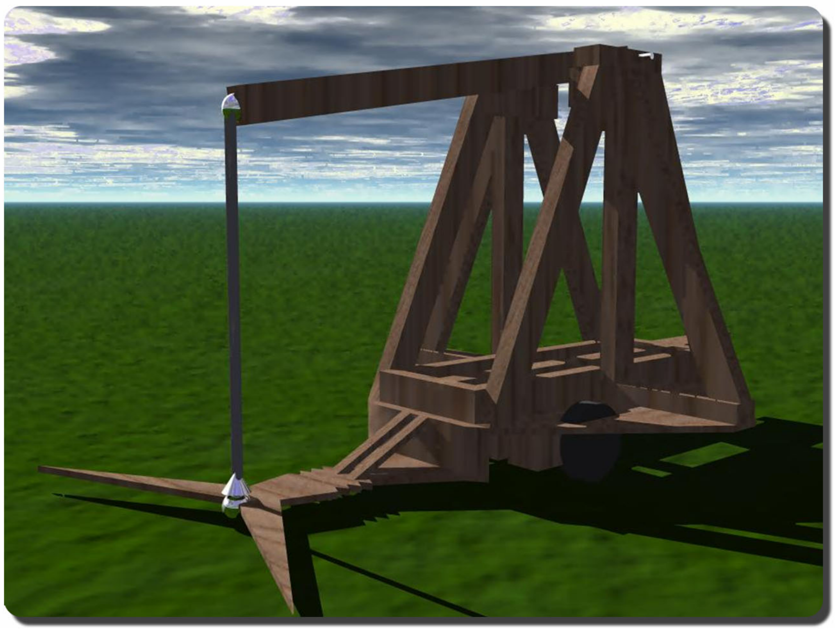
Mr. T



An Autonomous Trebuchet

Jeffrey Bergman
Intelligent Machine Design Lab
EEL 5666

Table of Contents

Abstract

Mr. T (short for Mr. Trebuchet) is an autonomous robotic trebuchet designed to find and throw ammunition at a target at least 10 yards away. It will continue to hurl objects until turned off. There is also a logical boundary that Mr. T will travel within. This boundary is the maximum and minimum distances the objects can be thrown. Mr. T will know his exact position and angle at all times, and can determine how far he is from the target.

Executive Summary

Mr. T is a modern version of one of the most powerful and devastating siege engines ever used. A trebuchet is similar to a catapult, but instead of using tension to launch an object, a trebuchet uses a pivot arm, a sling, and a massive counterweight to hurl the object over great distances.

The main purpose of Mr. T is to lay siege on a castle, positioned about 10 yards away from the robot. He will drive around semi-randomly, searching for appropriate objects to throw (specially designed ammunition) that can be thrown. Once ammunition is found, it is loaded onto an electromagnetic sling and hurled at the target. The round is released from the trebuchet at an optimum time for a direct hit on a target. By adjusting release times, Mr. T can compensate for being different distances from the target.

In order to stay aware of his body and the environment around him, Mr. T uses four types of sensors. These include IR sensors, bump sensors, potentiometers, and a special vector processing unit that can determine, through the use of an optical mouse, the exact position and angle the robot is facing.

The hardware used includes two Motorola HC11 EVBU boards with the Mekatronix ME11 memory and port expansion, a mouse, a AWCE PAK-VI, a UART, and several resistors, capacitor, transistors, and relays.

Introduction

The trebuchet was first introduced into western society by the Greeks or Romans in the 12th century, but was mostly developed during the middle ages by the French. Originating at around 300 B.C. in China, the trebuchet became one of the most popular and devastating siege engines during the middle ages. Characterized by its massive counterweight and long sling, trebuchets could throw heavy rocks and other objects (often dead horses or peasants infected with the plague) over hundreds of yards with amazing accuracy. Because trebuchets use a weight – counterweight design, it does not rely on tension like a catapult. This means that if you load ammunition with nearly the same weight every time, the trebuchet will throw it in the exact location.

Little has changed in trebuchet design in the last 500 years. The Hobbyists that compete in pumpkin throwing contests construct nearly the same devices that were used in the middle ages. My goal was to create a robot that merged state of the art processors and sensors with tried and true medieval weaponry to create and autonomous siege engine. Through this, Mr. T was created.

Mr. T is the first trebuchet that has a mind of its own. The only conditions it needs to work is for it to be a known distance away from and angle in relation to the target. It then searches its surrounding area for ammunition, loads that ammunition, and then launches it at the target. It has the intelligence to know where it is at all times (including angle and position), so an accurate, direct hit can be thrown every time.

