Chapter 7

Stepper Motion

Features

What is It All About?
How Do Stepper Motors Work?
Stepper Motor Accuracy and Induced Errors
Types of Stepper Motor Translators
Control Methods - ASIC or Processor
Stepper Pulse and Direction Algorithm
Stepper Phase Control Algorithm

What is It All About?

I've noticed that in most motion control write-ups, servo systems seem to be favored over stepper systems. Steppers appear to gain attention only through major breakthroughs. Micro-stepping is a clear example. A possible reason the servo system appears more popular in industry than the stepper could be because servos are relatively easy to analyze, are very forgiving, and there are general misunderstandings concerning the capabilities of stepper systems.

A servo system requires at least a position loop, some form of gain control, and phasing. As a result, servos require some form of gain selection, be it lead, lag, PID, PD, or other gain style. In addition, servos need to be operated via the update period, whether they are in motion or not. I believe that stepper motion operating in *open-loop* mode is far more complex to control than servo motion. Therefore, steppers require more analysis, and the end-result is that servos wind up in many applications otherwise ideally suited for steppers.

Load inflections, acceleration rates, top velocity, high velocity torque reduction, zero following error and other factors all contribute to the successful application of a stepper system. Most designers know relatively little about them, although a functionally comparable stepper system generally costs less than one based on a servo.

Successful stepper motor control requires at least a basic understanding of stepper motor internal functions, various stepper motor amplifiers or translators, control methods, and the system in which the stepper is to be incorporated. The objective of this chapter is to cover these items in as simple a manner as possible (all but the system the stepper is to be attached to . . . see Chapter 6). I will attempt to focus on piecing together the stepper puzzle. If you better understand what stepper is, and how it works, it will be much easier to incorporate into your design.

How Do Stepper Motors Work?

The Stepper Motor Principle

This section will describe how the stepper motor works by explaining the basic principle underlying its operation.

The first rudiment to understand is the difference between a DC (servo) motor and a stepper motor. When a voltage is applied to a DC servo motor, it will develop **both** torque, and rotation. When a voltage is applied to a stepper motor, it will develop **only** torque. This basic difference is the fact that a DC servo motor is designed to *self-commutate*.

Commutation . . . The principle by which the amplitude and/or direction of the current flowing in one or more of the electromagnetic coils within a motor is altered. Thus, the attracting and repelling magnetic force fields existing between the motor's stator and armature are altered, and rotational torque is developed.

However, the stepper is designed with no internal commutation capability, and until it is externally commutated (phase switched) it will only develop static torque.

As with all DC motors, the stepper motor operates within the basic principle of magnetic attraction, and repulsion. Most individuals at some point in their lives have observed this magnetic property by playing with magnets to make them either push away from each other or attract one another. The following is a simple illustration showing stepper motor operation using this idea:

1) Set up two magnets to attract each other. (call them #1, and #2).
2) Position them just far enough apart so that they do not physically move toward each other.
3) Place a third magnet (#3) of the same field strength as #1 next to magnet #1 in the identical orientation. You may need to force them together.

Note that magnet #2 will now move towards magnets #1 and #3. The magnetic field of magnet #1 is now strengthened by that of magnet #3. This is known as *magnetic field addition*. In turn, if magnet #3 is turned around and its magnetic field is made stronger, it will cancel the field strength of magnet #1 and repel magnet #2. This would be *magnetic field subtraction*.

The stepper motor operates by this principle, but it uses an electromagnet in place of magnet #3. It is constructed with a series of magnets on both the stator and the armature.

As you can see in Figure 7.1, the magnets will come to rest in some natural magnetic orientation called a *detent position*. If you turn the shaft of the stepper motor by hand, you will feel the magnets try to maintain this detent position. However, when you apply enough rotational force to the shaft to overcome the magnetic attraction, you will feel the armature *jump* to the next detent position. Figure 7.1 shows a stator with eight magnets and an armature with six magnets, which yields a total of 24 detent positions. As the armature is turned counterclockwise from the position in Figure 7.1A to the position in Figure 7.1B, then to positions 7.1C and 7.1D, different magnet sets will exhibit a stronger attraction and will pull the armature into new detent positions. In this example, there are three *full-step* positions between each of the stator magnets.

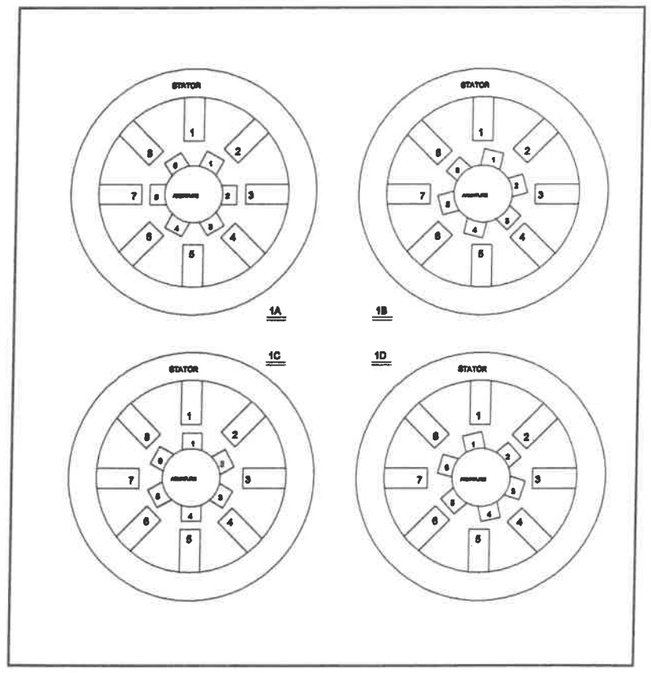


Figure 7.1 Stepper motor pole interaction.

To provide some manner of controlling the armature position, the principle of electro-magnetics is used to generate the rotational force required within the stepper motor. A series of coils surrounds each of the stator magnet poles, and by having current flow through these coils with certain amplitudes and in given directions, the magnetic attraction and repulsion of the magnetic fields will cause the armature to rotate to new positions. By continuously changing the coil currents, and/or directions, the result is continuous rotation will result.

Figure 7.2 shows how rotational motion can be achieved in the stepper motor by combining static magnetic, and dynamic electromagnetic forces.

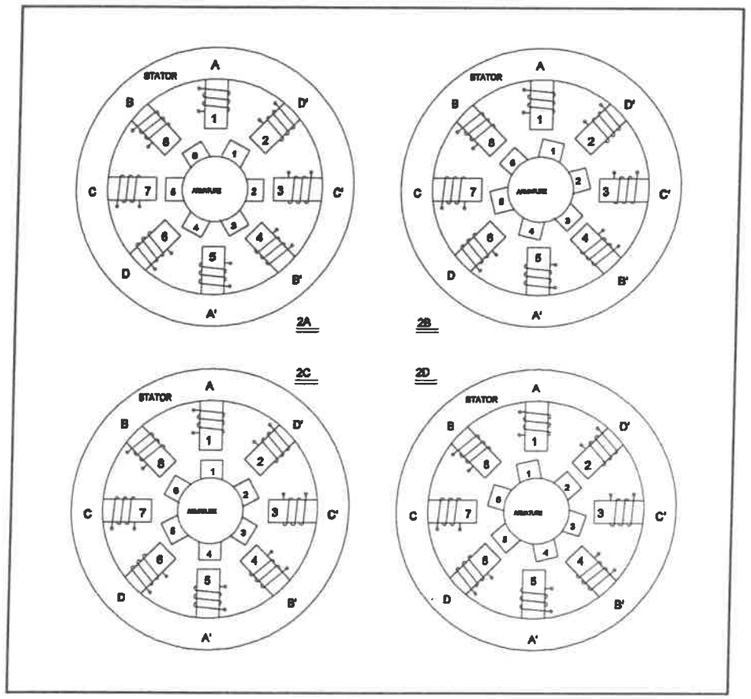


Figure 7.2 Stepper motor electromagnetic pole interaction.

In Figure 7.2A, the motor is oriented with armature magnets 2 and 5 aligned with stator magnets 3 and 7. If a current is passed through electromagnetic coils C and C', the motor will exhibit only a *holding* torque. If the current is then removed from coils C and C' and passed through coils B and B', the armature will then rotate and align armature magnets 6 and 3 with stator magnets 8 and 4, respectively (Figure 7.2B). The effect of altering, or removing existing coil currents is the external *commutation* required by the stepper motor to produce rotation. If the current commutation is halted, the stepper motor will stop rotating (stepping). At this point, an alignment of the armature and stator magnets will remain in the existing orientation, depending on which coils were carrying current and the current values and directions within those coils.

The typical stepper motor contains 200 orientation positions in *full-step* mode (50 stator poles) yielding a resolution of 1.8 degrees-per-step. Operation of the stepper motor between these full-step positions (called half or microstep operation) is accomplished by the stepper motor amplifier (translator).

