

Application Design

Chapter Objectives

The information in this chapter will enable you to the following:

- Recognize and understand important considerations that must be addressed before you implement your application
- Understand the capabilities of the system
- Customize the system to meet your requirements

The X language command examples in this chapter are tailored to single-axis operation. To use these examples for multi-axis simulation, change the axis-specific prefix for each command (1, 2, or 3). If an axis is not specified, the OEMØ23-AT defaults to axis #1.

Application Considerations

Successful application of a rotary indexer system requires careful consideration of the following important factors:

- Mechanical resonance
- Ringing or overshoot
- Move times (calculated vs. actual)
- Positional accuracy and repeatability

Mechanical Resonance

Resonance, a characteristic of all stepper motors, can cause the motor to stall at low speeds. Resonance occurs at speeds that approach the natural frequency of the motor's rotor and the first and second harmonics of those speeds. It causes the motor to vibrate at these speeds. The speed at which fundamental resonance occurs is typically between 0.3 and 0.8 revs per second and is highest for small motors and lowest for large motors.

Most full-step motor controllers *jump* the motor to a set minimum starting speed to avoid this resonance region. This causes poor performance below one rev per second. In nearly all cases, using a microstepping drive with the OEMØ23-AT will overcome these problems.

Motors that will not accelerate past one rev per second may be stalling due to resonance. The resonance point may be lowered to some extent by adding inertia to the motor shaft. This may be accomplished by putting a drill chuck on the back shaft. *This technique is applicable only to double-shaft motors with the shaft extending from both ends of the motor.* In extreme cases, you may also need a viscous damper to balance the load. Changing the velocity (V command) and acceleration (A command) may resolve a resonance problem.

Helpful Hint

Viscous damper manufacturer:
Ferrofluidics Corp.
40 Simon Street
Nashua, NH 03061
(603) 883-9800

Ringling or Overshoot

The motor's springiness, along with its mass, form an underdamped resonant system that *rings* in response to acceleration transients (such as at the end of a move). Ringing at the end of a move prolongs settling time. *Overshoot* occurs when the motor rotates beyond the actual final position. The actual settling time of a system depends on the motor's stiffness, the mass of the load, and any frictional forces that may be present. By adding a little friction, you can decrease the motor's settling time.

Move Times: Calculated vs. Actual

You can calculate the time required to complete a move by using the acceleration, velocity, and distance values that you define with the **A**, **V**, and **D** software commands respectively. However, you should not assume that the values that you use will constitute the actual move time. After you issue the Go (**G**) command, the OEMØ23-AT may take up to 50 ms to calculate the move before the motor starts moving. You should also expect some time to elapse for the motor and the load to settle.

Positional Accuracy vs. Repeatability

In positioning systems, some applications require high absolute accuracy. Others require repeatability. You should clearly define and distinguish these two concepts when you address the issue of system performance.

If the positioning system is taken to a fixed place and the coordinates of that point are recorded, the only concern is how well the system repeats when you command it to go back to the same point. For many systems, what is meant by accuracy is really repeatability. Repeatability measures how accurately you can repeat moves to the same position.

Accuracy, on the other hand, is the error in finding a random position. For example, suppose the job is to measure the size of an object. The size of the object is determined by moving the positioning system to a point on the object and using the move distance required to get there as the measurement value. In this situation, basic system accuracy is important. The system accuracy must be better than the tolerance on the measurement that is desired.

For more information on accuracy and repeatability, consult the technical data section of the *Compumotor Programmable Motion Control Catalog*.

Open Loop Accuracy

Open-loop absolute accuracy of a step motor is typically less than a precision-grade system, but is better than most tangential drive systems. When you *close the loop* with an incremental encoder, the accuracy of these systems is equivalent to the encoder's accuracy.

The worst-case accuracy of the system is the sum of the following errors.
Accuracy = A + B.

- A **Uni-directional Repeatability**: The error measured by repeated moves to the same point from different distances in the same direction.
- B **Hysteresis**: The backlash of the motor and mechanical linkage when it changes direction due to magnetic and mechanical friction.

Using Terminal Emulation Mode

Compumotor recommends that you become familiar with the OEM023-AT's X language commands in *Terminal Emulation Mode* before you create your custom programs in another language. Terminal Emulation mode is a user-friendly menu-driven program that allows you to use X Language commands directly. After you are familiar with the X language, you can develop custom program routines (pre-tested in Terminal Emulation mode).

Helpful Hint
Steps to enter Terminal Emulation mode

- ① Type `OEM23TRM` at the OEM023-AT prompt.
- ② Enter device address `768`.

Positioning Modes

You can use one of three positioning modes to run the motor with X language commands—*normal* (preset), *alternating*, and *continuous*.

Normal (Preset) Mode

You can select preset moves by putting the OEM023-AT into normal mode using the Mode Normal (`MN`) command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (incremental moves) or in relation to a defined zero reference position (absolute moves). You can select incremental moves by using the Mode Position Incremental (`MPI` or `FSA0`) command. You can select absolute moves using the Mode Position Absolute (`MPA` or `FSA1`) command.

Incremental Mode Moves

When using the Incremental mode (`MPI` or `FSA0`), a preset move moves the shaft of the motor the specified distance from its starting position. For example, to move the motor shaft 1.5 revolutions, specify a preset move with a distance of `+37,500` steps (1.5 revs @ 25,000 steps/rev). Every time the indexer executes this move, the motor moves 1.5 revs from its resting position. You can specify the direction of the move with the optional sign (`D+37500` or `D-37500`), or define it separately with the Set Direction (`H`) command (`H+` or `H-`) after the `D` command. If you do not specify the direction (e.g., `D25000`), the unit defaults to the positive (CW) direction.

Sample Incremental Mode Moves

The moves below are incremental moves. The distance specified is relative to the motor's current position. *This is the default (power-up) positioning mode.*

Helpful Hint
The motor moves one CW revolution and stops. It then moves one more CW revolution in the same direction and stops. The motor changes direction and moves one CCW revolution.

Command	Description
<code>MPI</code>	Sets unit to Incremental Position mode
<code>A2</code>	Sets acceleration to 2 rps ²
<code>V5</code>	Sets velocity to 5 rps
<code>D25000</code>	Sets distance to 25,000 steps
<code>G</code>	Executes the move (Go)
<code>G</code>	Repeats the move (Go)
<code>H</code>	Reverses direction of next move
<code>G</code>	Executes the move (Go)

Helpful Hint
The motor returns to its original starting position.

Command	Description
<code>D-25000</code>	Sets the distance to 25,000 steps in the negative (CCW) direction
<code>G</code>	Executes the move (Go)

Helpful Hint
The motor moves 25,000 steps in the positive (CW) direction.

Command	Description
<code>H</code>	Toggles the motor direction of the next move, but maintains existing acceleration, velocity, and distance parameters.
<code>G</code>	Executes the same move profile as the previous move, but in opposite direction (Go)

To load all the commands before executing them, use the Pause (`PS`) and Continue (`C`) commands.

Command	Description
<code>PS</code>	Pauses execution until the indexer receives a Continue (<code>C</code>) command
<code>G</code>	Executes a 25,000-step move (Go)
<code>T3</code>	Waits 3 seconds after the move
<code>G</code>	Executes another 25,000-step move (Go)
<code>C</code>	Cancel pause and executes the <code>G T3 G</code> commands

Absolute Preset Mode Moves

A preset move in the Absolute mode moves the motor the distance that you specify (in motor steps) from the *absolute zero position*. You can set the absolute position to zero with the Position Zero (PZ) command, issuing the Go Home (GH) command, or by cycling the power to the indexer.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. For example, if the motor is at absolute position +12,500, and you instruct the motor to move to position +5,000, the motor will move in the negative direction a distance of 7,500 steps to reach the absolute position of +5,000.

When you issue the Mode Position Absolute (MPA or FSA1) command, it sets the mode to absolute. When you issue the Mode Position incremental (MPI or FSA0) command the unit switches to Incremental mode. The OEM023-AT retains the absolute position, even while the unit is in the incremental mode. You can use the Position Report (PR) command to read the absolute position.

Sample Absolute Moves

The moves shown below are absolute mode moves. The distance specified is relative to the OEM023-AT's absolute zero position.

Command	Description
MN	Sets the OEM023-AT to Normal mode
MPA	Sets the OEM023-AT to the Absolute Positioning mode
PZ	Sets the current absolute position to zero
A5	Sets acceleration to 5 rps ²
V3	Sets velocity to 3 rps
D5000	Sets move to absolute position 5,000
G	Executes move (motor moves to absolute position 5,000)

Command	Description
D10000	Sets the motor to absolute position 10,000. (Since the motor was already at position 5,000, it will move 5,000 additional steps in the CW direction.)
G	Executes the move (Go)
D0	Sets the motor to absolute position 0. (Since the motor is at absolute position 10,000, the motor will move 10,000 steps in the CCW direction.)
G	Executes the move (Go)

Alternating Mode

Helpful Hint

This indexing mode is not functional when the absolute positioning mode is selected (MPA or FSA1).