Integrated System

Mr. T is a multiprocessor based robotic trebuchet. Its system can be broken down into several categories. Some of these are the hardware used, as well as the behaviors it can perform, and the sensors used. This hardware can perform all of the functions required of the robot

Hardware

Mr. T is based on two Motorola 68HC11 evaluation boards each equipped with a Motorola MC68HC11E9 processor and the Mekatronix ME11 memory and port expansion board. The robot uses two NiCad battery packs to provide energy, as well as one 12 V Lead Acid battery which provides energy for the on board electromagnet. A system of random searching was used by the robot to find the ammunition, 32 bit precision floating point vector calculations to find its current location, as well as floating point calculations to find the target.

Sensors and Additional Hardware

Sensors in the device included a mouse, two IR transmitter / receivers, a potentiometer, and a bump switch. An AWCE PAK-VI, a National Semiconductor UART, and an Eriez EM-R1 small flat faced electromagnet were some additional hardware that was required. The connections to the device included two parallel port interfaces, an AC adapter plug, and several switches and buttons.

Behaviors

Mr. T has three main modes of operation. These modes are programming, calibration, and running. The running mode can be broken down into several smaller modes; searching, loading, returning, aiming, and firing. Also, during running mode the device is constantly performing obstacle avoidance.

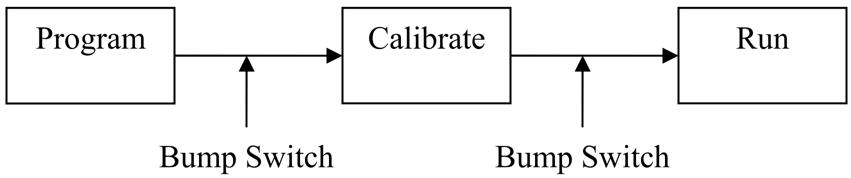


Figure 1 - Behaviors

Mobile Platform

The mobile platform needed no only to be able to withstand the large amount of force generated by the throwing arm, but to also be able to be maneuvered accurately. The platform can be divided into several sections: the platform, the arm and pivot, the wheels, the catcher, sling, and the weight crate.

Platform

To make the platform strong enough to survive the force laid upon it, 1" x 2" pine wood was used instead of the conventional airline plywood. To connect all of the pieces of wood together, #8 wood screws were used along with two supporting nails. The final, reinforced design was the third attempt at making the structure. The first implementation used larger wood and was a little too large for what was needed. The second design used the same size wood as my final implementation, but was built without reinforcing nails and with larger screws. The structure could not handle the weight of the test throws and began to lose stability quickly. The final design, shown below in figure 2, was created without the side reinforcements which were not being used. See appendix A for scaled representations of the beams required.

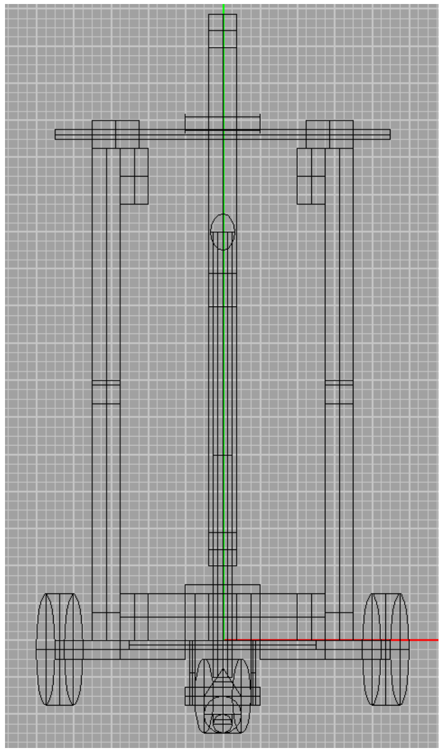


Figure 2a – Front View

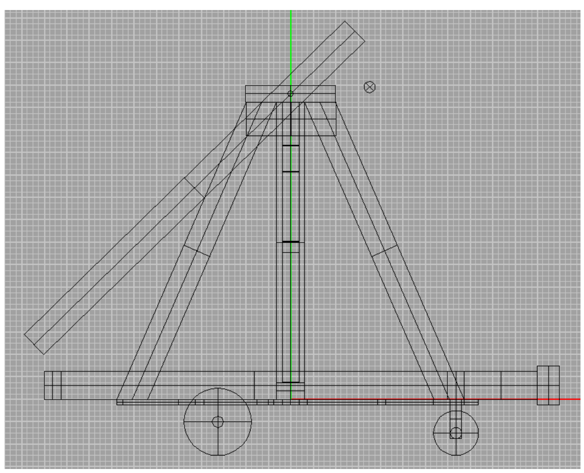


Figure 2b – Side View

Arm and Pivot

The arm was constructed with the same size wood as the body. The arm was 25" in length. At a point ½" in on both sides a hole was drilled. There was also a larger whole drilled in 4 ½" from the end of the short side.