However, you must remember that since the stepper functions in electromagnetically generated fields, it suffers all of the losses associated with any motor, including viscous damping, hysterisis, eddy currents, back-EMF, resistive and inductive losses, and so on. Many of these losses are only evident when the stepper is in motion, since it takes either varying current or rotational motion to generate them (eddy currents are an example). Windage is a rotational loss more commonly referred to within a combined group term known as *Viscus Damping Torque*.

Basic Stepper Motor Calculations

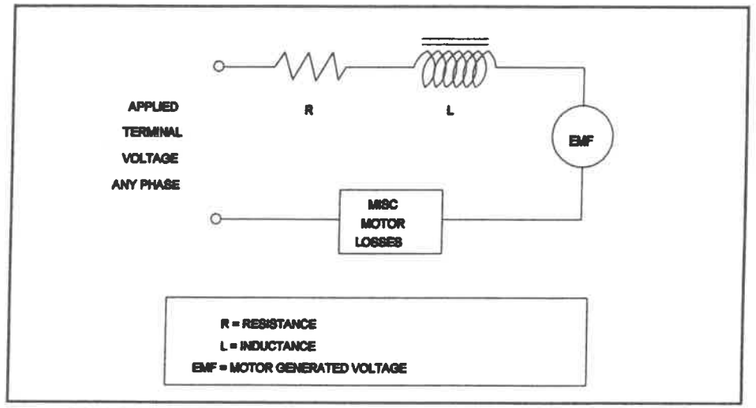


Figure 7.3 Basic motor equivalent circuit.

In writing, I try to avoid math whenever possible, but I must address two properties inherent in stepper motors here. They are winding **Resistance**, and **Inductance**. You need to understand these two properties and their effect on the response of the system.

Figure 7.3 shows a simple equivalent circuit for a stepper motor.

The DC static current in the stepper motor is limited only by coil resistance, and behaves according to Ohms law:

$$E = IR$$

. . .where, **E** is given in volts, **I** is current in amperes and **R** represents the DC resistance in ohms.

For a given motor current, the higher the coil resistance of the motor windings, the lower the winding current is for a given voltage.

The electromagnetic coils inside the stepper motor provide not only motor rotation (via magnetic field alteration), but also provide the motor with *inductance*. To define *inductance* would require a

research paper in itself, and there are many good resources. But suffice to say that ***inductance*** is a property of a magnetic field generator that resists a *change* in current flow. In other words, if current is flowing in a coil of wire, and the voltage that is producing that current is removed, the inductance property of the coil will try to maintain that current level by building a self-generated EMF force (this phenomena can be observed by an arc being produced when a relay contact removes power from a motor or transformer). What will be discussed here is the effect of motor inductance on motor performance.

Current in a motor winding is given by the following relationship:

$$I_{coil} = \frac{E}{R}(1 - e^{-\frac{Rt}{L}})$$

where:

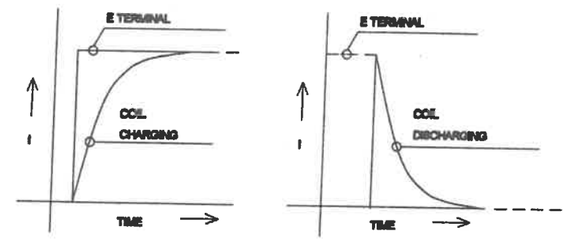
I	=	coil current (amps)
E	=	terminal voltage (volts)
R	=	coil resistance (ohms)
L	=	coil inductance (henrys)
t	=	elapsed time (seconds)
e	=	2.718281828 (natural number)

This equation relates the instantaneous current (**I**) through the coil as a function of time (**t**). The ratio of wire resistance to coil inductance determines the time-rate-of-change of current for a given applied voltage.

The static DC current is limited by (E/R). As the coil DC resistance increases, the static DC motor current decreases (with constant E). As the coil resistance increases due to a rise in wire temperature, the **time** that it will take to reach the limiting static DC motor current decreases. Although, note here that this reduction in current is not by design, and therefore, the motor power will be reduced as well. The coil inductance, on the other hand, only works on the **time** portion of the equation. Therefore, as the inductance goes up, so does the **time** it takes to reach the static DC current value, and vice-versa. Refer to Figure 7.4.

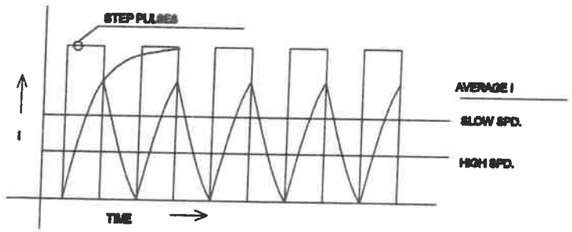
The time element comes into play during the motor coil switching operation (external commutation). As a constant voltage value is switched on and off to various coil pairs, the current in any given motor coil must *charge* up to the static DC current value, or *discharge* to zero. The time period to do this is determined by the motor resistance and inductance (approximately five R/L time periods). This calculated time period is **fixed** since the motor coil resistance and inductance do not change after the motor is built, in spite of wire temperature, and so on.

As the stepping frequency increases, coil charging time decreases; therefore, average coil current decreases. Refer to Figure 7.4C. Since motor torque is a product of current, the motor torque generated will also decrease.



4A

4B



4C

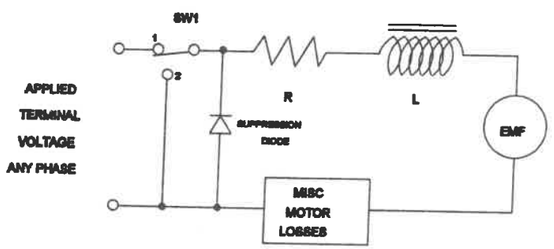


Figure 7.4 Stepper motor electric response characteristics.

To increase the stepping rate ability of a stepper motor, the applied terminal voltage must be increased. To prevent current overload, the motor current must be monitored. This becomes the task of the stepper motor amplifier (Translator). To produce high-speed/high-torque stepper operation, the translator is used to vary the motor terminal voltage, switching waveform, coil current(s), current direction, and/or other factors.

Stepper Motor Accuracy and Induced Errors

A characteristic of a servo motor is that sudden changes in motor loading will not cause adverse consequences in motor operation. The motor will perhaps slow down or speed up, but it will not stop (if the load is still within the torque range of the motor). A stepper motor, on the other hand, operates best with smooth transitions (no drastic discontinuities). The stepper is a *zero following error* device. This means that the acceleration, deceleration, slew rate (running velocity), loading, and other operating conditions, must all remain within the stepper's immediate capability, or it will *stall*. A sudden change in the slew rate can cause it to stop, so can too high an acceleration rate.

However, a stepper system offers attractive benefits such as simplicity, low cost, ease of operation, and open loop capability; but it comes with a price. That price is a lengthy list of considerations that can affect its accuracy, stability, and overall operation.

The following specifics will show that when you are ready to incorporate a stepper system, you must not to take it for granted. The one thing a stepper does great is **STALL**.

Stepper Accuracy

This is typically within *3 to 5 percent of one Full Step* at no load, or ±0.054 angular degrees for a 1.8 degree-per-step (200 step per revolution) 3 percent unit. Check the stepper motor specification for the accuracy of the motor you are using.

Due to mechanical placement of the stator and armature magnets and placement of the stator electromagnetic coils, stepper accuracy is not quite as perfect as one might think. However, the repeatability of a given step position in a given rotation is much tighter, generally five arc-seconds or ±0.00139 angular degrees. The effect of accuracy must be taken into account when working out your system resolution. If, for example, you require system resolution to be 0.001 ±0.0 inch, and a full step is 0.002 inch, a half step would be 0.001 ±0.00006 inches (3% x 0.002) or a 6 percent error. This may not seem much, but it is outside the specification.

As another example, what if you were micro-stepping, and had to achieve a resolution of 0.00003125 ±1 step (64 steps-per-step = 0.002/64 = 0.00003125 inch, the repeatability being ±0.0000015 inch). You could possibly accomplish this move due to the repeatability of the stepper, but definitely not because of its accuracy. You would have to construct a stepper motor *Step Angle Compensation Table* to precisely position the motor shaft similar to a Leadscrew Error Compensation Table used in high grade CNC controls. The other alternative would be to use encoder feedback.

Another point about the 3 percent accuracy rule (5% in some units), is that although the step angle is *guaranteed* not to exceed this, the actual step position within the full step can change due to motor loading, *two-phase-on* operation, *one-phase-on* operation, as well as by phase coil-current matching, and other factors explained in the following section.

Stepper Errors

- **Phase errors** – are due to non-linearity of the phase waveforms, amplitudes, or amplifier current generated when developing the stepping power signals. *Quadrature*, *One-Phase-On*, and *Two-Phases-On* are terms that describe specific phase errors due to either physical coil positions, and/or current imbalance through these coils.

