position +20. If one performed an absolute closed-loop move to absolute position +55 the desired closed-loop position after the move was completed would be the absolute encoder position +55. Now, if one was to turn the position maintenance function off, perform an open-loop move, and then turn the position maintenance function back on, the desired encoder position would be that position the encoder was at when the open-loop move was completed.

It should be noted that an open-loop incremental or absolute move cannot be performed when the position maintenance function is turned on. Therefore, in order to perform an open-loop move (this excludes the open-loop go HOME move and continuous moves) one must first turn the position maintenance function off and then perform the open-loop move.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

19 1 Turn off/on move termination on stall detect

00 or nonzero, 00 = "off" and nonzero = "on"

Turns on or off the move termination on stall detect function according to the parameter byte passed. A nonzero parameter byte turns move termination on and a 00 parameter byte will turn the move termination feature off. After power up or reset this function is off.

After the motor to encoder resolution, the motor resolution, the backlash sigma, and the number of rotor teeth have been defined the indexer will watch for stall detect on ALL moves (open-loop and closed-loop). If this function is on when a stall is realized the motor velocity will be zeroed, which results in prematurely ending the current move. If the position maintenance function was turned on when the stall is realized (see command 18h) the encoder position at the time of the stall becomes the new desired encoder position. If this function is turned off when a stall occurs the motor will continue to stall, until the move has completed or is stopped by the host!

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

2F 4 Define absolute closed-loop position

00 -> FF,	most significant byte of the absolute closed-loop position
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the relative of the absolute closed-loop position

This command is used to define the current position as the absolute closed-loop position X, where X is specified with the parameter bytes. X Must be within the absolute closed-loop positioning range of the indexer.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode

VALID-in-VELOCITY-STREAMING-mode

35 4 Go to relative encoder position "X"

00 -> FF,	most significant byte of the relative encoder position
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the relative encoder position

This command tells the indexer to move "X" encoder pulses at the default velocity and acceleration (defined with the 31h command). The relative encoder distance specified is interpreted as a signed, two's complement number. A relative closed-loop move may be performed for any distance in the signed range "X" * (motor pulse to encoder pulse ratio) = 80000001h through "X" * (motor pulse to encoder pulse ratio) = 7FFFFFFFh (-7FFFFFFFh through +7FFFFFFFh or -(2^31)-1 through +(2^31)-1). A distance of "X" * (motor pulse to encoder pulse ratio) = 80000000h is invalid and will be ignored. In other words, the absolute value of "X" must be less than or equal to 7FFFFFFFh divided by the motor pulse to encoder pulse ratio. If the default move has not been defined (with command 31h) and the B0h through B4h commands have not been defined no move can be performed.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

36 4 Go to absolute encoder position "X"

00 -> FF,	most significant byte of the absolute encoder position
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the absolute encoder position

This command tells the indexer to move to absolute encoder position "X" ("X" pulses away from the absolute zero position). The absolute encoder distance specified as a part of this command is interpreted as a signed, two's complement number. Absolute

positions may only be specified such that "X" minus the current absolute encoder position does not exceed the values given above, in the 35h command. In other words, the relative distance traveled by the motor must be less than or equal to 7FFFFFFFh divided by the motor pulse to encoder pulse ratio. The indexer will move the motor at the default velocity and acceleration (defined with the 31h command). If the default velocity and acceleration have not been defined and commands B0h through B4h have not been defined no move cam be performed.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

3A 1 Go to encoder HOME at the default
velocity & acceleration

00 or nonzero, direction in which search is to begin,
00 = + and nonzero = -

This command will cause the indexer to search for the channel Z HOME position at the velocity and acceleration specified with command 31h. If the channel Z HOME position is not found after striking both the CW limit switch and the CCW limit switch the search is abandoned.

The parameter byte specifies the direction the indexer will begin moving the motor as it begins its search for the HOME input (as defined with command 38h), i.e. whether to begin searching for the HOME position in the + direction or in the -direction. This command will not be performed if the 38h command has not been performed (the HOME position has not been defined), if the default move parameters have not been defined, if the motor is moving, or if the joystick is enabled.

The channel Z home position is defined via the CH Z+ and the CH Z-inputs. These inputs are typically found on incremental encoders and are used to indicate a known reference point. For rotary encoders it is necessary to supply an additional signal which is referred to as the HOME ENABLE input. Activating the HOME ENABLE input validates the channel Z home position (which occurs once during each revolution of a rotary encoder). Therefore, the indexer will search for the channel Z home position that occurs when HOME ENABLE is active.

One must issue the 38h command before any "go HOME" moves can be performed. The 38h command is typically performed only

once after power up or system initialization. It allows the user to define the precise home position with respect to a HOME input signal of finite width and fixed position, regardless of the initial approach direction. It also defines the velocity and direction at which the final approach to the HOME position is performed. Additional information regarding the closed-loop "go HOME" command is provided on the following pages.

SINGULAR
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode

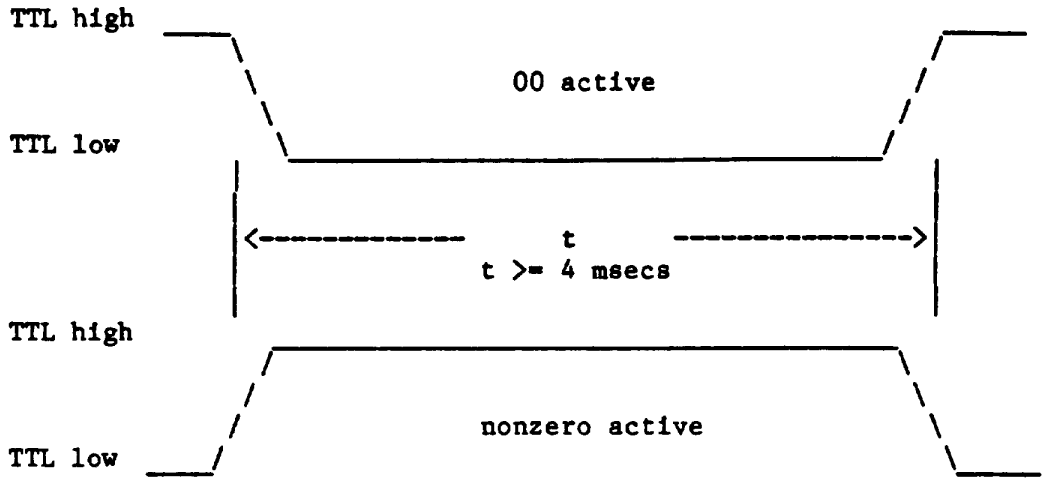
NOTES ON THE CLOSED-LOOP GO HOME COMMAND

The 38h command is used to define the HOME position for both open-loop and closed-loop "go HOMES." When used to define a closed-loop "go HOME" its parameter bytes take on slightly different meanings. Following is a description of the 38h command as it pertains to the closed-loop "go HOME" move. (Note: if position maintenance is turned off a closed-loop "go HOME" cannot be performed. An open-loop "go HOME" can be performed regardless of the state of position maintenance.)

38 4 Define HOME location and final search parameters (for c-l)

00 or nonzero,	active state of HOME ENABLE input, 00 = TTL low
XX	don't care, (not ignored by open-loop go "HOME")
01 -> 64	final approach velocity as % of default velocity
00 or nonzero,	final approach direction, 00 = CW and nonzero = CCW

The first parameter byte supplied with the 38h command defines the active state of the HOME ENABLE input. A 00 parameter byte means the HOME ENABLE input is active low (TTL 0) and a nonzero parameter byte means the HOME ENABLE input is active high (TTL 1). The HOME ENABLE input, like TRIGGERs 2 through 5, is a level sensitive input and must be active for a minumum of four milliseconds at the default velocity, (defined with the 31h command), in order for the indexer to "see" HOME properly. Following is an illustration of an active HOME ENABLE input.