The pivot used was a ¼" threaded rod. Two ½" hollow metal tubes were used where the arm connects to the body. First, the two tubes were attached to the wood with strong epoxy, and then secured with metal strips that were screwed into the surface. Next, the rod was placed through the tube with the lock nut configuration shown in figure 3. The nuts around the tube were loosely secured, while the two nuts holding the wooden arm were secured as tightly as possible. The goal of this was to make the rod and the arm turn together, while having causing as little rotational friction as possible with the body. The rod needed to always turn with the arm because the rod is then connected to a potentiometer (with at least a 270° range of motion) for determining the position of the arm at all times.

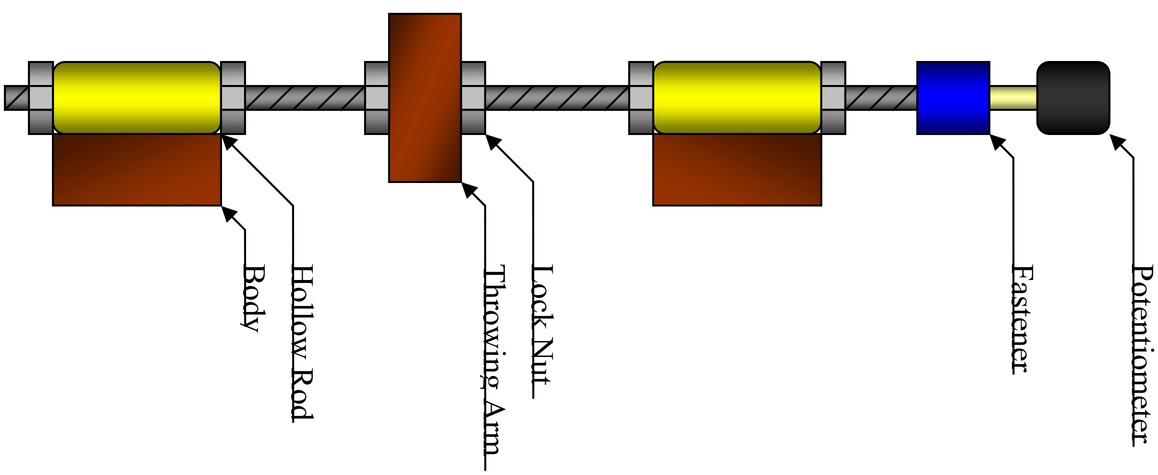


Figure 3 – Arm and Pivot

Wheels

For the wheels, three 4" tires with supports were used. These can be found at any hardware store. For the drive wheel, the tire was removed from the axel and secured to the large round servo connector. This was secured by drilling eight small holes into the all rubber tire and then hammering in small nails to connect it. Because of the nature of rubber, when the nails were hammered in, it stretches the rubber which then holds them tightly in place. Figure 4 shows how the tire is connected.

