Engineering Solutions

Chuck Raskin P.E. EE MSCS
Blaine MN 55434
Cell: 612-207-5285 <ChuckR@EngSolu.com>

Microprocessor Trajectory Generation

'In the beginning', when I first got into Motion Control, I wanted to understand how a motion generator was implemented in a microprocessor. I tried to find out the 'secret', as it was, but it was a well-kept secret! As time went on, ASIC's and DSP's made the world better and faster. But the big question was still there . . . .

How did the software manage to make it all happen at such high speed?

In this case, the problem wasn't one of formula use, but of formula implementation to produce high speed math in multi-axis systems.

The objective of this paper, therefore, is to pass on some of what I have learned. To remove the servo control mystery, I will first present a detailed description of Trapezoidal and 'S curve motion generation. Following this, I will present the typical PID gain algorithm employing Acceleration and Velocity FeedForward, to form the motor command output signal (DAC). Finally, I will show how I used Log math to perform high speed multiply, divide, and powers calculations

****

Generating the Trajectory and Controlling the DAC

The code I will be presenting will be pseudo code. Pseudo code will allow me more latitude for explaining the operation of the various algorithm components.

When motion is to be resolved, we need to be knowledgeable of four things . . . the desired motion parameters, the calculated motion vs. time, the motion limits, and the actual motion taking place. It doesn't do any good to begin moving if we don't know where we are going or how to control it! So what we will do, is break the move down into the four indicated parts, so we can deal with it in software.

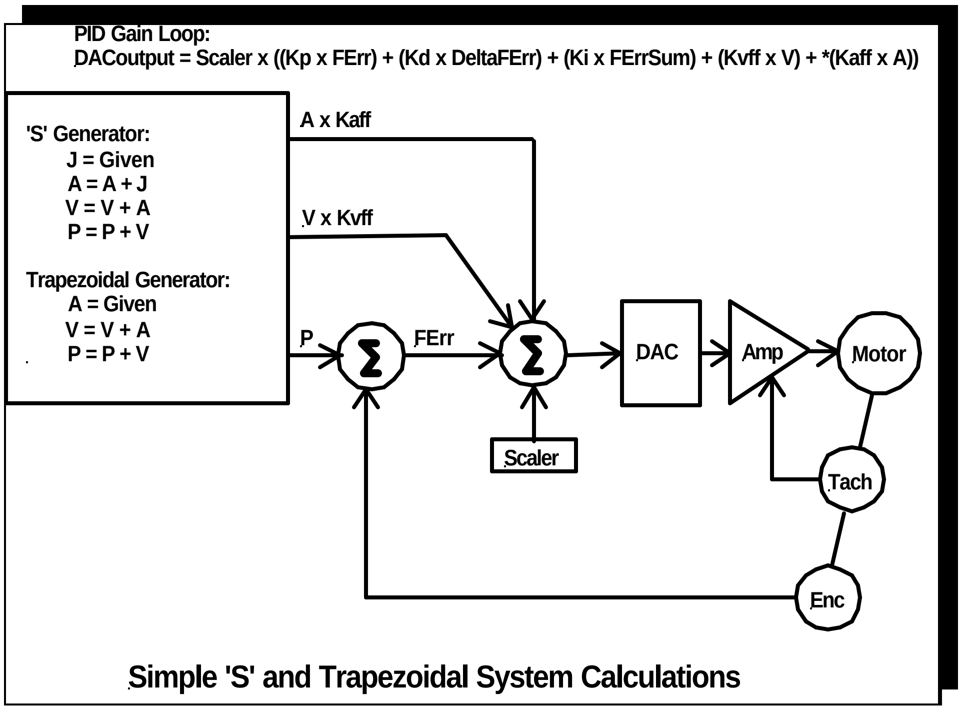

Figure 1

As figure 1 shows, the basics to a Trajectory Generator are very straightforward. The real issue is in detailing all of the math checks that have to be done as each section of math is accomplished.

Component	**Description**		**Variable**
Target Position	= Position to move to if in Position mode	=	DesPos
Desired Velocity	= Speed at which we want the axis to move	=	DesVel
Desired Acceleration	= Calculated Axis Acceleration	=	DesAcl
Desired Deceleration	= Calculated Axis Deceleration	=	DesDcl
Desired Jerk	= Calculated Axis Deceleration	=	DesJrk
Calculated Position	= Calculated trajectory position @ any time	=	CalcPos
Calculated Velocity	= Calculated trajectory velocity @ any time	=	CalcVel
Calculated Acceleration	= Calculated trajectory Accel @ any time	=	CalcAcl
Calculated Deceleration	= Calculated trajectory Decel @ any time	=	CalcDcl
Maximum Velocity	= A limit above which the velocity does not go	=	MaxVel
Maximum Acceleration	= A limit above which the Accel does not go	=	MaxAcl
Maximum Deceleration	= A limit above which the Decel does not go	=	MaxDcl
l			
Actual Position	= Accumulated position Counter @ any time	=	ActPos
Actual Velocity	= Speed at which we want the axis to move	=	ActVel
Actual Acceleration	= Speed at which we want the axis to move	=	ActAcl
Actual Deceleration	= Speed at which we want the axis to move	=	ActDcl

From the above list, you can see that the only variable we are missing is TIME. In all motion formulas, time is what makes it all come together . . . . the glue so to speak. Examine the following motion formulas for example . . . .

$S = 1/2at^2$ where **S** is DISTANCE, **a** is ACCELERATION, and **t** is TIME

$S = 1/2dt^2$ where **S** is DISTANCE, **d** is DECELERATION, and **t** is TIME

$V = S/t$ where **S** is DISTANCE, **V** is VELOCITY, and **t** is TIME

$V = at$ where **V** is VELOCITY, **a** is ACCELERATION, and **t** is TIME

Notice that time is the only true variable because <u>TIME is the only true unknown.</u>

For those who doubt this, think about the following . . . . We know motor RPM or Speed in revolutions per unit time! If we know revolution per unit time at given time intervals, then we can compute Accel and Decel rates. We know Revolutions or linear distance because we will have some form of position feedback (even for a stepper)! Thus, the only thing we don't have is a time stamp! But what if we were to make TIME (t) a constant? It would then become a non-variable, or known quantity, and allow us to easily solve all of the formulas required, in a very short time (pun intended).

Let's begin by setting a timer interrupt at one millisecond, and lets call this the update time. Everyone millisecond, or every update period, we will have an opportunity to look at the motion and determine if it is doing what we want it to do. If it is, then we do nothing. But if it isn't, we can speed it up or slow it down as required. What we really need to know then, is how fast the system can react. For example, if the system can change at a rate of 10 RPM per millisecond, and we need to change 1 RPM, we certainly cannot make a change then go away for one full millisecond expecting the motion to be one RPM different. That is, unless we slow down (lower) the DAC command gain control values so that the motor changes at a rate of less than 1 RPM per millisecond. We need to insure that the system does not change faster than our ability to control it to achieve the desired stability, which is not a variable, but a specification.

Setting up system physical requirements

Since over 80% of motion systems I have dealt with do not require time updates (sample times) faster than 100 microseconds, I will set my example update time to that. In addition, most CNC's employ system resolutions of 10,000 counts/inch, so we will set our system for that value. Assuming a five turn per inch resolution of the motor to load travel, the encoder will be 2000 counts per revolution (10,000counts per inch / 5 pitch). The speed (velocity) will be limited to 2500 RPM, with Accel and Decel rates of 231.48 Revs/Sec^2, which yields a velocity of 2500 RPM in 180 milliseconds. 180 milliseconds is a value that I have used for years. It was derived from some of the earlier work that was done with CNC velocity control prior to DSP's and the high-performance motion takeover. It was a very comfortable value to use which insured mechanical systems did not over stress. I have used it ever since I learned about it.

Thus, our system breakdown is as follows . . . (note the many ways the variables are described) . . . .

Update time = 100 microseconds (us)

Motor Velocity = 2500 RPM

(2500 RPM) / (60 seconds/minute) x (2000 counts/Rev)
= 83,333.34 counts per second

83,333.34 counts per second x 0.0001sec
= 8.3334 counts / sample-time

Motor Acceleration = 231.48 Revs / Sec^2

(231.48Revs/sec^2) x (2000 counts/Rev)
= 462960 counts per Sec^2

462960 counts per Sec^2 x .0001sec /sample-time x 0.0001 sec /sample-time
= 0.00462960 counts / sample-time^2

Motor Deceleration = Motor Acceleration Rate

Motor Jerk = 1/10 Acceleration Rate (a number I just picked)

Set the MaxVel to 110% of DesVel, and MaxAcl, and MaxDcl to 500% of the Desired values.

MaxVel	=	9.16674 counts per sample time
MaxAcl	=	0.023148 counts / sample-time^2
MaxDcl	=	0.023148 counts / sample-time^2

Now that the basic trajectory variables have been assigned, let's set up some move parameters, and make some **calculated** moves.

****

Trapezoidal Motion Profile Generator

For this example, I am intentionally setting the Accel and Decel values equal to each other. The target position will be three inches which makes the move 30000 encoder counts long.

Since the time constant, and **known**, the acceleration change will be 0.00462960 added to the calculated acceleration register (RampReg) every sample time until either DesVel has been achieved or the current position register times two is equal to or greater than the TgtPos (30,000).

If a move is not underway, keep the motor in position

```
If (MveDoneFlag) GoTo DoFolErr
```

Next, test to see if the slew is complete. If it is, then go to the CalcDcl.

```
If (DoDclFlag) GoTo DoDcl;
```

Next, test to see if acceleration is complete. If it is, then go to the CalcSlew.

```
If (DoSlewFlag) GoTo DoSlew;
```

DoAccel:

The question is, why am I adding the acceleration to a totalizing register every sample time? Simple, the formula for velocity is V=a/t. Since we know the acceleration per unit time, and that time is constant, then the velocity is increasing at a rate of the growth of acceleration. The growth of Velocity will be linear, or change at a constant rate, in each constant time interval (which is a constant value). Remember, this is a calculated velocity value, not what is actually happening. Thus, perfection can prevail as . . . .

```
CalcVel = CalcVel + DesAcl
```

The next step is to test the CalcVel value against DesVel to determine if the target velocity has been reached or exceeded. If it has, then we will set the CalcVel to the DesVel value.

```
If (CalcVel > DesVel) then {  CalcVel = DesVel and DoSlewFlag = 1 }
```

In this equation, if the CalcVel is determined to be greater than the desired value (DesVel) it will get set to DesVel, and the trapezoidal profile that is generated by this will actually get a ‘beveled’ edge one sample time prior to going into the flat line velocity calculation. This hurts nothing, puts the velocity at the exact value it should be, and creates a softening of the final step into DesVel just prior to flat line entry, which actually reduces system ‘Jerk’.

Next, calculate the current position using the formula: $S = Vt$

This is an easy one. If we were traveling at a rate of 10mph for one hour, then we would have covered 10 miles. In our case, we just finished traveling at a rate of CalcVel for one sample period, thus . . . .

```
CalcPos = CalcPos + CalcVel
```

Next, determine if we are halfway to the target position. If we are, jump to the deceleration routine.

```
If ((CalcPos + CalcPos) >= TgtPos) then { DoDclFlag = 1 and  GoTo DoDecel }
```

Save the current position for later use in the Slew routine.

```
CalcAclDist = CalcPos
```

Exit the acceleration module at this point . . . .

```
GoTo DoFolErr
```

CalcSlew:

At this point, the velocity is never changing, so we simply run along incrementing the position register.

```
CalcPos = CalcPos + CalcVel
```

Next check to determine if the value of the Position register plus the distance it took to reach the velocity we are currently running at (CalcAclDist) is equal to or greater than the Target position. If it is, we set the deceleration request flag.

If ((CalcPos + CalcAclDist) >= TgtPos) then DoDclFlag = 1

Exit the Slew module at this point . . . .

Goto DoFolErr

DoDcl:

It is now time to decelerate from the running velocity. Since we got to CalcVel by adding DesAcl to CalcVel, it only makes sense to stop by subtracting DesDcl from CalcVel until the velocity reaches zero, or a negative value . . .

CalcVel = CalcVel - DesDcl

The next step is to test the CalcVel value to determine if Zero velocity has been reached or exceeded. If it has, then we will set the MveDoneFlag.

If (CalcVel <= 0) then { CalcVel = 0 and MveDoneFlag = 1 }

Next, calculate the current position. . . .

CalcPos = CalcPos + CalcVel

Exit the acceleration module at this point. . . .

GoTo DoFolErr

****

‘S’ Curve Motion Generator

For this example, I will be using the Jerk Factor to ramp the acceleration values in segments 1, 3, 5, and 7 which generates the ‘S’ portions of the curve. This profile is sometimes called a seven (7) segment curve since it is broken down into seven component parts as shown in figure 2.

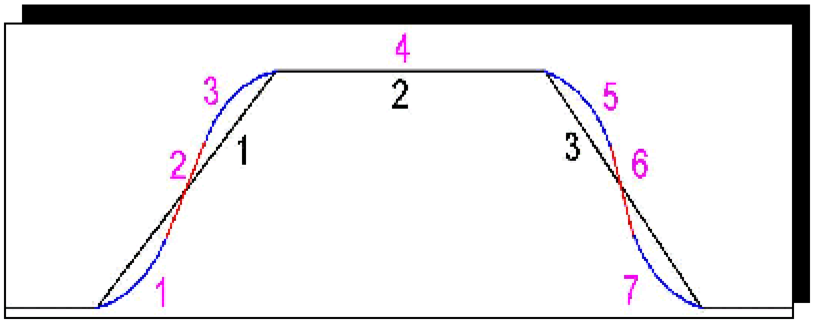

Figure 2

Several things of interest about the ‘S’ curve is that to accelerate in the same time as the trapezoid, the acceleration rate in sections 2 and 6 are twice what they are in the trapezoid sections 1 and 3. But if you were to make the rates the same, then the S curve would take twice as long to get to the running velocity (4) as the trapezoid (2)

Once again, since the time constant is **<u>known</u>**, the acceleration will change by an amount of DesJrk every update, and the velocity will change by the amount of CalcAcl every update. The Jerk value is added to the calculated acceleration register (RampReg) every sample time until either CalcVel has achieved one half the value of DesVel, or the current position register times Four is equal to or greater than the

TgtPos (30,000), or MaxAcl has been achieved. The four times position factor comes from the fact that the 'S' curve must form a 'bell' shaqpe if the motion is to remain continuous. Therefore, sections 1, 3, 5, and 7 are all equal in distance if the acceleration and deceleration ramps are set equal to each other.

```
If a move is not underway, keep the motor in position . . . .
        If (MveDoneFlag) GoTo DoFolErr
Next, test to see if deceleration-6 is complete. If it is, then go to the DoDecl-7 routine . . . .
        If (DoDcl7Flag) GoTo DoDcl-7
Next, test to see if deceleration-5 is complete. If it is, then go to the DoDecl-6 routine . . . .
        If (DoDcl6Flag) GoTo DoDcl-6
Next, test to see if the slew is complete. If it is, then go to the DoDecl-5 routine . . . .
        If (DoDcl5Flag) GoTo DoDcl-5
Next, test to see if acceleration-3 is complete. If it is, then go to the CalcSlew . . . .
        If (DoSlew4Flag) GoTo CalcSlew-4
During Acceleration, keep track of the acceleration distance . . . .
        CalcAclDist = CalcPos + CalcAclDist + CalcVel
Next, test to see if acceleration-2 is complete. If it is, then go to the DoAccel-3 routine . . . .
        If (DoAcl3Flag) GoTo DoAcl-3
Next, test to see if acceleration-1 is complete. If it is, then go to the DoAccel-2 routine . . . .
        If (DoAcl2Flag) GoTo DoAcl-2
```

DoAcl-1:

Prior to making the Move, Zero the AclRampReg & DclRampReg. This will be used to insure the Accel and Decel Ramps are ended at the right velocity.

Increment the base S curve sample time counter for use in the DoAcl-3 segment . . . .

ScurvCntr = ScurvCntr + 1

The question is, why am I adding the Jerk to an acceleration totalizing register every sample time? Simple, the formula for S acceleration is A = J/t. Since we know the Jerk per unit time, and that time is constant, then the acceleration will increase at a rate of the growth of Jerk. The growth of Jerk will be linear, or change at a constant rate, in each constant update time interval. Remember, this is a calculated velocity value, not what is <u>actually</u> happening. Thus, perfection can prevail as . . . .

CalcAcl = CalcAcl + DesJrk

The next step is to test the CalcAcl value against DesAcl to determine if the target acceleration has been reached or exceeded. If it has, then we will set the CalcAcl equal to the DesAcl value.

If (CalcAcl > DesAcl) then { CalcAcl = DesAcl and DoAcl2Flag = 1 }

In this equation, if CalcAcl is determined to be greater than the desired value (DesAcl) it will get set to DesAcl, and the acceleration ramp that is generated by this will actually get a 'beveled' edge one sample time prior to going into the flat line acceleration calculation (segment 2 of figure 1). This hurts nothing, puts the acceleration ramp at the exact value it should be, and reducing the 'Jerk' effect on the system just prior to flat line entry into ramp segment 2.

Next, calculate the current velocity using the formula: V = at

CalcVel = CalcVel + CalcAcl

At this point, there test for ½ velocity since we are in the beginning of a three-step acceleration curve. The velocity test is simply . . . .

```
If (CalcVel >= ½ DesVel) then {  DoAcl3Flag = 1 }
```

Next, calculate the current position using the formula: $S = Vt$

This is an easy one. Once again, if we were traveling at a rate of 10mph for 1hr, then we would have covered 10m. In our case, we just finished traveling at a rate of CalcVel for one sample period, thus . . . .

```
CalcPos = CalcPos + CalcVel
```

Next, determine if we are one fourth of the way to the target position. If we are, jump to the completion of the acceleration routine.

```
If ((CalcPos x 4) >= TgtPos) then { DoAcl3Flag = 1 and  GoTo DoAcl3 }
```

Save the current position for later use in the Slew routine.

```
CalcAclDist = CalcAclDist + CalcVel
```

Exit this module . . . .

```
GoTo DoFolErr
```

DoAcl-2:

Count the number of samples it takes to either reach ½ velocity, or the MaxDcl Value . . . .

```
ScurvCntr = ScurvCntr + 1
```

If the Velocity is less than ½ the desired velocity increment the AclRampReg counter, otherwise decrement the AclRampReg counter, and when it is equal to Zero, perform the DoAcl-3 segment.

```
If (CalcVel >= DesVel / 2) then {
        AclRampReg = AclRampReg - 1
        If (AclRampReg <= 0)    { DoAcl3Flag = 1
                                  GoTo DoAcl3
                                }
}
Else    {  AclRampReg = AclRampReg + 1 }
```

Next, Increment the Velocity Register remembering that the velocity is now linear with acceleration..

```
CalcVel = CalcVel + CalcAcl
```

Next, keep track of position as the acceleration ramp continues to climb.

```
CalcPos = CalcPos + CalcVel
```

Exit this module . . . .

```
GoTo DoFolErr
```

DoAcl-3:

Round out the acceleration ramp to meet the desired velocity with zero transition.