The second parameter byte is ignored when performing a closed-loop "go HOME." It still has meaning if one attempts to perform an open-loop "go HOME" move! The edge of HOME to be chosen as the HOME position by the indexer is a function of the final approach direction (parameter byte number four) and a jumper on the 1811 board.

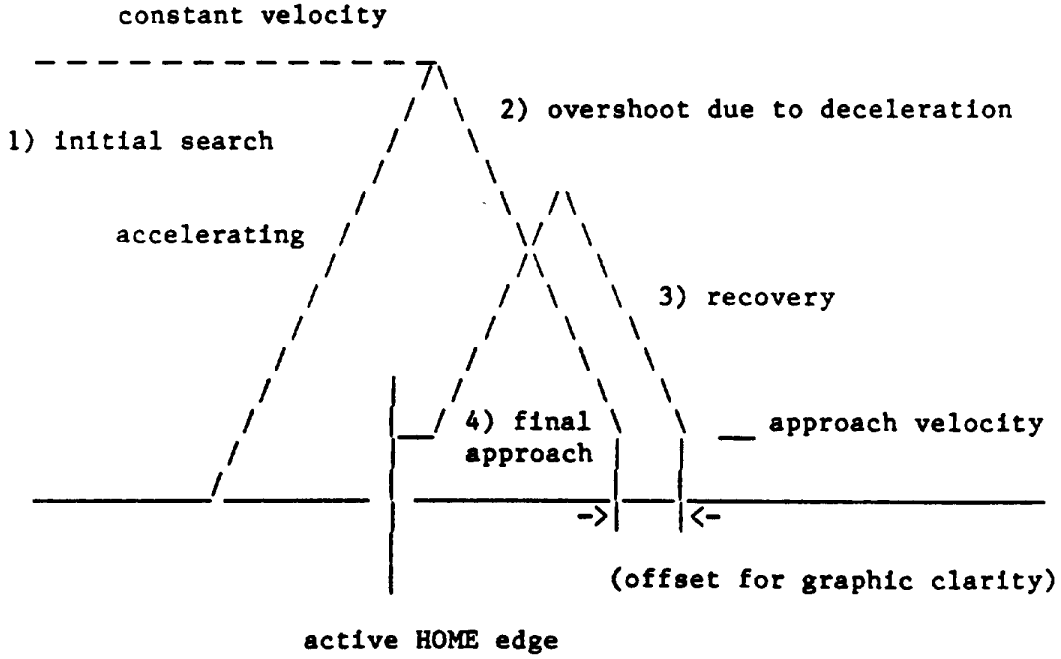
The edge approach velocity, specified by the third parameter byte, is the velocity at which the final search for the HOME position is performed. There is no acceleration to nor deceleration from this velocity, the motor must be able to step up to and down from this velocity. The repeatability of the HOME position is a function of the magnitude of the final approach velocity. The lower the final approach velocity the more repeatable, and more accurate, will be the HOME position. The final approach velocity is determined as a function of the default velocity (defined with command 31h). The third parameter byte supplied with the 38h command expresses the final approach velocity to be a percent of the default velocity. Percents greater than 100 or equal to 0 are invalid and will cause this command to be ignored. To determine the actual final approach velocity use the following formula:

$$\text{approach velocity} = (\text{default velocity}) * \frac{\text{third parameter byte}}{100}$$

The fourth, and final, parameter byte supplied with this command specifies the final approach direction for finding the HOME position. This means that the HOME position will always be

approached from the same absolute direction. If this parameter byte is a 00 the final approach direction is in the CW direction, towards the CW limit. If the parameter byte is nonzero the final approach direction is in the CCW direction, towards the CCW limit.

The "go HOME" move is designed to be rapid and repeatable. The initial search direction is specified with the go HOME command (3Ah), but if an end of limit is encountered the motor will reverse and try searching in the other direction. If the opposite end of travel limit is hit the search for HOME is abandoned. The following diagram illustrates what happens after the specified HOME edge is found once. Item number one indicates the initial search for the HOME input. When the active edge of interest is found (either at the default velocity or during acceleration to the default velocity) the indexer will begin to decelerate the motor; this begins item number two, the overshoot due to deceleration. The indexer notes the distance it has overshot and will perform a recovery move, item number three, to get back to the approximate location of the HOME position. The final approach to the HOME position will be performed at the specified percent of default velocity (parameter byte number three) in the specified absolute direction (parameter byte number four). The diagram below does not illustrate the motions the indexer will send the motor through to acheive the final approach in the specified absolute direction.



96 0 Request the state of the Channel Z home input

This command is used to determine the state of an encoder home bit input. There are two home inputs to the 1810 Indexer, the home bit on the auxiliary connector and the home bit on the encoder connector. They are not the same home bit. One is the open-loop home bit; the other is the closed-loop home bit. This command refers to the closed-loop home bit.

To determine the state of the open-loop home bit, use the 8A command. When using the closed-loop home bit as a home bit, the open-loop home bit becomes the home enable bit. This is for use with rotary encoders. Rotary encoders have a home bit that occurs once every revolution. The home enable bit is used to determine which revolution contains the true home. Therefore, two bits are required for a closed-loop home--the home enable bit and the closed-loop home bit.

A response looks like, 96 00 or FF, followed by bytes 3-16 which are always zero and are always present. 00 indicates the input is low, FF indicates the input is high.

This command should be used only on 1811 indexers. The command is active on 1810 Indexers, but there is no hardware on the board supporting the Z-channel input.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

98 0 Request relative encoder count

This command returns the relative encoder count. If the indexer is performing a move the relative encoder count is the number of encoder pulses that have been seen by the indexer since the most recent zero velocity position. If the indexer is not performing a move the relative encoder count is the number of encoder pulses the indexer saw during the last move (zero velocity to nonzero velocity and back to zero velocity). The position returned is expressed in two's complement form, most significant byte first and least significant byte last. The sign of the distance reflects the direction of the relative move.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a response to this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

98	identifies the command sending the message
00 -> FF	MS byte of the relative encoder count
00 -> FF	second MS byte
00 -> FF	third MS byte
00 -> FF	LS byte of the relative encoder count
00	bytes six through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

99 0 Request relative error from desired closed-loop position

This command returns the difference between the desired absolute closed-loop position and the actual absolute closed-loop position. The error returned is expressed in two's complement form, most significant byte first and least significant byte last. The sign of the error reflects the direction the motor must move to make up the error. The error is expressed in units of encoder pulses as seen by the indexer (see command B0h). The error is calculated using the following formula:

$$error = A - B$$

where:

A = desired absolute closed-loop position
B = actual absolute closed-loop position

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

```
99              identifies the command sending the message
00 -> FF        MS byte of the relative error
00 -> FF        second MS byte
00 -> FF        third MS byte
00 -> FF        LS byte of the relative error

00              bytes six
 :              through sixteen are always zero
00              and are always present
```

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

9A 0 Request absolute encoder count

This command returns the absolute encoder count. The position returned is expressed in two's complement form, most significant byte first and least significant byte last. The sign of the position reflects the side of absolute zero the motor is residing on. The absolute encoder count is reset to zero when one redefines absolute zero with the 30h command. The absolute position counter is a 32 bit up/down binary counter, thus, the maximum absolute encoder count (maximum number of encoder pulses) the indexer can track is 80000000h through 7FFFFFFFh (-2,147,483,648 through 2,147,483,647). The number of encoder pulses is not equal to the number of encoder lines, the closed-loop circuitry performs a "times 4" function on the incoming encoder line pulses. Thus, the number of encoder pulses is equal to four times the number of encoder lines.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

```
9A              identifies the command sending the message
00 -> FF        MS byte of the absolute encoder count
00 -> FF        second MS byte
00 -> FF        third MS byte
00 -> FF        LS byte of the absolute encoder count

00              bytes six
```

```
 :              through sixteen are always zero
00              and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING
```