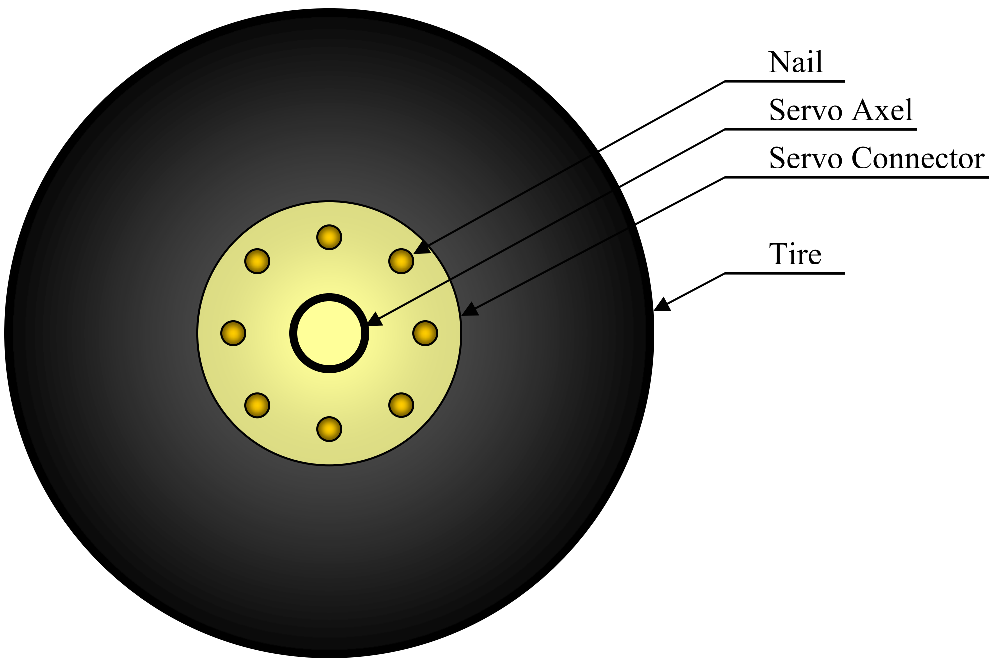


Figure 4 – Tire Connection

Catcher

The purpose of the catcher is to catch the ammunition as the robot moves towards it, and position the ammunition so that when the arm lowers to the "load" position the magnet is resting upon it. The easiest way to do this was to create a "V" shaped aircraft plywood structure, with the widest point of the "V" the same width as the tires. The bottom of the "V" is about the width of the ammunition (1"), and there is a bump sensor there to tell the robot when the ammunition is properly placed. This section is shown in figure 4.

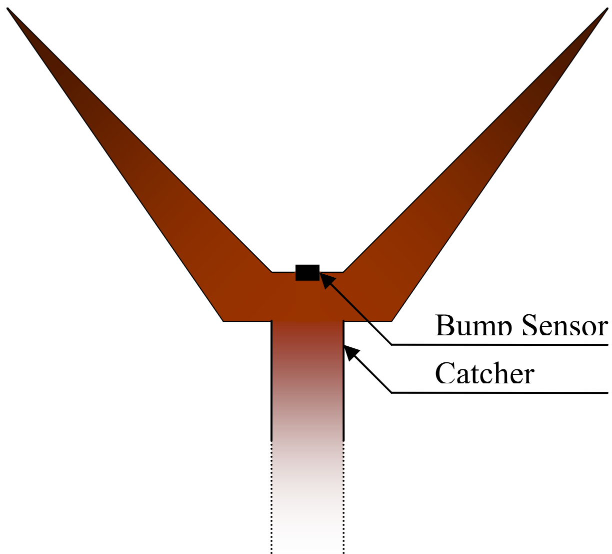


Figure 4 – Catcher

Sling

The sling consists of the electromagnet, and a 13" piece of picture wire, as well as a #10 screw. The end of the wire is inserted into the hole in the back of the electromagnet, and a #10 screw is screwed in. This fastens the wire to the magnet. The other end of the wire is threaded through the arm and secured by either a knot or some other method (fastener, solder, etc.).

Weight Crate

The final piece of the platform is the weight crate. This is essentially a box that surrounds the 12 Volt lead-acid battery. It needs to be very strong, or the battery will fly out of the bottom, and it also must be able to let the battery be removed. In addition to this, it needs to minimize the sideways swing so that when it moves it will not strike the sides of the body. It also must be able to swing freely forwards and backwards to properly transfer its energy. In building testing, the box was destroyed several times. Because of this, the final version of the box is reinforced with metal sheets, and made to be almost the exact size of the battery (to minimize that movement of the battery in the crate). This piece is shown below in figure 5.