First decrement the 'S' curve counter . . . .

```
ScurvCntr = ScurvCntr -1
```

If it is equal to or less than Zero, do the Slew . . . .

```
If (ScurvCntr <= 0) { CalcSlew-4 = 1 }
```

Subtract the Jerk from the calculated acceleration rate register . . .

```
CalcAcl = CalcAcl - DesJrk
```

If the calculated acceleration register is less than or equal to Zero, do the slew . . .

```
If (CalcAcl <= 0) { DoSlew4Flag = 1 }
```

Next, calculate the current velocity . . . .

```
CalcVel = CalcVel + CalcAcl
```

If the calc'd velocity is equal to or greater than the desired velocity, set the calculated velocity to the desired velocity then set the slew flag. . . .

```
If (CalcVel >= CalcVel) { CalcVel = DesVel and DoSlew4Flag = 1 }
```

Next, keep track of the calculated position as the velocity continues to climb.

```
CalcPos = CalcPos + CalcVel
```

If the calculated position is equal to or greater than ½ the Target position, set the DoDcl-5 flag . . . .

```
If ((CalcPos x 2) >= TgtPos) then { DoDcl5Flag = 1 }
```

Exit this module . . . .

```
GoTo DoFolErr
```

DoSlew-4:

At this point, the velocity is never changing, so we simple run along incrementing the position register.

```
CalcPos = CalcPos + CalcVel
```

Next check to determine if the value of the Position register plus the distance it took to reach the velocity during acceleration is equal to or greater than the Target position. If it is, set the deceleration flag . . . .

```
If ((CalcPos + CalcAclDist) >= TgtPos) then DoDcl5Flag = 1
```

Exit the Slew module . . . .

```
GoTo DoFolErr
```

DoDcl-5:

It is now time to decelerate from some velocity. Since we got to this velocity by adding CalcAcl to CalcVel, it only makes sense deceleration is accomplished by subtracting DesDcl from CalcVel until the calculated velocity reaches zero or a negative value . . .

First, get the ½ velocity point for later use . . . .

```
SlewVelMem = CalcVel / 2
```

Next, we need to compute the CalcDcl Value . . . .

```
CalcDcl = CalcDcl + DesJrk
```

Next, we need to determine if the MaxDcl Rate has been reached, and if it has, limit the CalcDcl to the MaxDcl value . . . .

```
        If (CalcDcl >= MaxDcl) then {CalcDcl = MaxDcl and DoDcl6Flag = 1 }
```

Next, calculate the change in velocity . . . .

```
        CalcVel = CalcVel - CalcDcl
```

Next, determine if we are ½ way to Zero velocity. If we are, then go to DoDcl-7 to round out into 0 velocity . . .

```
        If (CalcVel <=  SlewVelMem) then { DoDcl7Flag = 1 }
```

Next, calculate the current position . . . .

```
        CalcPos = CalcPos + CalcVel
```

Exit this module . . . .

```
        GoTo DoFolErr
```

DoDcl-6:

If the Velocity is greater than ½ way to Zero velocity, increment the DclRampReg counter, otherwise decrement the DclRampReg counter, and when it is equal to Zero, perform the DoDcl-7 segment.

```
    If (CalcVel <= SlewVelMem) then {
                DclRampReg = DclRampReg - 1
                If (DclRampReg <= 0)   {  DoDcl7Flag = 1
                                          GoTo DoDcl-7
                                       }
            }
    Else      { DclRampReg = DclRampReg + 1 }
```

Next, Increment the Velocity Register remembering that the velocity is now linear with acceleration..

```
        CalcVel = CalcVel + CalcDcl
```

Next, keep track of position as the acceleration ramp continues to climb.

```
        CalcPos = CalcPos + CalcVel
```

Exit this module . . . .

```
        GoTo DoFolErr
```

DoDcl-7:

Round out the acceleration ramp into 0 velocity with zero transition.

First decrement the S curve counter . . . .

```
        ScurvCntr = ScurvCntr -1
```

If it is equal to or less than Zero, Stop the move . . . .

```
        If (ScurvCntr <= 0) { CalcVel = 0 and  MveDoneFlag = 1 }
```

Subtract the Jerk from the calculated acceleration rate register . . .

```
        CalcDcl = CalcDcl - DesJrk
```

If the calculated acceleration register is less than or equal to Zero, do the slew . . .

```
        If (CalcDcl <= 0) { CalcVel = 0 and  MveDoneFlag = 1 }
```

Next, calculate the current velocity register . . . .

CalcVel = CalcVel - CalcDcl

If the calc'd velocity is equal to or less than Zero velocity, set the calculated velocity to Zero and set the Move Done flag. . . .

If (CalcVel <= 0) { CalcVel = 0 and MveDoneFlag = 1 }

Next, keep track of the calculated position . . . .

CalcPos = CalcPos + CalcVel

Exit this module . . . .

GoTo DoFolErr

****

Do the Gain Algorithm Here

DoFolErr:

This section will discuss the PID gain algorithm and at what part of the trajectory the various parameters are to be adjusted. I will start off by defining a typical PID formula . . . .

DAC Value = SetPoint+(Pgain x FErr)+(Igain x ErrSum)+(Dgain x DeltaErr)+(Ka x CalcAcl)+(Kv x CalcVel)

FErr (Following Error):	This is the difference between the Actual Position Register (ActPos) and the desired position register (TgtPos). The actual position count is captured first when the update interrupt is entered, so as to maintain a constant time interval between captures. When the DoFolErr routine is entered, ActPos is subtracted from TgtPos to compute the Following Error Value.
DAC Output Value:	This is the result actually applied to the motor control amplifier.
SetPoint:	This is a user-preset value that can be used to offset the DAC output. For example, it can be used to offset the effect of gravity in vertical systems. If used, it is directly applied to the DAC output value.
Kp: Proportional:	This value is derived by multiplying the Kp gain term and the Following Error.
Ki: Integral:	This value is derived by adding the newly computed Following Error to a Following Error totalizing register, and then multiplying the total value by the Ki gain term.
Kd: Derivative:	This value is derived by multiplying the Kd gain term and the difference between the Following Error determined in the current update and Following Error determined in the previous update. In addition, some motion controllers allow the Derivative element to be updated in a different time interval than other gain elements. This will generally increase the dynamic range of control capability.
Ka: AclFeedFwd:	For the 'S curve, this value is derived by multiplying the Ka value and the calculated Accel (CalcAcl). For the Trapezoidal profile, this value is derived by multiplying the Ka value and the Desired Accel when the move is started.
Kv: VelFeedFwd:	This value is derived by multiplying the Kv value and the calculated velocity (CalcVel)

****

A PID Analogy

To understand how to tune the PID, you first need to understand what the PID gain loop is up against when coordinating motion. The following analogy will allow you to better grasp how the elements interact. Have you ever read about or tried to think of an analogy for the PID gain structure you could quickly relate to? The generally accepted analogy relating shock absorbers to Kd and springs to Kp has not allowed for good PID understanding since not everyone is mechanically inclined. In addition, an analogy for Ki to this point has yet to be solidly defined. So, let's begin understanding PID gain structures by relating the PID elements to something we are all intimately familiar with, which is our body.

Activating your Kp

Stand up straight with your arms at your sides. Carefully lift one foot off the floor, and place it behind the ankle of the other. Immediately, you will feel your supporting ankle reacting to changes in your body's balance. This is your body Kp error correction system reacting to instantaneous changes in balance your mind is sensing. The update period for this reaction is the time it takes for you to mentally record and correct for the changes in balance. Your mind is using its learned Kp gain value to compensate for any shift in your balance. As you "sense/feel" your balance change, your mind multiplies your Kp value by your balance shift, or PosErr, and then applies the adjustment to your ankle. If you consciously put your learned Kp to zero, your ankle (system) will stop reacting and you will fall. If on the other hand you raise your Kp too high, your ankle will constantly overreact (i.e., oscillate out of control). This action is exactly how the Kp will respond in your motion system.

Activating your Kd

Slowly raise your arms until they are pointing to the left and right, parallel to the ground. Try to maintain them in this position for the remainder of the analogy. Note this position as Kd equal to zero. After a few moments, or minutes, you'll notice that your arms will have shifted (or be shifting) to new positions in order to help maintain your balance while standing on one foot. Also notice that your arms may maintain the new position for a period of time longer than the normal update period being used by your ankle Kp. This time interval is sometimes referred to as the Kd sample time or DST.

Your arm(s) 'adjust-and-hold-over-time' method of reacting to the body change in balance is its Kd reacting to a projected error based on previous error differences (or velocity changes for the motion controller). Note that your arms are responding to your sense of 'feel' of what you project will happen, not to what is actually happening as the Kp does. The anticipation of falling based on your body rate of change in vertical attitude is the same as a motion controller sensing a change in velocity over time (acceleration). Both cases are predicting what is expected to happen if the existing condition is allowed to prevail.

Activating your Ki

At this point, has your body torso also shifted in position? This is the body Ki. The Ki term builds slowly over time and as well, reduces to zero slowly over time. Note that as the Ki term is invoked, your ankles (Kp), and arms (Kd) never stop correcting. All three work as a team. Also note, however, that as one gain term takes over the control of the balance for a given moment, the other gain term effects are reduced, and might even go to zero. The long term Ki effect will readjust itself over time. As your body moves to 'catch up' to where your mind (the controller) says it should be (standing), the newly computed Ki output value may still be rather large (your body is still leaning or bent over). In time new position adjustments produced by the Ki will become relatively small in comparison to the total Ki reaction first encountered.

Once your body has stabilized, the Ki term will do what is required to restore you to your initial upright position (zero following error). Notice that your Kp and Kd body elements meanwhile are still working to help you balance while the Ki element is working toward straightening you up. It should be obvious that if the totaled Ki error is allowed to grow to a value too large for your body to handle, you will lose control and fall. In a motion system, however, too much Ki will react as a "rubber-band" oscillating back and forth around the desired position point. To prevent an excessive Ki reaction, your mind invokes a limiting value to the growth of the Ki result, generally called "Integral Limit" (IL). Your body will try not to lean beyond this limit.

Activating your FeedForward Gain

Actually, this gain loop has been in effect since the beginning of the exercise! As you were 'thinking' about your balance and what your Kp, Ki, and Kd was doing, you were also contemplating (projecting) balance problems. You were making adjustments for these projections before they even occurred! You were "looking ahead," so to speak, at potential instability and making corrections based on what your balance feedback device(s) (eyes, ears, and body 'feel', etc.) were telling you.

****

Where the PID gain terms should be adjusted

Figure 3 shows a typical Trapezoidal motion profile. Superimposed on this profile is the placement of the various PID gains, and where they should be adjusted. However, just as important as where they should be adjusted is **when** they should be adjusted.

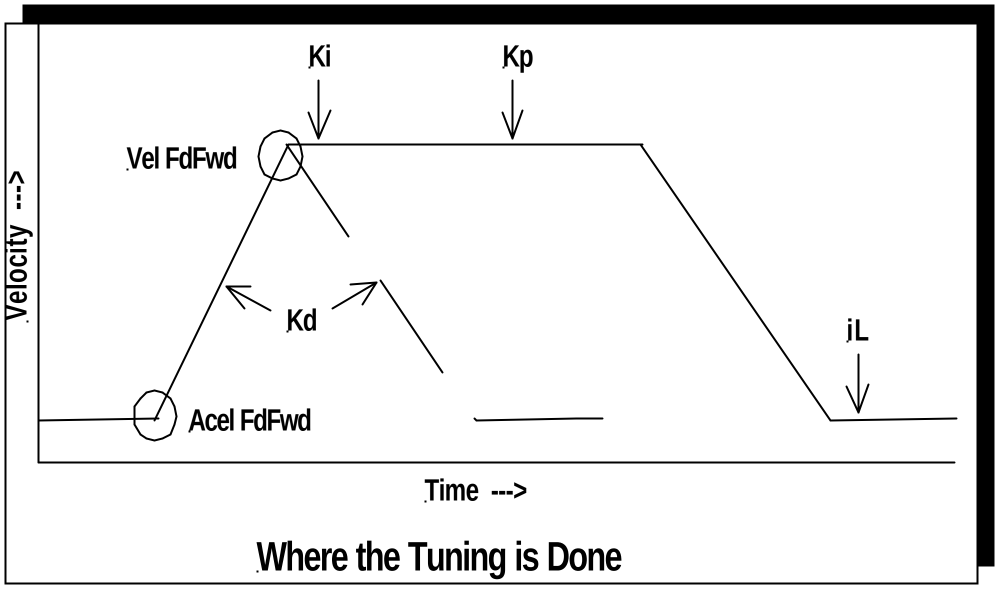

Figure 3

The tune should start out by tuning OFF the KI, IL, Velocity and Acceleration FeedForward gains. The Kp should be set to a low value, and the Kd to five times the Kp value depending on the scaling value for KD. With the Motor operating using a superficial load - an inertia disk of the same J as the load, begin a continuous velocity move in any direction. When the motor finishes the acceleration ramp, monitor the Following Error value. It may be large, but it should be stable. If it is growing or stable, begin increasing the Kp term until the Following Error reduces to ((3 degrees) / (360 degrees)) x Counts_per_motor_revolution if operating in the Current Mode, or ((90 degrees) / (360 degrees)) x Counts_per_motor_revolution if operating in the velocity Mode

If the Following Error begins to oscillate during this operation, increase the Kd gain slowly to stabilize it. Stop at the indicated Following Error values, or at a point where the motor is stable, but further adjustment does not allow stable tight Following Error operation.

Next, ramp the motor up from zero velocity to the top required operating velocity, and adjust the Kd for best Following Error. This is when the Acceleration ramp Following Error (FErr) is the Same as the Deceleration ramp Following Error. If Kd has to be reduced to balance the FErr value during the Kd ramping test, rerun the Kp test and insure the motor is still stable. If it is not, only adjust the Kp to stabilize it.

Next set the Ki to the value of Kp and the IL to a value ½ Kp. Make a Trapezoidal move. Adjust the Ki for a simple single round-out into the test velocity. When done, adjust the IL so that the motor stops with a simple single round-out into zero velocity, yielding a very minor position reset or an acceptable stop with no further motion.

The Acceleration FeedForward should be adjusted at the onset of the Acceleration ramp, and the Velocity FeedForward at the Top of the Velocity ramp.

On the Engineering Solutions web site is an article titled “The Science of Tuning.” This article gives a play-by-play description of the tuning process.

****

Logarithm Math

The idea behind logarithms is to accomplish divide, multiply and powers math is a hurry! Appendix A demonstrates the three log mechanisms. First, convert a number to a Log, second, convert a log to a number, and third, use a correction table to allow for finite math handling errors. The correction table can be expanded to accommodate more exacting answers.

I found that in 8 bit microprocessors, and processors without good math handlers, using logs can save a lot of time. For example, in a one microsecond per instruction 8051, a 4 byte divide can take over 500 microseconds to solve. By using logs, however, the solution time can be cut down to under 100 microseconds.

A correction table is used to accommodate conversion errors, and can be expanded on to allow for more precise answers.

****

Epilog

I hope the information presented in this paper answers many of the questions about the math required to make motors move with purpose, in both velocity and position modes. Right, I forgot about velocity mode calculations. Wait, isn’t that the same set of calculations except without a Desired Target Position. Simply tell the unit to move to the absolute maximum allowed Target position the DesPos register can handle and say ‘GO’. Just when the counter reaches DesPos, roll the position to Zero, and keep going until a STOP is commanded.

There are so many things one can do once the formula methodology is known. Down the road, various contouring algorithms will be explained. I hope you can put the methods shown here to good use.