9B 0 Request slip detect status

This command returns the slip detect status word. The slip detect status word is the number of motor rotor teeth (+1/-0 if the number is nonzero and +/-0 if the number is zero) that have slipped with respect to the stator teeth. If no slip has occured this number will be zero. The slip status word can only be calculated if the motor to encoder ratio, the motor resolution, and the number of rotor teeth have been defined. If these quantities have not been defined a zero will always be returned, even if slip has occured. Slip is defined as follows:

```
if (abs_value_of (A - B*C) > sigma)
     slip status word  =  (abs_value_of (A - B*C) - sigma) / (D/E);

else
     slip status word  =  0;
```

where:

```
A  =  current absolute motor pulse count
B  =  current absolute encoder count
C  =  motor pulse to encoder pulse ratio
sigma  =  system backlash (motor steps)
D  =  motor resolution (# of motor steps per revolution)
E  =  number of rotor teeth
```

The motor to encoder ratio is defined with the B0h command. The motor resolution is defined with the B1h command. The number of rotor teeth is defined with the B4h command. The slip status word is the number of rotor teeth that the motor has "slipped" with respect to the stator teeth. In other words, if a slip does occur it means the motor has lost sychronism with the input pulses being sent by the indexer. It is recommended that the slip detect status word only be requested when the motor is not moving. To set this quantity back to zero, if a slip has occurred, one must issue the 30h command, which results in

zeroing both the open-loop absolute position and the closed-loop absolute position.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

9B	identifies the command sending the message
00	always
00	always
00 -> FF	MS byte of the slip status word
00 -> FF	LS byte of the slip status word
00	bytes six
:	through sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

9C 0 Request motor pulse to encoder pulse ratio

This command returns the motor pulse to encoder pulse ratio. This ratio is used to convert encoder pulses counted by the 1800 indexer to motor pulses when calculating various quantities (see commands 19h and 9Bh). The second and third byte returned are the numerator of this ratio and the fourth and fifth byte are the denominator. This ratio is defined by the host with the "define motor pulse to encoder pulse ratio" command (B0h). The numbers returned are interpreted as ABSOLUTE numbers, not as signed numbers.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

9C	identifies the command sending the message
00 -> FF	MS byte of the ratio's numerator
00 -> FF	LS byte of the ratio's numerator
00 -> FF	MS byte of the ratio's denominator
00 -> FF	LS byte of the ratio's denominator

```
00              bytes six
 :              through sixteen are always zero
00              and are always present
```

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

9D 0 Request motor resolution

This command returns the defined motor resolution. The motor resolution is required for a number of calculations, including stall detect and slip detect. If this quantity is not supplied those functions that require this number will not be operable. This quantity is defined with the B1h command. The motor resolution is defined as the number of pulses that constitute one complete revolution of the motor. Note that the number returned is always a positive number.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

```
9D              identifies the command sending the message
00              always
00              always
00 -> FF        MS byte of the motor resolution
00 -> FF        LS byte of the motor resolution

00              bytes six
 :              through sixteen are always zero
00              and are always present
```

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

9E 0 Request backlash sigma (motor steps)

The backlash sigma number is the number of motor steps required to traverse any deadband in the mechanical system between the motor's shaft and the encoder. Typically, the deadband is due to backlash in a geartrain being driven by the motor. If the encoder is mounted directly to the motor's shaft there should be no backlash or deadband (between the motor and the encoder). This number is named sigma because it is not expected to be very large. Too large a deadband can result in an unstable closed-loop system.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed. Note that the backlash sigma is defined in units of motor pulses, not encoder pulses, and is the entire width of the deadband region.

```
9E              identifies the command sending the message
00 -> 7F        MS byte of the backlash sigma
00 -> FF        second MS byte
00 -> FF        third MS byte
00 -> FF        LS byte of the backlash sigma

00              bytes six
 :              through sixteen are always zero
00              and are always present


VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING
```

9F 0 Request position maintenance alg., const. & max. velocity

Position maintenance is performed only when enabled and only when a move is not being performed. The position maintenance algorithm is very simple and consists of creating a correction velocity based on the positional error (the difference between the desired absolute encoder position and the actual absolute encoder position). Three velocity generation functions are

available and are chosen with parameter byte number one. Those three functions are:

parameter byte # one	correction algorithm
01	correction velocity = K * error
02	correction velocity = K * (error)^(1/2)
03	correction velocity = K * (error)^2

In the first algorithm, 01, the correction velocity is K times the positional error. For algorithm 02 the correction velocity is equal to K times the square root of the positional error. And in algorithm 03 the correction velocity is equal to K times the square of the positional error.

K is the correction gain constant and is specified with the second parameter byte. It may be any number between 0 and FFh (0 -255). Obviously, if 0 is chosen no correction will take place!

The last five parameter bytes specify the maximum correction velocity allowed. The method of choosing the maximum correction velocity is similar to that used to choose move velocities. That is, a velocity range number must be supplied with the four byte velocity number. The velocity range number corresponds to the velocity range numbers one uses when defining a move.

The following list illustrates the message one will receive in the OUTPUT DATA BUFFER as a result of this command. A "message ready interrupt" will be generated if that interrupt is enabled and the appropriate jumpers have been installed.

9F	identifies the command sending the message
01 -> 03	position maintenance algorithm
00 -> FF	correction gain constant (K)
0X -> 5X	velocity range number (see C8 command)
00 -> 7F	MS byte of the maximum correction velocity
00 -> FF	second MS byte
00 -> FF	third MS byte
00 -> FF	LS byte of the maximum correction velocity
00	bytes nine through
:	sixteen are always zero
00	and are always present

VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

A6 1 Interrupt on next stall detect

00 or nonzero, 00 = disable the interrupt,
nonzero = enable the interrupt

Interrupt the host bus when the next stall detect occurs. The interrupt occurs only once, when the next stall detect occurs. Subsequent stall detects will not cause an interrupt. The interrupt one will receive from the 1800 indexer is a "message ready" interrupt, the message in the OUTPUT DATA BUFFER is this command (A6h) followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

A7 1 Interrupt on every stall detect

00 or nonzero, 00 = disable the interrupt,
nonzero = enable the interrupt

Same as A6h except an interrupt will be issued every time a stall detect occurs. The message in the OUTPUT DATA BUFFER will be an A7h followed by fifteen zeroes. After power on or reset this interrupt is disabled.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B0 4 Define motor pulse to encoder pulse ratio

00 -> FF, MS byte of the numerator
00 -> FF, LS byte of the numerator
00 -> FF, MS byte of the denominator
00 -> FF, LS byte of the denominator

This command defines the motor pulse to encoder pulse ratio. This ratio is used for a variety of calculations (see commands 19h and 9Bh). The second and third parameter bytes are the numerator of this ratio and the fourth and fifth parameter bytes are the denominator. The numbers in the ratio are interpreted as absolute integers, not as two's complement numbers, (the ratio is a rational number). The 1800 closed-loop circuitry performs a "times 4" function on the incoming encoder pulses. Therefore, one incoming encoder period equals four encoder periods as counted by the 1800 indexer. To determine the ratio do the following:

1) x = number of motor pulses per revolution of the motor
2) y = number of encoder lines per revolution of the motor
3) z = y * 4 (number of encoder pulses per motor revolution)
4) ratio = x / z

The numbers one must send with this command are "x" and "z." These two numbers are interpreted as positive sixteen bit numbers. If the numbers one determines for the above ratio do not fit into the two byte numerator and denominator format it may be necessary to simplify the ratio. For example, if one has a ratio of 70,000 / 10,000 they cannot define the ratio using this command because 70,000 does not fit into sixteen bits. However, 7 / 1 does fit, and will work just as well!