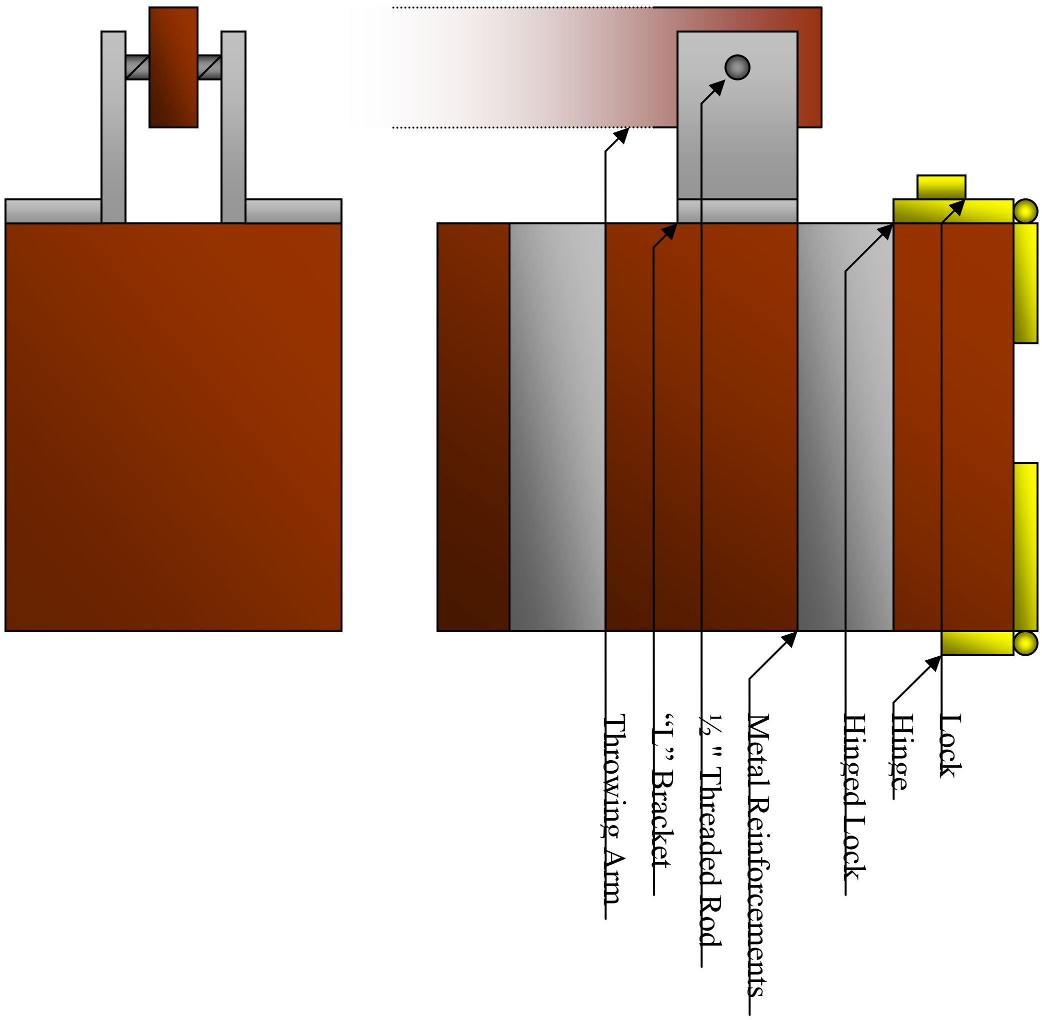


Figure 5 – Weight Crate

Actuation

The only actuation devices used for the robot were four servos. These were used for four tasks; steering, driving, winching, and locking. The same system calls were used to work all of the servos. The value in the A register is decimal 0, 50, or 100. When it is 50, the servo turns to the neutral position (or off if hacked), when it 100 is entered, the servo will rotate to +90° (or continuously rotate clockwise), and if 0 is entered the servo will rotate to -90°(or counter clockwise). The code for this can be found in appendix B.

Steering

The same wheel was used for both steering and driving. This was to make it easier for the robot to travel in a straight line and to pivot around a single point. The servo used for steering was a Cirrus CS-80. The CS-80 is a metal geared, two ball bearing high torque servo. At 6 Volts (the voltage it is used at), the CS-80 has 129.86 oz-in. of torque, and the speed is .25 sec/60°. The steering servo of robot is only needed to go to 0°, 90°, and -90°. This servo was not hacked.

Drive

The drive servo is massive. Because so much weight must be moved, the servo chosen was the Cirrus CS-600 FET. This servo can produce a huge 333.29 oz-in. of torque at 6 Volts. Its speed is .22 sec/60°. The drive servo was hacked so it can continuously go forwards or backwards.

Winch

The winch needed to be strong enough to lift the 6 lbs counterweight while moving the arm to the loading and arming position. This servo was a hacked version of the CS-80. It has the same specifications as the steering servo.

Lock

The lock servo is used to hold the arm in place while the winch releases the wire. This is needed because the winch would otherwise produce too much rotational friction in the arm and it would waste too much energy. This servo does not need to be very powerful, so a smaller Cirrus CS-60 was used. The speed of this servo is .14 sec/60°, and the torque is 56.38 oz-in.

Sensors

Four different sensors were used in the design of Mr. T.: a bump sensor, Infrared Range Detectors, a potentiometer, and a special vector processing unit.

Bump Sensor

Since only one bump sensor was used, it was connected in a simple voltage divider circuit to one of the analog ports. If the button was depressed the analog port read about 2.5 V, and if it was not pressed the port would read about 0 V.

Infrared Range Detector

The Infrared sensor used was the Sharp GP2DI2. This is a 40 kHz transmitter and receiver combined into one. The inputs to it are ground, +5 V, and it outputs an analog signal that represents the range from an object. The IR sensors are situated with one above the other so that if an object is detected on the lower IR but not the upper IR, than it is ammunition.

Potentiometer

The third sensor used was also very straight forward. The potentiometer attached to the pivot of the arm is connected to a voltage divider circuit. The value received by the analog port represents exactly where the arm is at any time. The potentiometer used had a range of 0 – 10 kΩ. When placed in series with another 10 kΩ resistor, the values read to the analog port ranged from 5 V – 2.5 V, and the circuit did not draw too much amperage to function properly. Initially lower resister values were used, but too much current was trying to be drawn, and it drained the batteries too quickly.

Vector Processing Unit

The vector processing was a special sensor designed specifically for Mr. T. Appendix C shows all of the code written for the VPU.