- **Bias errors** – due to offsetting DC current levels within a phase coil. These offset currents will produce a variance in coil torques offsetting the actual armature position from the desired position.

- **Hysterisis errors** – will generally show a deadband area in which no motion will occur despite microsteps generated. This could be due to unbalance of stepper coil currents, voltage waveforms, or the mechanics. A stepper should always move one step per Full or Half step generated.

- **System Mechanics** – including the Load will tend to increase or decrease the friction force. It will also affect the inertial loading and offsets in system balance. As these changes occur, stopping positions will physically be disrupted in either direction, causing a motion deadband. In other words, the controller will step, but nothing will move. The accumulated effect of this problem will cause major positioning errors especially those in microstep applications.

Reducing Errors

- Buy more accurate motors when necessary. Check the error patterns of similar motors to verify consistency.

- Use the optimum motor current based on the manufacturer's charts. Use all of the coils in their optimum configuration to derive the best torque for a given power dissipation.

- Generally, a motor with low detent and harmonic torque proves to be the best choice. Definitely, this is the best choice for microstepping system applications.

- If positioning is paramount to the application, use a form of position feedback such as an encoder or a scale to insure the required accuracy.

- Buy a stepper and motor amp as a matched pair—in which the motor is "mapped" and the amplifier automatically compensates for motor errors.

Stepper Annoyances

- **Velocity Ripple**

When a motion step is made, the stepper system inertia will try to oscillate or ring about the new position point. This can be attributed to the fact that the motor's magnetic field(s) will not allow the two mechanical masses (stator and armature) to come to rest in a perfect trajectory. There will always be some overshoot since the magnetic fields are moving laterally to each other, and there is no position feedback to tell the control to correct for this action. The larger the step size, the more torque is required to make the step; and, generally, the larger the mechanical oscillation will be. Even at velocity, the velocity profile can have this variance to it, called *velocity ripple*. To minimize this oscillation, reduce the step size (half step or microstep), reduce the step current required to make the move, or mechanically dampen the system. Figure 7.5 shows an oscillation occurring in the *physical* positioning of step 1, when the step 1 impulse is issued. It is important to note, that the *physical movement* of the stepper armature *is reversed* (refer to point A) to the original armature motion direction. The armature is actually oscillating back and forth. The degree of movement is not necessarily great, but if a step impulse is applied to a stepper motor at the point of reversal (Point A), the motor can actually begin stepping continuously in the reverse direction even though it was commanded to operate in the forward direction!

- **Step Ringing**

The coils that create motion will form a tuned circuit with the internal coil capacitance in addition to any other capacitance in the circuitry. This will create ringing on the control step waveforms. Damping circuitry is required to either reduce or eliminate this condition. Severe step ringing can cause interference with other electronic equipment including the controlling computer. This can have dangerous side effects.

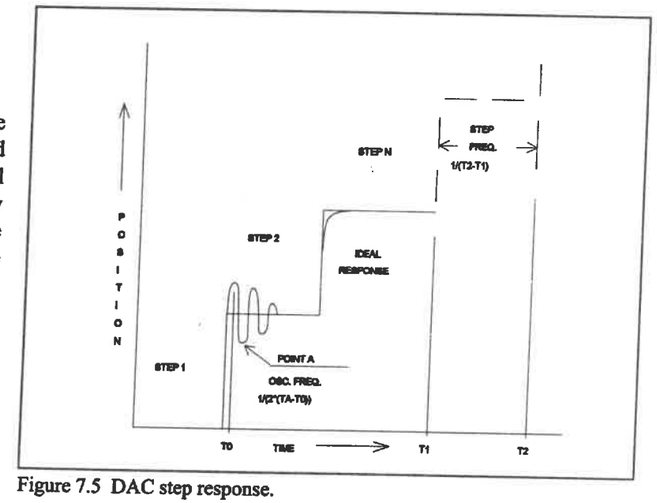


Figure 7.5 DAC step response.

- **Resonance in certain step frequency ranges**

There can be one or more natural resonant frequency harmonics associated with any stepper motor. When operating at these frequencies, a heavy oscillation will be heard and/or felt from the stepper motor. This can have detrimental consequences on the motor torque. To eliminate this problem, either refrain from running at these frequencies, use reduced stepping techniques and/or low current stepping in these regions.

- **Possible step misses without stall**

Since the motor and load system exhibit inertia, it is possible for the motor and control step generator to fall out of and then back into synchronization. If this happens, the position of the system will be corrupted. To correct this, reduce the ramping and/or velocity rates. If this reduction is done in steps, then it will be possible to find the problem area.

Never allow this error to occur since stalling is just around the corner.

Types of Stepper Motor Translators (Drives)

Motion Definitions:

Full Step – This term is used to denote a stepper motor operating in a detent-to-detent stepping mode. If power is removed, the stepper motor will not advance or retract from its current position (no load, horizontal attitude) since it is presently located at a detent position (see Section on stepper motor principles) any time a step is made. A 200 Full Step motor will rotate at 1.8 degrees-per-step.

Half Step – This term is used to denote a stepper motor operating in a mode such that its magnetic fields allow the armature to be physically placed half way between actual detent positions. If power is removed, the stepper motor will advance, or retract from its current half step position to a true physical detent position—if it is not presently located at one (no load, horizontal attitude). A 200 Full Step motor will produce 400 Half Steps at 0.9 degrees-per-step.

Microstep – This term is used to denote a stepper motor operating in a mode such that its magnetic fields allow the armature to be physically positioned *anywhere* between the actual detent positions. If power is removed, the stepper motor will advance or retract from its current position to a true detent position—if it is not presently located at one (no load, horizontal attitude). Typically, a 200 Full Step motor can subdivide a 1.8 degree step into specified even or odd divisions you set up. This can give a direct resolution of up to 128 microsteps per full step without the need for gearing.

Current Flow Definitions:

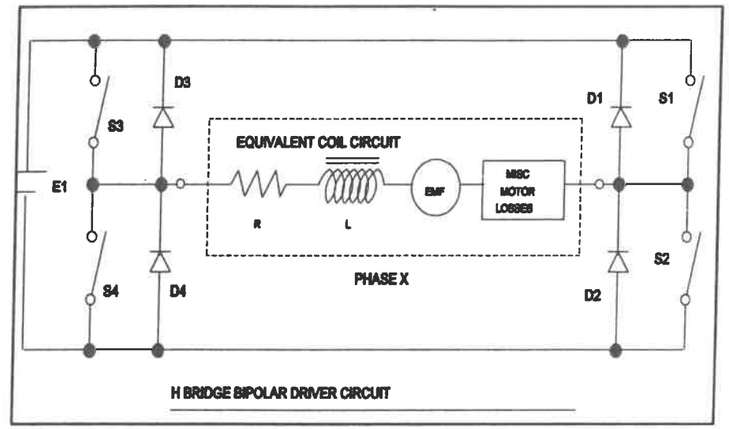


Figure 7.6 H bridge bipolar driver circuit.

BiPolar – Bipolar operation simply means that coil current in any given coil is allowed to flow in either direction. The coils are constructed so that by sequencing the coil current, armature movement will be generated. Figure 7.6 illustrates an H bridge bipolar driver. When S1 and S4 are closed, S2 and S3 are open and vice-versa. The drive allows current to flow through the phase winding in both directions. Figures 7.7 and 7.8 are examples of two other styles of bipolar drives.

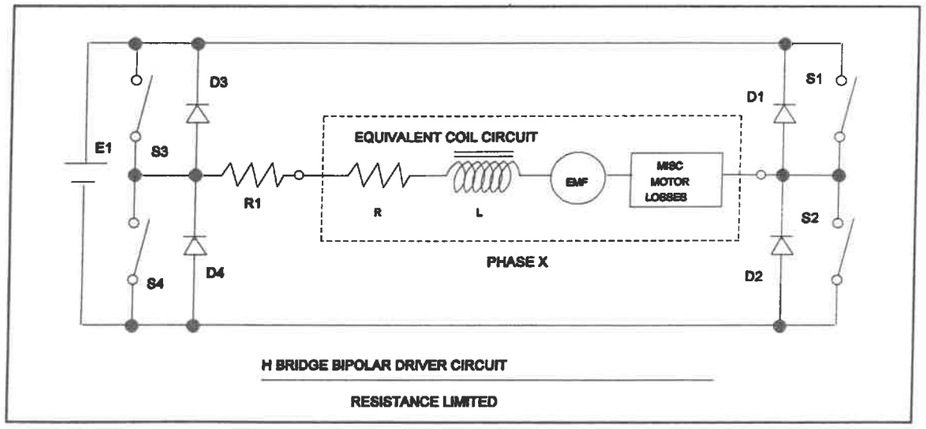


Figure 7.7 H bridge bipolar driver circuit—resistance limited.