You can select the Alternating mode with the MA command. Set-up for Alternating mode is identical to that for Normal mode (MN). The difference is that when you issue the Go (G) command, the motor shaft rotates to the commanded position that corresponds to the value set by the D command, and then retraces its path back to the start position. The shaft continues to move between the start position and the command position. Normally, the motor stops immediately when you issue an S command. However, if you issue the SSD1 command before the G command, then when you issue the S command, the motor completes the cycle and stops at the start position. Another Go (G) command will repeat the same motion pattern.

Continuous Mode

The Continuous mode (MC) is useful for applications that require constant movement of the load, when the motor must stop after a period of time has elapsed (rather than after a fixed distance), or when the motor must be synchronized to external events such as trigger input signals. You can manipulate the motor movement with either buffered or immediate commands. After you issue the G command, buffered commands are executed in the order in which they were programmed. While the motor is in continuous motion, you can change the velocity and acceleration by issuing a new V and A command followed by a G command.

Sample
Continuous
Mode Move

The following example demonstrates a continuous mode sequence.

Command	Description
MC	Sets to Continuous mode
A10	Sets acceleration to 10 rps ²
V1	Sets velocity to 1 rps
G	Executes the move (Go)

Motor accelerates to 1 rps and continues at that same rate until you issue the **s** command, or until the load triggers an end-of-travel limit switch.

The following example demonstrates how to change the acceleration and velocity while the motor is moving (*on-the-fly*) in Continuous mode.

Command	Description
MC	Sets all moves to the Continuous mode
A1	Sets acceleration to 1 rps ²
V.5	Sets final velocity to 0.5 rps
G	Executes move (motor travels at 1 rps)
A10	Sets acceleration to 10 rps ²
V.4	Sets velocity to 0.4 rps
G	Changes velocity to 0.4 rps at 10 rps ²
V.1	Sets velocity to 0.1 rps
G	Changes velocity to 0.1 rps

Move Parameters

For rotary motors, velocity and acceleration parameters are expressed in units of motor revolutions. For accurate speed control, each axis needs to know its motor's resolution. Use the **MR** command to change motor resolution from the default (25,000 steps/rev) as required. This parameter also controls step pulse width and velocity range. Refer to the description of the **MR** command in ***OEM023-AT Software Reference Guide***.

It is desirable in many applications for the distance parameter to be specified in units other than motor steps, such as hundredths of a degree or thousandths of an inch. If the desired units correspond to an integer number of motor steps, the indexer's position multiplier (**US** command) capability can be used to scale the **D** command position parameter to those units for any axis.

Example

A 25,000-step/rev motor drives a 5-turn/inch lead screw. The system resolution is 125,000 steps per inch. To express the distance parameter in thousandths of an inch, the operator uses the **US125** command to set a distance multiplier of 125. If he then gives a distance command **D1000**, the motor will move 1,000 units of 125 steps each (one inch) on the next go (**G**) command. For more information, refer to the **US** command description in ***OEM023-AT Software Reference Guide***.

When scale factors greater than 1 are used, and an indexed move is interrupted by a Stop (s) command or by activation of a limit switch, the final position may not be a multiple of the scale factor. Consequently, the repeatability of that move is lost.

Program Control

This section discusses the X language program control features of the OEM023-AT.

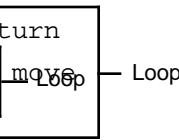
Loops

You may use the Loop (L) command to repeat certain sequences. You can nest loop commands up to 8 levels deep.

Command	Description
PS	Pauses command execution until the indexer receives a Continue (C) command
MPI	Sets mode to incremental
A5	Sets acceleration to 5 rps ²
V5	Sets velocity to 5 rps
L5	Loops 5 times
D2000	Sets distance to 2,000 steps
G	Executes the move (Go)
T2	Delays 2 seconds after the move
N	Ends Loop structure
C	Initiates command execution to resume

The example below shows how you can nest a small loop inside a major loop. In this example, the motor makes 2 moves and returns a carriage return. The unit repeats these procedures and will continue to repeat until you instruct the unit to stop immediately with an s (Stop) or κ (Kill) command. If you issue a Y (Stop Loop) command, the OEM023-AT finishes the current loop of commands and then stops the motor.

Command	Description
PS	Pauses command execution until the indexer receives a Continue (C) command
L	Loops indefinitely
1CR	Sends a carriage return
L2	Loops twice
G	Executes 2,000-step move
T.5	Waits 0.5 seconds
N	Ends loop
N	Ends loop
C	Cancel pause and Initiates command execution



POBs (Programmable Output Bits)

You can turn the programmable outputs on and off using the Output (o) command. Zero (0) turns off a given output and one (1) turns the output on. The outputs conduct when they are on and do not conduct when they are off (see the o command description in the ***OEM023-AT Software Reference Guide***).

Command	Description
MN	Set move to Mode Normal
PS	Pauses execution until indexer receives a Continue (C) command
A10	Sets acceleration to 10 rps ²
V5	Sets velocity to 5 rps
D25000	Sets move distance to 25,000 steps
O10XXX	Sets POB 1 on and POB 2 off and leaves the rest unchanged
G	Executes the move (Go)
C	Cancel the Pause and executes the move

Move Completion Signals

When you complete a move, you may use the OEM023-AT's programming capability to signal the end of the current move. In a preset move, you may use one of the following commands:

- Carriage return (CR)
- Output command (o)
- Status request command (PR reports motor's absolute position)

You may also use the moving/stopped bits of the status byte (SB) register to determine if an axis has completed its move.

Helpful Hint

The motor moves 12,500 steps. When the move is complete, the OEM023-AT issues a carriage return, turns on POB #3, reports the motor's absolute position, and reports the encoder's absolute position.

Command	Description
PS	Pauses command execution until the indexer receives a Continue (C) command
A2	Sets acceleration to 2 rps ²
V.1	Sets velocity to 0.1 rps
D12500	Sets distance to 12,500 steps
G	Executes the move (Go)
1CR	Sends a carriage return over the interface
OXX1XXX	Turns on Output #3 (ignores the other POBs)
1PR	Report absolute position
C	Cancel the Pause and executes the move

Custom Profiles

You can define a custom non-linear motion profile with the OEM023-AT's Velocity Streaming modes. The three Velocity Streaming modes that the OEM023-AT offers are listed below.

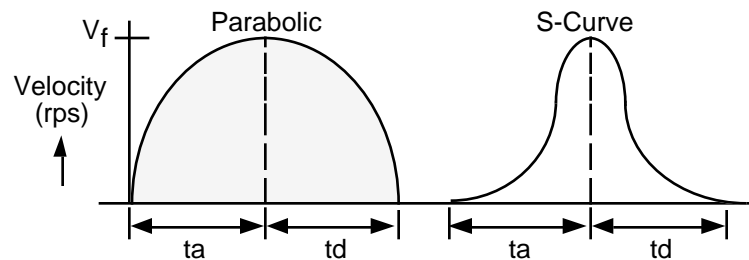
- Immediate Velocity (Q1 command)
- Time-Distance (Q2 command)
- Time-Velocity (Q3 command)

When you select Velocity Streaming mode (Q1), the OEM023-AT makes immediate velocity changes based on parameters you set with Set Immediate Velocity (RM) commands. The timing between issuing each RM command determines the exact move profile (refer to the Immediate Velocity Streaming section later in this chapter).

The Time-Distance (Q2) mode allows you to specify an update period at which the OEM023-AT travels the next distance that you specify.

The Time-Velocity (Q3) mode is very similar to the Time-Distance (Q2) mode, except that the variable parameter is the velocity.

By using these modes, you can define any motion profile. You can use a parabolic motion profile to optimize the motor's available running torque. An S-Curve profile provides smoother handling of sensitive loads by eliminating sudden acceleration changes.



For more information on custom profiling modes, refer to the RM, Q2, and Q3 commands in ***OEM023-AT Software Reference Guide***.

Immediate Velocity Streaming Mode

In this mode, the host computer software supplies a stream of velocity data (X language **RM** commands) for real-time control of motor speed. The OEMØ23-AT has no control of acceleration. *The operator controls all velocity changes. If the motor changes too abruptly for the motor to follow, the motor will stall or fault.*

Once this operating mode is entered with the **Q1** command, the OEMØ23-AT clears out any buffered commands waiting for execution and waits for Rate Multiplier in Velocity Streaming mode (**RM**) commands. With these commands, the OEMØ23-AT makes instantaneous changes to the specified velocity. The timing between issuing each **RM** command determines the exact move profile. Sending **RM** commands to the OEMØ23-AT in rapid succession provides smoother motion by virtue of an S-curve acceleration. Testing and modification may be required to establish the correct sequence of **RM** commands.

CAUTION

Do not make the velocity changes too abrupt. Abrupt velocity changes may stall the motor.

In the default high velocity range, with the default motor resolution (25,000 steps/rev) and a 25,000-step motor, a data increment of 1 produces a velocity increment of about 15.259 steps per second.