Function

The main function of the sensor is to give an accurate position measurement, including angle, based on an initial position.

Guidelines

The sensor had several important guidelines to make it useful. The most important one was that it must have a very small error. This is important for two main reasons. Since the device is going to sample many times per second and the same numbers are going to be used (because current location is always a function of previous location), any small error will continue to be compounded and make the later results very inaccurate. The other main reason is that since the target is so far away, any inaccuracies will be multiplied by this distance and make the final calculation off.

Implementation

To implement this, the robot uses an optical mouse and a dedicated 68HC11. Mr. T samples the mouse every 10 ms, and computes the calculations with the value returned. Figure 6 shows the block structure of this sensor.

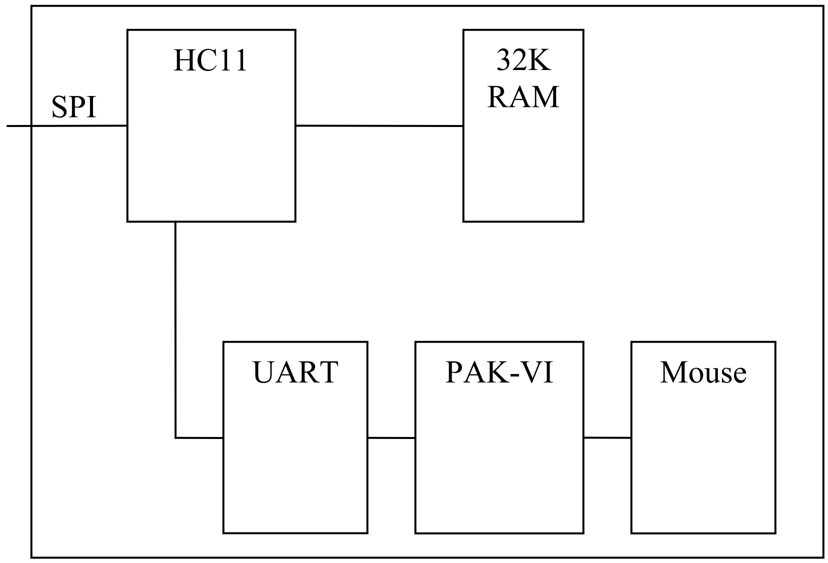


Figure 6 – Block Diagram

The intention is to make the sensor a "black box" device with a small instruction set. The only way the main processor of the robot can communicate with it is through the SPI interface. The instruction set is shown in figure 7.

Instruction	Opcode	Input	Output
Get X	$01		float XAbsolute
Get Y	$02		float YAboslute
Get Theta	$03		float ThetaAbsolute
Start Mouse	$04		
Stop Mouse	$05		
Reset Mouse Counter	$06		
Reset Angle	$07		
Compute distance	$08	float XTarget, YTarget	int Distance

Figure 7 – Instruction Set of Vector Processing Unit

The PAK-VI is a keyboard controller that can also be used as an all purpose PS/2 controller. There is an instruction that you can send it that passes the next byte of data directly to the PS/2 port. This function was used to put the mouse into remote mode. Remote mode is a method where instead of the mouse constantly streaming data to the PS/2 port, it only reports data when requested. The data it returns is three 8 bit values. From these values two 9 bit values representing changes in X and Y positions can be extracted. These values are then put through the mathematical calculations shown in figure 8. All calculations are done as 32 bit floating point numbers.

$$\Delta X = \text{Sampled X}$$
$$\Delta \text{Y} = \text{Sampled Y}$$
$$\text{R} = \text{Radius from mouse to wheels}$$
$$\Delta\Theta = \frac{\Delta X}{R}$$
$$\Theta = \Theta + \Delta\Theta$$
$$X = \Delta Y \cdot \cos(\Theta) + X$$
$$Y = \Delta Y \cdot \sin(\Theta) + Y$$

Figure 9 – Mathematical Vector Calculations

Errors

This sensor was never fully functional. The calculations simulated correctly, but the interface between the HC11 and the UART never worked properly. Because of this, no data concerning the accuracy of the sensor has yet to be created.

Behaviors

Mr. T has three main modes of operation. These modes are program, calibrate, and run.

Programming

In this mode, the robot remains stationary as new code is loaded into its memory. This mode is the initial mode the robot is in

Calibrate

In the calibration mode, the robot calibrates its sensor values for the room it is in. It should be placed 12" from a piece of ammunition, and 20" from the wall. It will then calibrate the IR. The pushbutton is pressed to confirm it is in the correct location. Since many problems were found with navigation, this mode has yet to be implemented.

Run

During the entire run mode, the robot is performing obstacle avoidance. The run mode can be broken down into five smaller modes: searching, loading, returning, aiming, and firing. The code for this mode can be found in appendix D.