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B1 4 Define motor resolution

```
00,       always
00,       always
00 -> FF, MS byte of the motor resolution
00 -> FF, LS byte of the motor resolution
```

This command defines the motor resolution. The motor resolution is used for a variety of calculations (see commands 19h and 9Bh). The motor resolution is defined as the number of pulses that constitute one complete revolution of the motor (in a standard Compumotor this would be 25,000 steps). Note that the motor resolution is always defined as a positive number.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B2 4 Define backlash sigma (motor steps)

```
00 -> 7F, MS byte of the backlash sigma
00 -> FF, second MS byte
00 -> FF, third MS byte
00 -> FF, LS byte of the backlash sigma
```

The backlash sigma number is the number of motor steps required to traverse any deadband in the mechanical system between the motor's shaft and the encoder. Typically the deadband is due to backlash in a geartrain being driven by the motor. If the encoder is mounted directly to the motor's shaft there should be no backlash or deadband (between the motor and the encoder). This number is named sigma because it is not expected to be very large. Too large a deadband can result in an unstable closed-loop system.

Note that the backlash sigma is defined in units of motor pulses, not encoder pulses, and is the entire width of the deadband region.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B3 7 Define position maintenance alg., const. & max. velocity

01 -> 03, choose position maintenance algorithm
00 -> FF, correction gain constant (K)
0X -> 5X, velocity range number
00 -> 7F, MS byte of the maximum correction velocity
00 -> FF, second MS byte
00 -> FF, third MS byte
00 -> FF, LS byte of the maximum correction velocity

Position maintenance is performed only when enabled and only when a move is not being performed. The position maintenance algorithm is very simple and consists of creating a correction velocity based on the positional error (the difference between the desired absolute encoder position and the actual absolute encoder position). Three velocity generation functions are available and are chosen with parameter byte number one. Those three functions are:

parameter byte # one	correction function
01	correction velocity = K * error
02	correction velocity = K * (error)^(1/2)
03	correction velocity = K * (error)^2

In the first algorithm, 01, the correction velocity is K times the positional error. For algorithm 02 the correction velocity is equal to K times the square root of the positional error. And in algorithm 03 the correction velocity is equal to K times the square of the positional error.

K is the correction gain constant and is specified with the second parameter byte. It may be any number between 0 and FFh (0 -255). Obviously, if 0 is chosen no correction will take place! The first two correction algorithms work the best. Always start off with a small value for K and work up to the optimum value for K. The correct algorithm and K value are very strong functions of the your mechanical system.

The last five parameter bytes specify the maximum

correction velocity allowed. The maximum correction velocity is specified in units as one specifies a move velocity. The velocity range number used here is exactly the same as the velocity range numbers used in move definitions. (See the D4h command below).

No position maintenance will be performed until all of these parameters have been supplied. These parameters may be changed at any time, it is not necessary to turn the position maintenance function off when changing them.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B4 4 Define the number of rotor teeth

00, always
00, always
00 -> FF, MS byte of the number of rotor teeth
00 -> FF, LS byte of the number of rotor teeth

This command defines the number of rotor teeth on the motor being driven. The number is required to calculate the slip detect status word. Most hybrid permanent magnet step motors have 50 rotor teeth. (All Compumotors at the time of this writing have 50 rotor teeth). The number of rotor teeth determines the distance over which a slip will occur.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-in-VELOCITY-STREAMING-mode
VALID-concurrent-with-VELOCITY-STREAMING

B5 4 Define the deadband region in encoder pulses

00 -> FF ignored
00 -> FF ignored
00 -> FF most significant byte of deadband size
00 -> FF least significant byte of deadband size

This command is required for use with the code 80 option of the 0F command. Code 80 of the 0F command is used to indicate

whether the motor is within the user defined closed-loop deadband. The B5 command is used to define that desired closed-loop deadband. It can only be issued to an 1811. This command will be ignored by an 1810.

The four parameter bytes are the deadband region to be defined. The first two parameter bytes are unused and ignored. The last two parameter bytes define the deadband region. The number defined is the absolute value of the deadband region on each side of the desired position. The number is not a two's complement number.

The reason for this command, and the 80 code of the 0F command, is to allow a way of knowing when an 1811--which is performing position maintenance--has achieved the desired closed loop position, or is within a certain distance of the desired closed-loop position

For example, when performing a closed-loop move, the motor may not acheive the desired position on the first try. The motor may overshoot or undershoot. The 1811 will recover that overshoot by moving the motor to the desired closed-loop position.

This command, in conjunction with the 0F command, gives a means for determining whether or not the motor has acheived its programmed position. To find out if the motor is positioned to within one encoder pulse, the deadband should be set to be zero. To know only when the motor happens to be within 5 encoder pulses of the desired position, define the deadband to be 5. In this case, whenever the encoder is within 5 encoder pulses of the desired position, the 0F command--along with the 80 code--would cause the programmed user defined status bit to indicate active. If it were outside of 5 encoder pulses from the desired position, the same user defined status bit would indicate inactive.

This number has no effect on the position maintenance algorithm--the 1811 always attempts to maintain a zero deadband.

VALID-in-INDEXER-mode
VALID-in-VELOCITY-STREAMING-mode
VALID-within-a-SEQUENCE-buffer
VALID-concurrent-with-SEQUENCE-buffer-execution
VALID-concurrent-with-VELOCITY-STREAMING

D4 E define move "X," define it as a relative, closed-loop move

01 -> FF,	move number
XY	X = velocity range and Y = acceleration range (see below)
00 -> 7F,	most significant byte of the velocity
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the velocity
00 -> 7F,	most significant byte of the acceleration
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the acceleration
00 -> FF,	most significant byte of the relative distance (enc. pulses)
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the relative distance

This command will define move "X" as a relative, closed-loop move to be performed by the "perform move number 'X'" command (40h). The first parameter byte specifies the move number, a move number of 00h is illegal and will be ignored. The most significant nibble of the second parameter byte denotes the velocity range of the move and the least significant nibble denotes the acceleration range. The velocity range determines the units of the velocity number sent, the maximum velocity achievable, and the velocity resolution. The acceleration range determines the units of the acceleration number with respect to the velocity number's units. The last twelve bytes specify the velocity, acceleration, and distance (in encoder pulses); all three numbers are interpreted in two's complement form. The velocity and acceleration numbers must be specified as positive numbers. The distance specified is a relative encoder distance! The relative distance specified must be in the range (relative encoder distance) * (motor pulse to encoder pulse ratio) = 80000001h through (relative encoder distance) * (motor pulse to encoder pulse ratio) = 7FFFFFFFh. In other words, the absolute value of the relative encoder position specified must be less than or equal to 7FFFFFFFh divided by the motor pulse to encoder pulse ratio. All of the bytes must be present in order for a relative move to be defined. Redefinition of a move is allowed by reissuing a complete define a move command.

It should be noted that this move may only be performed after the motor pulse to encoder pulse ratio has been defined (see the B0h command). Without this ratio no moves may be performed. The distance specified in this move definition is in units of encoder pulses as seen by the indexer; that is, the distance is specified in terms of encoder lines times four (see

the B0h command) not in terms of motor pulses.

Only one move number "X" may be defined at any time; for example, one can only define one move number 9, whether move 9 is a relative move, an absolute move, or a continuous move is of no relevance, there may be only one move 9.

There are six velocity ranges defined with three acceleration ranges per velocity range. If one fails to specify a defined range the move definition is invalid. The velocity and acceleration ranges are as follows.