***RM** commands are executed immediately. Consequently, it is not possible to download a list of **RM** values to the OEMØ23-AT, and execute it afterward.*

To exit Velocity Profiling mode, you must use the **QØ** command. When a **Q1** or **QØ** command is received, it is performed before any buffered commands.

Syntax, Range, and Output Control for RM Commands

The syntax for **RM** commands is **RMn₁n₂n₃n₄** where n₁n₂n₃n₄ represents a 4-digit hexadecimal number. The most significant bit of n₁ is interpreted as a direction bit. If the most significant bit = 1, the motor will turn CW. If the most significant bit = Ø, the motor will turn CCW.

For example, issuing **RMØ666** changes the velocity of Axis #1 to 24,994 Hz (666 Hex □ 15.259 Hz) in the CCW direction. *This assumes a motor resolution of 25,000 steps/rev.*

To change the direction, you must enter a zero (Ø) point (**RMØØØØ** if the motor was traveling CCW or **RM8ØØØ** if the motor was traveling CW). *If you do not enter the zero point, the new **RM** data point will be ignored.* The following table summarizes the **RM** command parameter range.

Parameter	CW Rotation		CCW Rotation	
	Decimal	Hex	Decimal	Hex
Zero	32,768	8000	0	000
Maximum	65,523	FFF3	32,755	7FF3

The hex values 7FF4 through 7FFF, and FFF4 through FFFF are not listed above. These are special function values that do not produce a motor speed value. Instead, they provide signals on the programmable outputs in the middle of the velocity profiling operation. These values may be inserted in the data stream to generate a signal when the OEMØ23-AT gets to critical points in the process. The functions of these values are shown in the following table.

Output	To set low	To set high
POB #1	7FF4 or FFF4	7FF5 or FFF5
POB #2	7FF6 or FFF6	7FF7 or FFF7
POB #3	7FF8 or FFF8	7FF9 or FFF9
POB #4	7FFA or FFFA	7FFB or FFFB
POB #5	7FFC or FFFC	7FFD or FFFD

Immediate Streaming Example

Command	Description
Q1	Enter Velocity Profiling mode
RM0011	Go to RM velocity of 249 Hz in CCW direction
RM0055	Go to RM velocity of 1,105 Hz in CCW direction
RM0100	Go to RM velocity of 3,906 Hz in CCW direction
RM0055	Go to RM velocity of 1,105 Hz in CCW direction
RM0011	Go to RM velocity of 249 Hz in CCW direction
RM0000	Go to zero velocity (mandatory) to change direction
RM8011	Go to RM velocity of 249 Hz in CW direction
RM8055	Go to RM velocity of 1,105 Hz in CW direction
RM8100	Go to RM velocity of 3,906 Hz in CW direction
RMFFF5	Set Output #1 high
RM8055	Go to RM velocity of 1,105 Hz in CW direction
RM8011	Go to RM velocity of 249 Hz in CW direction
Q0	Exit velocity profiling mode

Timed Data Streaming Modes

The Timed Data Streaming modes allow precise multi-axis distance and velocity control. Timed Data Streaming is accomplished by dividing the profile into small straight-line segments, allowing greater accuracy and control of the profile shape.

- ❑ **Time-distance streaming** allows control over the *number* of steps output over a given period of time.
- ❑ **Time-velocity streaming** allows control over the *frequency* of steps output over a given period of time.

In both time-distance and time-velocity streaming modes, outputs can be turned on or off *on-the-fly* (while the motor is in motion). Wait-on-trigger and looping are also possible in these timed data streaming modes. To produce a data streaming profile, complete the following steps.

Helpful Hint

* Steps ④ and ⑤ can be reversed, refer to Continuous Streaming section.

- ① Enter the timed data streaming mode for the appropriate axes.
- ② Define the update interval for each axis.
- ③ Identify the clock source for each axis.
- ④ Provide the time-distance or time velocity streaming data.*
- ⑤ Start the master clock.*
- ⑥ Exit the timed data streaming mode after the motion is completed.

① Enter Timed Data Streaming Mode

There are two modes associated with timed data streaming. They are time-distance streaming and time-velocity streaming.

To enter the *Time-Distance Streaming mode*, enter the Q2 command for the appropriate axes. For example, entering 1Q2 and 2Q2 puts axes one and two in the time-distance streaming mode.

To enter the *Time-Velocity Streaming mode*, enter the Q3 command for the appropriate axes. For example, entering 2Q3 and 3Q3 puts axes two and three in the time-velocity streaming mode.

CAUTION

When either the Q2 or Q3 mode is entered, motion stops and the command buffers are cleared on the specified axes.

② Define the Update Interval

The update interval is defined with the TD_{nn} command, where **nn** is the update period ranging from 2 to 50 ms in 2-ms increments. Smaller update intervals produce finer discrete line segments. For example, issuing 1TD50 and 2TD50 establishes an update interval of 50 ms for axes one and two. The default update interval is 10 ms for each axis.

CAUTION

All axes in a master/slave relationship (using the MSL command) must have the same update interval and the same minimum pulse width (set with the MR command).

③ Identify Clock Source

Using the **MSL** command, you must specify the clock source for each axis. In doing this, you determine the master/slave relationship of the axes to one another or to other boards. The **MSL** command must be specified in the form **MSLn₁n₂n₃**, where each **n_n** value represents the clock source for axes ① through ③.

Axis #1

For Axis #1, the possible values for **n₁** are as follows:

- 1 If axis #1 uses its own internal clock (master)
- 2 If axis #1 uses axis #2's internal clock (slave to axis #2)
- 3 If axis #1 uses axis #3's internal clock (slave to axis #3)
- x If axis #1 is not in Data Streaming mode
- ∅ If axis #1 uses an external clock connected to pins 1 and 2 on the RMCLK slave-to-external connector (refer to *Chapter ⑤ Hardware Reference* for connections)

Axis #2

For Axis #2, the possible values for **n₂** are as follows:

- 1 If axis #2 uses axis #1's internal clock (slave to axis #1)
- 2 If axis #2 uses its own internal clock (master)
- 3 If axis #2 uses axis #3's internal clock (slave to axis #3)
- x If axis #2 is not in Data Streaming mode
- ∅ If axis #1 and #2 uses an external clock (via the RMCLK connector)

Axis #3

For Axis #3, the possible values for **n₃** are as follows:

- 1 If axis #3 uses axis #1's internal clock (slave to axis #1)
- 2 If axis #3 uses axis #2's internal clock (slave to axis #2)
- 3 If axis #3 uses its own internal clock (master)
- x If axis #3 is not in Data Streaming mode
- ∅ If axis #1, #2, and #3 uses an external clock (via the RMCLK connector)

CAUTION

Never overlap clock sources. For instance, do not issue **MSL21X**, where axis #2 would use axis #1's clock source and axis #1 would use axis #2's clock.

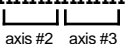
When using the external clock, axis #1 must always use the external clock and be set to a data streaming mode (1Q2 or 1Q3). If the command string 1Q2 2Q2 3Q2 MSLX∅∅ was issued, the MSLX∅∅ command would be invalid because axis 1 is not using an external clock.

The most common use of the **MSL** command is to specify one axis as the master while other axes follow the master. For example, if you use the **MSL111** command, axis #1 is the master and axes #2 and #3 follow axis #1.

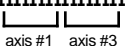
④ Establish Timed Data Streaming Format and Data

The Streaming Data (**SD**) command specifies either the distance to travel (in the Q2 mode) or the velocity output (in Q3 mode) in one update period. Special data points implement certain functions such as loops, setting the POBs, and waiting for a specific trigger input pattern. *Data-intensive streaming on one axis is likely to reduce command throughput on the other axes that are not in streaming mode.*

The format for the **SD** command is **SDnnnn(nnnn[nnnn])**, where **nnnn** is a 2, 4, or 6 byte HEX code. If one axis is in Streaming mode, the format is **SDnnnn**. If two axes are in Streaming mode, the format is **SDnnnnnnnn**. If all three axes are in Streaming mode, the format is **SDnnnnnnnnnnnnnn**.

SDnnnnnnnnnn

 axis #2 axis #3

The order of the four HEX digits corresponds from the lowest to the highest axis number value. For example, if axes #2 and #3 are in Timed Data Streaming mode, the order is as shown in the graphic to the left.

SDnnnnnnnnnn

 axis #1 axis #3

If axes #1 and #3 are in Timed Data Streaming mode, the order is as shown in the graphic to the left.

The range of values for the four HEX digits is 0000 - 7EFF or 8000 - FEFF. These values can represent *distance* in Q2 mode or *velocity* in Q3 mode. HEX digits 0000 - 7EFF correspond to a data range of 0 - 32511 (decimal).

The most significant bit of the four hex digits sets the direction. A 1 (binary) in this bit position means CW. The remaining three bits specify a magnitude. Therefore, the data range 0000 - 7EFF corresponds to CCW, and 8000 - FEFF corresponds to CW. For example, to specify a CW distance of 100 motor steps, the SD command would be SD8064. To specify a CCW distance of 100 steps, the SD command would be SD0064 (64 hex = 100 decimal).