Searching

In the searching mode, the robot is looking for ammunition. It starts out by rotating around in a circle, all the while checking the IR sensors for any objects. If an object is detected in the lower IR but not the upper, than the robot drives towards it. This object will be a piece of ammunition. If nothing is detected, or an obstacle is detected, it travels in a random direction away from the object. It continues to do this until an object is found. When it drives towards a piece of ammunition, it continues to drive forward until the bump switch is depressed, meaning an object is in the ready position. One this happens, the mode changes to the loading state.

Loading

When the robot is ready to load, it begins to winch down the arm. It continues to do this while checking the position of arm until the arm reaches a predetermined "loading" position. Once in the loading position, the electromagnet is turned on. After the electromagnet, the robot changes to arming mode.

Arming

When arming, the robot winches the arm down more while slowly moving away from the target. This causes the sling to slide under the robot. It again checks the arms position. Once the armed position is found, the lock is lowered and the winch is unwound for a predetermined amount of time.

Aiming

When aiming, the robot first turns its steering wheel to 90°, and then revolves until the vector processing unit says that it is pointed directly at the target. Once it is aimed at the target, the vector processing unit is queried for the position to release the ammunition for it to strike the target. Once this position is found, the robot emits a warning sound and counts down with an LED display for 5 seconds and then moves to the fire state.

Firing

In the fire state, the robot stops all other calculations and spends all of its time sampling the analog port. As soon as the value determined in the aiming state is found, the magnet is released and the target flies towards its target. After this, the robot returns to the searching state.

Experimental Results

The robot is not yet function enough to acquire proper experimental results. This section will be updated when the robot becomes functional.