Chuck Raskin P.E. MSCS

****

A DSP version of the Pseudo Code

The few pages (~32) gives a simple example of what DSP code might look like when doing a trajectory generator, gain algorithm and output to the DAC.

```
DoUpdate:
        ENA SEC_REG;                    {protect the main body data}

        {Every Update cycle------------------}
        {Get the Current Elapsed Position Count}
        {ASIC_Seq reg tells the ASIC to run a math seq}
        {AxisSeq  reg returns the next seq the ASIC is to do from the ASIC}
        {        the ASIC will then clear the ASIC_Seq reg    }
        { Flag the ASIC to compute ALL pos/vel/acl & errors }

{*************************************************************
  1. Increment the Update counter in the ASIC -
        Increment the sample interval counter - 32 bit w/rollover DPM
        DSP sets a bit for the ASIC to compute pos/vel/acl and errors
        The ASIC maintains the actual position count.
                This is a 32 bit position counter (+/-31 bit).
        The ASIC maintains the current position error (this err - last err),
                This is a signed 16 bit integer.
        The ASIC maintains the current velocity (16/16 bit integer/frac.)
                Change in pos in 1msec intervals / update time
        The ASIC maintains the current acceleration (16/16 bit integer/frac.)
                Change in vel in 1msec intervals / update time^2
********* *****************************************************
 Section 1.
        Flag the ASIC to compute ALL errors
 Section 2.
        Activate the previously calc'd DAC value 1st
        ffff -> ffff        16 Bit Temp DAC -> 16 Bit Act DAC

 Section 3. Get ALL Errors from the ASIC as needed
************************************************************************}
        AR=DM(SeqToDo);                         {Point at the Segment to do}
        TOPPCSTACK=AR;                          {put it on the stack}
        RTS;                                    {go to the proper Seg routine}
{***********************************************************************}
{***********************************************************************}
Segment1a:
        AX0=0x1;                                {set TC=0  acl bit=1}
        DM(SpdFlg)=AX0;                 {store indicator}
        I4=^Segment1;                           {get seg 1 ready}
        DM(SeqToDo)=I4;                 {store}
Segment1:
        {       DO THE S ACL Curve_Up
          C     if v >= Mxv/2 then set segment 3 flag
          *     if a >= Mxacl then a=Mxacl - begin segment 2
                 *********************************}
        {       If 'trap goto segment 2}
        {set up the seg counter ... seg1cnt=seg1cnt+1 ... seg2cnt=1}

/               {*********************************}
```

```
            { 'IF' statements to determine when this segment is done }

{           Update time         = 5 usec/axis
            quad cnt rate       = 8 MHz
            Mx vel              = 255 cts/sample
            resolution          = 1000000 cts/rev = 8 RPS = 480 RPM

            Pos = 48 bits = 00000000 00000000 00000000 00000000.00000000 00000000  = +/-2^31
            Vel = 32 bits =                            00000000.00000000 00000000 00000000  =  255 cts/smp
            Acl = 32 bits =                            00000000.00000000 00000000 00000000 = 255 cts/smp^2
            Jrk = 32 bits =                            00000000.00000000 00000000 00000000 = 255 cts/smp^3

            NOTE:  Vel/Acl/Jrk values are positive values only ....
}

            {Test_Jerk:}                                       { if jerk = 0 begin seg 2 }
            I0=^AclJrk;                                        {get the Ajrk pointer}
            AX0=DM(I0,M1);                         {get   j1 : -> j2}
            AY0=DM(I0,M2);                         {get   j2 : -> j1}
            AR=AX0 OR AY0;                         {get the jerk value}
            IF EQ JUMP Segment2a;                              {if the jerk == 0 goto seg2}
            {5}
            AX0=DM(ForceTgFlg);
            AR=PASS AX0;                                       {test the stop flag}
            IF NE JUMP Segment3a;                              {if 1 do acl curve down}

            {**********************************}
                { Pos = Pos + Vel  (32.16 + 16.32) }
                        { p = p+v->store p }

            I0=^DesPos;                                        {get the pos buf pointer}
            I1=^DesVel;                                        {get the vel buf pointer}
            SE=-8;
            MR0=DM(I1,M1);                                     {get v1 ->v2}
            MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);                  {get v2 ->v1 : shift the LSW v1}
            AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);                    {get p1 ->p2 : shft MSW v2}
            AY1=DM(I0,M1), AR=SR0+AY0;             {get p2 ->p3 : p1=p1+v1}
            AX0=DM(I0,M3);                                     {get p3 ->p1  : p2=p2+v2}
            DM(I0,M1)=AR, AR=SR1+AY1+C;                        {store p1 ->p2}
            DM(I0,M1)=AR,  AR=AX0+C;                           {store p2   ->p3 : p3=p3+c}
            DM(I0,M3)=AR;                                      {store p3 -> p1}
            SE=-1;
                                {//////////////}
                                                               {is DesPos>=TgtPos*.1681  (=0x2b08)}
            I1=^TgtPos;                                        {get the Tgpos buf pointer}
            MX0=0x2b08;                                        {get .1681 multiplier}
            MY0=DM(I1,M1);                         {get Tgp1 -> tgp2}
            MY1=DM(I1,M2),MR=MX0*MY0(UU);                      {get tgp2 - tgp1 : Compute Lo}
            SR=LSHIFT MR0(LO);                                 {Shift LSW right 1 bit}
            SR=SR OR LSHIFT MR1(HI);                           {before saving}
            AX1=SR0;                                           {Save LSW}
            MR0=MR1;                                           {shift 16 right}
            MR=MR+MX0*MY1(US);
            SR=LSHIFT MR0 (LO);                                {Shift LSW right 1 bits}
            SR=SR OR LSHIFT MR1 (HI);                          {Shift MSW right 1 bits}
            {SR0:MR0}                                          {Save LSW:MSW}
                                {//////////////}
            AY0=DM(I0,M1);                          {get p2 -> p3}
            AY1=DM(I0,M3), AR=SR0-AY0;              {get p3 -> p1 : tgp1*.1681 - p2}
            AR=MR0-AY1+C-1;                         {tgp2*.1681 - p3}
            IF GT JUMP Seg1_Acl;                    { if if Tgp*.1681 > p continue}

            I0=^AclJrk;                             {get the acl Jrk buf pointer}
            I1=^DclJrk;                             {get the dcl Jrk buf pointer}
            AX0=DM(I0,M1);                          {get aj1    : ->aj2}
            DM(I1,M1)=AX0;                          {store in dj1 -> dj2}
            AX0=DM(I0,M1);                          {get aj2    : ->a1}
            DM(I1,M1)=AX0;                          {store in dj2 -> d1}

            I0=^DesAcl;                             {get the acl buf pointer}
            I1=^DesDcl;                             {get the dcl buf pointer}
            AX0=DM(I0,M1);                          {get a1    : ->a2}
            DM(I1,M1)=AX0;                          {store a1 in d1 -> d2}
            AX0=DM(I0,M2);                          {get a2    : ->a1}
            DM(I1,M2)=AX0;                          {store a2 in d2 -> d1}

            I4=^Segment3;                           {if yes - get seg 3 ready}
            DM(SeqToDo)=I4;                 {store}
            JUMP Seg3_Vel;                          {continue with Seg3}

            {**********************************}
               { Acl = Acl + Jrk  (32.32 + .32)  }

Seg1_Acl:
            I0=^DesAcl;                             {get the acl buf pointer : I1=velptr}
            I1=^AclJrk;                             {get the Ajrk buf address}
            AX0=DM(I1,M1);                          {get j1       : ->j2}
            AY0=DM(I0,M1);                          {get a1       : ->a2}
            AX1=DM(I0,M2), AR=AX0+AY0;              {get a2       : ->a1   : a1=a1+j1}
            AY1=DM(I1,M2);                          {get j2           : ->j1}
            AY0=AR,  AR=AX1+AY1+C;                  {store a1 in ay0 : a2=a2+c}
            AY1=AR;                                 {store a2 in ay1}

            I1=^TgtAcl;                             {get the mxacl buf pointer}
            AX0=DM(I1,M1);                          {get   mxa1 : -> mxa2}
            AX1=DM(I1,M2),  AR=AX0-AY0;             {get   mxa2 : -> mxa1 : mxa1 - a1}
            AR=AX1-AY1+C-1;                         {mxa2 - a2}
            IF GT JUMP Stor_Seg1acl;                { if > 0 continue}
            {5}                                     { else set acl=mxacl}
            AY0=AX0;                                {xfer the mxacl1}
            AY1=AX1;                                {xfer the mxacl2}
            I4=^Segment2b;                          {if yes - get seg 2 ready}
            DM(SeqToDo)=I4;                 {store}
            }
Stor_Seg1acl:
            DM(I0,M1)=AY0;                  {store   a1 ->   a2}
            DM(I0,M2)=AY1;                  {store   a2 ->   a1}

            {**********************************}
              { Vel = Vel + Acl   (16.32 + 16.32) }
                        { v = v+a->store v }

            I0=^DesAcl;                             {get the acl buf pointer : I1=velptr}
            I1=^DesVel;                             {get the vel buf pointer}
            AX0=DM(I0,M1);                          {get a1    : ->a2}
            AY0=DM(I1,M1);                          {get v1    : ->v2}
            AX1=DM(I0,M2), AR=AX0+AY0;              {get a2    : ->a1   : v1=v1+a1}
            AY1=DM(I1,M2);                          {get v2    : ->v1}
            AY0=AR,  AR=AX1+AY1+C;                  {store new v1 : v2=v2+a2}
            AY1=AR;                                 {store new v2}
            {7}
            I0=^TgtVel;                             {get the tgtvel}
            MR0=DM(I0,M1);                          {get tgv1 -> tgv2}
            MR1=DM(I0,M2), SR=LSHIFT MR0 (LO);      {get tgv2 -> tgv1 : tgv1/2}
            AY0=DM(I1,M1), SR=SR OR LSHIFT MR1 (HI)          ;{get v1 -> v2 : tgv2/2}
                                                    {answer is in SR1 SR0}

            AR=SR0-AY0;                             {a=tgv1/2-v1}
            AR=SR1-AY1+C-1;                         {a=tgv2/2-v2}
            IF GT JUMP Stor_Seg1vel;                {is v > tgtvel/2 ?}
            {3}
            AY0=SR0;                                {xfer the mxacl1}
            AY1=SR1;                                {xfer the mxacl2}
            I4=^Segment3;                           {if yes - get seg 3 ready}
            DM(SeqToDo)=I4;                 {store}

Stor_Seg1vel:
            DM(I0,M1)=AY0;                  {store   a1 ->   a2}
            DM(I0,M2)=AY1;                  {store   a2 ->   a1}

            {**********************************}
ExtS1:
            AR=DM(GainToDo);                        {Point at the Gain to do}
            TOPPCSTACK=AR;                          {put it on the stack}
            RTS;                                    {go to the gain routine}

{*************************************************************************}
{*************************************************************************}

Segment2a:
            I4=^Segment2;                           {if yes - get seg 2 ready}
            DM(SeqToDo)=I4;                 {store}
            {2}

Segment2b:
            {clear the segment counter for seg2 'S requirement}
            I0=^SegCntr;                            {point to the seg counter}
            AX0=0x0;                                {set 0}
            DM(I0,M3)=AX0;                          {store segcnt1}
            DM(I0,M3)=AX0;                          {store segcnt2}
            {4}

Segment2:
{
            Update time   = 5 usec/axis
            quad cnt rate = 10 MHz
            Mx vel        = 255 cts/sample
            resolution    = 1000000 cts/rev = 10 RPS = 600 RPM

            Pos = 48 bits = 00000000 00000000 00000000 00000000.00000000 00000000         = +/-2^31
            Vel = 32 bits =                   00000000.00000000 00000000 00000000 = 255 cts/smp
            Acl = 32 bits =                   00000000.00000000 00000000 00000000 = 255 cts/smp^2
            Jrk = 32 bits =                   00000000.00000000 00000000 00000000 = 255 cts/smp^3

            NOTE:  ALL Vel/Acl/Jrk values are positive values only ....
}
            AX0=DM(ForceTgFlg);
            AR=PASS AX0;                                {test the stop bit}
            IF EQ JUMP Seg2_ProfChk;                    {if 0 do }
            {3}
            {Stop has been issued}                      {check for 'S or 'T mode}
            I0=^AclJrk;                                 {get the jrk pointer}
            AY0=DM(I0,M1);                              {get   j1 : -> j2}
            AX0=DM(I0,M2);                      {get   j2 : -> j1}
            AR=AX0 OR AY0;                      {jx = j1+j2}
            IF EQ JUMP Segment6a;                       {if Jrk==0 do seg6 trap down}
            JUMP Segment3a;                             { else begin  seg3 acl down}
            {6}

Seg2_ProfChk:
            AY0=DM(I0,M1);                              {get   j1 : -> j2}
            AX0=DM(I0,M2);                      {get   j2 : -> j1}
            AR=AX0 OR AY0;                      {jx = j1+j2}
            IF NE JUMP Seg2_Sprof;                      {if > 0 do the S curve}
            {4}

            {////////////////////////////////////////////////////////}
            {AT THIS POINT SEG2 IS DOING THE TRAP MOVE}
            {////////////////////////////////////////////////////////}
                                {p = p + v}
            I0=^DesVel;                                 {get the vel buf pointer}
            I1=^DesPos;                                 {pnt at the despos}
            SE=-8;
            MR0=DM(I0,M1);                              {get v1 ->v2}
            MR1=DM(I0,M2),SR=LSHIFT MR0 (LO);           {get v2 ->v1 : shift the LSW v1}
            AY0=DM(I1,M1),SR=SR OR LSHIFT MR1 (HI);             {get p1 ->p2 : shft MSW v2}
            AY1=DM(I1,M1), AR=SR0+AY0;                  {get p2 ->p3 : p1=p1+v1}
            AX0=DM(I1,M3);                              {get p3 ->p1  : p2=p2+v2}
            DM(I1,M1)=AR, AR=SR1+AY1+C;                 {store p1 ->p2}
            DM(I1,M1)=AR,  AR=AX0+C;                    {store p2   ->p3 : p3=p3+c}
            DM(I1,M3)=AR;                               {store p3 -> p1}
            SE=-1;

            {**********************************}
                    {is DesPos>=TgtPos/2?}

            I0=^TgtPos;                                 {get the Tgpos buf pointer}
            MR0=DM(I0,M1);                              {get tgp1 -> tgp2}
            MR1=DM(I0,M2), SR=LSHIFT MR0 (LO);          {get tgp2 -> tgp1 : shft tgps1}
            AY0=DM(I1,M1),SR=SR OR LSHIFT MR1 (HI);             {get p1 -> p2 : tgp2/2}
                                                        {tp/2 answer is in SR1 SR0}
            AY0=DM(I0,M1);                              {get p2 -> p3}
            AY1=DM(I0,M3), AR=AX0-AY0;                  {get p3 -> p1 : tgp1/2 - p2}
            AR=MR0-AY1+C-1;                             {tgp2/2 - p3}
            IF GT JUMP Seg2_TrapVel;                    { if Tgp/2 > p continue w/trap acl}

            I0=^DesAcl;                                 {get the acl buf pointer}
            I1=^DesDcl;                                 {get the dcl buf pointer}
            AX0=DM(I0,M1);                          {get a1    : ->a2}
            DM(I1,M1)=AX0;                          {store a1 in d1 -> d2}
            AX0=DM(I0,M2);                          {get a2    : ->a1}
            DM(I1,M2)=AX0;                          {store a2 in d2 -> d1}

            I4=^Segment6a;                          {if p>=tgp set seg 6a flag}
            DM(SeqToDo)=I4;                 {store}

            AR=DM(GainToDo);                        {Point at the Gain to do}
            TOPPCSTACK=AR;                          {put it on the stack}
            RTS;                                    {go to the gain routine}

            {***********************************}
             { Vel = Vel + Acl   (16.32 + 16.32) }
                    { v = v+a->store v }

Seg2_TrapVel:
            I0=^DesAcl;                             {get the acl buf pointer}
            I1=^DesVel;                             {get the vel buf pointer}
            AX0=DM(I0,M1);                          {get a1    : ->a2}
            AY0=DM(I1,M1);                          {get v1    : ->v2}
            AX1=DM(I0,M2), AR=AX0+AY0;              {get a2    : ->a1   : v1=v1+a1}
            AY1=DM(I1,M2);                          {get v2    : ->v1}
            AY0=AR,  AR=AX1+AY1+C;                  {save v1  : v2=v2+a2}
            AY1=AR;                                 {save v2}

            I0=^TgtVel;                             {get the tgtvel}
            AX0=DM(I0,M1);                          {get tgv1 -> tgv2}
            AX1=DM(I0,M2), AR=AX0-AY0;              {get tgv2 -> tgv1  : tgv1-v1}
            AR=SR1-AY1+C-1;                         {tgv2-v2}
            IF GT JUMP ExtSeg2T;                    { if v < tgtvel continue}
                                                    { else set the vel to the mx vel}
            AY0=AX0;                                {xfer mxv1 -> v1}
            AY1=AX1;                                {xfer mxv1 -> v1}

            I4=^Segment4b;                          {if v>=tgv set seg 4a flag}
            DM(SeqToDo)=I4;                 {store}
ExtSeg2T:
            DM(I1,M1)=AY0;                  {store   v1 : -> v2}
            DM(I1,M2)=AY1;                  {store   v2 : -> v1}

            AR=DM(GainToDo);                        {Point at the Gain to do}
            TOPPCSTACK=AR;                          {put it on the stack}
            RTS;                                    {go to the gain routine}



            {////////////////////////////////////////////////////////}
            {AT THIS POINT SEG2 IS DOING THE S MOVE}
            {////////////////////////////////////////////////////////}
Seg2_Sprof:
                          {p=p+v}
            I0=^DesPos;                             {get the pos buf pointer}
            I1=^DesVel;                             {get the vel buf pointer}
            SE=-8;
            MR0=DM(I1,M1);                          {get v1 ->v2}
            MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);        {get v2 ->v1 : shift the LSW v1}
            AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);           {get p1 ->p2 : shft MSW v2}
            AY1=DM(I0,M1), AR=SR0+AY0;      {get p2 ->p3 : p1=p1+v1}
            AX0=DM(I0,M3);                           {get p3 ->p1  : p2=p2+v2}
            DM(I0,M1)=AR, AR=SR1+AY1+C;              {store p1 ->p2}
            DM(I0,M1)=AR,  AR=AX0+C;                 {store p2   ->p3 : p3=p3+c}
            DM(I0,M3)=AR;                            {store p3 -> p1}
            SE=-1;

            {**********************************}
            { if (v+a >= Mxv/2)
                        1.          SegCntr = SegCntr - 1
                        2.           if      SegCntr <= 0 jump to segment3
                                    else   SegCntr += 1
            }

            I0=^DesAcl;                              {get the acl buf pointer : I1=velptr}
            I1=^DesVel;                              {get the vel buf pointer}
            AX0=DM(I0,M1);                           {get a1    : ->a2}
            AY0=DM(I1,M1);                           {get v1    : ->v2}
            AX1=DM(I0,M2), AR=AX0+AY0;               {get a2    : ->a1   : v1=v1+a1}
            AY1=DM(I1,M2);                           {get v2    : ->v1}
            DM(I1,M1)=AR,  AR=AX1+AY1+C;             {store v1  : ->v2   : v2=v2+a2}
            DM(I1,M2)=AR;                            {store v2  : ->v1}

            I0=^TgtVel;                              {get the MxVel buf pointer}
            MR0=DM(I0,M1);                           {get mxv1 -> mxp2}
            MR1=DM(I0,M2), SR=LSHIFT MR0 (LO);       {get mxv2 -> mxp1 : shft mxps1}
            AY0=DM(I1,M1),SR=SR OR LSHIFT MR1 (HI);           {get v1 -> v2 : mxp2/2}
                                                     {mxv/2 answer is in SR1 SR0}
            AY1=DM(I1,M2) ,AR=SR0-AY0;               {get v2 : mxv1/2 - v1}
            AR=SR1-AY1+C-1;                          {mxv2/2 - v2}
            IF LT JUMP Inc_Seg2_Cnt;                 { if tgv/2 < v inc the smpl count}
{Dec_Seg2_Cnt:}
            I0=^SegCntr;                             {point to the seg2 1st half counter}
            AY0=DM(I0,M1);                           {get   cnt1 : -> cnt2}
            AX1=DM(I0,M2), AR=AY0-1;                 {get   cnt2 : -> cnt1 : cnt1 - 1}
            DM(I0,M1)=AR,   AR=AX1+C-1;              {store cnt1 : -> cnt2 : cnt2 - c}
            DM(I0,M1)=AR;                            {store cnt2 : -> xxxx}
            IF GT JUMP ExtSeg2S;                     {continue if cnt > 0}

            I4=^Segment3a;                           {get seg 3a ready}
            DM(SeqToDo)=I4;                 {store}
            JUMP ExtSeg2S;                           {finish the last seq of segment 2}
Inc_Seg2_Cnt:
            I0=^SegCntr;                             {point to the seg2 1st half counter}
            AY0=DM(I0,M1);                           {get   cnt1 : -> cnt2}
            AX1=DM(I0,M2), AR=AY0+1;                 {get   cnt2 : -> cnt1 : cnt1 + 1}
            DM(I0,M1)=AR,   AR=AX1+C;                {store cnt1 : -> cnt2 : cnt2 + c}
            DM(I0,M2)=AR;                            {store cnt2 : -> cnt1}
ExtSeg2S:
            AR=DM(GainToDo);                {Point at the Gain to do}
            TOPPCSTACK=AR;                           {put it on the stack}
            RTS;                            {go to the gain routine}