In Figure 7.8 (BiLevel driver) switch S5 is closed at the beginning of a step, allowing the higher E2 voltage to drive the motor. S5 is then opened after a prescribed period of time or when a current monitor indicates that the coil current is at the required value. Opening S5 then places the lower voltage E1 in control of motor current.

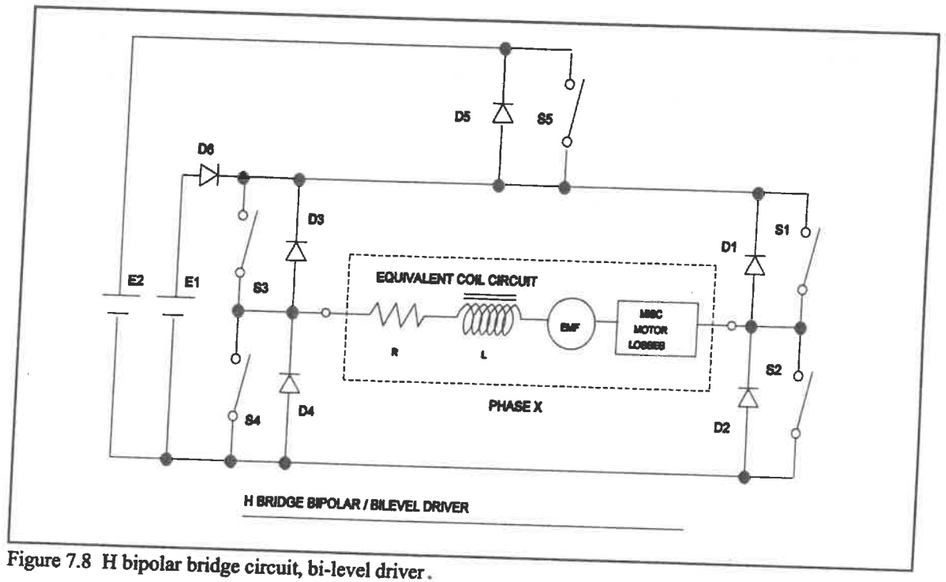


Figure 7.8 H bipolar bridge circuit, bi-level driver.

UniPolar — Unipolar operation simply means that coil current in any given coil will only be allowed to flow in one direction. The coils are constructed so that by sequencing the coil current, armature movement will be generated.

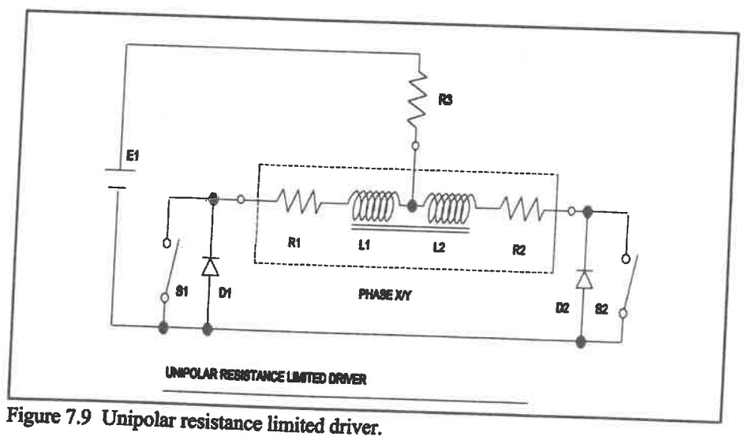


Figure 7.9 Unipolar resistance limited driver.

Figures 7.9 and 7.10 show examples of Unipolar drivers. The phase currents are allowed to flow in one direction only. In Figure 7.9 (BiLevel driver) switch S3 is closed at the beginning of a step, allowing the higher E2 voltage to drive the motor. S3 is then opened after prescribed period of time or when a current monitor indicates that the coil current is at the required value. Opening S3 then places the lower voltage E1 in control of generating motor current.

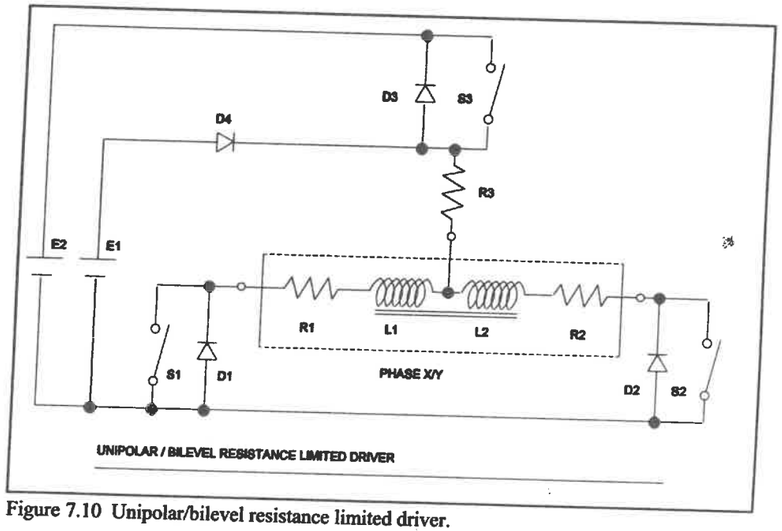


Figure 7.10 Unipolar/bilevel resistance limited driver.

Translator Types

The following section lists the major classifications of stepper motor drivers. However, this list does not mean to imply that these are the only types available. It is most important to understand the differences between each type so that you can select the proper type for your application. The differences I speak of not only relate to how the translators operate, but also to how the stepper motor will react when using a specific translator in your system. For example, a chopper drive always causes the stepper motor to vibrate slightly when **not** commanded to move. In the film handling industry, a stepper motor that is directly coupled to a film drive roller will cause the film to vibrate as well. This would be unacceptable when positioning film for developing or splicing. A BiLevel, current limited (non-chopper) type drive would be a better fit.

If you limit current using an external resistor, you enjoy an added feature. This approach allows the power supply voltage to be increased, resulting in an improved speed-torque curve. However, practical limitation of the power supply size, power dissipation in the external resistors, and motor driver constraints must be taken into account.

BiPolar
- BiPolar Resistance Limited
- BiPolar BiLevel
- BiPolar V.S.I
- BiPolar Two Quadrant Chopper
- BiPolar Four Quadrant Chopper

UniPolar
- UniPolar Resistance Limited
- UniPolar BiLevel
- UniPolar V.S.I.
- UniPolar Two Quadrant Chopper
- UniPolar Four Quadrant Chopper

MicroStep
- 2, 4, 8, 16, 32, 64 and 128 steps per full step are the most popular ranges available at reasonable cost. However, odd divisions are available as well.

Control Methods – ASIC and Processor

Open Loop verses Closed Loop Stepper Control

Considering open loop versus closed loop control is not a matter of which is *better*, it is a matter of which is *necessary*. Revisiting a previous discussion on the accuracy of the stepper system and your application should determine which method you need to use.

ASIC operation

The preprogrammed ASIC stepper control chips on the market today provide an invaluable tool for generating stepper motion. I read an article not to long ago that asked, "*Why use another processor to generate a move, when your computer already has one?*" Although that company is in the business of selling motion, it sells software not hardware. My response to that question would be:

One axis granted, but have you ever really done multi-axis, processor-based, stepper control? PC systems must handle the operator, the I/O, data collection, motion, inter-machine or factory communications, and possibly a host of other things. PC processors are capable of only so much, and even the CNC Industry uses multitasking.

If the ASIC fits use it. It will only aid you in achieving your goal, which is to get the project done quick. A hardware solution of comparable or lower cost to a software solution is generally the best solution.

Processor Based Stand-Alone Operation

Processor-based stepper control does not necessarily imply a PC-based processor, but rather a separate processor such as an *Intel* 8052. I have used this processor many times to generate 1, 2 and 3 axes of stepper control and had excellent results. The only limitation with a software (assembly code bit manipulation) controlled processor, is the *real-time* stepping frequency that can be generated. Ten thousand (10,000) steps per second is quite comfortable, but depending on the required motion, fifteen thousand (15,000) pulses per second is a maximum. The ASIC generators, on the other hand, can generate up to three million (3,000,000) pulses per second and not even get tired.

The advantage of a stand-alone device, would be to control the lower pulse rate needs with a processor, and also feed information to the ASIC processors to control the higher rates. A good stand-alone unit can quite satisfactorily control up to eight axes of stepper motion. These types of controllers will relieve the host processor of the motion work, and reduce the amount of software necessary to do the job at hand.

Since stand-alone units presently incorporate a software processor on board, they also maintain the ability to collect data and control I/O that the host can call up anytime.

Stepper Trapezoidal Algorithm

Below are the necessary considerations for developing any profile:

- Type of move – position or velocity
- Accel/decel rate – pulses/sec^2
- Max move velocity (slew rate) – pulses/sec
- Distance to move – pulses (if a position move)
- Time to accel/decel from initial velocity to final velocity.
- Desired update time – 1 to 65535 milliseconds.