0 velocity number is pulses/second (pps)
maximum velocity is 546,133.3 Hertz
velocity resolution is 16.667 Hertz

1 velocity number is pulses/ten seconds (0.1 pps)
maximum velocity is 54,613.33 Hertz
velocity resolution is 1.6667 Hertz

2 velocity number is pulses/hundred seconds (0.01 pps)
maximum velocity is 5,461.333 Hertz
velocity resolution is 0.16667 Hertz

3 velocity number is pulses/thousand seconds (0.001 pps)
maximum velocity is 546.1333 Hertz
velocity resolution is 0.016667 Hertz

4 velocity number is pulses/sixty seconds (ppm)
maximum velocity is 9,102.222 Hertz
velocity resolution is 0.27778 Hertz

5 velocity number is pulses/second (pps)
maximum velocity is 1,092,266.67 Hertz
velocity resolution is 33.333 Hertz

0 acceleration number is velocity units/second
1 acceleration number is velocity units/ten seconds
2 acceleration number is velocity units/hundred seconds

A relative, preset move is defined to be an excursion from zero velocity to a nonzero velocity and back to zero velocity. The goal of a preset move is to get from the current position to a new position. Therefore, if the indexer is performing a preset move it considers the action of "performing a move" to be synonymous with the "motor moving." Thus, a relative, preset move can only be performed if the motor is not moving. It is a good

idea not to mix continuous moves with preset moves.

VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution

D5 E Define move "X," define it as an absolute, closed-loop move

01 -> FF,	move number
XY	X = velocity range and Y = acceleration range (see below)
00 -> 7F,	most significant byte of the velocity
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the velocity
00 -> 7F,	most significant byte of the acceleration
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the acceleration
00 -> FF,	most significant byte of the absolute position (enc. pulses)
00 -> FF,	second most significant byte
00 -> FF,	third most significant byte
00 -> FF,	least significant byte of the absolute position

This command will define move "X" as an absolute, closed-loop move to be performed by the "perform move number 'X'" command (40h). The first parameter byte specifies the move number, a move number of 00h is illegal and will be ignored. The most significant nibble of the second parameter byte denotes the velocity range of the move and the least significant nibble denotes the acceleration range. The velocity range determines the units of the velocity number sent, the maximum velocity achievable, and the velocity resolution. The acceleration range determines the units of the acceleration number with respect to the velocity number's units. The last twelve bytes specify the velocity, acceleration, and distance (in encoder pulses); all three numbers are interpreted in two's complement form. The velocity and acceleration must be specified as positive numbers. The distance specified is an absolute distance! All of the bytes must be present in order for an absolute move to be defined. Redefinition of a move is allowed by reissuing a complete define a move command.

Absolute positions can only be defined such that the difference between the current absolute encoder position and the desired absolute encoder position meets the relative encoder

distance specifications outlined in the D4h command above. In other words, the relative encoder distance to be travelled must be less than or equal to 7FFFFFFFh divided by the motor pulse to encoder pulse ratio. An absolute position specified outside of this range is invalid.

It should be noted that this move may only be performed after the motor pulse to encoder pulse ratio has been defined (see the B0h command). Without this ratio no moves may be performed. The distance specified in this move definition is in units of encoder pulses as seen by the indexer; that is, the distance is specified in terms of encoder lines times four (see the B0h command) not in terms of motor pulses.

Only one move number "X" may be defined at any time; for example, one can only define one move number 9, whether move 9 is a relative move, an absolute move, or a continuous move is of no relevance, there may be only one move 9.

There are six velocity ranges defined with three acceleration ranges per velocity range. If one fails to specify a defined range the move definition is invalid. The velocity and acceleration ranges are as follows.

0 velocity number is pulses/second (pps)
maximum velocity is 546,133.3 Hertz
velocity resolution is 16.667 Hertz

1 velocity number is pulses/ten seconds (0.1 pps)
maximum velocity is 54,613.33 Hertz
velocity resolution is 1.6667 Hertz

2 velocity number is pulses/hundred seconds (0.01 pps)
maximum velocity is 5,461.333 Hertz
velocity resolution is 0.16667 Hertz

3 velocity number is pulses/thousand seconds (0.001 pps)
maximum velocity is 546.1333 Hertz
velocity resolution is 0.016667 Hertz

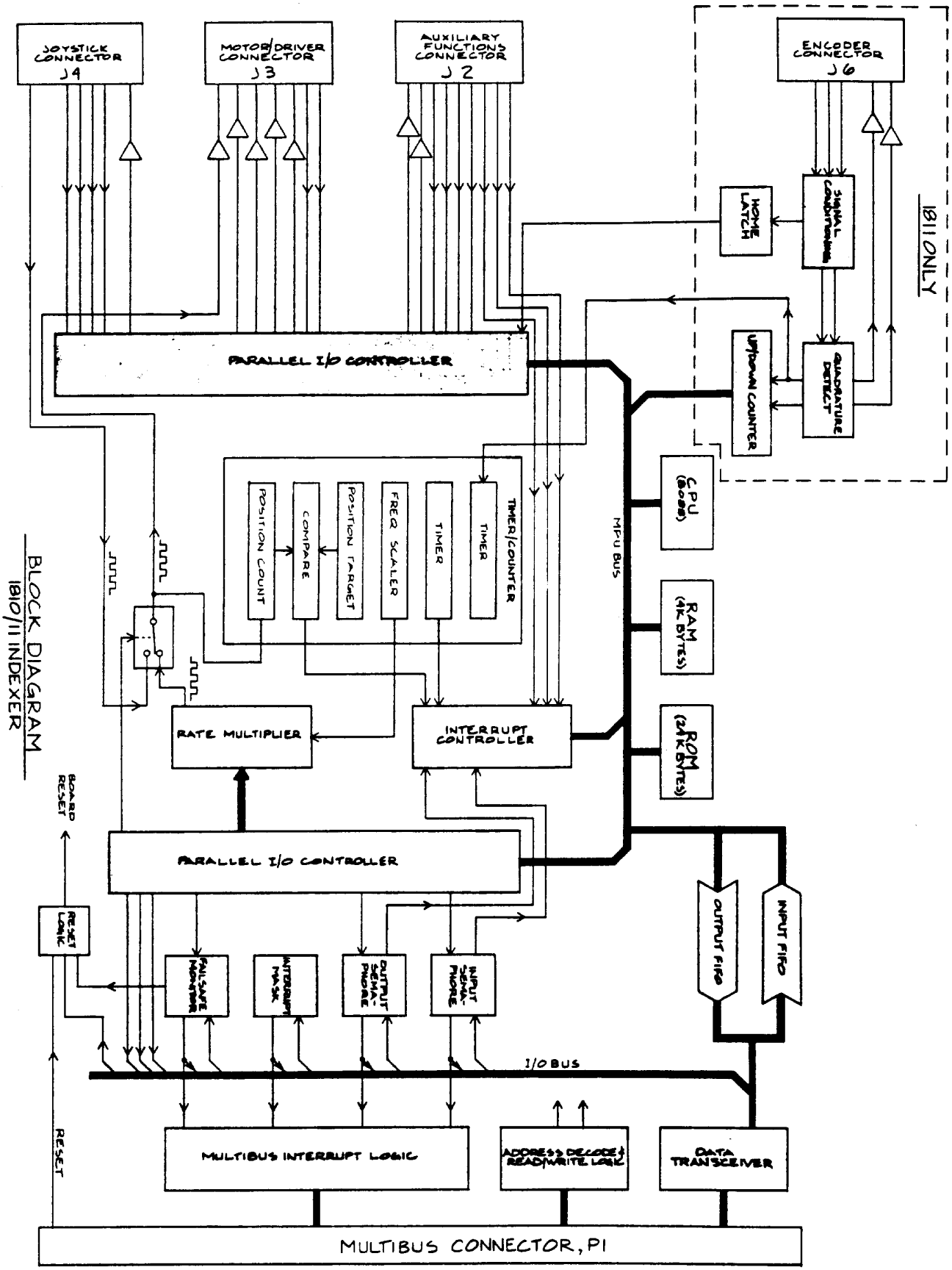
4 velocity number is pulses/sixty seconds (ppm)
maximum velocity is 9,102.222 Hertz
velocity resolution is 0.27778 Hertz