{**************************************************************************}
{**************************************************************************}
```

```
            { lower the acl by the jerk val until at top vel, or the acl <= 0}
                                { p = p+v->store p }
                                { v = v+a->store v }
                                { a = a-j->store a }
Segment3a:
            I4=^Segment3;                                   {if yes - get seg 3 ready}
            DM(SeqToDo)=I4;                         {store}
Segment3:
            I0=^DesPos;                                     {get the pos buf pointer}
            I1=^DesVel;                                     {get the vel buf pointer}
            SE=-8;
            MR0=DM(I1,M1);                                  {get v1 ->v2}
            MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);               {get v2 ->v1 : shift the LSW v1}
            AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);                 {get p1 ->p2 : shft MSW v2}
            AY1=DM(I0,M1), AR=SR0+AY0;      {get p2 ->p3 : p1=p1+v1}
            AX0=DM(I0,M3);                                  {get p3 ->p1  : p2=p2+v2}
            DM(I0,M1)=AR, AR=SR1+AY1+C;                     {store p1 ->p2}
            DM(I0,M1)=AR,  AR=AX0+C;                        {store p2   ->p3 : p3=p3+c}
            DM(I0,M3)=AR;                                   {store p3 -> p1}
            SE=-1;

            {**********************************}
             { Vel = Vel + Acl   (32.32 + 32.32) }
                        { v = v+a->store v }
Seg3_Vel:
            I0=^DesAcl;                                     {get the acl buf pointer : I1=velptr}
            AX0=DM(I0,M1);                                  {get a1    : ->a2}
            AY0=DM(I1,M1);                                  {get v1    : ->v2}
            AX1=DM(I0,M2), AR=AX0+AY0;                      {get a2    : ->a1   : v1=v1+a1}
            AY1=DM(I1,M2);                                  {get v2    : ->v1}
            AY0=AR, AR=AX1+AY1+C;                           {save v1   : v2=v2+a2}
            AY1=AR;                                         {save v2}

            I0=^TgtVel;                                     {get the tgtvel}
            AX0=DM(I0,M1);                                  {get tgv1 -> tgv2}
            AX1=DM(I0,M2), AR=AX0-AY0;                      {get tgv2 -> tgv1  : tgv1-v1}
            AR=SR1-AY1+C-1;                                 {tgv2-v2}
            IF GT JUMP Seg3Acl;                             { if v < tgtvel continue seg3}
                                                            { else set the vel to the mx vel}
            DM(I0,M1)=AY0;                          {store   Mxvel1 in v1 : -> v2}
            DM(I0,M2)=AY1;                          {store   Mxvel2 in v2 : -> v1}

            I4=^Segment4b;                                  {vel >= Mxvel .. set seg 4 flag}
            DM(SeqToDo)=I4;                         {Point at Seg 2}

            AR=DM(GainToDo);                                {Point at the Gain to do}
            TOPPCSTACK=AR;                                  {put it on the stack}
            RTS;                                            {go to the gain routine}

            {**********************************}
              { Acl = Acl - Jrk  (16.32 + .32)  }
Seg3Acl:
            DM(I0,M1)=AY0;                          {store vel1 in v1 : -> v2}
            DM(I0,M2)=AY1;                          {store vel2 in v2 : -> v1}

            I1=^AclJrk;                                     {get the Ajrk buf address}
            AX0=DM(I1,M1);                                  {get j1       : ->j2}
```

```
        AY0=DM(I0,M1);                          {get a1      : ->a2}
        AX1=DM(I0,M2), AR=AX0-AY0;              {get a2      : ->a1   : a1=a1+j1}
        AY1=DM(I1,M2);                          {get j2          : ->j1}
        AX0=AR, AR=AX1-AY1+C-1;                 {save a1 : compute a2=a2+c}
        IF LE JUMP ExtSeg3;                     { if <= 0 do not store : goto gain }

        DM(I0,M1)=AX0;                  {store in a1 : ->   a2}
        DM(I0,M2)=AR;                           {store in a2 : ->   a1}

        {**********************************}
ExtSeg3:
        AR=DM(GainToDo);                        {Point at the Gain to do}
        TOPPCSTACK=AR;                          {put it on the stack}
        RTS;                                    {go to the gain routine}

{****************************************************************************}
{****************************************************************************}

Segment4a:
        I4=^Segment4;                           {if yes - get seg 4 ready}
        DM(SeqToDo)=I4;                 {store}
        {2}

Segment4b:
        AX0=0x2;                                {set AtSpeed bit=1}
        DM(SpdFlg)=AX0;                 {store indicator}

        {First In - if at the dcl point, set the STOP flag}
        {bit 0 = calc needs to be done}
        {bit 1 = calc is in progress}
        {bit 2 = calc is done}
        {bit 3 = move is in VelMode}

        AX0=DM(CalcDclPt);                      {get the calcs flag}
        AR=PASS AX0;                            {test the dcl calc flag}
        IF NE JUMP Segment4;                    {bypass flag setting if already done}
                                                {this might have been done in MAIN}
        AX0=0x01;                               {set the do a dcl calc bit}
        DM(CalcDclPt)=AX0;                      {tell exit routine to do a dp calc}

Segment4:
        {**********************************}
        { Pos = Pos + Vel  (32.32 + 16.32) }

        I0=^DesPos;                             {get the pos buf pointer}
        I1=^DesVel;                             {get the vel buf pointer}
        SE=-8;
        MR0=DM(I1,M1);                          {get v1 ->v2}
        MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);       {get v2 ->v1 : shift the LSW v1}
        AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);         {get p1 ->p2 : shft MSW v2}
        AY1=DM(I0,M1), AR=SR0+AY0;      {get p2 ->p3 : p1=p1+v1}
        AX0=DM(I0,M3);                          {get p3 ->p1  : p2=p2+v2}
        DM(I0,M1)=AR, AR=SR1+AY1+C;             {store p1 ->p2}
        DM(I0,M1)=AR,  AR=AX0+C;                {store p2   ->p3 : p3=p3+c}
        DM(I0,M3)=AR;                           {store p3 -> p1}
        SE=-1;
        {**********************************}
        { if (p >= decl point)
                if mode = Trap begin segment 6
                if mode = 'S'  begin segment 5
                else continue with segment 4
        }

        AY0=0x03;                               {Set the I'm doing it bit}
        AX0=DM(CalcDclPt);                      {get DP calc flag}
        AR=AX0 - AY0;                           {is the DP in place ? }
        IF NE JUMP DP_Bypass;                   {if not ready go around}
        {4}
        I0=^DclPos;                             {point at the decel point}
        I1=^DesPos;                             {point at the pos buf}
        {2}
        AX0=DM(I0,M1);                          {get dpt1 -> dpt2}
        AY0=DM(I1,M1);                          {get p1 -> p2}
        AX1=DM(I0,M1),  AR=AX0-AY0;             {get dpt2 -> dpt3 : dpt1-p1}
        AY1=DM(I1,M1);                          {get p2 -> p3}
        AX0=DM(I0,M2),  AR=AX1-AY1+C-1;         {get dpt3 -> dpt1 : dpt2-p2}
        AY0=DM(I1,M2);                          {get p3 -> p1}
        AR=AX0-AY0+C-1;                         {dpt3-p3}
        IF LE JUMP Segment4c;                   {if <= 0 time to stop}

        {**********************************}
DP_Bypass:
        AX0=DM(ForceTgFlg);
        AR=PASS AX0;                            {test the stop flag}
        IF EQ JUMP Segment4d;                   {if=0 continue seg4}

        {**********************************}
Segment4c:
        {Save DesVel in mxvel for 'S and 'T curve decl calcs}
        I0=^DesVel;
        I1=^TgtVel;
        AX0=DM(I0,M1);                          {get v1 -> v2}
        DM(I1,M1)=AX0;                          {store in mxv1->mxv2}
        AX0=DM(I0,M2);                          {get v2 -> v1}
        DM(I1,M2)=AX0;                          {store in mxv2 -> mxv1}

        I0=^DclJrk;                             {get the dcl jrk pointer}
        AY0=DM(I0,M1);                          {get   j1 : -> j2}
        AX0=DM(I0,M2);                  {get   j2 : -> j1}
        AR=AX0 OR AY0;                  {jx = j1+j2}
        IF EQ JUMP Segment4d;                   { if Jrk==0 do the trap down}

        I4=^Segment5a;                          {if yes - get seg 5 ready}
        DM(SeqToDo)=I4;                 {store}

        AR=DM(GainToDo);                        {Point at the Gain to do}
        TOPPCSTACK=AR;                          {put it on the stack}
        RTS;                                    {go to the gain routine}

        {**********************************}
Segment4d:
        I4=^Segment6a;                          {if yes - get seg 6 ready}
        DM(SeqToDo)=I4;                 {store}
        AR=DM(GainToDo);                        {Point at the Gain to do}
        TOPPCSTACK=AR;                          {put it on the stack}
        RTS;                                    {go to the gain routine}

{***************************************************************************}
{***************************************************************************}

Segment5a:
        AX0=0x4;                                {set dcl flag}
        DM(SpdFlg)=AX0;                 {store indicator}

        I4=^Segment5;                           {if yes - get seg 5 ready}
        DM(SeqToDo)=I4;                 {store}

Segment5:
          {             DO THE S DCL Curve_Dn
                          P = P + V
                          V = V + A
          *      if v <= Mxv/2 then set segment 6 flag
          *     if a >= Mxdcl then a=Mxdcl - begin segment 6

                 **********************************}
                  { Pos = Pos + Vel  (32.32 + 16.32) }
                        { p = p+v->store p }

        I0=^DesPos;                             {get the pos buf pointer}
        I1=^DesVel;                             {get the vel buf pointer}
        SE=-8;
        MR0=DM(I1,M1);                          {get v1 ->v2}
        MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);       {get v2 ->v1 : shift the LSW v1}
        AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);         {get p1 ->p2 : shft MSW v2}
        AY1=DM(I0,M1), AR=SR0+AY0;      {get p2 ->p3 : p1=p1+v1}
        AX0=DM(I0,M3);                          {get p3 ->p1  : p2=p2+v2}
        DM(I0,M1)=AR, AR=SR1+AY1+C;             {store p1 ->p2}
        DM(I0,M1)=AR,  AR=AX0+C;                {store p2   ->p3 : p3=p3+c}
        DM(I0,M3)=AR;                           {store p3 -> p1}
        SE=-1;

        {**********************************}
          { Vel = Vel + Acl   (16.32 + 16.32) }

        I0=^DesDcl;                             {get the decel pointer}
        AX0=DM(I0,M1);                          {get d1    : ->d2}
        AY0=DM(I1,M1);                          {get v1    : ->v2}
        AX1=DM(I0,M2), AR=AX0+AY0;              {get a2    : ->a1   : v1=v1+a1}
        AY1=DM(I1,M2);                          {get v2    : ->v1}
        DM(I1,M1)=AR,  AR=AX1+AY1+C;            {store v1  : ->v2   : v2=v2+a2}
        DM(I1,M2)=AR;                           {store v2  : ->v1}

        I0=^TgtVel;                             {get the tgtvel}
        SE=-1;
        MR0=DM(I0,M1);                          {get tgv1 -> tgv2}
        MR1=DM(I0,M2), SR=LSHIFT MR0 (LO);      {get tgv2 -> tgv1 : tgv1/2}
        AY0=DM(I1,M1), SR=SR OR LSHIFT MR1 (HI);        {get v1 -> v2 : tgv2/2}
        {answer is in SR1 SR0}

        AY1=DM(I1,M2), AR=SR0-AY0;              {get v2 -> v1  : a=tgv1/2-v1}
```

```
            AR=SR1-AY1+C-1;                         {a=tgv2/2-v2}
            IF LT JUMP Seg5a;                       {is v < tgtvel/2 ?}
                                                    { if yes -------------+}
            I4=^Segment6;                           {get seg 6 ready <----+}
            DM(SeqToDo)=I4;                 {store}

            AR=DM(GainToDo);                        {Point at the Gain to do}
            TOPPCSTACK=AR;                          {put it on the stack}
            RTS;                                    {go to the gain routine}

            {**********************************}
              { Acl = Acl - DJrk  (16.32 + .32)  }
Seg5a:
            I1=^DclJrk;                             {get the Djerk pointer}
            AY0=DM(I1,M1);                          {get Dj1 -> Dj2}
            AX0=DM(I0,M1);                          {get d1  -> d2}
            AY1=DM(I1,M2),  AR=AX0-AY0;             {get Dj2 -> Dj1 : d1=d1-j1}
            AX1=DM(I0,M2);                          {get d2  -> d1}
            DM(I0,M1)=AR,   AR=AX1-AY1+C-1
                                                    {store d1 -> d2 : d2=d2-j2+c-1}
            DM(I0,M2)=AR;                           {store d2 -> d1}

            I1=^TgtDcl;                             {get the MxDcl pointer}
            AX0=DM(I1,M1);                          {get   mxd1 -> mxd2}
            AY0=DM(I0,M1);                  {get   d1 -> d2}
            AX1=DM(I1,M2),  AR=AX0+AY0;             {get   mxd2 -> mxd1 : mxd1 - d1}
            AY1=DM(I0,M2);                          {get   d2 -> d1}
            AR=AX1+AY1+C;                   {mxd2 - d2}
            IF GT JUMP ExtSeg5;                     { if > 0 continue}

            I4=^Segment6;
            DM(SeqToDo)=I4;                 {Point at Seg 5}

            {**********************************}

ExtSeg5:
            AR=DM(GainToDo);                        {Point at the Gain to do}
            TOPPCSTACK=AR;                          {put it on the stack}
            RTS;                                    {go to the gain routine}

{*************************************************************************}
{*************************************************************************}
{*************************************************************************}
{*************************************************************************}
{*************************************************************************}
            {get the tgpos-pos .. if despos >= set the pos=tgpos set segment8}

            I0=^TgtPos;                             {get the Tgpos buf pointer}
            I1=^DesPos;                             {pnt at the despos}
            AY0=DM(I1,M1);                          {get p1 -> p2}
            AY0=DM(I1,M1);                          {get p1 -> p3}
            AX0=DM(I0,M1);                          {get tp1 -> tp2}
            AY1=DM(I1,M3), AR=AX0-AY0;              {get p3 -> p1 : tgp1 - p2}
            AX0=DM(I0,M2);                          {get tp2 -> tp1}
            AR=MR0-AY1+C-1;                         {tgp2 - p3}
            IF GT JUMP Seg6_OK;                     { if > 0 continue}
```

```
        I4=^Segment8b;                          {end of move}
        DM(SeqToDo)=I4;                  {Point at Seg 5}

        AR=DM(GainToDo);                        {Point at the Gain to do}
        TOPPCSTACK=AR;                          {put it on the stack}
        RTS;                                    {go to the gain routine}

{**************************************************************************}
{**************************************************************************}
{**************************************************************************}
{**************************************************************************}

Segment6a:
        {clear the segment counter for seg6 'S requirement}
        I0=^SegCntr;                            {point to the seg counter}
        AX0=0x0;                                {set 0}
        DM(I0,M3)=AX0;                          {store segcnt1}
        DM(I0,M3)=AX0;                          {store segcnt2}

        AX0=0x4;                                {set dcl flag}
        DM(SpdFlg)=AX0;                  {store indicator}

        I4=^Segment6;                           {if yes - get seg 6 ready}
        DM(SeqToDo)=I4;                  {store}

Segment6:
        I0=^DclJrk;                             {point at the Djerk buf}
        AY0=DM(I0,M1);                          {get   Dj1 : -> Dj2}
        AX0=DM(I0,M2);                   {get   Dj2 : -> Dj1}
        AR=AX0 OR AY0;                   {Djx = Dj1+Dj2}
        IF NE JUMP Seg6_S_Chk;                  {if 'S continue the acl ramp}

Seg6_S_Chk:
        {AT THIS POINT SEG2 IS DOING THE TRAP MOVE}
                                                {is DesPos>=TgtPos-1?}
        I0=^DesDcl;                             {get the decel pointer}
        I1=^TgtDcl;
        AX0=0x0;                                {set 0}
        AY0=DM(I1,M1);                          {get tgd1 ->tgd2}
        AY1=DM(I1,M2), AR=AX0-AY0;              {get tgd2 ->tgd1 : ar=-d1}
        DM(I0,M1)=AR, AR=AX0-AY0+C-1;           {save d1 ->d2 : ar=-d2}
        DM(I0,M1)=AR;                           {save d2}


        I0=^TgtPos;                             {get the Tgpos buf pointer}
        I1=^DesPos;                             {pnt at the despos}
        AY0=DM(I1,M1);                          {get p1 -> p2}
        AY0=DM(I1,M1);                          {get p2 -> p3}
        AX0=DM(I0,M1);                          {get tp1 -> tp2}
        AY1=DM(I1,M3), AR=AX0-AY0;              {get p3 -> p1 : tgp1 - p2}
        AX1=DM(I0,M2);                          {get tp2 -> tp1}
        AY0=AR, AR=AX1-AY1+C-1;                 {save the result : tgp2 - p3}
        AX0=AR, AR=AY0-1;                       {save the result, ans - 1}
        AR=AX0+C-1;                             {do the top half of ans -1}
        IF GT JUMP Seg6_OK;                     { if > 0 continue}

        I4=^Segment8b;                          {end of move}
```

```
          DM(SeqToDo)=I4;                       {Point at Seg 5}

          AR=DM(GainToDo);                              {Point at the Gain to do}
          TOPPCSTACK=AR;                                {put it on the stack}
          RTS;                                          {go to the gain routine}

Seg6_Counter:
          {AT THIS POINT SEG2 IS DOING THE S MOVE}
          { if (v <= Mxv/2)
                    1.         Seg6Cntr = Seg6Cntr - 1
                    2.         if      Seg6Cntr <= 0 jump to segment7
                               else  Seg6Cntr += 1
          }

          I0=^TgtVel;                                   {get the mxvel pointer}
          I1=^DesVel;                                   {get the vel pointer}

          MR0=DM(I0,M1);                                {get mxv1 : -> mxv2}
          MR1=DM(I0,M2), SR=LSHIFT SR0 (LO);            {get mxv2 -> mxv1 : mxv1/2}
          AY0=DM(I1,M1), SR=SR OR LSHIFT SR1 (HI);              {get v1->v2 : mxv2/2}
                                                        {answer is in SR1.SR0 MR0}
          AY1=DM(I1,M2),  AR=SR0-AY0;                   {get   v2->v2 : mxv1/2 - v1}
          AR=SR1-AY1+C-1;                               {mxv2/2 - v2}
          IF LT JUMP Inc_Seg6_Cnt;                      { if < 0 continue}

{Dec_Seg6_Cnt:}
          I0=^SegCntr;                                  {point to the seg2 1st half counter}
          AY0=DM(I0,M1);                                {get   cnt1 : -> cnt2}
          AX1=DM(I0,M2), AR=AY0-1;                      {get   cnt2 : -> cnt1 : cnt1 - 1}
          DM(I0,M1)=AR,   AR=AX1+C-1;                   {store cnt1 : -> cnt2 : cnt2 - c}
          DM(I0,M1)=AR;                                 {store cnt2 : -> xxxx}
          IF GT JUMP Seg6_OK;                           {continue if cnt > 0}

          I4=^Segment7;                                 {get seg 3 ready}
          DM(SeqToDo)=I4;                       {store Seg 3}
          JUMP Seg6_OK;                                 {finish the last seq of segment 2}

Inc_Seg6_Cnt:
          I0=^SegCntr;                                  {point to the seg2 1st half counter}
          AY0=DM(I0,M1);                                {get   cnt1 : -> cnt2}
          AX1=DM(I0,M2), AR=AY0+1;                      {get   cnt2 : -> cnt1 : cnt1 + 1}
          DM(I0,M1)=AR,   AR=AX1+C;                     {store cnt1 : -> cnt2 : cnt2 + c}
          DM(I0,M2)=AR;                                 {store cnt2 : -> cnt1}

          {**********************************}
Seg6_OK:
            { Pos = Pos + Vel  (32.32 + 16.32) }
                    { p = p+v->store p }

          I0=^DesPos;                                   {get the pos buf pointer}
          I1=^DesVel;                                   {get the vel buf pointer}
          SE=-8;
          MR0=DM(I1,M1);                                {get v1 ->v2}
          MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);             {get v2 ->v1 : shift the LSW v1}
          AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);               {get p1 ->p2 : shft MSW v2}
          AY1=DM(I0,M1), AR=SR0+AY0;            {get p2 ->p3 : p1=p1+v1}
          AX0=DM(I0,M3);                                {get p3 ->p1  : p2=p2+v2}
          DM(I0,M1)=AR, AR=SR1+AY1+C;              {store p1 ->p2}
          DM(I0,M1)=AR,  AR=AX0+C;                 {store p2   ->p3 : p3=p3+c}
          DM(I0,M3)=AR;                            {store p3 -> p1}
          SE=-1;

          {*********************************}
           {note: d is a neg val}
           { Vel = Vel + Dcl   (16.32 + 16.32) }
                   { v = v+d->store v }

          I0=^DesDcl;                              {get the decel pointer}
          AX0=DM(I0,M1);                           {get d1    : ->d2}
          AY0=DM(I1,M1);                           {get v1    : ->v2}
          AX1=DM(I0,M2), AR=AX0+AY0;               {get d2    : ->d1   : v1=v1+d1}
          AY1=DM(I1,M2);                           {get v2    : ->v1}
          DM(I1,M1)=AR,  AR=AX1+AY1+C;             {store v1  : ->v2   : v2=v2+d2}
          DM(I1,M2)=AR;                            {store v2  : ->v1}
          IF GT JUMP ExtSeg6;
                                                   {set a vel in place}
          AY1=0x0;                                 {load v1}