With this information the formula pattern and the software algorithm can take shape. The trapezoidal example given below is for a *pulse and direction* type translator. First, figure out the minimum pulse width that can be accepted by the translator (consult the translator specification). For this example, the translator requires a 20-microsecond minimum positive or *true* pulse signal.

Next, knowing the maximum velocity requirement for any of the required moves, figure out the worst case time that your software can spend in the pulse-generating interrupt handler. For this example, maximum velocity will never exceed 9,000 steps-per-second. Therefore, I will set the worst case update time to 100 microseconds (10,000 steps/sec). Knowing that other functions besides stepper pulsing must also be handled, I will restrict the maximum time spent in the interrupt handler to 75 microseconds. That will allow a minimum of 25 percent of the software time to be spent in the main program.

Thirdly, we need to find out if *real-time* math can be used, or if you must develop a move table. My only objection to using tables is that they need to be constructed before each move, which takes valuable time (the tables need to be pre-processed just prior to or just after the issued *start* command). This takes both time and memory.

The processor we will use for this example is an *Intel* 8052, clocked at 12 MHz providing a one microsecond average instruction cycle. The equation used to figure out the required stepping interrupt time during an update is:

$$Interrupt\ \ time = \left(\frac{1}{velocity}\right)(1{,}000{,}000\ \mu sec)$$

Using Logarithm math, the fastest this equation can be solved, is around 100 microseconds. Obviously, this will not meet the update time criteria. A move table will be required ... ***or will it?*** Examining the move requirement closer, there is nothing that suggests that the acceleration must be done in one-step jumps. If we allow another indicator to control the number of pulses per second the velocity will be incremented each time an acceleration step is completed, then we can force the time interval between increments to at least 1.5 times longer than the time required to calculate it. Another interesting point is that when we reach the slew velocity, no further lengthy calculations are required until reaching the deceleration point. The deceleration position can be calculated prior to the move, and then it can be counted out in the pulse generating interrupt to flag when the deceleration must begin.

In this example I will not allow changing profile parameters while in motion, although it will only take your imagination to do so.

The accel/decel increment that will be used to change the velocity during the respective ramps will be calculated before the move, based on the accel/decel time, the indicated update time and the maximum velocity. When we are within one velocity increment of the slew velocity, we will calculate an interrupt value that will place the desired speed right on the number.