Distance

For an axis in the normal velocity range (SSF0), the maximum *distance* that can be specified is calculated as follows: $[(\text{UPDATE INTERVAL IN MS}) \cdot 1000/2] - 1$

For example, if the update interval is 10 ms, the maximum distance is 4,999 steps (SD1387 or SD9387).

For an axis in the low velocity range (SSF1), the maximum *distance* that can be specified is calculated as follows: $[(\text{UPDATE INTERVAL IN MS}) \cdot 100/2] - 1$

For example, if the update interval is 10 ms, the maximum distance is 499 steps (SD01F3 or SD81F3).

Velocity

For an axis in the normal velocity range, the **velocity** value for nnnn in the SDnnnn command is determined by dividing the desired velocity by 15.259 Hz (15.259 steps/rev). For example, to achieve 1,526 steps/sec CW, use the SD8064 command (1,526/15.259 = 100 = 64rps HEX). If you wanted to move CCW, you would use the SD0064 command.

For an axis in the low velocity range, the **velocity** value for nnnn in the SDnnnn command is determined by dividing the desired velocity by 1.526 Hz (1.526 steps/rev). For example, to achieve 153 Hz (steps/sec) CW, use the SD8064 command (153/1.526 = 100 = 64 HEX) for one update interval. If you wanted to move CCW, you would use the SD0064 command.

CAUTION

When specifying consecutive data points that are different in direction, you must insert a zero (0) data point between the two non-zero data points that are different in direction. If this is not done, Timed Data Streaming mode for this axis will be exited.

Special Data Points

The maximum value for a normal data point is 7EFF or FEFF. The range of values 7F00 - 7FFF and FF00 - FFFF performs specialized functions. *Data values in this range are interpreted not as data, but as special commands.*

Loops

SD7F01 thru SD7F7F and SDFE01 thru SDFE7F are loop commands with the loop count as the two least significant hex digits. SD7F80 and SDFE80 are endloop commands. The most significant bit corresponds to the direction. Use SD7F00 through SD7F80 for loops beginning CCW. Use SDFE00 through SDFE80 for loops beginning CW.

The loop count range is 1 - 127. SD7F00 (CCW) and SDFE00 (CW) are loop commands with an infinite loop count.

Helpful Hint:
To create an infinite loop that outputs 100 steps for each update period, use the following commands.

Command	Description
SDFE00	Loop an infinite number of times
SD8064	Move CW 100 steps
SDFE80	End loop

Helpful Hint:
For a finite loop count of 5, you could issue the following commands

Command	Description
SD7F05	Loop 5 times
SD0064	Move CCW 100 steps
SD7F80	End loop

Any loop can be terminated on a *loop boundary* by sending another endloop data point. For instance, if the OEM023-AT was half way through the third cycle of a 5-cycle loop and you issued another SD7F80 (or SDFE80) command, the OEM023-AT would finish the current loop cycle and then stop.

Outputs

SD7FC0 thru **SD7FFF** commands set or clear POBs 1 - 5. In the **SD** data point, bit positions 0 thru 5 correspond to POBs 1 - 6 respectively as follows:

SD Bit:	7	6	5	4	3	2	1	0
POB #:	X	X	X	5	4	3	2	1

To set or clear a POB, two POB data points are required: a *set/clear mask* and a *don't care mask*, specifying which bits are to be affected by the set/clear mask and which ones are to be unaffected.

When a pair of POB set/clear and don't care mask data points are encountered after the POB action is taken, the next distance or velocity data point is taken from the data buffer. This means that only 1 POB data pair is allowed per update interval. If you specify more than one POB data point pair, the motion may not be smooth.

Helpful Hint:

This example demonstrates how to calculate the two POB data points needed to set and clear POBs 1 - 5.

Desired outcome:	POB1 = 0	POB2 = 1	POB3 = 1
	POB4 = X	POB5 = X	

- ① Calculate the first **SD7Fnn** command (*set/clear mask*). The set/clear mask specifies which bits (outputs) are not being turned off (output = 0).
 - a Since there are only five outputs, put 111 on the front of the POB pattern (1, 1, 1, POB5, POB4, POB3, POB2, POB1). This is done to make two hex characters. The result is 111XX110.
 - b Replace Xs with 1s. The result is 11111110.
 - c Convert the 8-bit binary number to HEX (result is FE).
 - d Thus, you should specify the data point as **SD7FE**.
- ② Calculate the second **SD7Fnn** command (*don't care mask*). The don't care mask specifies which bits (outputs) are to be left in their current state (output = X).
 - a Replace 1s with 0s, and Xs with 1s in the POB pattern—011000.
 - b Put 11 on the front of the new POB pattern (to make two hex characters). The result is 11111000.
 - c Convert the 8-bit binary number to HEX (result is F8).
 - d Thus, you should specify the data point as **SD7FD8**.

When axes are in a master/slave arrangement via the **MSL** command, **SD** data alignment is based on valid data points and not on Trigger, POB, or Loop/Endloop data points. The following commands will output 100 steps on each axis for the first update interval.

Helpful Hint:

POB #1 will be high and both axes will output 50 steps during the second update interval.

Command	Description
SD80648064	Move both axes CW 100 steps
SD7FFF7FFF	Set POB #1 high—leave the rest SD7FFE7FFE unchanged (equivalent to 1XXXXX)
SD80328032	Move both axes CW 50 steps

Triggers

The **SDFFC0** - **SDFFFF** commands implement a *buffer pause* based on the pattern of trigger bits 1 - 6. *Buffer pause* repeats the previous data value until the trigger condition is met. In the **SD** data point bit, positions 0 - 5 correspond to triggers 1 - 6 respectively (i.e., bit 0 is trigger bit 1).

To implement a buffer wait function, two trigger *data* points are required. A *wait mask* and a *don't care mask* specify which bits are to be affected by the wait mask and which ones are to be unaffected. The calculations to compute each mask are performed just like the output mask calculations discussed in the previous section.

Once a valid trigger wait pair of data points is encountered, data points in the buffer are not fetched in subsequent update intervals (buffer-paused) until the pattern is satisfied. For example, to wait for trigger bits 2 and 3 to be set and bit 1 to be cleared while ignoring bits 4, 5, and 6, the following **SD** commands would be used.

Command	Description
SDFFFE	Wait for bit 1 to clear, bits 2 & 3 to be set, and SDFFF8 ignore bits 4, 5, & 6 (011XXX)

⑤ Start Master Clock

The Master/Start (**MSS**) command initiates the start of the clock for the pulse generation circuitry on any axis that is designated as a master.

Data Streaming Restrictions

The following are restrictions to be considered when using the Timed Data Streaming mode:

- ❑ If the master/slave relationship with another board is desired, then axis #1 must be included as a streaming axis, and must use the external clock.
- ❑ The minimum motor resolution required is 5,000 steps/rev.
- ❑ All master/slave axes must have the same velocity range (*SSF* command).
- ❑ If a limit is hit while in Timed Data Streaming mode (*Q2* or *Q3*), Timed Data Streaming mode is exited for that axis only (equivalent to a *Q0* command).
- ❑ For update periods of 6 ms or less, binary input mode is recommended (see below).
- ❑ While streaming in a master/slave relationship, all slave axes should be exited first, before the master axis. For example, when operating axis 1 through 3 under the command *MSL111*, you should exit as follows: *3Q0 ... 2Q0 ... 1Q0*.
- ❑ Discrete data streaming is limited by the OEM023-AT buffer size (see *Continuous Streaming* discussed later in this chapter).

Time-Distance Streaming Example

An application requires axes #1 and #2 to run in the Time-Distance Streaming mode and axis #3 to be independent.