          DM(I1,M1)=AR;                            {store v1->v2}
          AY1=0x01;                                {load v2}
          DM(I1,M2)=AR;                            {store v2->v1}

          {*********************************}

ExtSeg6:
          AR=DM(GainToDo);                         {Point at the Gain to do}
          TOPPCSTACK=AR;                           {put it on the stack}
          RTS;                                     {go to the gain routine}

{************************************************************************}
{************************************************************************}

Segment7a:
          I4=^Segment7;                            {if yes - get seg 7 ready}
          DM(SeqToDo)=I4;                  {store}

Segment7:
          {if p >= Tgpos then p = Tgpos .. end move}
          {if v <= Dj then v= Djrk}
             { Pos = Pos + Vel  (32.32 + 16.32) }
                   { p = p+v->store p }

          I0=^DesPos;                              {get the pos buf pointer}
          I1=^DesVel;                              {get the vel buf pointer}
          SE=-8;
          MR0=DM(I1,M1);                           {get v1 ->v2}
          MR1=DM(I1,M2),SR=LSHIFT MR0 (LO);        {get v2 ->v1 : shift the LSW v1}
          AY0=DM(I0,M1),SR=SR OR LSHIFT MR1 (HI);          {get p1 ->p2 : shft MSW v2}
          AY1=DM(I0,M1), AR=SR0+AY0;       {get p2 ->p3 : p1=p1+v1}
          AX0=DM(I0,M3);                           {get p3 ->p1  : p2=p2+v2}
          DM(I0,M1)=AR, AR=SR1+AY1+C;              {store p1 ->p2}
          DM(I0,M1)=AR,  AR=AX0+C;                 {store p2   ->p3 : p3=p3+c}
          DM(I0,M3)=AR;                            {store p3 -> p1}
          SE=-1;
```

```
        {**********************************}



         {note: d is a neg val}
         { Vel = Vel + Dcl   (16.32 + 16.32) }
                 { v = v+d->store v }

        I0=^DesDcl;                          {get the decel pointer}
        AX0=DM(I0,M1);                       {get d1    : ->d2}
        AY0=DM(I1,M1);                       {get v1    : ->v2}
        AX1=DM(I0,M2), AR=AX0+AY0;           {get d2    : ->d1   : v1=v1+d1}
        AY1=DM(I1,M2);                       {get v2    : ->v1}
        DM(I1,M1)=AR,  AR=AX1+AY1+C;         {store v1  : ->v2   : v2=v2+d2}
        DM(I1,M2)=AR;                        {store v2  : ->v1}
        IF GT JUMP ExtSeg6;
                                             {set a vel in place}
        AY1=0x0;                             {load v1}
        DM(I1,M1)=AR;                        {store v1->v2}
        AY1=0x01;                            {load v2}
        DM(I1,M2)=AR;                        {store v2->v1}

        {**********************************}
          { Acl = Acl + Jrk  (16.32 + .32)  }

        I1=^DclJrk;                          {get the Djrk buf address}
        AX0=DM(I1,M1);                       {get j1       : ->j2}
        AY0=DM(I0,M1);                       {get d1       : ->d2}
        AX1=DM(I0,M2), AR=AX0+AY0;           {get d2       : ->d1   : d1=d1+j1}
        AY1=DM(I1,M2);                       {get j2           : ->j1}
        DM(I0,M1)=AR,  AR=AX1+AY1+C;         {store d1     : ->d2   : d2=d2+c}
        DM(I0,M2)=AR;                        {store d2     : ->d1}
        IF LT JUMP ExtSeg7;                  { if < 0 continue}
                                             { else set acl=mxacl}
        AX0=0x0;
        DM(I0,M1)=AX0;                {store   d1 -> d2}
        DM(I0,M2)=AX0;                {store   d2 -> d1}
ExtSeg7:
        I0=^TgtPos;                          {get the Tgpos buf pointer}
        I1=^DesPos;                          {pnt at the despos}
        AY0=DM(I1,M1);                       {get p1 -> p2}
        AY0=DM(I1,M1);                       {get p2 -> p3}
        AX0=DM(I0,M1);                       {get tp1 -> tp2}
        AY1=DM(I1,M3), AR=AX0-AY0;           {get p3 -> p1 : tgp1 - p2}
        AX1=DM(I0,M2);                       {get tp2 -> tp1}
        AY0=AR, AR=AX1-AY1+C-1;              {save the result : tgp2 - p3}
        AX0=AR, AR=AY0-1;                    {save the result, ans - 1}
        AR=AX0+C-1;                          {do the top half of ans -1}
        IF GT JUMP ESeg7;                    { if > 0 continue}

        I4=^Segment8b;                       {end of move}
        DM(SeqToDo)=I4;               {Point at Seg 5}
ESeg7:
        AR=DM(GainToDo);                     {Point at the Gain to do}
        TOPPCSTACK=AR;                       {put it on the stack}
        RTS;                                 {go to the gain routine}
```

```
{***************************************************************************}
{***************************************************************************}

Segment8a:
            I4=^Segment8;                                   {if yes - get seg 8 ready}
            DM(SeqToDo)=I4;                     {store}
Segment8b:
            AX0=0x8;                                        {set TC}
            DM(SpdFlg)=AX0;                     {store indicator}
Segment8:
            AR=DM(GainToDo);                                {Point at the Gain to do}
            TOPPCSTACK=AR;                                  {put it on the stack}
            RTS;                                            {go to the gain routine}

{***************************************************************************}
{***************************************************************************}
                                    {DO THE KP GAIN HERE}
DoKPGain:
            {load axis 1 buffer pointers}
            I0=^KpGain;                                     {KpErr:LastKpDAC1/2}
            I1=^KpError;                                    {Load from the ASIC}
            I2=^TempDAC;

{           Fe1 *  Kp
            pointers  I0=Kp gain, I1=Fe, I2=TempDAC
}
            MX0=DM(I0,M1);                      {get Kp1 : I0->KLKpDac1}
            MY0=DM(I1,M1);                      {get FE1 : I1->tFE1}
            MR=MX0*MY0 (US);                                {x0*y0   signed FE}
                                                            {Get the Last KpDac Value}
            AY0=DM(I0,M1),SR=LSHIFT MR0 (LO);               {Shift LSW right 1 bits}
                                                            {get the Temp DAC1 value}
            AX0=DM(I2,M1),SR=SR OR LSHIFT MR1 (HI);                 {Shift MSW right 1 bits}

            AY1=DM(I0,M1), AR=AX0-AY0;                      {sub LKpdac from Dac->tDAC1}
            AX1=DM(I2,M2);                                  {get the DAC2 value ->tDAC1}
            AY0=AR, AR=AX1-AY1+C-1;                         {sub LKpdac from Dac->tDAC2}
            {AX0=LSW TDac1  AR=MSW TDac2}

            AY1=AR, AR=SR0+AY0;                             {xfer MSW : add new Kp1DAC}
            DM(I2,M1)=AR, AR=MR0+AY1+C;                     {save LSW : add new Kp2DAC}
            DM(I2,M2)=AR;                                   {save MSW}

            I4=^DoKIGain;                                   {reset the gain pointer}
            DM(GainToDo)=I4;                                {Point at the Ki}
            JUMP SetDAC;                                    {set the control voltage}

{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
                                    {DO THE KI GAIN HERE}
DoKIGain:
{            Te2 Te1  *  Ki
            pointers  I0=Ki gain, I1=TotFe, I2=TempDAC
}
            I0=^KiGain;                                     {Ki gain}
            I1=^KiError;                                    {Load total error from ASIC}
            I2=^TempDAC;
            MX0=DM(I0,M1);                              {get Ki1 : I0->+IL1}
            MY0=DM(I1,M1);                      {get TFE1: I1->TFE2}
                                                        {get TFE2: I1->DeltaKdEr}
            MY1=DM(I1,M1),MR=MX0*MY0(UU);               {Compute LSW}
            SR=LSHIFT MR0(LO);                          {Shift LSW right 1 bit}
            SR=SR OR LSHIFT MR1(HI);                    {before saving}
            AX1=SR0;                                    {Save LSW}
            MR0=MR1;                                    {shift 16 right}
            MR=MR+MX0*MY1(US);
            SR=LSHIFT MR0 (LO);                         {Shift LSW right 1 bits}
            SR=SR OR LSHIFT MR1 (HI);                   {Shift MSW right 1 bits}
            {SR0:MR0}                                   {Save LSW:MSW}

            {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

            {if IL > TKiDAC) then set TKiDAC = IL}

            AY0=0x8000;                                 {set the neg bit}
            AF = SR0 AND AY0;                           {test for a neg err val}
            IF NE JUMP TestNegLim;

            AY0=DM(I0,M1);                              {get the +IL1 -> KdCntr}
            AR=SR0-AY0;                                 {get MSW - Lim}
            IF POS JUMP SetIl;                          {if il lim > KiEr exit}
            JUMP SetMaxIl;
TestNegLim:
            AY0=DM(I0,M1);                              {get the +IL1 -> KdCntr}
            AR=SR0-AY0;                                 {get MSW - Lim}
            IF NEG JUMP SetIL;                          {yes - set the tdac = KiEr}
SetMaxIL:                                               {set the DACki = -IL}
            AX1=0;                                      {load the max IL1 Lim}
            SR0=AY0;                                    {load the max IL2 Lim}
SetIl:
            AY0=DM(I2,M1);                              {get tDAC1}
            AY1=DM(I2,M2), AR=AX1+AY0;                  {get tDAC2}
            DM(I2,M1)=AR,  AR=SR0+AY1+C;                {save DACp1 in tDAC}
            DM(I2,M2)=AR;                               {save DACp2}

            I4=^DoKDGain;
            DM(GainToDo)=I4;                            {Point at the Ki}
            JUMP SetDAC;                                {set the control voltage}

{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
                              {DO THE KD GAIN HERE}
DoKDGain:
{            DeltaEr  *  Kd
}
            I0=^KdGain;                                 {->kdcounter}
            I1=^KdError;                                {Load Delta error from ASIC}
            I2=^TempDAC;

            AY0=DM(I0,M0);                              {Get the KD count}
            AR = AY0 -1;                                {decrement the count}
            DM(I0,M1)=AR;                               {store the new KD count -> cntval}
            AR=DM(I0,M1);                               {nop to inc the reg -> Kd}
            IF GE JUMP ExtKDGain;                       {jump if KDcount >= 0}
            AX0=DM(I0,M2);                          {nop to -> KdUpdtCntr}
            AX0=DM(I0,M2);                          {get the KDUpdtCnt -> KdCntr}
            DM(I0,M3)=AX0;                          {reload the KDcntr -> KdtempDAC1}
            AX0=DM(I0,M2);                          {nop to -> Kd}

            {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

            MX0=DM(I0,M1);                  {get Kd1 : I0->Ka1}
            MY0=DM(I1,M1);                  {get DeltaEr : I1-> AclEr1}
            MR=MX0*MY0 (US);                        {x0*y0}
            SR=LSHIFT MR0 (LO);                     {Shift LSW right 1 bits}
            SR=SR OR LSHIFT MR1 (HI);               {Shift MSW right 1 bits}
                                                    {add the Kd to the DAC}
            AY0=DM(I2,M1);                          {get tDAC1}
            AY1=DM(I2,M2), AR=MR0+AY0;              {get tDAC2}
            DM(I2,M1)=AR,  AR=SR0+AY1;              {save DACp1 in tDAC}
            DM(I2,M2)=AR;                           {save DACp2}

            {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

ExtKDGain:

            I4=^DoKAGain;
            DM(GainToDo)=I4;                        {Point at the Ki}
            JUMP SetDAC;                            {set the control voltage}

{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
                                    {DO THE KA GAIN HERE}
DoKAGain:
{           Ae1 *  Ka
}
            I0=^KaGain;                             {->kacounter}
            I1=^KaError;                            {Load Delta error from ASIC}
            I2=^TempDAC;

            AY0=DM(I0,M0);                          {Get the KA count}
            AR = AY0 -1;                            {decrement the count}
            DM(I0,M1)=AR;                           {store the new KA count -> cntval}
            AR=DM(I0,M1);                           {nop to inc the reg -> Ka}
            IF GE JUMP ExtKAGain;                   {jump if KAcount >= 0}

            AX0=DM(I0,M2);                          {nop to -> KaUpdtCntr}
            AX0=DM(I0,M2);                          {get the KAUpdtCnt -> KaCntr}
            DM(I0,M3)=AX0;                          {reload the KAcntr -> KatempDAC1}
            AX0=DM(I0,M2);                          {nop to -> Ka}

            {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

            MX0=DM(I0,M1);                  {get Ka1 : I0->Kv1}
            MY0=DM(I1,M1);                  {get AclEr : I1-> VelEr1}
            MR=MX0*MY0 (US);                        {x0*y0}
            SR=LSHIFT MR0 (LO);                     {Shift LSW right 1 bits}
            SR=SR OR LSHIFT MR1 (HI);               {Shift MSW right 1 bits}
            MR0=SR0;                                {save the lobyte -> tDAC2}
            SR0=SR1;                                {swap the hi byte}
            SR=SR OR LSHIFT MR2 (HI);               {Shift LSW right 1 bits}
                                                    {add the Ka to the DAC}
            AY0=DM(I2,M1);                          {get tDAC1}
            AY1=DM(I2,M2), AR=MR0+AY0;              {get tDAC2}
            DM(I2,M1)=AR,  AR=SR0+AY1;              {save DACp1 in tDAC}
            DM(I2,M2)=AR;                           {save DACp2}
ExtKAGain:
            I4=^DoKAGain;
            DM(GainToDo)=I4;                        {Point at the Ki}
            JUMP SetDAC;                            {set the control voltage}

{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

DoKVGain:
{            Ve1 *  Kv
}
            I0=^KvGain;                             {->kvcounter}
            I1=^KvError;                            {Load Delta error from ASIC}
            I2=^TempDAC;

            AY0=DM(I0,M0);                          {Get the KV count}
            AR = AY0 -1;                            {decrement the count}
            DM(I0,M1)=AR;                           {store the new KV count -> cntval}
            AR=DM(I0,M1);                           {nop to inc the reg -> Kv}
            IF GE JUMP ExtKVGain;                   {jump if KVcount >= 0}

            AX0=DM(I0,M2);                          {nop to -> KvUpdtCntr}
            AX0=DM(I0,M2);                          {get the KvUpdtCnt -> KaCntr}
            DM(I0,M3)=AX0;                          {reload the KVcntr -> KvtempDAC1}
            AX0=DM(I0,M2);                          {nop to -> Kv}

            {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

            MX0=DM(I0,M1);                  {get Ka1 : I0->Kv1}
            MY0=DM(I1,M1);                  {get AclEr : I1-> VelEr1}
            MR=MX0*MY0 (US);                        {x0*y0}
            SR=LSHIFT MR0 (LO);                     {Shift LSW right 1 bits}
            SR=SR OR LSHIFT MR1 (HI);               {Shift MSW right 1 bits}
            MR0=SR0;                                {save the lobyte -> tDAC2}
            SR0=SR1;                                {swap the hi byte}
            SR=SR OR LSHIFT MR2 (HI);               {Shift LSW right 1 bits}
                                                    {add the Ka to the DAC}
            AY0=DM(I2,M1);                          {get tDAC1}
            AY1=DM(I2,M2), AR=MR0+AY0;              {get tDAC2}
            DM(I2,M1)=AR,  AR=SR0+AY1;              {save DACp1 in tDAC}
            DM(I2,M2)=AR;                           {save DACp2}
ExtKVGain:
            I4=^DoKVGain;
            DM(GainToDo)=I4;                        {Point at the Ki}
            {JUMP SetDAC;}                          {set the control voltage}

{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

SetDAC:
            {DAC = 0000 0000 0000|0000 0000 0000 0000 0000}
            {         |--------+--------|                             }

            SE=-12;                                 {shift 12 bits right}
            AY0=0xf000;                             {setup}
```

```
        AX0=DM(I2,M1);                          {get tDAC1}
        AR=AX0 AND AY0;                         {get the upper nibble}
        AX1=DM(I2,M1),SR=LSHIFT AR (LO);        {get tDAC2 : shift}
        AY1=0x0fff;                             {setup}
        AR=AX1 AND AY1;                         {get the lower 3 nibbles}
        SR=SR OR LSHIFT AR (HI);                {shift}
                                                {SR0 holds the DAC}
        AY0=0x8000;                             {add the offset to set DAC to 0}
        AR=SR0+AY0;                             {get the totalized DAC}

        IF AC JUMP LoadPDAC;                    {if over flow occurred}

        AR=AX1 AND AY0;                         {get the upper MSW nibble}
        IF NE JUMP LoadPDAC;                    {if over flow occurred}
        JUMP LoadDAC;
LoadPDAC:
        SR0=0xffff;                             {set max positive}
LoadDAC:
        DM(I2,M0)=SR0;                          {set the DAC output}
        SE=-1;                                  {shift 12 bits right}

        AX0=DM(CalcDclPt);                      {check if a calc has been requested}
        AY0=0x01;                               {Check the do-it bit}
        AR=AX0 AND AY0;
        IF EQ JUMP ExtDAC;                      {exit if not requested}

        AY0=0x02;                               {Get the I'm doing it bit}
        AR=AX0 AND AY0;                         {in progress ? }
        IF EQ JUMP ExtDAC;                      {exit if in progress}

        AY0=0x03;                               {Set the I'm doing it bit}
        DM(CalcDclPt)=AY0;                      {store in progress flag}

        I4=^Calc_the_DclPt;                     {get the cal routine addr pointer}
        TOPPCSTACK=I4;                          {set up to return to it on rti}
ExtDAC:
        DIS SEC_REG;                            {reenable the main body data}
        RTI;

        {17 cy}
{****************************************************************************}
{****************************************************************************}

Calc_the_DclPt:
        {save the regs}
        TOPPCSTACK=AX0;
        TOPPCSTACK=AX1;
        TOPPCSTACK=AY0;
        TOPPCSTACK=AY1;
        TOPPCSTACK=MR0;
        TOPPCSTACK=MR1;
        TOPPCSTACK=SR0;
        TOPPCSTACK=SR1;
        TOPPCSTACK=SE;
        TOPPCSTACK=AR;

{
        Pos = 48 bits = 00000000 00000000 00000000 00000000.00000000 00000000 = +/-2^31
        Vel = 32 bits =                     00000000.00000000 00000000 00000000 = 255 cts/smp
        Acl = 32 bits =                     00000000.00000000 00000000 00000000 = 255 cts/smp^2
        Jrk = 32 bits =                     00000000.00000000 00000000 00000000 = 255 cts/smp^3
}
        {if in 'S divide the max acl by 2 and do the math}
        {then:}
        {formula to do s = 1/2 a (or d) t^2 then dp = tgp - s}
        {but t for a or d needs to be in samples = v/a or v/d}
        {and:  for acl ... s = (a >> 2) * (v/a) * (v/a)}
        {      for dcl ... s = (d >> 2) * (v/d) * (v/d)}
        {and:  dp = p - s}
        {then: set the calc complete flag}

        {***********************************}

        I4=^Log1;                               {point to the stor1}
        I5=^TgtDcl;                             {point to the Dcl}
        AX0=DM(I5,M5);                          {get mxd1->mxd2}
        DM(I4,M5)=AX0;                          {store mxd1 -> stor2}
        AX0=DM(I5,M6);                          {get mxd2->mxd1}
        DM(I4,M6)=AX0;                          {store mxd2 -> stor1}

        I5=^AclJrk;                             {get the jrk pointer}
        AR=DM(I5,M5);                           {get   j1 : -> j2}
        AY1=DM(I5,M6);                  {get   j2 : -> j1}
        AR=AR OR AY1;                           {jx = j1+j2}
        IF EQ JUMP Calc_DP;                     {if Jrk==0 do seg6 trap down}

        {If in 'S' mode divide the mxdcl by 2}
        SR=LSHIFT MR0 (LO);                     {lsb/2}
        SR=SR OR LSHIFT MR1 (HI);               {msb/2}
        DM(I4,M5)=SR0;                          {xfer stor1->stor2}
        DM(I4,M6)=SR1;                          {xfer stor2->stor1}

        {***********************************}

Calc_DP:
        { formula to solve ... s=1/2 a (v/a)^2
                               convert v and a to logs
                               do v/a by v-a = ans
                               take the square of v/a by ans+ans
                               convert back to antilog
        }
        {***********************************}
        {1.   convert the v and a to logs}

        I6=^Log1;                               {point to the stored accel}
        CALL Get_Log;
        {LogAcl is in Log1 storage}

        I4=^Log2;                               {point to the stor1}
        I5=^DesVel;                             {point to the Vel}
        AX0=DM(I5,M5);                          {get v1->v2}
        DM(I4,M5)=AX0;                          {store v1 -> v2}
        AX0=DM(I5,M6);                          {get v2->v1}
        DM(I4,M6)=AX0;                          {store v2 -> stor1}
        I6=^Log2;                                   {point to the stored vel}
        CALL Get_Log;                               {LogVel is in Log2 storage}

        {**********************************}
        {2.  divide the LogVel by the LogDcl}
                {logv-loga}
        I4=^Log1;                                   {point to the stored vel}
        I5=^Log2;                                   {point to the stored acl}
        I6=^Result;                                 {point at the result storage}
        AX0=DM(I4,M5);                              {get lv1 -> lv2}
        AY0=DM(I5,M5);                              {get la1 -> la2}