A Stepper Pulse and Direction Algorithm

```
;Set up the Scratch Pad Pointers without going into all of the required registers, the ones
;shown are to indicate a method for setup yielding fast handling in the 8052 processor.

X_CNTRL         EQU     20H             ; X Motor Control Byte
                                        ; 7   6   5   4   3   2   1   0
                                        ; |   |   |   |   |   |   |   |
MOV_X_REQ       BIT     X_CNTRL.0       ; |   |   |   |   |   |   |  VEL X
JOG_REQ         BIT     X_CNTRL.1       ; |   |   |   |   |   |  JOG X
MOV_TO_POS      BIT     X_CNTRL.2       ; |   |   |   |   |  POS X
X_MVE_DIR       BIT     X_CNTRL.3       ; |   |   |   |  DIR X  0=Fwd 1=Rev
RAMP_X_DWN      BIT     X_CNTRL.4       ; |   |   |  RAMP  0=Acl 1=Dcl
X_SHIFT         BIT     X_CNTRL.5       ; |   |  Next Acl/Dcl Vel. Jump

STEP_LREG       EQU     30H             ;Accel value 1 to 65535 step jumps
STEP_HREG       EQU     31H
X_POS1  EQU     32H                     ;X Absolute Counter 4 Bytes - 32 Bits
X_POS2  EQU     33H
X_POS3  EQU     34H
X_POS4  EQU     35H
X_GTP_1         EQU     36H             ;X Position to GoTo Register
X_GTP_2         EQU     37H             ;to begin the deceleration
X_GTP_3         EQU     38H
X_GTP_4         EQU     39H
X_DIST_1        EQU     3AH             ;X Position to GoTo Register
X_DIST_2        EQU     3BH             ;  = the position to goto
X_DIST_3        EQU     3CH
X_DIST_4        EQU     3DH
X_INT_LTME      EQU     3EH             ;TMR0 Interrupt reload regs
X_INT_HTME      EQU     3FH             ;  for current velocity
X_TOP_LVEL      EQU     40H             ;Top Velocity = steps/update
X_TOP_HVEL      EQU     41H
```



```
X_ACT_LVEL      EQU     42H             ;Actual velocity steps/update
X_ACT_HVEL      EQU     43H
X_UPDTE_LTME    EQU     44H             ;Update time for X axis speed check
X_UPDTE_HTME    EQU     45H
S_CURVE_MLTP1   EQU     46H             ;"S" curve operator shaper
S_CURVE_MLTP2   EQU     47H             ;1 msec/cnt = 4.66 Hrs.
S_CURVE_MLTP3   EQU     48H
S_UPDTE_CTR     EQU     49H             ;Num of updates bypassed
S_BYPASS_CNT    EQU     4AH             ;Num of updates to bypass

;XRAM POINTERS

RAM_STRT        XDATA   0000H
Axs_Port        XDATA   0C000H          ;X Axis Port.0 = Pulse
                                        ;       Port.1 = Direction
                                        ;       Port.2 = Hold I


                ;*****************************************************************
;The Program Starts Here

                ORG     0000H           ;Set Address
                JMP     START           ;Jump to the Program Beginning

                ORG     0003H           ;Set Address
                JMP     EXT_INT0        ;Ext Intrpt if used

                ORG     000BH           ;Set Address
                JMP     UPDTE_TMER      ;Update Timer Routine

                ORG     001BH           ;X Axis Control Routine

                ;*****************************************************************
                  ;Timer 1 Interrupt - This routine controls the X axis motor.
MOTOR_X:        MOV     TL1,X_INT_LTME  ;Load New Value into timer 1
                MOV     TH1,X_INT_HTME

                PUSH    DPL             ;Save generally used registers
                PUSH    DPH
                PUSH    ACC
                PUSH    PSW

                MOV     A,X_CNTRL       ;Is an X mov in progress ?
                ANL     A,#00000111B    ;Get the Move Request Bits
                JZ      MOTX1           ;If=0 then No move in progress

                JB      DCL_DONE,MOTX1  ;If Dcl is done, no Pulse

                MOV     DPTR,#Axs_Port  ;Get the port address
                MOVX    A,@DPTR         ;Get the port value
                SETB    ACC.0           ;Set the pulse bit
                MOVX    @DPTR,A         ;Output the pulse to the xslator
                ;*****************************************************************
                        ;Keep track of the absolute position
                JB      X_MVE_DIR,REV_MVE   ;Check the direction
FWD_MVE:        CLR     A
                CJNE    A,X_POS1,INC_CTR1   ;X POS CTR1 = 0 ?
                CJNE    A,X_POS2,INC_CTR2   ;X POS CTR2 = 0 ?
                CJNE    A,X_POS3,INC_CTR3   ;X POS CTR3 = 0 ?
INC_POS4:       INC     X_POS4              ;POS4 = POS4 - 1
INC_POS3:       INC     X_POS3              ;POS3 = POS3 - 1
INC_POS2:       INC     X_POS2              ;POS2 = POS2 - 1

INC_POS1:       INC     X_POS1              ;POS1 = POS1 - 1
                JMP     MOTX1
REV_MVE:        CLR     A
                CJNE    A,X_POS1,INC_CTR1   ;X POS CTR1 = 0 ?
                CJNE    A,X_POS2,INC_CTR2   ;X POS CTR2 = 0 ?
                CJNE    A,X_POS3,INC_CTR3   ;X POS CTR3 = 0 ?
DEC_POS4:       DEC     X_POS4              ;POS4 = POS4 - 1
DEC_POS3:       DEC     X_POS3              ;POS3 = POS3 - 1
```

```
DEC_POS2:   DEC     X_POS2              ;POS2 = POS2 - 1
DEC_POS1:   DEC     X_POS1              ;POS1 = POS1 - 1

            ;-------------------------------------------------------------------

MOTX1:      JB      MOV_TO_POS,XGOTO    ;Position move requested ?
            JB      MOV_X_REQ,CHKSTX    ;Any X Axis Move requested ?

            SETB    RAMP_X_DWN          ;Set the Decl Ramp indicator
            JMP     CHKSTX              ;Check the update indicator
            ;*****************************************************************
             ;A Position move was requested  -  Track the decel position

XGOTO:      JB      RAMP_X_DWN,CHKSTX   ;The DECL ramp is in progress

            CLR     A
            CJNE    A,X_DCTR1,DEC_CTR1  ;X GOTO POS CTR1=0 ?
            CJNE    A,X_DCTR2,DEC_CTR2  ;X GOTO POS CTR2=0 ?
            CJNE    A,X_DCTR3,DEC_CTR3  ;X GOTO POS CTR3=0 ?
            CJNE    A,X_DCTR4,DEC_CTR4  ;X GOTO POS CTR4=0 ?

            SETB    RAMP_X_DWN          ;Set the Decl Ramp
                                        ;indicator
            SETB    X_SHIFT             ;Do the first ramp shift
            SETB    ACL_DONE            ;No more accel. increments
            JMP     RAMPX

DEC_CTR4:   DEC     X_DCTR4             ;CNT4 = CNT4 - 1
DEC_CTR3:   DEC     X_DCTR3             ;CNT3 = CNT3 - 1
DEC_CTR2:   DEC     X_DCTR2             ;CNT2 = CNT2 - 1
DEC_CTR1:   DEC     X_DCTR1             ;CNT1 = CNT1 - 1

            ;*****************************************************************
             ;Check to see if the Update timer has flagged for a speed check

CHKSTX:     JNB     X_SHIFT,MTR_EXT     ;Request to check the speed

;A speed change was requested

RAMPX:      JNB     SPD_SET,MTR_EXT     ;New speed in place ?


            CLR     X_SHIFT             ;Clear the speed shift flag
            MOV     X_INT_LTME,X_LO_SPD ;Load the new Pulse rate
            MOV     X_INT_HTME,X_HI_SPD
            CLR     ACL_STP_REQ         ;Calc new Accl time
            CLR     DCL_STP_REQ         ;Calc new Decl time

            ;*****************************************************************
                        ;Clear the pulse from the port

MTR_EXT:    MOV     DPTR,#Axs_Port      ;Get the port address
            MOVX    A,@DPTR             ;Get the port value
            CLR     ACC.0               ;Clear the pulse bit
            MOVX    @DPTR,A             ;Output pulse to the translator

            ;*****************************************************************
                 ;Check if an accel. or decel. calculation is required

CHK_ACL:    JB      ACL_DONE,CHK_DCL
            JB      ACL_STP_REQ,MEXT

            SETB    ACL_STP_REQ

            MOV     DPTR,#ACL_MTH       ;ON RETURN -
            PUSH    DPL                 ;Goto the ACL Math routine
            PUSH    DPH
            RETI

            ;-------------------------------------------------------------------

CHK_DCL:    JNB     RAMP_DWN,MEXT
            JB      DCL_STP_REQ,MEXT

            SETB    DCL_STP_REQ

            MOV     DPTR,#DCL_MTH       ;ON RETURN -
            PUSH    DPL                 ;Goto the DCL Math routine
            PUSH    DPH
            RETI
```



```
            ;-------------------------------------------------------------------

MEXT:       POP     PSW                 ;SAVE STATUS
            POP     ACC                 ;RESTORE A
            POP     DPH                 ;RESTORE DATA POINTER
            POP     DPL
            RETI                        ;ENABLE ANOTHER INTERRUPT

;Note:  The time for this handler can be cut 40% using the newer Intel 20 Mhz processor.  Top
;velocity of 15,000 steps/sec can be realized using this style of code development.

            ;*****************************************************************
                                ;Do the Acceleration Math

ACL_MTH:    MOV     X_DCL_LVEL,X_LO_SPD ;Save the current vel
            MOV     X_DCL_HVEL,X_HI_SPD ;Time Interrupt for 1st Dcl

            MOV     A,X_ACT_LVEL        ;Add the increment to the vel
            ADD     A,STEP_LREG
            MOV     X_ACT_LVEL,A

            MOV     A,X_ACT_HVEL
            ADDC    A,STEP_HREG
            MOV     X_ACT_HVEL,A

            MOV     PSW,#000H           ;Chk if at the Top velocity
            MOV     A,X_TOP_LVEL
            SUBB    A,X_ACT_VEL

            MOV     A,X_TOP_HVEL
            SUBB    A,X_ACT_VEL

            JNC     ACL_MTH1

            MOV     X_ACT_LVEL,X_TOP_LVEL ;At the Top velocity
            MOV     X_ACT_HVEL,X_TOP_HVEL
            SETB    ACL_DONE

            ;-------------------------------------------------------------------

;calc the new acl increment time but save the old value in the decel reg prior to  each calc.
;When the slew rate is reached, the last stored value will be used for the first decel ramp
;step time.

;The equation which will be solved is  ....  Interrupt Time = 65536 - (1 / Vel) * 1,000,000

ACL_MTH1:   JB      BGN_UPDATE,ACL_MTH2

            SETB    BGN_UPDATE          ;Start the update timer

            MOV     TL2,X_UPDTE_LTME    ;Reload the timer 2 value
            MOV     TH2,X_UPDTE_HTME

ACL_MTH2:   MOV     VAL1,X_ACL_LVEL     ;Max Acel time > Update time
            MOV     VAL2,X_ACL_HVEL
            MOV     VAL3,#0
            MOV     VAL4,#0

            CALL    GET_LOG             ;Get the Log (Acl_Tme)
            CALL    STORE_LOG           ;Put the ans in storage reg1

            MOV     VAL1,#0             ;Get 1 sec * 1E6
            MOV     VAL2,#0             ;  = 1,000,000
            MOV     VAL3,#046H
            MOV     VAL4,#0E8H
            MOV     CHAR,#019H

            CALL    DIVDE_LOG           ;Get 1 / Vel = Intrpt time
            CALL    GET_ALOG            ;Return the decimal increment

            MOV     PSW,#0              ; 65536 - Time = Intrpt value
            MOV     A,#0
            SUBB    A,VAL1
            MOV     X_LO_SPD,A

            MOV     A,#0
            SUBB    A,VAL2
            MOV     X_HI_SPD,A

            POP     PSW                 ;RESTORE STATUS
            POP     ACC                 ;RESTORE A
            POP     DPH                 ;RESTORE DATA POINTER
```

```
            POP     DPL
            RET

;*****************************************************************
;Do the Deceleration Math

DCL_MTH:    MOV     PSW,#0
            MOV     A,X_ACT_LVEL        ;Sub the increm from the vel
            SUBB    A,STEP_LREG
            MOV     X_ACT_LVEL,A

            MOV     A,X_ACT_HVEL
            SUBB    A,STEP_HREG
            MOV     X_ACT_HVEL,A

            MOV     A,#0                ;Is the vel < 0 ?
            SUBB    A,#0

            JNC     DCL_MTH1

            MOV     X_ACT_LVEL,#0       ;Vel = 0
            MOV     X_ACT_HVEL,#0
            SETB    DCL_DONE

            ;-----------------------------------------------------------------