Command	Description
<i>3A10</i>	Acceleration = 10 rps ²
<i>3V10</i>	Velocity = 10 rps
<i>3D25000</i>	Distance = 25000 steps
<i>3L200</i>	Loop 200 times
<i>3G</i>	Start motion
<i>3T1</i>	Pause 1 second
<i>3H</i>	Change direction
<i>3G</i>	Start motion
<i>3H</i>	Change direction
<i>3T2</i>	Pause 2 seconds
<i>3N</i>	End loop
<i>1Q2</i>	Enter streaming mode axis #1
<i>2Q2</i>	Enter streaming mode axis 2
<i>MSL22X</i>	Synchronize axis 1 clock to axis 2, axis 3 independent
<i>1TD10</i>	Axis 1 update interval = 10 ms
<i>2TD10</i>	Axis 2 update interval = 10 ms
<i>SD00640096</i>	Move axis #1 100 steps CCW, move axis #2 150 steps CCW
<i>SD00000000</i>	Stop axis 1 and 2 - <i>required to change direction</i>
<i>SD806481CA</i>	Move axis 1 100 steps CW, move axis 2 458 steps C
<i>SDF05FF05</i>	Loop axes 1 and 2, 5 times CW
<i>SD80648064</i>	Move axes 1 and 2 100 steps CW
<i>SD7FF67FF6</i>	Two commands required to
<i>SD7FD07FD0</i>	Set POBs (X,5,4,3,2,1) to XX0110
<i>SD80968096</i>	Move axis 1 and 2 150 steps CW
<i>SD7FD97FD9</i>	Two commands required to
<i>SD7FD07FD0</i>	Set POBs (X,5,4,3,2,1) to XX1001
<i>SDF80FF80</i>	End loop
<i>SD00000000</i>	Stop axes 1 and 2
<i>SD7F7F7F7F</i>	Loop axis 1 and 2, 127 times CCW
<i>SD010000FF</i>	Move axis 1 256 steps CCW, move axes 2 255 steps CCW
<i>SD00000000</i>	Stop axis 1 and 2
<i>SDFFFFFFFF</i>	Both axes wait on trigger (6,5,4,3,2,1)
<i>SDFFBFFFB</i>	To be XXX1XX
<i>SD7F807F80</i>	End loop
<i>SD00000000</i>	Stop axis 1 and 2
<i>MSS</i>	Start master clock. Computer time delay needed (e.g., 1400 ms)
<i>1Q0</i>	Exit streaming on axis 1
<i>2Q0</i>	Exit streaming on axis 2

Time-Velocity Streaming Example

An application requires axis #1 and axis #3 to run in Time-Velocity Streaming mode and axis #2 to run independently.

Command	Description
2A10	Acceleration = 10 rps ²
2V10	Velocity = 10 rps
2D10000	Distance = 10000 steps
2L200	Loop 200 times
2G	Start motion
2T1	Pause 1 second
2H	Change direction
2G	Start motion
2H	Change direction
2T1	Pause 1 seconds
2N	End loop
SSF0	High velocity mode (default)
1Q3	Enter streaming mode axis 1
3Q3	Enter streaming mode axis 3
MSL1X1	Synchronize axis 1 clock to axis 3, axis 2 independent
1TD20	Axis 1 update interval = 20 ms
3TD20	Axis 3 update interval = 20 ms
SD7F7F7F7F	Loop axis 1 and 2, 127 times CCW
SD00050005	By end of update interval, velocity = 76.3 steps/sec
SD00100010	By end of update interval, velocity = 244.1 steps/sec
SD00000000	By end of update interval, velocity = 0 steps/sec
SD7F807F80	End loop
MSS	Start master clock) computer delay needed (e.g., 60 ms)
1Q0	Exit streaming mode axis 1
3Q0	Exit streaming mode axis 3
SSF1	Low velocity mode
1Q3	Enter streaming mode axis 1
3Q3	Enter streaming mode axis 2
MSL1X1	Synchronize axis 1 clock with axis 3, axis 2 independent
1TD50	Axis 1 update interval = 50 ms
3TD50	Axis 3 update interval = 50 ms
SDF005F005	By end of update interval, velocity = 76.3 steps/sec
SD7FF67FF6	Two commands required to
SD7FD07FD0	Set POBs (6,5,4,3,2,1,) to 1X0110
SD7F207F20	Loop axes 1 and 2, 20 times CCW
SDF005F005	By the end of update interval velocity = 76.3 steps/sec
SD7F807F80	End loop
SD7FD97FD9	Two commands required to
SD7FD07FD0	Set POBs (6,5,4,3,2,1,) to 1X1001
SD80108010	By end of update interval, velocity = 24.41 steps/sec
SD00000000	By end of update interval, velocity = 0 steps/sec
SD7F807F80	End loop
MSS	Start master clock. Computer delay needed (e.g., 150 ms)
1Q0	Exit streaming mode axis 1
3Q0	Exit streaming mode axis 3

Motion stops when the time velocity buffer is empty.

Discrete Data Streaming

The previous time-distance and velocity-distance examples use what is referred to as *discrete data streaming*, which follows this pattern.

- ① **Q2** or **Q3** Enter time-distance or velocity-distance mode
- ② **TD** Provide time-distance update rate or period
- ③ **MSL** Specify master/slave clock source
- ④ **SD** Specify stream data or send distance
- ⑤ **MSS** Master/slave start (start movement)
- ⑥ **QØ** Exit time-distance mode

This form of streaming is useful when the number of **SD** data points does not exceed the OEMØ23-AT's buffer capacity. Since each axis buffer can hold 1,000 characters, this corresponds to about 200 **SD** data points per axis (1,000 bytes @ 5 bytes per **SD** data point).

By using discrete streaming, the standard communication handshaking can be used (refer to **TERMINAL.BAS**, **OEM23P.PAS**, or **OEM23C.C** on the support disks). Discrete streaming also allows update intervals of 2 ms to be used without using the binary input mode or assembly language drivers.

Continuous Data Streaming

Continuous data streaming differs from discrete data streaming in that the **SD** data points are sent to the OEMØ23-AT **after** starting the master clock, instead of **before** starting the master clock. By choosing continuous streaming over discrete streaming, you are not limited by the OEMØ23-AT buffer. Therefore, you can send an infinite number of **SD** data points to the OEMØ23-AT.

CAUTION

You must check bit 7 of the status byte to ensure that the OEMØ23-AT data buffer is able to handle the data. The buffer cannot accept new data until bit 7 changes from Ø to 1. You can send up to 200 data points (2 bytes per point) at one time and wait for bit 7 to change from Ø to 1 to send 200 more.


The command sequence required to conduct a continuous streaming operation for each axis is as follows:

- ① **Q2** or **Q3** Enter time-distance or velocity-distance mode
- ② **MSL** Specify master/slave clock source
- ③ **TD** Provide time-distance update rate or period
- ④ **MSS** Master/slave start (start movement)
- ⑤ **SD** Specify velocity or # of steps to output
- ⑥ **QØ** Exit time-distance mode

Step ⑤ can be accomplished in two different ways. One way is by repeatedly using the standard OEMØ23-AT write drivers (refer to the **TERMINAL.BAS**, **OEM23P.PAS**, or **OEM23C.C** programs on the support disk). The other approach is to use the OEMØ23-AT's binary input capability (described below)—files **OEM23TDP.PAS** and **OEM23TDC.C**.

Binary Input Mode

By using the binary input mode, you can use a much faster update rate. Having a shorter update interval allows more precise control of the motion segments. Within the binary input mode, the data points are entered as binary words. Therefore, when you enter data this way, only two, four, or six bytes (for one, two, or three axes) per input session are allowed. During each input session, one *SD* data point (for 1, 2, or 3 axes) is transferred to the OEM023-AT.

 **Helpful Hint:**
The following steps are required to support a binary data input mode session.

- ① Read the status byte (base address + 1) until bit 4 is set (OEM023-AT ready to receive a byte).
- ② Send one byte of the 2, 4, or 6-byte data packet to the OEM023-AT data register (base address).
- ③ Send HEX 71 to the control byte (base address +1). This puts the OEM023-AT in binary input mode and informs the OEM023-AT that a data byte has been transferred to it.
- ④ Read the status byte until bit 4 is cleared (data byte accepted by OEM023-AT)
- ⑤ If the data byte that you sent in step 2 is the last byte of the 2, 4, or 6-byte packet, send HEX 60 to the control byte (exit binary input mode). If the data byte sent in step 2 is not the last byte, send HEX 61 (remain in binary input mode) to the control byte and repeat steps 1 through 5.

Binary Input mode will work only in Continuous Streaming mode.

Example

Axes #1 and #2 are in the time-distance mode (Q2) with axis #1 as the master and axis #2 as the slave, using the step pulse clock from axis #1 (i.e., *MSL11X*). The update rate is 2 ms (TD2).

To input a data point so that axis #1 outputs 100 steps CW for one update period and axis #2 outputs 200 CW steps for the same period, two binary words (4 bytes) are sent in one binary input mode session as follows (the sequential order is from top to bottom).

Command	Description
MSB1	HEX 80 (binary, <i>not</i> ASCII)
LSB1	HEX 64
MSB2	HEX 80
LSB2	HEX C8

On software support disk #2, binary input mode examples are written in two different program languages: PASCAL and C.

Language Considerations

When creating a program in the timed data streaming mode, you must consider the programming language you are using. Certain languages, such as C, run considerably faster than BASIC. PASCAL also runs faster than BASIC, even a compiled BASIC, such as Microsoft™ QuickBASIC.

On tests performed with an IBM compatible (80286 microprocessor), Borland TURBO C 2.0 was able to run a continuous timed data streaming program with an update interval of 6 ms (TD6). This program used the binary input mode to communicate *SD* data points to the OEM023-AT. Using Borland TURBO PASCAL 5.0, a similar program ran at 12 ms update intervals (TD12). Using Microsoft Quick BASIC 4.5, a comparable program ran at 30 ms update intervals (TD30).

If you require update intervals of 2 ms, you must use assembly language binary input mode drivers. By linking these assembly language routines into your BASIC, C, or PASCAL program, you will be able to obtain 2 ms update intervals, even on a standard PC with an 8088 microprocessor.