        AX1=DM(I4,M5),  AR=AX0-AY0;                 {get lv2 -> char3v : lv1-la1}
        AY1=DM(I5,M5);                              {get la2 -> char3a}
        DM(I6,M5)=AR,  AR=AX1-AY1+C-1{store ans1 -> ans2 : lv2-la2}
        AX0=DM(I4,M7);                              {get char3v -> lv1}
        AY0=DM(I5,M7);                              {get char3a -> la1}
        DM(I6,M5)=AR,  AR=AX1-AY1+C-1{store ans2 -> ans3 : char3v-char3a}
        DM(I6,M7)=AR;                               {store char3 ans}

        {**********************************}
          {3.  square the v/a answer}
                {logv-loga}

        AX0=DM(I6,M5);                              {get ans1->ans2}
        AY0=AX0;                                    {xfer ans1}
        AX1=DM(I6,M6), AR=AX0+AY0;                  {get ans2->ans1  : ans1=ans1*ans1}
        AY1=AX1;                                    {xfer ans2}
        DM(I6,M5)=AR,  AR=AX1+AY1+C;                {store ans1->ans2: ans2=ans2*ans2}
        DM(I6,M5)=AR;                               {store ans2 ->ans3}
        AX0=DM(I6,M4);                              {get ans3->ans3}
        AY0=AX0;                                    {xfer ans3}
        AR=AX1+AY1+C;                               {ans3=ans3*ans3}
        DM(I4,M7)=AR;                               {store ans3-> ans1}

        {**********************************}
        {4.   convert the result to number}

        I6=^Result;                                 {point to the log to convert back}
        CALL Anti_Log;
        {xfer the final answer to storage}

        {**********************************}
        {5.   mult the result by a}
        {hhhh (int) * h.hhh (fract) = ppppp.ppp}
{
         Integer Triple-Precision Multiplication          Z = X * Y
         Calling Parameters              I4 --> X Buffer
                                         I5 --> Y Buffer
                                         I6 --> Storage for Z
         Return Values                   Z Buffer Filled
         Altered Registers    MX0,MX1,MY0,MY1,MR,I0,I1,I2,SR
}
        I4=^TgtDcl;
        I5=^Result;

        MX0=DM(I5,M5);                              {get ans1->ans2}
        MY0=DM(I4,M5);                              {get ans2->ans3}
```

```
MX1=DM(I5,M5), MR=MX0*MY0(UU);        {get td1->td2 : MR=ans1*td1}
SR=LSHIFT MR0 (LO);                   {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}
MY1=DM(I4,M5);                        {get td2->td3}
AR=SR0, MR=MR+MX0*MY1(UU);            {save 1 : ans=ans1*td2}
SR=LSHIFT MR0(LO);                    {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}

AX0=SR0, MR=MX1*MY0(UU);              {save 2 : MR=ans2*td1}
SR=LSHIFT MR0 (LO);                   {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}
AY0=SR0, MR=MR+MX1*MY1(UU);           {save 3 : ans=ans2*td2}
SR=LSHIFT MR0(LO);                    {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}

MX1=DM(I5,M7);                        {get ans3->ans1}
AX1=SR0, MR=MX1*MY0(UU);              {save 4 : MR=ans3*td1}
SR=LSHIFT MR0 (LO);                   {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}
AY1=SR0, MR=MR+MX1*MY1(UU);           {save 5 : ans=ans3*td2}
SR=LSHIFT MR0(LO);                    {Shift LSW >> 1}
SR=SR OR LSHIFT MR1(HI);              {Shift NSW >> 1}
{SR0}                           {save 6}

DM(I5,M5)=AR, AR=AX0+AY0;             {save ans1->ans2 : ans2=save2+save3}
DM(I5,M5)=AR, AR=AX1+AY1+C;           {save ans1->ans3 : ans2=save4+save5}
DM(I5,M7)=AR, AR=SR0+C;               {save ans3->ans1 : ans2=save6+c}

{**********************************}
   {6.  do (a*(v/a)^2) / 2}
{then shift the answer 8 places right}
{and save lo 3 bytes (dist to dcl bbbb.bb)}

I6=^Result;                           {point at the result storage}
SE=-9;                                {divide by 2 then by eight}
SR0=DM(I6,M5);                        {get ans1->ans2}
MR0=DM(I6,M5), SR=LSHIFT SR0 (LO);    {get ans2->ans3 : shift}
MR1=DM(I6,M7), SR=SR OR LSHIFT MR0 (HI); {get ans3->ans1 :+shift}
DM(I6,M5)=SR0;                        {store ans1->ans2}
SR0=SR1;                              {shift ans2}
SR=SR OR LSHIFT MR1 (HI);             {+shift}
DM(I6,M6)=SR0;                        {store ans2->ans4}
MR1=DM(I6,M6);                        {get ans4->ans3}
SR0=SR1;                              {shift ans3}
SR=SR OR LSHIFT MR1 (HI);             {+shift}
DM(I6,M5)=SR0;                        {store ans3->ans4}
DM(I6,M4)=SR0;                        {store ans4->ans4}
SE=-1;                                {reset the shift value}
{15}
{**********************************}
{7.   get the BP position (p-ans)}

I4=^DesPos;                           {point at p}
I5=^Result;                           {point at the result storage}
I6=^DclPos;                           {point at the DP buf}
AX0=DM(I4,M5);                        {get p1 -> p2}
AY0=DM(I5,M5);                        {get res1 -> res2}
```

```
        AX1=DM(I4,M5),  AR=AX0-AY0;             {get p2 -> p3 : p1-res1}
        AY1=DM(I5,M5);                          {get res2 -> res3}
        DM(I6,M5)=AR,  AR=AX1-AY1+C-1;{store DP1 -> DP2 : p2-res2}
        AX0=DM(I4,M7);                          {get p3 -> p1}
        AY0=DM(I5,M4);                          {get res3 -> res3}
        DM(I6,M5)=AR, AR=AX1-AY1+C-1;           {store DP2-> DP3 : p3-res3}
        DM(I6,M7)=AR;                           {store DP3 -> DP1}

        {**********************************}

        AX0=0x07;                               {set the dcl calc point done bit}
        DM(CalcDclPt)=AX0;                      {store in progress flag}

        {restore the regs}
        TOPPCSTACK=AR;
        TOPPCSTACK=SE;
        TOPPCSTACK=SR1;
        TOPPCSTACK=SR0;
        TOPPCSTACK=MR1;
        TOPPCSTACK=MR0;
        TOPPCSTACK=AY1;
        TOPPCSTACK=AY0;
        TOPPCSTACK=AX1;
        TOPPCSTACK=AX0;
        RTS;                                    {return to the main body routine}

{**************************************************************************}
{**************************************************************************}

Get_Log:
        {wc conversion = 50 cy)
        {convert the value in the LogX reg's to a Log}
        {the pointer to the Logx reg's is I6}
         {AY0:AX1:AX0}
        AX0=DM(I6,M5);                          {get val1->val2}
        AX1=DM(I6,M6);                          {get val2->val1}
        AR=PASS AX1;                            {test val1}
        IF EQ JUMP Tst_Val2;

        {**********************************}

        AY0=0xff00;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high byte of val1}
        IF EQ JUMP Tst1_Bits0_7;

        AY0=0xf000;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high nibble of val1}
        IF EQ JUMP Tst1_Bits8_11;

        AY0=0xc000;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high 2 bits of val1}
        IF EQ JUMP Tst1_Bits12_13;

        AY0=0x8000;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high bit of val1}
        IF EQ JUMP Do1_Bit14;                   {if bit 8 - do bit 8 routine}
{Do1_Bit15:}
```

```
            SE=1;                                  {setup for shift}
            MX0=31;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do1_Bit14:
            SE=2;                                  {setup for shift}
            MX0=30;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}

            {*********************************}

Tst1_Bits12_13:
            AY0=0x1000;                            {set the test mask}
            AR=AX1 AND AY0;                        {test high bit of val1}
            IF NE JUMP Do1_Bit12;                  {if bit 8 - do bit 8 routine}
{Do1_Bit13:}
            SE=3;                                  {setup for shift}
            MX0=29;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do1_Bit12:
            SE=4;                                  {setup for shift}
            MX0=28;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}

            {*********************************}


Tst1_Bits8_11:
            AY0=0x0c00;                            {set the test mask}
            AR=AX1 AND AY0;                        {test low nibble of val1}
            IF EQ JUMP Tst1_Bits8_9;               {if 0 - do bit 8/9 routine}

            AY0=0x0800;                            {set the test mask}
            AR=AX1 AND AY0;                        {test low nibble of val1}
            IF EQ JUMP Do1_Bit10;                  {if 0 - do bit 8/9 routine}
{Do1_Bit11:}
            SE=5;                                  {setup for shift}
            MX0=27;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do1_Bit10:
            SE=6;                                  {setup for shift}
            MX0=26;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}

            {*********************************}

Tst1_Bits8_9:
            AY0=0x0100;                            {set the test mask}
            AR=AX1 AND AY0;                        {test low nibble of val1}
            IF NE JUMP Do1_Bit8;                   {if !=0 - do bit 8 routine}
{Do1_Bit9:}
            SE=7;                                  {setup for shift}
            MX0=25;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do1_Bit8:
            SE=8;                                  {setup for shift}
            MX0=24;                        {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
```

```
        {*********************************}

Tst1_Bits0_7:
        AY0=0x00f0;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high nibble of val1}
        IF EQ JUMP Tst1_Bits0_3;

        AY0=0x00c0;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high 2 bits of val1}
        IF EQ JUMP Tst1_Bits4_5;

        AY0=0x0080;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high bit of val1}
        IF EQ JUMP Do1_Bit6;                    {if bit 6 - do bit 6 routine}
{Do1_Bit7:}
        SE=9;                                   {setup for shift}
        MX0=23;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}
Do1_Bit6:
        SE=10;                                  {setup for shift}
        MX0=22;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}

        {*********************************}

Tst1_Bits4_5:
        AY0=0x0010;                             {set the test mask}
        AR=AX1 AND AY0;                         {test high bit of val1}
        IF NE JUMP Do1_Bit4;                    {if bit 4 - do bit 4 routine}
{Do1_Bit5:}
        SE=11;                                  {setup for shift}
        MX0=21;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}

Do1_Bit4:
        SE=12;                                  {setup for shift}
        MX0=20;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}

        {*********************************}

Tst1_Bits0_3:
        AY0=0x000c;                             {set the test mask}
        AR=AX1 AND AY0;                         {test low nibble of val1}
        IF EQ JUMP Tst1_Bits0_1;                {if 0 - do bit 0/1 routine}

        AY0=0x0008;                             {set the test mask}
        AR=AX1 AND AY0;                         {test low nibble of val1}
        IF EQ JUMP Do1_Bit2;                    {if 0 - do bit 2 routine}
{Do1_Bit3:}
        SE=13;                                  {setup for shift}
        MX0=19;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}
Do1_Bit2:
        SE=14;                                  {setup for shift}
        MX0=18;                         {set characteristic}
        JUMP ADD_Correct;                       {add the fixer upper}
        {**********************************}

Tst1_Bits0_1:
        AY0=0x0001;                     {set the test mask}
        AR=AX1 AND AY0;                 {test low nibble of val1}
        IF NE JUMP Do1_Bit0;            {if !=0 - do bit 0 routine}
{Do1_Bit1:}
        SE=15;                          {setup for shift}
        MX0=17;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}
Do1_Bit0:
        SE=16;                          {setup for shift}
        MX0=16;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}

        {**********************************}
        {**********************************}
        {**********************************}

Tst_Val2:
        AY0=0xf000;                     {set the test mask}
        AR=AX0 AND AY0;                 {test high nibble of val1}
        IF EQ JUMP Tst2_Bits8_11;

        AY0=0xc000;                     {set the test mask}
        AR=AX0 AND AY0;                 {test high 2 bits of val1}
        IF EQ JUMP Tst2_Bits12_13;

        AY0=0x8000;                     {set the test mask}
        AR=AX0 AND AY0;                 {test high bit of val1}
        IF EQ JUMP Do2_Bit14;           {if bit 8 - do bit 8 routine}
{Do2_Bit15:}
        SE=17;                          {setup for shift}
        MX0=15;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}
Do2_Bit14:
        SE=18;                          {setup for shift}
        MX0=14;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}

        {**********************************}

Tst2_Bits12_13:
        AY0=0x1000;                     {set the test mask}
        AR=AX0 AND AY0;                 {test high bit of val1}
        IF NE JUMP Do2_Bit12;           {if bit 8 - do bit 8 routine}
{Do2_Bit13:}
        SE=19;                          {setup for shift}
        MX0=13;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}
Do2_Bit12:
        SE=20;                          {setup for shift}
        MX0=12;                 {set characteristic}
        JUMP ADD_Correct;               {add the fixer upper}

        {**********************************}
```

```
Tst2_Bits8_11:
          AY0=0x0c00;                            {set the test mask}
          AR=AX0 AND AY0;                        {test low nibble of val1}
          IF EQ JUMP Tst2_Bits8_9;               {if 0 - do bit 8/9 routine}

          AY0=0x0800;                            {set the test mask}
          AR=AX0 AND AY0;                        {test low nibble of val1}
          IF EQ JUMP Do2_Bit10;                  {if 0 - do bit 8/9 routine}
{Do2_Bit11:}
          SE=21;                                 {setup for shift}
          MX0=11;                        {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}
Do2_Bit10:
          SE=22;                                 {setup for shift}
          MX0=10;                        {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}

          {**********************************}

Tst2_Bits8_9:
          AY0=0x0100;                            {set the test mask}
          AR=AX0 AND AY0;                        {test low nibble of val1}
          IF NE JUMP Do2_Bit8;                   {if !=0 - do bit 8 routine}
{Do2_Bit9:}
          SE=23;                                 {setup for shift}
          MX0=9;                                 {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}
Do2_Bit8:
          SE=24;                                 {setup for shift}
          MX0=8;                                 {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}

          {**********************************}

Tst2_Bits0_7:
          AY0=0x00f0;                            {set the test mask}
          AR=AX0 AND AY0;                        {test high nibble of val1}
          IF EQ JUMP Tst2_Bits0_3;

          AY0=0x00c0;                            {set the test mask}
          AR=AX0 AND AY0;                        {test high 2 bits of val1}
          IF EQ JUMP Tst2_Bits4_5;

          AY0=0x0080;                            {set the test mask}
          AR=AX0 AND AY0;                        {test high bit of val1}
          IF EQ JUMP Do2_Bit6;                   {if bit 6 - do bit 6 routine}
{Do2_Bit7:}
          SE=25;                                 {setup for shift}
          MX0=7;                                 {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}

Do2_Bit6:
          SE=26;                                 {setup for shift}
          MX0=6;                                 {set characteristic}
          JUMP ADD_Correct;                      {add the fixer upper}

          {**********************************}
```

```
Tst2_Bits4_5:
            AY0=0x0010;                            {set the test mask}
            AR=AX0 AND AY0;                        {test high bit of val1}
            IF NE JUMP Do2_Bit4;                   {if bit 4 - do bit 4 routine}
{Do2_Bit5:}
            SE=27;                                 {setup for shift}
            MX0=5;                                 {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do2_Bit4:
            SE=28;                                 {setup for shift}
            MX0=4;                                 {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}

            {*********************************}

Tst2_Bits0_3:
            AY0=0x000c;                            {set the test mask}
            AR=AX0 AND AY0;                        {test low nibble of val1}
            IF EQ JUMP Tst2_Bits0_1;               {if 0 - do bit 0/1 routine}

            AY0=0x0008;                            {set the test mask}
            AR=AX0 AND AY0;                        {test low nibble of val1}
            IF EQ JUMP Do2_Bit2;                   {if 0 - do bit 2 routine}
{Do2_Bit3:}
            SE=29;                                 {setup for shift}
            MX0=3;                                 {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do2_Bit2:
            SE=30;                                 {setup for shift}
            MX0=2;                                 {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}

            {*********************************}

Tst2_Bits0_1:
            AY0=0x0001;                            {set the test mask}
            AR=AX0 AND AY0;                        {test low nibble of val1}
            IF NE JUMP Do2_Bit0;                   {if !=0 - do bit 0 routine}
{Do2_Bit1:}
            SE=31;                                 {setup for shift}
            MX0=1;                                 {set characteristic}
            JUMP ADD_Correct;                      {add the fixer upper}
Do2_Bit0:
            SE=32;                                 {setup for shift}
            MX0=0;                                 {set characteristic}

            {*********************************}

Add_Correct:
                                                   {MR0 contains the characteristic value}
            MR0=AX0;                               {xfer answer}
            MR1=AX1;                               {xfer answer}
            SR=LSHIFT MR0 (LO);                    {get val2->val1 : shift}
            SR=SR OR LSHIFT MR1 (LO);              {shift}
            AX0=SR0;                               {xfer answer}
            AX1=SR1;                               {xfer answer}
            MR0=MX0;                               {xfer char}
```

```
        SE=-8;                                  {reset shifter}

        AY0=0xff00;                             {set mask for upper byte}
        AR=AX1 AND AY0;                         {get the upper byte}
        SR=LSHIFT AR (LO);                      {xfer the upper byte to the lo byte}
        SE=-1;                                  {reset shifter}

        I4=^LogTbl;                             {get the log table}
        M4=SR0;                         {xfer the addr offset}
        MODIFY(I4,M4);                          {set pointer to corr addr}

        AY0=DM(I4,M5);                          {get the 16 bit correction val}
        DM(I6,M5)=AX0, AR=AX1+AY0;              {save val1 : val2=val2+corr}
        DM(I6,M5)=AR,  AR=MR0+C;                {save val2 : char=char+c}
        DM(I6,M5)=AR;                           {save the char}
        RTS;

{**************************************************************************}
{**************************************************************************}

Anti_Log:
        AX0=DM(I6,M5);                          {get val1->val2}
        AX1=DM(I6,M5);                          {get val2->val3}
        MR0=DM(I6,M7);                          {get char->val1}

        SE=-8;                                  {reset shifter}
        AY0=0xff00;                             {set mask for upper byte}
        AR=AX1 AND AY0;                         {get the upper byte}
        SR=LSHIFT AR (LO);                      {xfer the upper byte to the lo byte}
        SE=-1;                                  {reset shifter}

        I4=^LogTbl;                             {get the log table}
        M4=SR0;                         {xfer the addr offset}
        MODIFY(I4,M4);                          {set pointer to corr addr}
{Sub_Correct:}
        AY0=DM(I4,M5);                          {get the 16 bit correction val}
        DM(I6,M5)=AX0, AR=AX1-AY0;              {save val1 : val2=val2-corr}
        DM(I6,M5)=AR,  AR=MR0+C-1;              {save val2 : char=char-c}
        DM(I6,M7)=AR;                           {save the char->val1}
        AY1=AR;

        {*********************************}

        MR0=AX0;                                {xfer val1}
        MR1=AX1;                                {xfer val2}
        SE=-1;                                  {set the shifter}
        SR=LSHIFT MR0 (LO);                     {get val2->val1 : shift}
        SR=SR OR LSHIFT MR1 (LO);               {shift}
        {answer is in SR1:SR0}
        MR0=SR0;                                {xfer val1}
        MR1=SR1;                                {xfer val2}
        AY0=0x8000;                             {set the mask}
        AR=MR1 OR AY0;                          {set the top bit}
        MR1=AR;                                 {xfer val2}

        AY0=31;                                 {set the max num of bits to shift}
        AX0=AY0;                                {xfer the char}
        AR=AX0-AY1;                        {get the difference}
        SE=AR;                             {set the shifter = char}
        SR=LSHIFT MR0 (LO);                {get val2->val1 : shift}
        SR=SR OR LSHIFT MR1 (LO);          {shift}

        DM(I6,M5)=SR0;                     {save val1->val2}
        DM(I6,M6)=SR1;                     {save val2->val1}
        RTS;

{****************************************************************************}
ENDMOD;
```