;calc the new dcl increment time.  When 0 slew velocity is reached, stop the move.
;The equation which will be solved is .... Interrupt Time = 65536 - (1 / Vel) * 1,000,000

DCL_MTH1:   MOV     VAL1,X_ACL_LVEL     ;Max Acel time > Update
                                        ;time
            MOV     VAL2,X_ACL_HVEL
            MOV     VAL3,#0
            MOV     VAL4,#0

            CALL    GET_LOG             ;Get the Log (Acl_Tme)
            CALL    STORE_LOG           ;Put the ans in storage reg1

            MOV     VAL1,#0             ;Get 1 sec * 1E6
            MOV     VAL2,#0             ;  = 1,000,000
            MOV     VAL3,#046H
            MOV     VAL4,#0E8H
            MOV     CHAR,#019H

            CALL    DIVDE_LOG           ;Get 1 / Vel = Intrpt time
            CALL    GET_ALOG            ;Return the decimal increment

            MOV     PSW,#0              ; 65536 - Time = Intrpt value
            MOV     A,#0
            SUBB    A,VAL1
            MOV     X_LO_SPD,A

            MOV     A,#0
            SUBB    A,VAL2
            MOV     X_HI_SPD,A

            POP     PSW                 ;RESTORE STATUS
            POP     ACC                 ;RESTORE A
            POP     DPH                 ;RESTORE DATA POINTER
            POP     DPL
            RET

;*****************************************************************
                    ;Update Timer Interrupt Handler

;A different versions of updating the step velocity uses a velocity pulse counting scheme.
;The idea of this is to set an update requirement based on a number of calculated velocity
;pulses that should occur during any velocity in the desired update time.  By counting the
;number of calculated pulses to zero and then activating the velocity shift, the whole
;operation can occur using only one timer interrupt.

;The reason why I use a separate interrupt timer for the velocity shift, is for "S" curve
;development and any "long" (linear) ramp requirements.  "S" curve math is easier, and more
;exact, the better you can control the time.  The counting method does not allow for as good
;a control at all velocities.  This can lead to oscillation (or generous velocity
;instability) during the ramp.  The second interrupt timer adds a degree of "smoothness" to
;profile ramps because of the more accurate timing.


UPDTE_TMER: MOV     TL2,X_UPDTE_LTME    ;Reload the timer 2 value
            MOV     TH2,X_UPDTE_HTME
```



```
            SETB    X_SHIFT

            ;This is a good place to do some general high speed I/O handling

            RETI

            ;*****************************************************************

;The following section would contain all of the various sub-routines required for the
;operation of the users software package.

            ;*****************************************************************
                        ;Hold current control bit On/Off

LO_AMPS:    MOV     DPTR,#Axis_PORT     ;Set the hold current bit
            MOVX    A,@DPTR
            SETB    ACC.2
            MOVX    @DPTR,A
            RET

HI_AMPS:    MOV     DPTR,#Axis_PORT     ;Clear the hold current bit
            MOVX    A,@DPTR
            CLR     ACC.2
            MOVX    @DPTR,A
            RET

            ;*****************************************************************
                        ;Check to see if the X axis is moving

CHK_MOTION: CLR     X_MOVING
            MOV     A,X_CNTRL           ;Get the move control byte
            ANL     A,#00000111B        ;Get the control bits
            JZ      CM1                 ; 0 = not moving
            SETB    X_MOVING

CMEXT:      RET

            ;*****************************************************************
                        ;Start the move (with Ramp)

STRT_MOVE:  CLR     RAMP_X_DWN
            CLR     ACL_DONE
            CLR     DCL_STP_REQ
            CLR     ACL_STP_REQ
            RET

            ;*****************************************************************
                        ;Stop the move (with Ramp)


STP_MOVE:   SETB    RAMP_X_DWN
            SETB    ACL_DONE
            CLR     DCL_STP_REQ
            CLR     ACL_STP_REQ
            RET

            ;*****************************************************************
STORE_LOG:  MOV     STORE1,VAL1
            MOV     STORE2,VAL2
            MOV     STORE3,VAL3
            MOV     STORE4,VAL4
            MOV     CHAR1,CHAR
            RET

            ;*****************************************************************
                        ;A / B = Log (A) - Log (B)

DIVDE_LOG:  MOV     PSW,#0
            MOV     A,VAL1
            SUBB    A,STORE1
            MOV     VAL1,A

            MOV     A,VAL2
            SUBB    A,STORE2
            MOV     VAL2,A

            MOV     A,VAL3
            SUBB    A,STORE3
            MOV     VAL3,A
```

```
                MOV     A,VAL4
                SUBB    A,STORE4
                MOV     VAL4,A

                MOV     A,CHAR
                SUBB    A,CHAR1
                MOV     CHAR,A
                RET

                ;*********************************************************************
                                ;A * B = Log (A) + Log (B)

MLTP_LOG:       MOV     A,STORE1
                ADD     A,VAL1
                MOV     VAL1,A

                MOV     A,STORE2
                ADD     A,VAL2
                MOV     VAL2,A

                MOV     A,STORE3
                ADD     A,VAL3


                MOV     VAL3,A

                MOV     A,STORE4
                ADD     A,VAL4
                MOV     VAL4,A

                MOV     A,CHAR1
                ADD     A,CHAR
                MOV     CHAR,A
                RET

                ;*********************************************************************

;Other Sub-Routines in this "Subroutine Section" would be to handle the communications, load
;the proper profile values as indicated in Section V.1 above, do all of the preparation math
;prior to axis release, and any other "House-Keeping" functions (I/O handling, etc.) your
;system may require prior to starting the ;move, during the move, or at the move completion
;point, such as activation or release of product clamps, lubricators on/off, etc.

                ;*********************************************************************
                                ;Insure the Profile is acceptable

TST_PROFILE:

;1) Get the position to begin the deceleration

;The Acel/Decl equation is ..... S = - A t^2
;By knowing the accel/Decel, the distance traveled during the Accel or Decel can be found for
;any time (t).

                CLR     A
                MOV     VAL4,A
                MOV     VAL3,A

                CLR     C
                MOV     A,STEP_HREG             ;Decl / 2
                RRC     A                       ;Time is not relevant since
                MOV     VAL2                    ;  the Decl is rated in
                                                ;  Steps/(1 sec^2)
                MOV     A,STEP_LREG
                RRC     A                       ;The equation then becomes
                MOV     VAL1                    ;       1/2 * D * 1^2

                MOV     PSW,#0
                MOV     A,X_DIST_1
                SUBB    A,VAL1
                MOV     X_GTP_1,A

                MOV     A,X_DIST_2
                SUBB    A,VAL2
                MOV     X_GTP_2,A


                MOV     A,X_DIST_3
                SUBB    A,VAL3
```



```
                MOV     X_GTP_3,A

                MOV     A,X_DIST_4
                SUBB    A,VAL4
                MOV     X_GTP_4,A               ;Store the decl Position

;2)  Set the starting speed Interrupt time

;The starting speed intrpt time is ..... Time1 = 1 / FL speed * 1,000,000 = 1,000,000/FL usec
;Interrupt Time = 65536 - Time1 = 0 - Time1 (Hex)   Note: Starting velocity (FL) must be >

;1/.065535 or 16 PPS

                MOV     VAL1,FL_LSPD            ;Get the FL speed
                MOV     VAL2,FL_HSPD
                MOV     VAL3,#0
                MOV     VAL4,#0

                CALL    GET_LOG                 ;Get the Log (FL)

                MOV     PSW,#0                  ;Log (1E6) - Log (FL)
                MOV     A,#0
                SUBB    A,VAL1
                MOV     VAL1,A

                MOV     A,#0
                SUBB    A,VAL2
                MOV     VAL2,A

                MOV     A,#04EH
                SUBB    A,VAL3
                MOV     VAL3,A

                MOV     A,#0EEH
                SUBB    A,VAL4
                MOV     VAL4,A

                MOV     A,#019H
                SUBB    A,CHAR
                MOV     CHAR,A

                CALL    GET_ALOG                ;Convert back to a number

                MOV  PSW,#0                     ;Convert to Hex Intrpt Time
                MOV     A,#0
                SUBB    A,VAL1
                MOV     X_INT_LTME,A
                MOV     X_LO_SPD,A              ;Load the new Pulse rate
                MOV     A,#0
                SUBB    A,VAL2


                MOV     X_INT_HTME,A
                MOV     X_HI_SPD,A

;3)  Set the Acceleration/Deceleration Increment to maintain a reasonably smooth Accel/Decel
;Profile, speed shifting of the stepper pulse rate should occur at "fixed" time intervals.
;The Update timer will be used for this purpose, and will be an operator entry.  A good
;starting point would be 2 ms updates.  For a velocity of 10,000 steps/sec this would be a
;speed correction (update) every 20 step pulses.

;The equations to determine the Acel/Decl Increments are then ...
;           Acel_time / Update_time = #_of_Acel_Updates
;           Top_Vel / #_of_Acel_Updates = Acel_Increment / Update

                MOV     VAL1,ACL_TME1           ;Max Acel time > Update time
                MOV     VAL2,ACL_TME2
                MOV     VAL3,ACL_TME3
                MOV     VAL4,ACL_TME4

                CALL    GET_LOG                 ;Get the Log (Acl_Tme)
                CALL    STORE_LOG               ;Put the ans in storage reg1

                MOV     VAL1,UPDTE_TME1         ;Get the Update time
                MOV     VAL2,UPDTE_TME2         ; Updte_Tme < 65535
                MOV     VAL3,#0
                MOV     VAL4,#0

                CALL    GET_LOG                 ;Get the Log (Updte_Tme)
                CALL    DIVDE_LOG               ;Get Acl_Tme / Updte_Tme
                CALL    STORE_LOG               ;Store the answer
```

```
        MOV     VAL1,X_TOP_LVEL         ;Get the Top Velocity
        MOV     VAL2,X_TOP_HVEL
        MOV     VAL3,#0
        MOV     VAL4,#0

        CALL    GET_LOG                 ;Get the Log (Top_vel)
        CALL    DIVDE_LOG               ;Top_vel / Previous answer
        CALL    GET_ALOG                ;Return the decimal increment

        MOV     STEP_LREG,VAL1          ;Store the increment
        MOV     STEP_HREG,VAL2

        CLR     A
        MOV     ELAPS_TME1,A            ;Zero the ramp elapsed time
        MOV     ELAPS_TME2,A            ;counter
        MOV     ELAPS_TME3,A



        MOV     S_UPDATE_CTR,S_BYPASS_CNT
        RET

        ;*****************************************************************
                        ;MAIN PROGRAM STARTS HERE
START:  ;VARIOUS HANDLING ROUTINES GO HERE
        ;
        ;
        CALL    GET_KEY_ENTRY           ;Get operator entry data
        ;
          CALL  DISPLAY_DATA            ;Send the data to the CRT
          ;
          CALL  SETUP_MOVE              ;Vel/Pos/Jog move handlers
          ;
          CALL  STRT_MOVE               ;Begin the move
          ;
          CALL  XYZ_Routine             ;Calls to various handlers
          ;
          ;
          ;
          JMP   START                   ;Do it all over again

          END
```