Examples of how to link the assembly language binary input mode drivers are provided on the support disk. For BASIC, reference the `\BAS_ASM` subdirectory. For C, reference the `\C_ASM` subdirectory. For PASCAL, reference the `\PAS_ASM` subdirectory.

Communicating with the OEMØ23-AT

This section describes how the computer communicates with the OEMØ23-AT.

Read and Write Registers

To operate the OEMØ23-AT with X language commands and sequences, you must be able to communicate with it via the computer's I/O bus. OEMØ23-AT communication involves two pairs of *registers*. A register is a temporary storage area for holding one character (or one eight-bit byte). Data transfer to and from the register occurs one character at a time.

☞ *Helpful Hint:*

Sample *read* and *write* routines that access the computer's I/O bus are on demonstration diskettes that accompany the OEMØ23-AT (routines written in BASIC, C, and PASCAL.)

Register pair #1 resides at the OEMØ23-AT's bases address set with the 8-position DIP switch and consists of the input data buffer (IDB) and the output data buffer (ODB). **Register pair #2** resides at one address location above the OEMØ23-AT's bases address and consists of the control byte (CB) and the status byte (SB). The ODB and the SB are *read-only registers*. The IDB and the CB are *write-only registers*.

Control Byte and Status Byte

X language commands are *strings* of ASCII characters. Sending a command to the OEMØ23-AT requires transferring each character in the string one at a time. Each character transfer requires the sender to notify the receiver that a character is ready and the receiver to notify the sender that the character has been received. This notification process involves setting or clearing control bits (flags) in the 8-bit SB and CB registers.

- Set = high = binary value of 1
- Clear = low = binary value of 0

Control Byte flags allow the host program to signal the OEMØ23-AT with messages such as "A1Ø v1Ø D25ØØØ G". *Status Byte flags* allow the host program to check the OEMØ23-AT's operating conditions (e.g., motor C or axis #2 is moving).

Control Byte (CB) Flags

The following table shows the control byte flags available to the programmer for signaling the OEMØ23-AT. Both this register and the status byte communicate with the OEMØ23-AT.

Bit	Definition
Ø	Binary Input Mode (active high)
1	<i>Unused</i>
2	Stop watchdog timer (active high)
3	Acknowledge interrupt (active high)
4	IDB command character ready (active high)
5	Restart watchdog timer (active low)
6	Reset interrupt output (active low)
7	ODB message character accepted (active high)

Bit 0	When set, indicates that the binary input mode of data streaming is being used.
Bit 2	When set, causes the OEMØ23-AT's watchdog timer to time out and stop. When the timer stops, it forces a hardware reset. The reset condition may be cleared by cycling power or restarting the timer (see Bit 5 below).
Bit 3	When set, tells the OEMØ23-AT that its interrupt signal to the computer has been noted and is no longer needed (see Bit 6 below).
Bit 4	When set, tells the indexer that a command character has been put into the IDB. The OEMØ23-AT then clears Bit 4 of the status byte (SB) to indicate that the IDB is unavailable, reads the character in the IDB, and sets Bit 4 of the SB to indicate to the host that the IDB is ready for a new character.
Bit 5	Restarts the watchdog timer. It must first be cleared, then the timer will start up when the bit is set again. This bit should never be toggled unless the timer has timed out.
Bit 6	Resets the hardware interrupt latch and the interrupt output. The interrupt output cannot be reset if the interrupt is not first acknowledged with Bit 3 above. These bits should be cleared during Reset or Interrupt acknowledge.
Bit 7	When set, tells the OEMØ23-AT that a response character, previously placed in the ODB by the OEMØ23-AT, has been received by the host, and a new character may be placed in the ODB.

Status Byte (SB) Flags

The status byte provides several information flags for the programmer. You can use the status byte to assist in the communications process and to provide run-time status information without the need to burden the indexer with routine status request commands.

Bit	Definition	Power-up State
0	Axis 2 stopped (active high)	Set
1	Axis 1 stopped (active high)	Set
2	Axis 3 stopped (active high)	Set
3	ODB ready (active high)	Cleared
4	IDB ready (active high)	Set
5	Board fail (active high)	Cleared
6	Interrupt active (active high)	Cleared
7	Timed data buffer > 1/2 full (active low)	Cleared

- Bits 0, 1, & 2 Indicate whether the motors for the three axes are moving. At the beginning of any move, the appropriate bit is cleared. Specifically, these bits indicate whether or not the indexer is sending step pulses to the drives. *These bits do not indicate if position maintenance is in effect.* Bits 3 and 4 are set when their corresponding data buffer is ready.
- Bit 3 Is set when the ODB contains an output character for the host, signaling the host to read the information it contains.
- Bit 4 Is set when the IDB is ready (computer may write a character to the IDB).
- Bit 5 When set, tells the PC that the OEM023-AT's watchdog timer has *timed out*, possibly indicating an internal failure from which it cannot recover. The only way to clear this bit is to reset the indexer (*cycle power to the OEM023-AT or send hex 60 to the CB to restore it*). Resetting the OEM023-AT is discussed in the Programming section of this chapter. Running the self-test function will also set this bit. When bit 5 is set, the drive shutdown output goes active, removing motor torque, and generating a drive fault.
- Bit 6 Indicates to the host that a conditional interrupt has been armed and that an interrupt has occurred. If either jumper JU27 (selects interrupt 3) or JU28 (selects interrupt 4) on the indexer board is installed, then the OEM023-AT has generated a hardware interrupt signal.
- Bit 7 When cleared, indicates that the timed data streaming buffer is over half full. This bit is only active when in time-velocity or time-distance streaming mode. At all other times, the bit is cleared.

Communication Process

The following sequence of events occurs when sending a character to the OEM023-AT:

- ① Read the SB until bit 4 is set.
- ② Write a byte (character) into the IDB.
- ③ Write to the CB and set bits 4, 5, and 6.
- ④ Read the SB until bit 4 is cleared.
- ⑤ Write to the CB to clear bit 4 and set bits 5 and 6.

The following sequence of events occurs when receiving a character from the OEM023-AT:

- ① Read the SB until bit 3 is set.
- ② Write a byte (character) from the ODB.
- ③ Write to the CB and set bits 5, 6, and 7.
- ④ Read the SB until bit 3 is cleared.
- ⑤ Write to the CB to clear bit 7 and set bits 5 and 6.

Programming

The OEMØ23-AT includes a support disk containing support routines written in Quick BASIC, C, and PASCAL. ASSEMBLY routines to reset, read, and write to the OEMØ23-AT are also included.

The support disk are divided logically into subdirectories. To go between these subdirectories type: `cd(subdirectory name)`. To back up one subdirectory type: `cd..`

Helpful Hint:

A terminal emulator program is also included with the support code. This program (OEM23TRM.EXE) talks to the OEMØ23-AT directly. *It is a good program to practice making motion with the OEMØ23-AT.*

- The Quick BASIC support routine is contained in the QBASIC subdirectory.
- The C support routine is contained in the C subdirectory.
- The PASCAL support routine is contained in the PASCAL subdirectory.

The ASSEMBLY routines are in the QBAS_ASM, C_ASM, and PAS_ASM subdirectories. Each of these subdirectories show how to link ASSEMBLY code to the appropriate programming language (Quick BASIC, C, or PASCAL).

Support Disk Files

To install the support disk onto your hard drive. Type `INSTALL` at the DOS prompt.

Support Disk File Structure

OEM23

OEM23.H							OEM23TRM.EXE							OEM23TRM.C							SEND23.EXE							SEND23.PAS							OEM23.C							CURSOR.C						
OEM23\C							OEM23\C_ASM							OEM23\PAS_ASM							OEM23\PASCAL							OEM23\QBAS_ASM							OEM23\QBASIC							OEM23\TIMEDIST						
OEM23C.EXE OEM23C.C							OEM23.INC O23ASMC.EXE O23BWRC.OBJ O23RDCF.OBJ O23RSCF.OBJ O23WRCF.OBJ O23WRCF.ASM O23RSCF.ASM O23RDCF.ASM O23BWRC.ASM O23ASMC.C CREATE_C.BAT							O23ASMP.EXE O23ASMP.PAS O23RSPF.OBJ O23BWRP.OBJ O23RDPF.OBJ O23RSPF.ASM O23RDPF.ASM OEM23.INC O23WRPF.ASM O23BWRP.ASM							OEM23P.EXE OEM23P.PAS							OEM23ASM.EXE OEM23.INC OEM23ASM.BAS OEM23QB.LIB OEM23QB.QLB CREATEQB.BAT O23BWRQ.OBJ O23RDQF.OBJ O23RSQF.OBJ O23WRQF.OBJ O23WRQF.ASM O23RSQF.ASM O23RDQF.ASM O23BWRQ.ASM BQLB45.LIB							TERMINAL.BAS TERMINAL.EXE MOVE.EXE MOVE.BAS							OEM23TDP.EXE OEM23TDP.PAS OEM23TDC.EXE OEM23TDC.C IO3.DAT IO2.DAT ONEREV.DAT IO.DAT SQUARE.DAT CIR.DAT						

Designing the Computer Program

Users are responsible for creating computer programs. Writing an interactive program is not difficult for an experienced programmer. If you are not a programmer, it would be prudent to seek advice or assistance from someone familiar with both programming and your computer.