Appendix A

8051 Logarithm Math

Appendix A will demonstrate the mechanics behind fixed point Logarithm Fast Math.
The correction tables and general calculation can be expanded per user requirements.

The three sections for Log Math are . . . .

1. Logarithmic Math Algorithms (Number -> Log)
2. Anti Logarithmic Math (Log -> Number)
3. Error Correction Tables

Logarithmic Math Algorithms

The following Algorithms show how to convert 4 byte binary numbers into base2 logarithms, and base$_2$ logarithms into numbers.

To convert a number into a base2 logarithms, load the VAL1-4 registers (low to high byte) with the number to convert then call GET_LOG. The base2 logarithm mantissa will be returned in the VAL1-4 registers (low to high byte), and the log characteristic in the CHAR register.

To convert a base2 logarithm into a number, load the VAL1-4 registers (low to high byte) with the mantissa, and the CHAR register with the Characteristic, and call ANTI_LOG. The number will be returned in the VAL1-4 registers (low to high byte).

```
                                    ;***********************************

                                          ;Math and Misc. Registers

VAL1 EQU     31H                                      ;Put the value here then call the
VAL2         EQU    32H                               ;GET_LOG routine, or put the Log
VAL3         EQU    33H                               ;here then call ANTI_LOG
VAL4         EQU    34H
CHAR         EQU    35H

;                                    ************************
;                        Get the base2 LOG of the number in the VAL1-4 Registers
;                            1) - convert the number to its logarithm value
;                            2) - put the answer in the VAL1-4 registers
;                            3) - correct the log value in the VAL registers

GET_LOG:     MOV    A,VAL4
             JNZ    LOG_4BT7
             JMP    LOG_TST_3BY

LOG_4BT7:    JNB    ACC.7,LOG_4BT6

             MOV    CHAR,#031

             MOV    A,VAL4
```

```
                ANL     A,#01111111B
                RL      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#10000000B
                RL      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#01111111B
                RL      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#10000000B
                RL      A
                XRL     VAL3,A

                MOV     A,VAL2
                ANL     A,#01111111B
                RL      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#10000000B
                RL      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#01111111B
                RL      A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT6:       JNB     ACC.6,LOG_4BT5

                MOV     CHAR,#030

                MOV     A,VAL4
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL3,A
                MOV     A,VAL2
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT5:       JNB     ACC.5,LOG_4BT4

                MOV     CHAR,#029

                MOV     A,VAL4
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL3,A

                MOV     A,VAL2
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT4:       JNB     ACC.4,LOG_4BT3

                MOV     CHAR,#028

                MOV     A,VAL4
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11110000B
                SWAP    A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11110000B
                SWAP    A
                XRL     VAL3,A

                MOV     A,VAL2
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11110000B
                SWAP    A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT3:       JNB     ACC.3,LOG_4BT2

                MOV     CHAR,#027

                MOV     A,VAL4
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11111000B
                SWAP    A
                RL      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11111000B
                SWAP    A
                RL      A
                XRL     VAL3,A

                MOV     A,VAL2
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11111000B
                SWAP    A
                RL      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT2:       JNB     ACC.2,LOG_4BT1

                MOV     CHAR,#026

                MOV     A,VAL4
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL3,A
                MOV     A,VAL2
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL1,A
                JMP     AD_CORR

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

LOG_4BT1:       JNB     ACC.1,LOG_4BT0

                MOV     CHAR,#025

                MOV     A,VAL4
                ANL     A,#00000001B
                RR      A
                MOV     VAL4,A
                MOV     A,VAL3
                ANL     A,#11111110B
                RR      A
                XRL     VAL4,A

                MOV     A,VAL3
                ANL     A,#00000001B
                RR      A
                MOV     VAL3,A
                MOV     A,VAL2
                ANL     A,#11111110B
                RR      A
                XRL     VAL3,A

                MOV     A,VAL2
                ANL     A,#00000001B
                RR      A
                MOV     VAL2,A
                MOV     A,VAL1
                ANL     A,#11111110B
                RR      A
                XRL     VAL2,A

                MOV     A,VAL1
                ANL     A,#00000001B
                RR      A
                RR      A
                MOV     VAL1,A
                JMP     AD_CORR

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

```
LOG_4BT0:       MOV     CHAR,#024

                MOV     VAL4,VAL3
                MOV     VAL3,VAL2
                MOV     VAL2,VAL1
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_TST_3BY:            MOV     A,VAL3
                JNZ     LOG_3BT7
                JMP     LOG_TST_2BY

LOG_3BT7:       JNB     ACC.7,LOG_3BT6

                MOV     CHAR,#023

                MOV     A,VAL3
                ANL     A,#01111111B
                RL      A
                MOV     VAL4,A
                MOV     A,VAL2
                ANL     A,#10000000B
                RL      A
                XRL     VAL4,A

                MOV     A,VAL2
                ANL     A,#01111111B
                RL      A
                MOV     VAL3,A
                MOV     A,VAL1
                ANL     A,#10000000B
                RL      A
                XRL     VAL3,A

                MOV     A,VAL1
                ANL     A,#01111111B
                RL      A
                MOV     VAL2,A
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT6:       JNB     ACC.6,LOG_3BT5

                MOV     CHAR,#022

                MOV     A,VAL3
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL4,A
                MOV     A,VAL2
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL4,A
                MOV     A,VAL2
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL3,A
                MOV     A,VAL1
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL3,A

                MOV     A,VAL1
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL2,A
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT5:       JNB     ACC.5,LOG_3BT4

                MOV     CHAR,#021

                MOV     A,VAL3
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL2
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL2
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL3,A
                MOV     A,VAL1
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL3,A

                MOV     A,VAL1
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL2,A
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT4:       JNB     ACC.4,LOG_3BT3
```

```
            MOV     CHAR,#020

            MOV     A,VAL3
            ANL     A,#00001111B
            SWAP    A
            MOV     VAL4,A
            MOV     A,VAL2
            ANL     A,#11110000B
            SWAP    A
            XRL     VAL4,A

            MOV     A,VAL2
            ANL     A,#00001111B
            SWAP    A
            MOV     VAL3,A
            MOV     A,VAL1
            ANL     A,#11110000B
            SWAP    A
            XRL     VAL3,A

            MOV     A,VAL1
            ANL     A,#00001111B
            SWAP    A
            MOV     VAL2,A

            MOV     VAL1,#000
            JMP     AD_CORR

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

LOG_3BT3:   JNB     ACC.3,LOG_3BT2

            MOV     CHAR,#019

            MOV     A,VAL3
            ANL     A,#00000111B
            SWAP    A
            RL      A
            MOV     VAL4,A
            MOV     A,VAL2
            ANL     A,#11111000B
            SWAP    A
            RL      A
            XRL     VAL4,A

            MOV     A,VAL2
            ANL     A,#00000111B
            SWAP    A
            RL      A
            MOV     VAL3,A
            MOV     A,VAL1
            ANL     A,#11111000B
            SWAP    A
            RL      A
            XRL     VAL3,A

            MOV     A,VAL1
            ANL     A,#00000111B
            SWAP    A
```

```
                RL      A
                MOV     VAL2,A

                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT2:       JNB     ACC.2,LOG_3BT1

                MOV     CHAR,#018

                MOV     A,VAL3
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL2
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL2
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL3,A
                MOV     A,VAL1
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL3,A

                MOV     A,VAL1
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL2,A

                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT1:       JNB     ACC.1,LOG_3BT0

                MOV     CHAR,#017

                MOV     A,VAL3
                ANL     A,#00000001B
                RR      A
                MOV     VAL4,A
                MOV     A,VAL2
                ANL     A,#11111110B
                RR      A
                XRL     VAL4,A

                MOV     A,VAL2
                ANL     A,#00000001B
                RR      A
                MOV     VAL3,A
                MOV     A,VAL1
                ANL     A,#11111110B
                RR      A
                XRL     VAL3,A

                MOV     A,VAL1
                ANL     A,#00000001B
                RR      A
                MOV     VAL2,A

                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_3BT0:       MOV     CHAR,#016

                MOV     VAL4,VAL2
                        MOV     VAL3,VAL1
                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_TST_2BY:            MOV     A,VAL2
                JNZ     LOG_2BT7
                JMP     LOG_TST_1BY

LOG_2BT7:       JNB     ACC.7,LOG_2BT6

                MOV     CHAR,#015

                MOV     A,VAL2
                ANL     A,#01111111B
                RL      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#10000000B
                RL      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#01111111B
                RL      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT6:       JNB     ACC.6,LOG_2BT5

                MOV     CHAR,#014
                MOV     A,VAL2
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11000000B
                RL      A
                RL      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00111111B
                RL      A
                RL      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT5:       JNB     ACC.5,LOG_2BT4

                MOV     CHAR,#013

                MOV     A,VAL2
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11100000B
                SWAP    A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT4:       JNB     ACC.4,LOG_2BT3

                MOV     CHAR,#012

                MOV     A,VAL2
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11110000B
                SWAP    A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT3:       JNB     ACC.3,LOG_2BT2

                MOV     CHAR,#011

                MOV     A,VAL2
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11111000B
                SWAP    A
                RL      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT2:       JNB     ACC.2,LOG_2BT1

                MOV     CHAR,#010

                MOV     A,VAL2
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11111100B
                RR      A
                RR      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT1:       JNB     ACC.1,LOG_2BT0

                MOV     CHAR,#009

                MOV     A,VAL2
                ANL     A,#00000001B
                RR      A
                MOV     VAL4,A
                MOV     A,VAL1
                ANL     A,#11111110B
                RR      A
                XRL     VAL4,A

                MOV     A,VAL1
                ANL     A,#00000001B
                RR      A
                MOV     VAL3,A

                MOV     VAL2,#000
                MOV     VAL1,#000
                JMP     AD_CORR

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

LOG_2BT0:       MOV     CHAR,#008

                MOV     VAL4,VAL1
                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_TST_1BY:            MOV     A,VAL1
                JNB     ACC.7,LOG_1BT6

                MOV     CHAR,#007

                CLR     C
                MOV     A,VAL1
                RLC     A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR
```

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

LOG_1BT6:       JNB     ACC.6,LOG_1BT5

                MOV     CHAR,#006

                MOV     A,VAL1
                CLR     C
                RLC     A
                CLR     C
                RLC     A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT5:       JNB     ACC.5,LOG_1BT4

                MOV     CHAR,#005

                MOV     A,VAL1
                ANL     A,#00011111B
                SWAP    A
                RR      A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT4:       JNB     ACC.4,LOG_1BT3

                MOV     CHAR,#004

                MOV     A,VAL1
                ANL     A,#00001111B
                SWAP    A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT3:       JNB     ACC.3,LOG_1BT2

                MOV     CHAR,#003

                MOV     A,VAL1
                ANL     A,#00000111B
                SWAP    A
                RL      A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT2:       JNB     ACC.2,LOG_1BT1

                MOV     CHAR,#002

                MOV     A,VAL1
                ANL     A,#00000011B
                RR      A
                RR      A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT1:       JNB     ACC.1,LOG_1BT0

                MOV     CHAR,#001

                MOV     A,VAL1
                ANL     A,#00000001B
                RR      A
                MOV     VAL4,A

                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                JMP     AD_CORR

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

LOG_1BT0:       JNB     ACC.0,LOG_0BT0

                MOV     CHAR,#000

                MOV     VAL4,#000H
                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#001H
                JMP     AD_CORR

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

LOG_0BT0:       MOV     CHAR,#000

                MOV     VAL4,#000H
                MOV     VAL3,#000H
                MOV     VAL2,#000H
                MOV     VAL1,#000H
                RET

;                                        ***********************
;Add the correction factor to the computed logarithm value.
;Since the Log correction table is 2 bytes per correction, the pointer will be incremented
;twice for any adjustment.

AD_CORR:        MOV     A,VAL4
                MOV     DPTR,#LOGTBL
                ADD     A,DPL
                MOV     DPL,A
                MOV     A,DPH
                ADDC    A,#0
                MOV     DPH,A

                MOV     A,VAL4
                ADD     A,DPL
                MOV     DPL,A
                MOV     A,DPH
                ADDC    A,#0
                MOV     DPH,A

                CLR     A
                MOVC    A,@A+DPTR
                ADD     A,VAL3
                MOV     VAL3,A

                CLR     A
                INC     DPTR
                MOVC    A,@A+DPTR
                ADDC    A,VAL4
                MOV     VAL4,A

                MOV     A,CHAR
                ADDC    A,#0
                MOV     CHAR,A
                RET

;                                        ***********************
```

Anti Logarithmic Math

The following algorithms show how to convert 4-byte base$_2$ logarithms into binary numbers. Retrieve the base$_2$ ANTI-LOG in the VAL1-4 Registers

;First - check to see if the number is zero - if it is - abort
;Second - correct the log value in the VAL registers
;Third - convert the log to its number value
;Fourth - put the answer in the VAL1-4 registers