5 velocity number is pulses/second (pps)
maximum velocity is 1,092,266.67 Hertz
velocity resolution is 33.333 Hertz

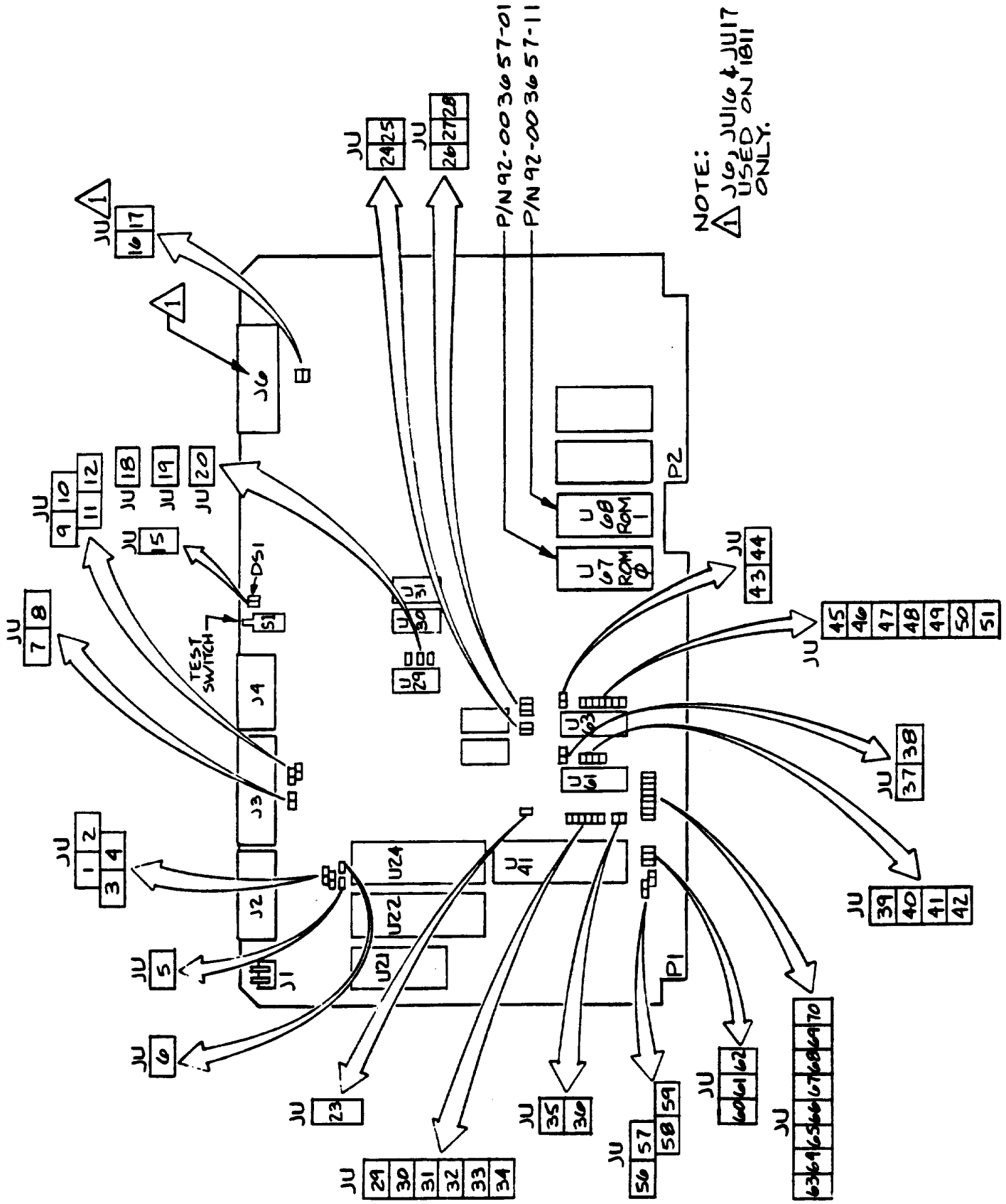
0	acceleration number is velocity units/second
1	acceleration number is velocity units/ten seconds
2	acceleration number is velocity units/hundred seconds

An absolute, preset move is defined to be an excursion from zero velocity to a nonzero velocity and back to zero velocity. The goal of a preset move is to get from the current position to a new position. Therefore, if the indexer is performing a preset move it considers the action of "performing a move" to be synonymous with the "motor moving." Thus, an absolute, preset move can only be performed if the motor is not moving. It is a good idea not to mix continuous moves with preset moves.

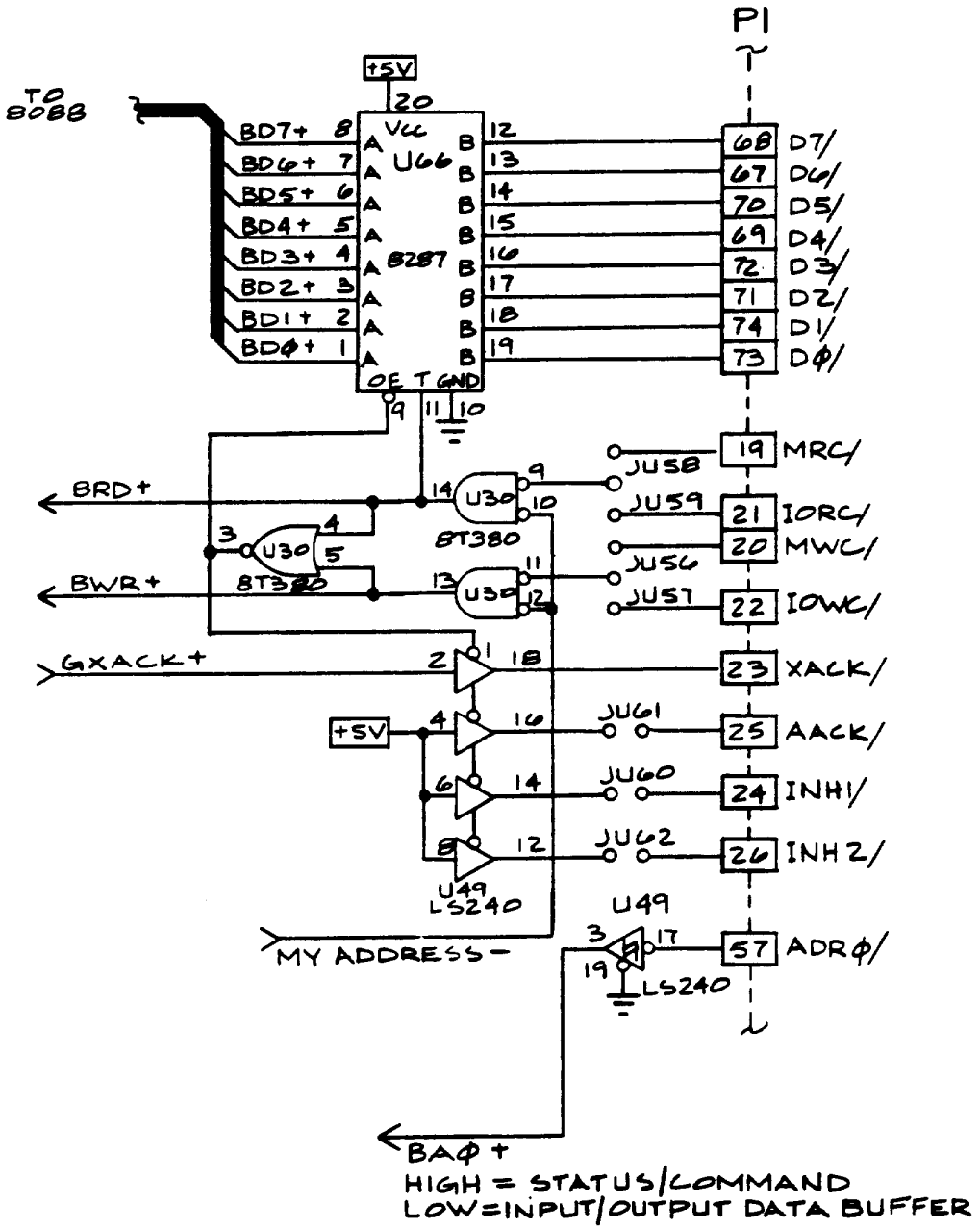
VALID-within-a-SEQUENCE-buffer
VALID-in-INDEXER-mode
VALID-concurrent-with-SEQUENCE-buffer-execution