The OEMØ23-AT is designed to operate motor axes in a fashion largely independent of the computer, requiring only a small number of high level commands and interaction. This interaction is almost exclusively in the form of characters and strings rather than numbers. The programmer will need knowledge of string handling in the programming language to be used. Regardless of the intended purpose of the program, it must include subroutines or procedures to do the following (in order of importance).

Helpful Hint:

The programs should allow a non-programmer to change the process being controlled without requiring the program to be rewritten or compiled.

- Reset the indexer
- Send a command string to the indexer
- Receive a character string from the indexer
- Decode Status Request responses
- Allow input of indexer command files

The examples illustrate the recommended approach to communicate with the indexer (*not necessarily optimum for any specific application*).

Sending and Receiving Strings

The algorithms shown in this chapter are required components for any program. Sending command strings of varying lengths is easy because the length of the string is easily known. Any general-purpose programming language will have the string length and string pointer functions needed to put together an iterative algorithm to send a string one character at a time.

Response strings are not always the same length. It is not always possible to predict when a response will occur (some responses are strings representing numbers and some responses are codes represented by strings).

The choice of commands will determine the data that the indexer returns. You must make provisions to interpret the indexer responses to status requests. When the response is a single character (**@**, **A**, **B**, **M**) etc., the meaning of the character is a function of the requesting command.

Helpful Hint:

In general, it is always a good programming practice to precede the status request commands with the appropriate axis specifier.

There is nothing inherent in a response that identifies which decoding process should be applied to the single character of the response. Position report responses are a function of the specific command to the OEM023-AT and can be decimal, hexadecimal, or binary data, with varying zero reference positions. In the default report format, axis #1 can respond in a format different from the other two axes.

Receiving Status Information and Data

Reading status with an INP instruction (see below) is a simple process.

```
BYTE = INP (ADDRESS+1)      QUICK BASIC
BYTE = INP (ADDRESS+1)      MICROSOFT C, BORLAND C
BYTE = PORT (ADDRESS+1)     TURBO PASCAL
```

The numeric variable named **ADDRESS** has previously been set equal to the base address of the OEM023-AT. This instruction sets the numeric variable named **BYTE** equal to the binary number corresponding to the bit pattern of the OEM023-AT's status byte.

This instruction may be executed at any time, regardless the OEM023-AT's status. The resulting variable **BYTE** may serve as a regular number or as a logic value. As a logic value, it can be logically **AND**ed with a *mask* logic value to reset all the bits in **BYTE** other than the status bit of interest.

Testing Individual Status Bits

In the status byte's (SB) definition, Bit 3 of the 8-bit status byte will be **1** if an output character is waiting, **0** if not (bits are numbered 0 - 7). To test this bit, set all other bits to zero by **AND**ing the byte with a mask and run a logical test on the result for zero (false). Bit 3 has a binary weight of 8.

Logical Masking Example

Suppose the OEM023-AT returned a status byte value of 89, Bit 3 is high.

	Dec	Hex	Binary
BYTE =	89 =	59 =	01011001 \
			> logic AND = 00001000
Mask =	8 =	08 =	00001000 / (TRUE)

If the OEM023-AT returns the value 70, a check on Bit 3 reveals that it is low.

	Dec	Hex	Binary
BYTE =	70 =	46 =	01000110 \
			> logic AND = 00000000
Mask =	8 =	08 =	00001000 / (FALSE)

Helpful Hint:

This example will *trap* the computer if the OEM023-AT has not received a status command.

In Quick BASIC, this check might appear as follows:

```
BYTE = INP ( ADDRESS+1 )
IF ( BYTE AND 8 ) > 0 THEN GOTO 3020 ELSE GOTO 3000
CHAR = INP ( ADDRESS )
CHAR$ = CHR$ ( CHAR )
ANSWER$ = ANSWER$ + CHAR$
etc.
```

This instruction sends the program back to read the OEM023-AT status until a character is ready. Then the character may be read.

Sending Control Information and Data

To write bytes to the OEMØ23-AT, use the following instructions.

OUT ADDRESS, ALPHA	BASIC
OUTPORTB(ADDRESS, ALPHA);	TURBO C
OUTP(ADDRESS, ALPHA);	MICROSOFT C
PORT[ADDRESS]: = ALPHA;	PASCAL

Resetting the OEMØ23-AT

The following are step-by-step procedures for writing your own subroutines for resetting the OEMØ23-AT:

- ① Write 64 Hex to the Control Port (Board Address +1).
- ② Read the Status Port (Board Address +1) until the status byte and 20 Hex > 0.
- ③ Write 40 Hex to the Control Byte (Board Address +1).
- ④ Write 60 Hex to the Control Byte (Board Address +1).
- ⑤ Read the Status Port (Board Address +1) until the status byte and 7F Hex = 17 Hex.
- ⑥ Write 20 Hex to the Control Port (Board Address +1).
- ⑦ Write 60 Hex to the Control Port (Board Address +1).

Reading Characters From the OEMØ23-AT

Use the following procedure to read characters from the OEMØ23-AT:

- ① Initialize the ASCII variable to null (Ø).
- ② Read the Status Port (Board Address +1) until Status Byte and 8 Hex >0.
- ③ Read the Data Port (Board Address) in to the ASCII variable.
- ④ Write EØ Hex to the Control Port (Board Address +1).
- ⑤ Read the Status Port (Board Address +1) until the status byte and 8 Hex = 0.
- ⑥ Write 60 Hex to the Control Port (Board Address +1).

Writing Characters to a OEMØ23-AT

Use the following procedure to write characters to the OEMØ23-AT:

- ① Convert the character to ASCII. This may not be necessary in some programming languages such as C (except for axes in Binary Input Data Streaming mode).
- ② Read the Status Port (Board Address +1) until the Status Byte AND 10 Hex > 0.
- ③ Write the ASCII character to the Data Port (Board Address+1).
- ④ Write 70 Hex to the Control Port (Board Address +1).
- ⑤ Read the Status Port (Board Address +1) until the Status Byte AND 10 Hex = 0.
- ⑥ Write 60 Hex to the Control Byte (Board Address +1).

Program Examples

The support disk files below provide read, write, and reset routines.

- MOVE.BAS (written in Quick BASIC)
- OEM23P.PAS (written in PASCAL)
- OEM23C.C (written in C)

These files provide the foundation from which you can design a motion control program for the OEMØ23-AT. The following are examples of how to use these files to create your own OEMØ23-AT program.

BASIC Program Example

Use the following steps as a guide to develop your custom OEMØ23-AT BASIC program.

- ① Make a copy of MOVE.BAS.
- ② Edit the CMD\$ code to fit your application's needs.
- ③ The response is returned in ANSWER\$.

```
CMD$ = "A1 V2 D25ØØØ G123 1PR 2PR 3PR"
SEND CMDSTRING (CMD$)
PRINT CMD$
PRINT
FOR I = 1 TO 3
PRINT "POSITION OF AXIS ";
DO
    BYTE = INP (ADDRESS + 1)
    LOOP WHILE (BYTE AND ALDONE) <> ALDONE
    DO
    ANSWER$ = GET RESPONSE$
    LOOP UNTIL ANSWER$ <> ""
PRINT ANSWER$
NEXT I
```

- ⑤ Compile and run the program.

C Program Example

Use the following steps as a guide to develop your custom OEM023-AT C program.

- ① Make a copy of OEM23C.C.
- ② Edit the main program of the copied file.
- ③ To send a command, place the X language in message (message = A10 V10 D25000 G). Then make a call to procedure writecmd (message).
- ④ To get a response from the OEM023-AT, use procedure readanswer (answer). The response is returned in answer.

⑤ Example:

```
main() {
    char *message,*answer;
    answer="";
    initialize();           /*resets OEM023-AT*/
    message = " 1PR ";
    writecmd (message);
    readanswer (answer);
    printf (answer);
} /* end of main*/
```

- ⑥ Compile and run the program.

PASCAL Program Example

Use the following steps as a guide to develop your custom OEM023-AT PASCAL program.

- ① Make a copy of OEM23P.PAS.
- ② Edit the main program of the copied file.
- ③ To send a command, place the X language commands in message (cmd = A10 V10 D25000 G). Then make a call to procedure writecmd (768,cmd) where 768 is the OEM023-AT board address.
- ④ To get a response from the OEM023-AT, use procedure readanswer (768,answer). The response is returned in answer.

⑤ Example:

```
begin
    answer:='';
    Initialize (768);
    cmd:=' A10 V10 D25000 G PR ';
    Writecmd (768,cmd);
    while answer='' do Readanswer (768,answer);
    writeln (answer);
End
```

- ⑥ Compile and run the program.