;Since the Log correction table is 2 bytes per correction, the pointer will be incremented twice for any adjustment.

```
;                            ************************************

ANTI_LOG:     MOV    A,CHAR ;Ok if regs. =/= 0
              JNZ    ALOG_OK
              MOV    A,VAL4
              JNZ    ALOG_OK
              MOV    A,VAL3
              JNZ    ALOG_OK
              MOV    A,VAL2
              JNZ    ALOG_OK
              MOV    A,VAL1
              JNZ    ALOG_OK
              RET

ALOG_OK:      MOV    A,VAL4                            ;Get the correction
              MOV    DPTR,#LOGTBL                      ;      table address
              ADD    A,DPL
              MOV    DPL,A
              MOV    A,DPH
              ADDC   A,#0
              MOV    DPH,A

              MOV    A,VAL4
              ADD    A,DPL
              MOV    DPL,A
              MOV    A,DPH
              ADDC   A,#0
              MOV    DPH,A

              MOV    PSW,#0                            ;subtract the correction
              CLR    A                                 ;factor from the log value
              MOVC   A,@A+DPTR
              XCH    A,VAL3
              SUBB   A,VAL3
              MOV    VAL3,A

              INC    DPTR
              CLR    A
              MOVC   A,@A+DPTR
              XCH    A,VAL4
              SUBB   A,VAL4
              MOV    VAL4,A

              MOV    A,CHAR
              SUBB   A,#0
              MOV    CHAR,A

;                                  -----------------------
;                               Convert the log to its number value

              MOV    A,CHAR
              CJNE   A,#023,ALOG_4BTHL                 ; <= 24 ?

ALOG_CHK_3BYTE:
              JMP    ALOG_A3_BT7                ; = 24

ALOG_4BTHL:
              JC     ALOG_CHK_3BYTE                    ; < 24

              CJNE   A,#031,ALOG_4BT6

              CLR    C
              MOV    A,VAL4
              RRC    A
              SETB   ACC.7
              MOV    VAL4,A
              MOV    A,VAL3
              RRC    A
              MOV    VAL3,A
              MOV    A,VAL2
              RRC    A
              MOV    VAL2,A
              MOV    A,VAL1
              RRC    A
              MOV    VAL1,A
              RET

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

ALOG_4BT6:    CJNE   A,#030,ALOG_4BT5

              CLR    C
              MOV    A,VAL4
              RRC    A
              MOV    VAL4,A

              MOV    A,VAL3
              RRC    A
              MOV    VAL3,A

              MOV    A,VAL2
              RRC    A
              MOV    VAL2,A

              MOV    A,VAL1
              RRC    A
              MOV    VAL1,A

              CLR    C
              MOV    A,VAL4
              RRC    A
              SETB   ACC.6
              MOV    VAL4,A

              MOV    A,VAL3
              RRC    A
              MOV    VAL3,A

              MOV    A,VAL2
              RRC    A
              MOV    VAL2,A

              MOV    A,VAL1
```

```
             RRC    A
             MOV    VAL1,A
             RET

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

ALOG_4BT5:   CJNE   A,#029,ALOG_4BT4

             MOV    A,VAL1
             ANL    A,#11111000B
             SWAP   A
             RL     A
             MOV    VAL1,A

             MOV    A,VAL2
             ANL    A,#00000111B
             SWAP   A
             RL     A
             XRL    VAL1,A

             MOV    A,VAL2
             ANL    A,#11111000B
             SWAP   A
             RL     A
             MOV    VAL2,A

             MOV    A,VAL3
             ANL    A,#00000111B
             SWAP   A
             RL     A
             XRL    VAL2,A

             MOV    A,VAL3
             ANL    A,#11111000B
             SWAP   A
             RL     A
             MOV    VAL3,A

             MOV    A,VAL4
             ANL    A,#00000111B
             SWAP   A
             RL     A
             XRL    VAL3,A

             MOV    A,VAL4
             ANL    A,#11111000B
             SWAP   A
             RL     A
             SETB   ACC.5
             MOV    VAL4,A
             RET

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

ALOG_4BT4:   CJNE   A,#028,ALOG_4BT3

             MOV    A,VAL1
             ANL    A,#11110000B
             SWAP   A
             MOV    VAL1,A

             MOV    A,VAL2
             ANL    A,#00001111B
             SWAP   A
             XRL    VAL1,A
```

```
              MOV    A,VAL2
              ANL    A,#11110000B
              SWAP   A
              MOV    VAL2,A

              MOV    A,VAL3
              ANL    A,#00001111B
              SWAP   A
              XRL    VAL2,A

              MOV    A,VAL3
              ANL    A,#11110000B
              SWAP   A
              MOV    VAL3,A

              MOV    A,VAL4
              ANL    A,#00001111B
              SWAP   A
              XRL    VAL3,A


              MOV    A,VAL4
              ANL    A,#11110000B
              SWAP   A
              SETB   ACC.4
              MOV    VAL4,A
              RET

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

ALOG_4BT3:    CJNE   A,#027,ALOG_4BT2

              MOV    A,VAL1
              ANL    A,#11100000B
              SWAP   A
              RR     A
              MOV    VAL1,A

              MOV    A,VAL2
              ANL    A,#00011111B
              SWAP   A
              RR     A
              XRL    VAL1,A

              MOV    A,VAL2
              ANL    A,#11100000B
              SWAP   A
              RR     A
              MOV    VAL2,A

              MOV    A,VAL3
              ANL    A,#00011111B
              SWAP   A
              RR     A
              XRL    VAL2,A

              MOV    A,VAL3
              ANL    A,#11100000B
              SWAP   A
              RR     A
              MOV    VAL3,A

              MOV    A,VAL4
              ANL    A,#00011111B
              SWAP   A
              RR      A
              XRL     VAL3,A

              MOV     A,VAL4
              ANL     A,#11100000B
              SWAP    A
              RR      A
              SETB    ACC.3
              MOV     VAL4,A
              RET

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

ALOG_4BT2:    CJNE    A,#026,ALOG_4BT1

              MOV     A,VAL1
              ANL     A,#11000000B
              RL      A
              RL      A
              MOV     VAL1,A

              MOV     A,VAL2
              ANL     A,#00111111B
              RL      A
              RL      A
              XRL     VAL1,A

              MOV     A,VAL2
              ANL     A,#11000000B
              RL      A
              RL      A
              MOV     VAL2,A

              MOV     A,VAL3
              ANL     A,#00111111B
              RL      A
              RL      A
              XRL     VAL2,A

              MOV     A,VAL3
              ANL     A,#11000000B
              RL      A
              RL      A
              MOV     VAL3,A

              MOV     A,VAL4
              ANL     A,#00111111B
              RL      A
              RL      A
              XRL     VAL3,A

              MOV     A,VAL4
              ANL     A,#11000000B
              RL      A
              RL      A
              SETB    ACC.2
              MOV     VAL4,A
              RET

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

ALOG_4BT1:    CJNE    A,#025,ALOG_4BT0

              MOV     A,VAL1
              ANL     A,#10000000B
              RL      A
              MOV     VAL1,A

              MOV     A,VAL2
              ANL     A,#01111111B
              RL      A
              XRL     VAL1,A

              MOV     A,VAL2
              ANL     A,#10000000B
              RL      A
              MOV     VAL2,A

              MOV     A,VAL3
              ANL     A,#01111111B
              RL      A
              XRL     VAL2,A

              MOV     A,VAL3
              ANL     A,#10000000B
              RL      A
              MOV     VAL3,A

              MOV     A,VAL4
              ANL     A,#01111111B
              RL      A
              XRL     VAL3,A
              MOV     A,VAL4
              ANL     A,#10000000B
              RL      A
              SETB    ACC.1
              MOV     VAL4,A
              RET

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

ALOG_4BT0:    CJNE    A,#024,ALOG_3BT7

              MOV     VAL1,VAL2
              MOV     VAL2,VAL3
              MOV     VAL3,VAL4
              MOV     VAL4,#001H
              RET

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

ALOG_3BT7:    CJNE    A,#015,ALOG_3BTHL                   ; <= 16 ?

ALOG_CHK_2BYTE:
              JMP     ALOG_A2_BT7                         ; = 16
ALOG_3BTHL:
              JC      ALOG_CHK_2BYTE                            ; < 16

              CJNE    A,#023,ALOG_3BT6

              MOV     A,VAL2
              ANL     A,#11111110B
              RR      A
              MOV     VAL1,A

              MOV     A,VAL3
              ANL     A,#00000001B
              RR      A
              XRL     VAL1,A
```

```
             MOV     A,VAL3
             ANL     A,#11111110B
             RR      A
             MOV     VAL2,A

             MOV     A,VAL4
             ANL     A,#00000001B
             RR      A
             XRL     VAL2,A

             MOV     A,VAL4
             ANL     A,#11111110B
             RR      A
             SETB    ACC.7
             MOV     VAL3,A

             MOV     VAL4,#000H
             RET

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

ALOG_3BT6:   CJNE    A,#022,ALOG_3BT5

             MOV     A,VAL2
             ANL     A,#11111100B
             RR      A
             RR      A
             MOV     VAL1,A

             MOV     A,VAL3
             ANL     A,#00000011B
             RR      A
             RR      A
             XRL     VAL1,A

             MOV     A,VAL3
             ANL     A,#11111100B
             RR      A
             RR      A
             MOV     VAL2,A

             MOV     A,VAL4
             ANL     A,#00000011B
             RR      A
             RR      A
             XRL     VAL2,A

             MOV     A,VAL4
             ANL     A,#11111100B
             RR      A
             RR      A
             SETB    ACC.6
             MOV     VAL3,A

             MOV     VAL4,#000H
             RET

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

ALOG_3BT5:   CJNE    A,#021,ALOG_3BT4

             MOV     A,VAL2
             ANL     A,#11111000B
             SWAP    A
             RL      A
             MOV    VAL1,A

             MOV    A,VAL3
             ANL    A,#00000111B
             SWAP   A
             RL     A
             XRL    VAL1,A

             MOV    A,VAL3
             ANL    A,#11111000B
             SWAP   A
             RL     A
             MOV    VAL2,A

             MOV    A,VAL4
             ANL    A,#00000111B
             SWAP   A
             RL     A
             XRL    VAL2,A


             MOV    A,VAL4
             ANL    A,#11111000B
             SWAP   A
             RL     A
             SETB   ACC.5
             MOV    VAL3,A

             MOV    VAL4,#000H
             RET

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

ALOG_3BT4:   CJNE   A,#020,ALOG_3BT3

             MOV    A,VAL2
             ANL    A,#11110000B
             SWAP   A
             MOV    VAL1,A

             MOV    A,VAL3
             ANL    A,#00001111B
             SWAP   A
             XRL    VAL1,A

             MOV    A,VAL3
             ANL    A,#11110000B
             SWAP   A
             MOV    VAL2,A

             MOV    A,VAL4
             ANL    A,#00001111B
             SWAP   A
             XRL    VAL2,A

             MOV    A,VAL4
             ANL    A,#11110000B
             SWAP   A
             SETB   ACC.4
             MOV    VAL3,A

             MOV    VAL4,#000H
             RET

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

```
ALOG_3BT3:    CJNE   A,#019,ALOG_3BT2

              MOV    A,VAL2
              ANL    A,#11100000B
              SWAP   A
              RR     A
              MOV    VAL1,A

              MOV    A,VAL3
              ANL    A,#00011111B
              SWAP   A
              RR     A
              XRL    VAL1,A

              MOV    A,VAL3
              ANL    A,#11100000B
              SWAP   A
              RR     A
              MOV    VAL2,A

              MOV    A,VAL4
              ANL    A,#00011111B
              SWAP   A
              RR     A
              XRL    VAL2,A

              MOV    A,VAL4
              ANL    A,#11100000B
              SWAP   A
              RR     A
              SETB   ACC.3
              MOV    VAL3,A

              MOV    VAL4,#000H
              RET

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

ALOG_3BT2:    CJNE   A,#018,ALOG_3BT1

              MOV    A,VAL2
              ANL    A,#11000000B
              RL     A
              RL     A
              MOV    VAL1,A

              MOV    A,VAL3
              ANL    A,#00111111B
              RL     A
              RL     A
              XRL    VAL1,A

              MOV    A,VAL3
              ANL    A,#11000000B
              RL     A
              RL     A
              MOV    VAL2,A

              MOV    A,VAL4
              ANL    A,#00111111B
              RL     A
              RL     A
              XRL    VAL2,A

              MOV    A,VAL4
```

```
             ANL    A,#11000000B
             RL     A
             RL     A
             SETB   ACC.2
             MOV    VAL3,A

             MOV    VAL4,#000H
             RET

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

ALOG_3BT1:   CJNE   A,#017,ALOG_3BT0

             MOV    A,VAL2
             ANL    A,#10000000B
             RL     A
             MOV    VAL1,A

             MOV    A,VAL3
             ANL    A,#01111111B
             RL     A
             XRL    VAL1,A

             MOV    A,VAL3
             ANL    A,#10000000B
             RL     A
             MOV    VAL2,A

             MOV    A,VAL4
             ANL    A,#01111111B
             RL     A
             XRL    VAL2,A

             MOV    A,VAL4
             ANL    A,#10000000B
             RL     A
             SETB   ACC.1
             MOV    VAL3,A

             MOV    VAL4,#000H
             RET

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

ALOG_3BT0:   CJNE   A,#016,ALOG_2BT7

             MOV    VAL1,VAL3
             MOV    VAL2,VAL4
             MOV    VAL3,#001H
             MOV    VAL4,#000H
             RET

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

ALOG_2BT7:   CJNE   A,#007,ALOG_2BTHL                 ; <= 16 ?

ALOG_CHK_1BYTE:
             JMP    ALOG_A1_BT7                       ; = 16
ALOG_2BTHL:
             JC     ALOG_CHK_1BYTE                           ; < 16

             CJNE   A,#015,ALOG_2BT6

             MOV    A,VAL3
             ANL    A,#11111110B
```

```
                RR      A
                MOV     VAL1,A

                MOV     A,VAL4
                ANL     A,#00000001B
                RR      A
                XRL     VAL1,A

                MOV     A,VAL4
                ANL     A,#11111110B
                RR      A
                SETB    ACC.7
                MOV     VAL2,A

                MOV     VAL3,#000H
                MOV     VAL4,#000H
                RET

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

ALOG_2BT6:      CJNE    A,#014,ALOG_2BT5

                MOV     A,VAL3
                ANL     A,#11111100B
                RR      A
                RR      A
                MOV     VAL1,A

                MOV     A,VAL4
                ANL     A,#00000011B
                RR      A
                RR      A
                XRL     VAL1,A

                MOV     A,VAL4
                ANL     A,#11111100B
                RR      A
                RR      A
                SETB    ACC.6
                MOV     VAL2,A

                MOV     VAL3,#000H
                MOV     VAL4,#000H
                RET

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

ALOG_2BT5:      CJNE    A,#013,ALOG_2BT4

                MOV     A,VAL3
                ANL     A,#11111000B
                SWAP    A
                RL      A
                MOV     VAL1,A

                MOV     A,VAL4
                ANL     A,#00000111B
                SWAP    A
                RL      A
                XRL     VAL1,A
                MOV     A,VAL4
                ANL     A,#11111000B
                SWAP    A
                RL      A
                SETB    ACC.5
                MOV     VAL2,A

                MOV     VAL3,#000H
                MOV     VAL4,#000H
                RET

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

ALOG_2BT4:      CJNE    A,#012,ALOG_2BT3

                MOV     A,VAL3
                ANL     A,#11110000B
                SWAP    A
                MOV     VAL1,A

                MOV     A,VAL4
                ANL     A,#00001111B
                SWAP    A
                XRL     VAL1,A

                MOV     A,VAL4
                ANL     A,#11110000B
                SWAP    A
                SETB    ACC.4
                MOV     VAL2,A

                MOV     VAL3,#000H
                MOV     VAL4,#000H
                RET

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

ALOG_2BT3:      CJNE    A,#011,ALOG_2BT2

                MOV     A,VAL3
                ANL     A,#11100000B
                SWAP    A
                RR      A
                MOV     VAL1,A

                MOV     A,VAL4
                ANL     A,#00011111B
                SWAP    A
                RR      A
                XRL     VAL1,A

                MOV     A,VAL4
                ANL     A,#11100000B
                SWAP    A
                RR      A
                SETB    ACC.3
                MOV     VAL2,A

                MOV     VAL3,#000H
                MOV     VAL4,#000H
                RET

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

ALOG_2BT2:      CJNE    A,#010,ALOG_2BT1

                MOV     A,VAL3
                ANL     A,#11000000B
                RL      A
                RL      A
```

```
             MOV     VAL1,A

             MOV     A,VAL4
             ANL     A,#00111111B
             RL      A
             RL      A
             XRL     VAL1,A

             MOV     A,VAL4
             ANL     A,#11000000B
             RL      A
             RL      A
             SETB    ACC.2
             MOV     VAL2,A

             MOV     VAL3,#000H
             MOV     VAL4,#000H
             RET

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

ALOG_2BT1:   CJNE    A,#009,ALOG_2BT0

             MOV     A,VAL3
             ANL     A,#10000000B
             RL      A
             MOV     VAL1,A

             MOV     A,VAL4
             ANL     A,#01111111B
             RL      A
             XRL     VAL1,A

             MOV     A,VAL4
             ANL     A,#10000000B
             RL      A
             SETB    ACC.1
             MOV     VAL2,A

             MOV     VAL3,#000H
             MOV     VAL4,#000H
             RET

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

ALOG_2BT0:   CJNE    A,#008,ALOG_1BT7

             MOV     VAL1,VAL4
             MOV     VAL2,#001H
             MOV     VAL3,#000H
             MOV     VAL4,#000H
             RET

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

ALOG_1BT7:   CJNE    A,#007,ALOG_1BT6

             CLR     C
             MOV     A,VAL4
             RRC     A
             SETB    ACC.7
             MOV     VAL1,A


             MOV     VAL2,#000H
```

```
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT6:    CJNE   A,#006,ALOG_1BT5
              MOV    A,VAL4
              ANL    A,#11111100B
              RR     A
              RR     A
              SETB   ACC.6
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT5:    CJNE   A,#005,ALOG_1BT4

              MOV    A,VAL4
              ANL    A,#11111000B
              SWAP   A
              RL     A
              SETB   ACC.5
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT4:    CJNE   A,#004,ALOG_1BT3

              MOV    A,VAL4
              ANL    A,#11110000B
              SWAP   A
              SETB   ACC.4
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT3:    CJNE   A,#003,ALOG_1BT2

              MOV    A,VAL4
              ANL    A,#11100000B
              SWAP   A
              RR     A
              SETB   ACC.3
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET
```

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

ALOG_1BT2:    CJNE   A,#002,ALOG_1BT1

              MOV    A,VAL4
              ANL    A,#11000000B
              RL     A
              RL     A
              SETB   ACC.2
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT1:    CJNE   A,#001,ALOG_1BT0

              MOV    A,VAL1
              ANL    A,#10000000B
              RL     A
              SETB   ACC.1
              MOV    VAL1,A

              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

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

ALOG_1BT0:    MOV    VAL1,#001H
              MOV    VAL2,#000H
              MOV    VAL3,#000H
              MOV    VAL4,#000H
              RET

;                                  ************************
```

Error Correction Tables

```
;This is the correction table for the logarithm generators.

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

LOGTBL:      DW     07000H,0DF00H,04D01H,0B901H,02502H,08E02H,0F602H,05D03H,0C303H,02704H
             DW     08904H,0EB04H,04B05H,0AA05H,00706H,06406H,0BF06H,01807H,07107H,0C807H
             DW     01E08H,07308H,0C608H,01809H,06909H,0B909H,0080AH,0550AH,0A20AH,0ED0AH
             DW     0370BH,0800BH,0C80BH,00E0CH,0540CH,0980CH,0DB0CH,01D0DH,05E0DH,09E0DH
             DW     0DD0DH,01B0EH,0580EH,0930EH,0CE0EH,0080FH,0400FH,0780FH,0AE0FH,0E40FH
             DW     01810H,04C10H,07E10H,0B010H,0E010H,01011H,03E11H,06C11H,09911H,0C411H
             DW     0EF11H,01912H,04212H,06A12H,09012H,0B712H,0DC12H,00013H,02313H,04613H
             DW     06713H,08813H,0A813H,0C713H,0E513H,00214H,01F14H,03A14H,05514H,06E14H
             DW     08714H,0A014H,0B714H,0CD14H,0E314H,0F814H,00C15H,01F15H,03215H,04315H
             DW     05415H,06415H,07415H,08215H,09015H,09D15H,0A915H,0B515H,0BF15H,0C915H
             DW     0D215H,0DB15H,0E315H,0EA15H,0F015H,0F615H,0FA15H,0FE15H,00216H,00516H
             DW     00616H,00816H,00816H,00816H,00716H,00616H,00416H,00116H,0FD15H,0F915H
             DW     0F415H,0EF15H,0E915H,0E215H,0DA15H,0D215H,0C915H,0C015H,0B615H,0AB15H
             DW     0A015H,09415H,08715H,07A15H,06C15H,05D15H,04E15H,03E15H,02E15H,01D15H
             DW     00C15H,0F914H,0E714H,0D314H,0BF14H,0AB14H,09614H,08014H,06A14H,05314H
             DW     03C14H,02414H,00B14H,0F213H,0D813H,0BE13H,0A313H,08813H,06C13H,05013H
             DW     03313H,01513H,0F712H,0D812H,0B912H,09A12H,07912H,05912H,03712H,01612H
             DW     0F311H,0D011H,0AD11H,08911H,06511H,04011H,01B11H,0F510H,0CE10H,0A710H
             DW     08010H,05810H,03010H,00710H,0DD0FH,0B40FH,0890FH,05E0FH,0330FH,0070FH
             DW     0DB0EH,0AE0EH,0810EH,0540EH,0250EH,0F70DH,0C80DH,0980DH,0680DH,0380DH
             DW     0070DH,0D60CH,0A40CH,0720CH,03F0CH,00C0CH,0D80BH,0A40BH,0700BH,03B0BH
             DW     0060BH,0D00AH,09A0AH,0630AH,02C0AH,0F409H,0BD09H,08409H,04C09H,01209H
             DW     0D908H,09F08H,06408H,02A08H,0EE07H,0B307H,07707H,03A07H,0FD06H,0C006H
             DW     08306H,04406H,00606H,0C705H,08805H,04805H,00805H,0C804H,08704H,04604H
             DW     00404H,0C203H,08003H,03D03H,0FA02H,0B702H,07302H,02F02H,0EA01H,0A501H
             DW     06001H,01A01H,0D400H,08E00H,04700H,02300H
```

The following BASIC program was used to develop the 2-byte error correction table shown alone. To reduce the required code time for log and antilog conversions, the bytes were reversed in the actual code table. The BASIC program, however, will produce a normal HiByte, LoByte arrangement.

```
5  'SAVE "A:LOG.BAS
7  CLOSE #1
8  OPEN "O",1,"LOGTBL"
10 FOR MANT = 1 TO 255
20 ER = 65536 * (LOG(MANT + 256) / LOG(2) - 8 - MANT / 256)
30 PRINT MANT,INT(ER),HEX$(INT(ER))"H"
32 NUM$=HEX$(INT(ER)) + "H"
35 PRINT #1,NUM$
40 NEXT MANT
45 CLOSE #1

END
```