A Stepper Phase Control Algorithm

```
        ;Timer 1 Interrupt - This routine controls the x axis motor via pulse/sec req.

T1INT:  MOV     TL1,X_INT_LTME          ;Load time count into timer 1
        MOV     TH1,X_INT_HTME
        JNB     HLDN_TMR_ACTVE,T1_CONT  ;Is part clamp active ?
        RETI

T1_CONT:PUSH    DPL                     ;Save generally used registers
        PUSH    DPH
        PUSH    ACC
        PUSH    PSW

        ;Output phase sequences to the stepper motor (fwd or rev motion)

MOTORX: MOV     A,XSPD_TBL_PTR          ;Is the x speed table pointer = 0
        JZ      MOTX1                   ;Yes - Therefore no sequence change is required

        MOV     A,R0                    ;Get the phase sequence pointer.
        INC     A                       ;Point to the next sequence.
        ANL     A,#037H                 ;Limit = addresses 30H to 37H
        MOV     R0,A                    ;Save the phase sequence pointer.

        MOV     A,Y_CODE                ;Get the current y axis phase step from Y storage
        ADD     A,@R0                   ;Combine for a single output

        MOV     DPTR,#Axis_Port         ;Set the port addres
        MOVX    @DPTR,A                 ;Output phase sequences to the motors.

MOTX1:  JB      MOV_TO_POS,XGOTO        ;Position move requested ?
        JB      MOV_X_REQ,CHKSTX        ;Any move requested ?

        SETB    RAMP_X_DWN              ;Set the ramp down flag
        JMP     CHKSTX                  ;Check the update counter

                ;Any move Run or Goto Position requested

XGOTO:  JB      RAMP_X_DWN,XGOTO_EXT
        JB      RMP_CONTRL,XGT1

XGT1:   CLR     A
        CJNE    A,X_LCTR,DEC_LCTR       ;Goto Pos low Cntr = 0 ?
        CJNE    A,X_HCTR,DEC_HCTR       ;Goto Pos high Cntr = 0 ?
                                        ;Low and High Cntrs = 0
        SETB    RAMP_X_DWN              ;Begin the decl rampdown
        JMP     XGOTO_EXT1

DEC_HCTR:DEC    X_HCTR                  ;Decrement the High Cntr
DEC_LCTR:DEC    X_LCTR                  ;Decrement the Low  Cntr

XGOTO_EXT:DJNZ  XGOTO_INTRP,MTRX_EXT
        JNB     RAMP_X_DWN,CHKSTX1

XGOTO_EXT1:MOV  A,XSPD_TBL_PTR
        ANL     A,#11111110B            ;Ignore the base speed
        JZ      XGOTO_EXT2
        JNB     ACC.7,XGOTO_EXT3
        ANL     A,#01110000B            ;Get the top speed
        JZ      XGOTO_EXT2

XGOTO_EXT2:MOV  X_CONTRL,#000H          ;Mask off Motion Request

XGOTO_EXT3:JNB  RAMP_X_DWN,CHKSTX1
        JMP     RAMPDX

                ;Check the Interrupt counter for the x/y axis.

CHKSTX: JNB     X_SHIFT,MTRX_EXT        ;Time to update the X Axis ?
        CLR     X_SHIFT                 ;Yes
        JB      RAMP_X_DWN,RAMPDX       ;If ramping down exit test

                ;Check and change the speed if required

CHKSTX1:MOV     A,XSPD_TBL_PTR          ;Get the Speed Table Pointer
        CJNE    A,X_MAX_SPD,CHK_HI_LO   ; = to Max req value ?
        JMP     NEW_XSTP_TMR            ;If at speed - no change req
```

```
CHK_HI_LO:      JC      RAMPUX              ;If Low - goto next higher speed

RAMPDX:         MOV     A,XSPD_TBL_PTR      ;Get the speed table pointer
                JZ      NEW_XSTP_TMR        ; = to zero ?
                                            ;No
RDX1:           DEC     XSPD_TBL_PTR        ;Goto next lower speed selection
                JMP     NEW_XSTP_TMR

RAMPUX:         INC     XSPD_TBL_PTR        ;Goto next Higher speed selection

                ;Setup for next timer 0 interrupt time period

NEW_XSTP_TMR:   MOV     A,XSPD_TBL_PTR      ;Get the current speed table pointer
                JNB     MOV_TO_POS,NXT1

                CLR     C
                RRC     A
                CLR     C
                RRC     A
                CLR     C
                RRC     A

NXT1:           INC     A
                MOV     XGOTO_INTRP,A

                MOV     A,XSPD_TBL_PTR      ;Get the current speed table pointer
                RL      A                   ; x2 (Addresses are 2 bytes each)
                MOV     X_INT_HTME,A        ;place in temp storage

                MOV     DPTR,#XSPD_TBL      ;Time table interrupt addresses
                MOVC    A,@A+DPTR           ;Gget the high time data byte

                XCH     A,X_INT_HTME        ;Store time and restore the pointer
                INC     A                   ;Pointer address + 1

                MOVC    A,@A+DPTR           ;Get the low time data byte
                MOV     X_INT_LTME,A        ;Store it

        ;Check for a y axis movement or miscellaneous action requirement on exit

MTRX_EXT:       JB      DO_MISC_OPER,MEXT
                SETB    DO_MISC_CHK         ;MISC REQ OPERATION

                MOV     DPTR,#DO_MISC       ;ON RETURN - GOTO THE MISC OPER
                PUSH    DPL                 ;CHECK ROUTINE
                PUSH    DPH
                RETI

MEXT:           POP     PSW                 ;Restore Status
                POP     ACC                 ;RESTORE A
                POP     DPH                 ;RESTORE DATA POINTER
                POP     DPL
                RETI                        ;ENABLE ANOTHER INTERRUPT

          ;Do any other miscellaneous operations that are interrupt controlled

DO_MISC:        xxx     xxxxxxxx
                yyy     yyyyyyyy
                zzz     zzzzzzzz

MISC_EXT:       POP     PSW                 ;Restore Ststus
                POP     ACC                 ;RESTORE A
                POP     DPH                 ;RESTORE DATA POINTER
                POP     DPL

                CLR DO_MISC_CHK
                RET                         ;EXIT THIS ROUTINE

                        ;X Axis Speed Time Interrupt Table
;Interrupt times = (65536 - (indicated count)) x 1.0 usec

XSPD_TBL:                                                   ;STEPS /  SEC
                DW      55536,60536,62203,63036,63536       ; 100 ->  500
                DW      63869,64107,64286,64425,64536       ; 600 -> 1000
                DW      64627,64703,64767,64822,64869       ;1100 -> 1500
                DW      64911,64948,64980,65010,65036       ;1600 -> 2000
                DW      65060,65081,65101,65119,65136       ;2100 -> 2500
                DW      65151,65166,65179,65191,65203       ;2600 -> 3000
                DW      65213,65224,65233,65242,65250       ;3100 -> 3500
                DW      65258,65266,65273,65280,65286       ;3600 -> 4000
                DW      65292,65298,65303,65309,65314       ;4100 -> 4500
                DW      65319,65323,65328,65332,65336       ;4600 -> 5000
                DW      65340,65344,65347,65351,65354       ;5100 -> 5500
                DW      65357,65361,65364,65367,65369       ;5600 -> 6000
                DW      65372,65375,65377,65380,65382       ;6100 -> 6500
                DW      65384,65387,65389,65391,65393       ;6600 -> 7000
```



```
                DW      65395,65397,65399,65401,65403       ;7100 -> 7500
                DW      65404,65406,65408,65409,65410       ;7600 -> 8000
                DW      65412,65414,65415,65417,65418       ;8100 -> 8600

                                    ;************
     ;X Axis motor Accl/Deccl (or general speed shift) update timer handling routine

UPDTE_TMR:      PUSH    DPL                 ;TO THE STACK
                PUSH    DPH
                PUSH    ACC                 ;SAVE GENERALLY USED REGISTERS
                PUSH    PSW

                JB      MOV_TO_POS,UPDTE_CONT

                DJNZ    X_INTRPT_CTR,UPDTE_CONT ;INTRPT CTR = 0 ?
                SETB    X_SHIFT
                JB      RAMP_X_DWN,XDCL
                MOV     X_INTRPT_CTR,ACL_REG

                JNB     JOG_REQ,UPDTE_CONT  ;HAS A JOG BEEN REQUESTED  ?
                MOV     X_INTRPT_CTR,#060H  ;5 SEC ACL PROFILE
                JMP     UPDTE_CONT

XDCL:           MOV     X_INTRPT_CTR,DCL_REG

UPDTE_CONT:     ppp     pppppp              ;DO OTHER PLC ETC OPERATIONS HERE
                qqq     qqqqqq
                rrr     rrrrrr

UPDT_EXT:       POP     PSW                 ;Restore the Status word
                POP     ACC                 ;Restore the Accumulator
                POP     DPH                 ;Restore the Data pointer
                POP     DPL
                RETI
```