Special Modes of Operation

This section discusses special modes of operation for the OEMØ23-AT. These special modes include the following:

- Using multiple OEMØ23-ATs with one computer
- X-Y linear interpolation
- Using interrupts

Multiple OEMØ23-AT Addressing

Multiple OEMØ23-ATs can be synchronized by using one master clock. By making the connections shown below and using the `MSL` command to specify the master/slave relationship of the different axes, any number of OEMØ23-ATs can be made synchronous.

Master OEMØ23-AT Output


- RMCLK+ (2) on J1
- RMCLK- (4) on J1

Slave OEMØ23-AT Input

- XMRMCLK+ (32) on J3
- XMRMCLK- (34) on J3

Connections

<u>Master</u>	<u>Slave #1</u>	<u>Slave #2</u>
RMCLK+ (2) on J1	→ XMRMCLK+ (32) on J3	→ XMRMCLK+ (32) on J3...
RMCLK- (4) on J1	→ XMRMCLK- (34) on J3	→ XMRMCLK- (34) on J3...

 *Helpful Hint:*
The table below lists parts and tools needed to make a synchronization cable.

20-22 AWG Wire
<i>AMP Connectors</i>
Tools:
Manual tool for double row connector
Conversion kit for single row connector
Single row 4 position (cover)
Single row 4 position (housing)
Double row 8 position (kit)

X-Y Linear Interpolation

To move multiple orthogonal linear axes in a straight line, you must make all the axes start, finish accelerating, start decelerating, and stop, in a synchronized fashion.

The simplest case involves producing a 45° angle line of movement with two axes. Both axes (X and Y) are given the same velocity, acceleration, and distance parameters (*to produce other angles, these three parameters must be proportionally scaled*).

Typically, the task is to derive appropriate move parameters to get from the current location to a new location, where each position is specified by a set of *Cartesian coordinates*. Linear acceleration and velocity are specified.

In the following example, the incremental distance parameter for each axis is the difference between the target position coordinate and the current position coordinate for that axis. The ratio of incremental distance for one axis to that of the next establishes the ratio of the respective accelerations and velocities. Linear acceleration and velocity is the *vector sum* of these components. The *Pythagorean theorem* provides the formula for calculating the velocity.

Example

The two-axis X-Y positioning system must move from its current position to a new position at a linear speed of 1 inch/second ($V_L = 1$). A motor velocity of 4 rps translates to 1 inch per second (ips) on both axes. Acceleration on any axis must not exceed 100 rps².

Helpful Hint:
The following procedure is used to determine the move parameters.

Current Position: $X_0 = -60,000$ **Target Position:** $X_1 = 180,000$
 $Y_0 = 200,000$ $Y_1 = 20,000$

① **Determine incremental distances (D) and the ratio:**
 X axis distance = $X_1 - X_0 = +240,000 = D+240000$
 Y axis distance = $Y_1 - Y_0 = -180,000 = D-180000$
 Ratio = $Y/X = -0.75$

② **Determine the velocity (v) settings:**
 Velocity ratio = $V_Y = 0.75(V_X)$
 Pythagoras = $V_L^2 = V_X^2 + V_Y^2$
 Substitution = $V_L^2 = V_X^2 + (0.75(V_X))^2 = 1.56(V_X^2)$
 $V_X^2 = V_L^2/1.56 = 0.8006\text{ips} = 3.20\text{rps} = \mathbf{V3.20}$
 $V_Y = 0.75(V_X) = 0.6004\text{ips} = 2.40\text{rps} = \mathbf{V2.40}$

③ **Determine the acceleration (A) settings:**
 Acceleration ratio = $A_Y = 0.75(A_X)$
 $A_X = 100 \text{ rps}^2 = \mathbf{A100}$
 $A_Y = 75 \text{ rps}^2 = \mathbf{A75}$

④ **Enter the commands derived from steps ① through ③ above:**

Command	Description
1A100	Set acceleration on axis 1 to 100 rps ²
1V3.024	Set velocity on axis 1 to 3.024 rps
1D240000	Set distance on axis 1 to 240,000 steps CW
2A75	Set acceleration on axis 2 to 75 rps ²
2V2.268	Set velocity on axis 2 to 2.268 rps
2D-180000	Set distance on axis 2 to 180,000 steps CCW

When polar coordinates are used, the distance to the endpoint (radius) and the angle from the X axis are known. In this case, the distance (D) for the X axis is equal to the radius multiplied by the cosine of the angle. The distance for the Y axis is equal to the radius multiplied by the sine of the angle. The X and Y velocity and acceleration parameters may be derived from linear parameters in the same way.

Using Interrupts

The OEM023-AT has one interrupt signal available to be activated upon specific conditions. This interrupt signal can be directed at either of two interrupt request lines, IRQ3 or IRQ4. Jumper 27/28, located just above the edge connector on the OEM023-AT indexer card, determines which interrupt request line is activated. Installing a jumper on the JU27 pins selects IRQ3, and installing a jumper on the JU28 pins selects IRQ4.

You, the user, are responsible to enable the 8259 interrupt controller chip in the PC to accept the interrupt request. You must also set the corresponding interrupt vectors to accommodate the interrupt request.

8259 Interrupt Controller Chip

Use the following procedure to enable the 8259 interrupt controller chip:

- ① Read in the current interrupt mask register (IMR) at the I/O address (21 HEX).
- ② Clear the proper bit to enable interrupt hardware (IRQ3 = bit #3, IRQ4 = bit #4).
- ③ Write the IMR back to the 8259 chip.

```
unsigned char int_8259;           /* Byte to read IMR into */
int_8259 = inportb(0x21);        /* Read IMR for current settings */
int_8259 = int_8259 & 0xF7;     /* Clear bit 3 */
outportb(0x21, int_8259);       /* Write new IMR */
```


Interrupt Vector

The IRQ3 interrupt corresponds to the 11th interrupt vector. Since each vector is 4 bytes long, the user must modify the 4 bytes located at memory address 0000:002C to point to the interrupt service routine. For IRQ4, modify the 4 bytes at memory address 0000:0030.

Usually, IRQ3 and IRQ4 are assigned to the computer's serial ports, COM 2 and COM 1 respectively. Therefore, when making changes to these vectors. *Avoid conflicts with communication cards that may require these vectors.*

The following is an example of changing the interrupt vector for IRQ3:

```
VOID INTERRUPT(*OLDVECT) ();           /* VARIABLE TO HOLD ORIGINAL INTERRUPT VECTOR */
VOID INTERRUPT_IRQ3_SERV(VOID);        /* FUNCTION PROTOTYPE FOR INTERRUPT ROUTINE */
OLDVECT = GETVECT(0XB);                /* GET ORIGINAL VECTOR AND STORE */
SETVECT (0XB, IRQ3_SERV);              /* SET VECTOR TO POINT TO INTERRUPT ROUTINE */
SETVECT (0XB, OLDVECT);                /* BEFORE EXITING PROGRAM, RESET ORIGINAL VECTOR */
```

Interrupt Commands

You can activate the OEM023-AT's interrupt output based on several conditions for each axis. The QS command allows you to enable or disable the *interrupt-on-condition* functions. No hardware interrupt will be generated until a specific QS command has been issued and the corresponding condition is met. The possible interrupt-on-condition functions are listed below.


- Interrupt on Trigger #1 high (QSA command)
- Interrupt on Move Complete (QSB command)
- Interrupt on Encountered Limit Switch (QSD command)
- Interrupt on Ready to Respond (QSE command)
- Interrupt on Command Buffer Full (QSG command)
- Interrupt on Motor Stall (QSH command)

The QR command reports which interrupt-on-condition functions have been selected with the QSA - QSH commands. The QI command reports which interruptible conditions are active. *The QI command always shows a response to in position (QSB1), even if the interrupt is not enabled (QSB0).*

You can take advantage of the OEM023-AT's interrupt capabilities without actually generating any host interrupts. This is done by arming whatever interrupt conditions are desired for each axis, and then polling the status byte (SB) to see if the *Interrupt Active* bit (bit #6) is set. Then, each axis must be polled with the QI command to determine the source of the signal.

Clearing the Interrupt Signal

Once an interrupt signal is generated, two steps are required to remove that signal. First, the hardware device (*latch*) that holds the signal must be cleared (toggle the *Reset Interrupt Output* bit (bit #6) in the control byte (CB). Second, the host must acknowledge the interrupt. This calls for toggling the OEM023-AT's Interrupt Acknowledge bit (bit #3) in the CB.

 **Helpful Hint:**
The following is example code used to clear an interrupt.

```
#INCLUDE <CONIO.H>
MAIN()
INT PC21ADDR = 0X3000;
OUTP(PC21ADDR+1, 0X68);
WHILE( INPORTB(PC21ADDR+1) & 0X40);
OUTP(0X20, 0X20);
}
```

*When your interrupt service routine is complete, **remember** to send an EOI (end-of-interrupt) signal to the 8259 chip, as shown below:*

```
outportb(0x20, 0x20);           /* clear interrupt from 8259 chip */
```