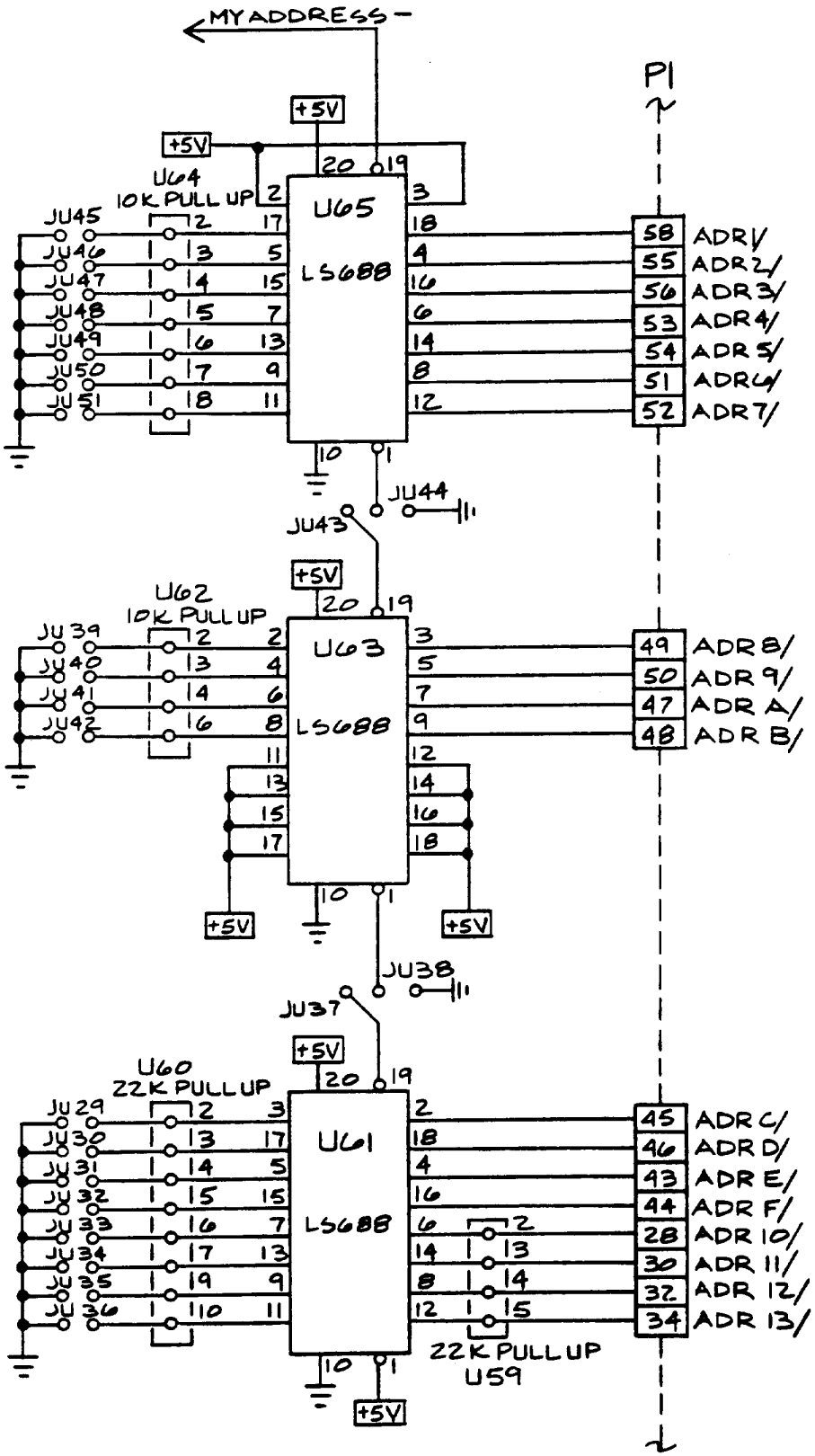
BLOCK DIAGRAM 1810/11 MOTOR CONTROLLER



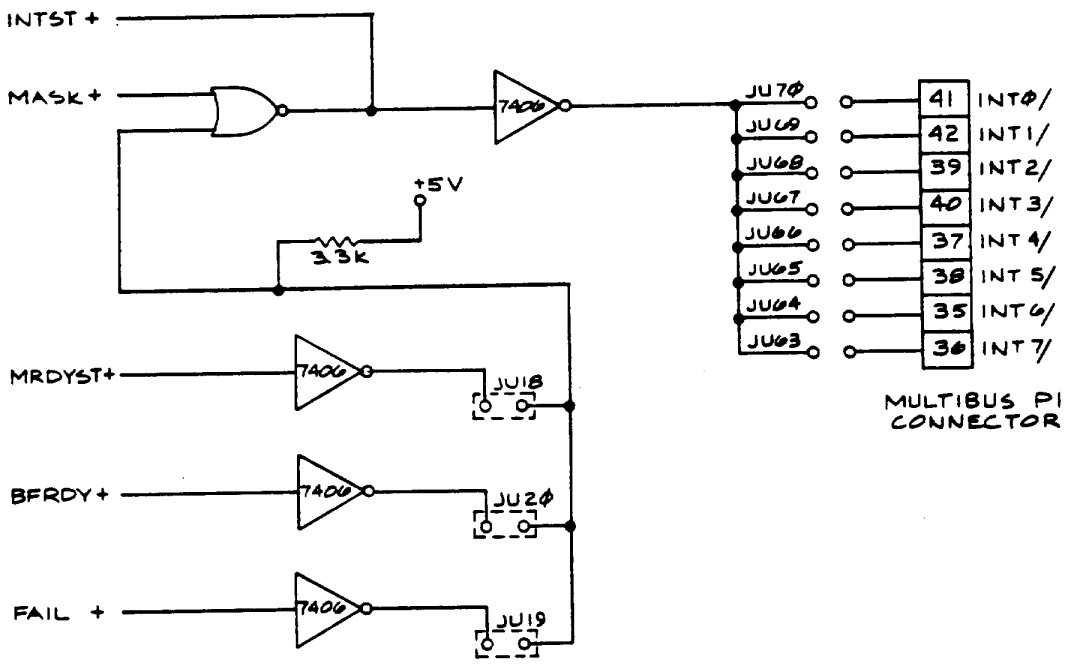
1810/11 CONFIGURATION JUMPER/EPROM LOCATION DIAGRAM



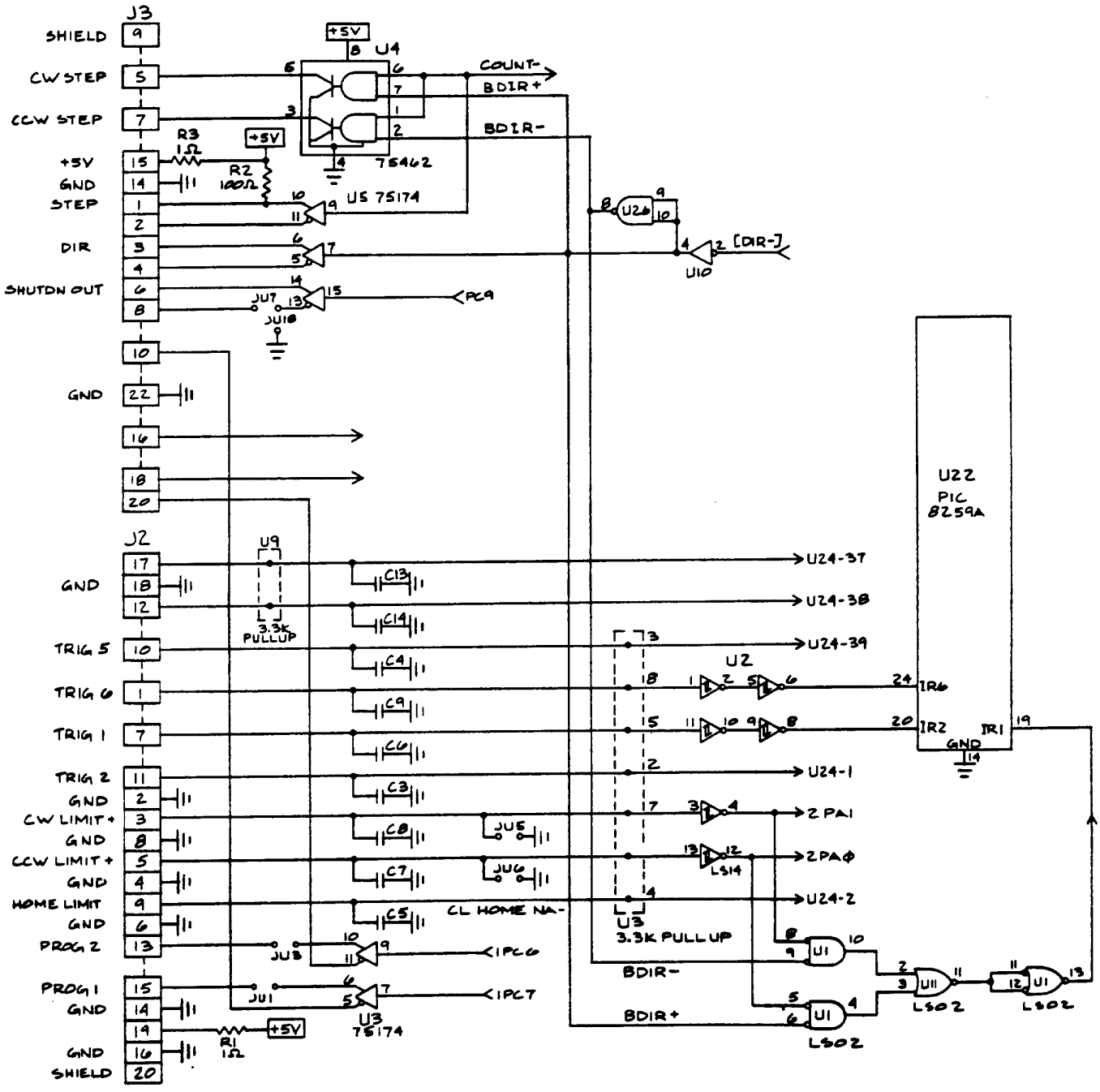
1810/11 ADDRESS SELECTION DIAGRAM, SHT 1 OF 2



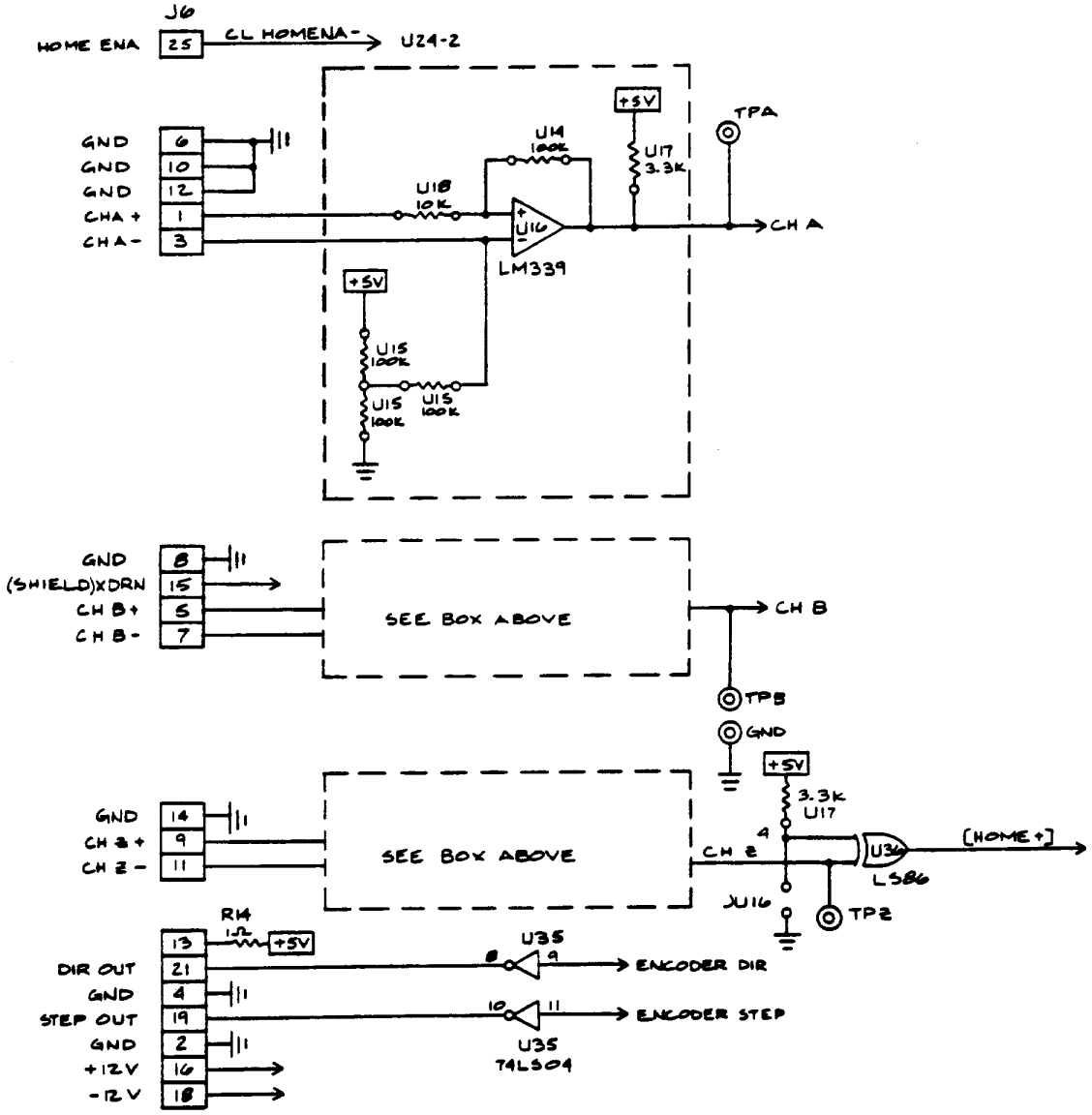
1810/11 ADDRESS SELECTION DIAGRAM, SHT 2 OF 2



1810/11 INTERRUPT SELECTION DIAGRAM



1810/11 AUXILIARY INTERFACE DIAGRAM



1810/11 ENCODER INTERFACE DIAGRAM

P/N 88-010945-01

Addendum to 1811 Manual

NEW COMMAND

37H - Resume stopped closed-loop move. This command will cause the 1811 to move to the last commanded closed-loop position. It was implemented as a fix for remotely resuming closed-loop moves on the 3000. The 3000 will need D4 rev software to take advantage of the new command.