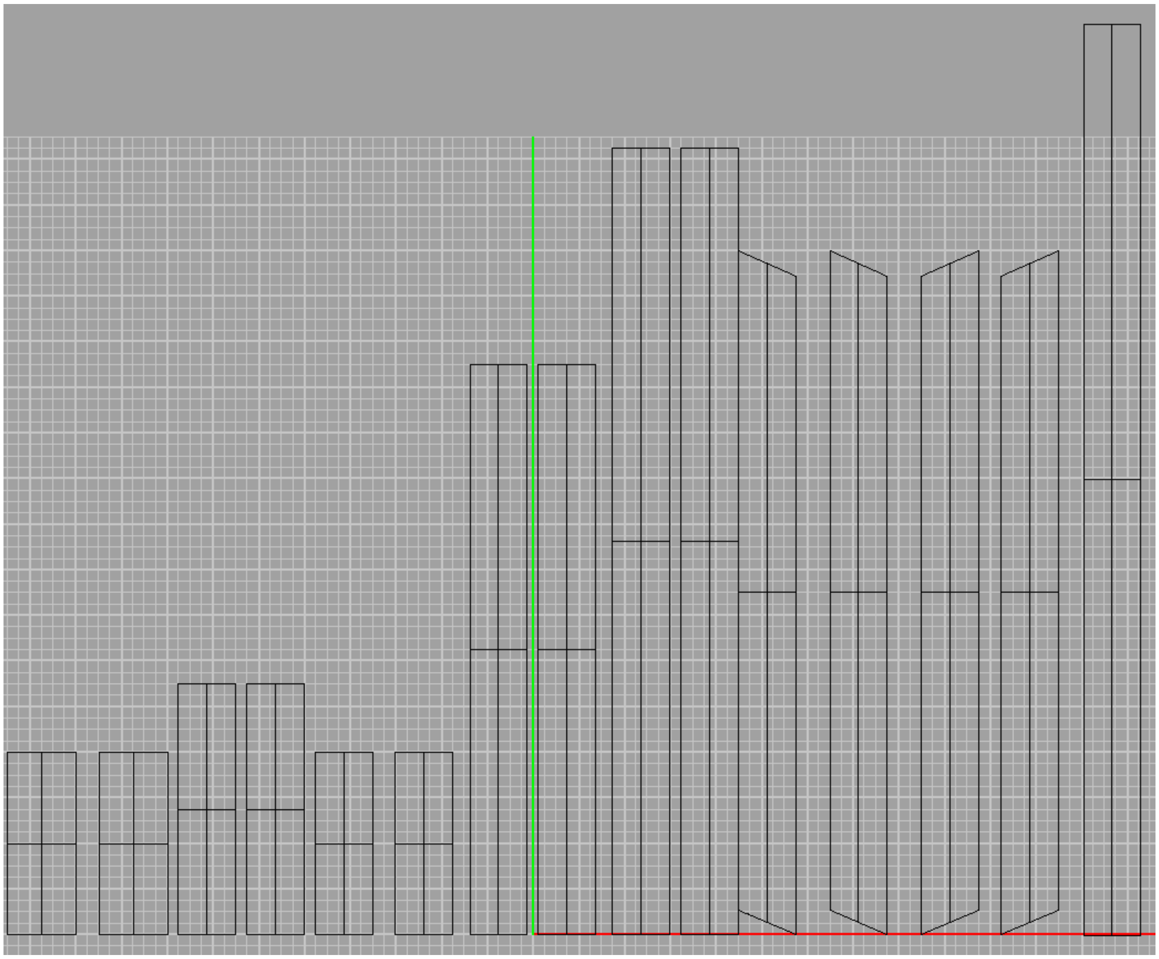
Conclusion

Although countless hours were spent in working on this robot, it has yet to do much of anything. I am still working on it, and it will be mostly functional soon, but at this point it does not work.

Many of my problems came from problems with hardware. It took me a very long time to discover that one of the ME11 boards I am using might not me working properly. Whenever my servos change direction or stop it resets my board. This even happens when the servos are on independent power supplies. I was also having problems with my UART. I can not check to see if my mouse sensor works correctly because the serial interface does not work.

I plan on finishing this project in the near future. I am very frustrated that it does not work correctly, and hope to have it fully functional soon.

Appendix A – Scale Drawing



Scale – 1 square = ¼"

Appendix B – Servo Code

```
*************************
* INIT INTERRUPT VECTORS *
*************************

      ORG   $00D3
      JMP   HANDLE_OC5
      JMP   HANDLE_OC4
      JMP   HANDLE_OC3
      JMP   HANDLE_OC2

      ORG   $8000
      LDX   #BASE
      CLRA
      STAA  TMSK2,X
      LDAA  #%01111000
      STAA  TMSK1,X
      STAA  TFLG1,X
      CLI

*******************
* WINCH IN THE ARM *
*******************

      LDAA  #0
      JSR   S_WINCH

LOWER JSR   GET_ARM
      LDAB  #$4F
      CBA

      BEQ   LOWER_DONE
      BRA   LOWER

LOWER_DONE
      LDAA  #50
      JSR   S_WINCH


here  bra   here



***************
* DRIVE SERVO *
***************

      ORG   $9000
S_DRIVE     LDAB  #50
      CBA
      BGT   DR_CW
      BLT   DR_CCW

      LDAA  #$0B
      STAA  $02
```

```
      RTS

DR_CW LDAA  #$04
      STAA  $02
      RTS

DR_CCW      LDAA  #$14
      STAA  $02
      RTS

*****************
* TURNING SERVO *
*****************

      ORG   $9100
S_TURN      LDAB  #50
      CBA
      BGT   TN_CW
      BLT   TN_CCW

      LDAA  #$0B
      STAA  $02
      RTS

TN_CW LDAA  #$04
      STAA  $02
      RTS

TN_CCW      LDAA  #$14
      STAA  $02
      RTS

***************
* WINCH SERVO *
***************

      ORG   $9200
S_WINCH     LDAB  #50
      CBA
      BGT   WN_CW
      BLT   WN_CCW

      LDAA  #$0B
      STAA  $08
      LDAA  #$A0
      STAA  $09
      RTS

WN_CW LDAA  #$04
      STAA  $08
      LDAA  #$A0
      STAA  $09
      RTS

WN_CCW      LDAA  #$14
      STAA  $08
      LDAA  #$A0
      STAA  $09
```

```
      RTS

**************
* LOCK SERVO *
**************

      ORG   $9300
S_LOCK      LDAB  #50
      CBA
      BGT   LK_CW
      BLT   LK_CCW

      LDAA  #$0B
      STAA  $02
      RTS

LK_CW LDAA  #$04
      STAA  $02
      RTS

LK_CCW      LDAA  #$14
      STAA  $02
      RTS

*************************
* INTURRUPT SUBROUTINES *
*************************

HANDLE_OC5:
      LDX   #BASE
      BCLR  TFLG1,X     %11110111
      LDAA  0,X
      ANDA  #%00001000
      BEQ   OC5DOWN
      LDD   $08
      BRA   OC5END
OC5DOWN
      LDD   #28960
OC5END
      ADDD  TOC5,X
      STD   TOC5,X
      RTI

HANDLE_OC4:
      LDX   #BASE
      BCLR  TFLG1,X     %11101111
      LDAA  0,X
      ANDA  #%00010000
      BEQ   OC4DOWN
      LDD   $08
      BRA   OC4END
OC4DOWN
      LDD   #28960
OC4END
      ADDD  TOC4,X
      STD   TOC4,X
      RTI
```

```
HANDLE_OC3:
      LDX   #BASE
      BCLR  TFLG1,X     %11011111
      LDAA  0,X
      ANDA  #%00100000
      BEQ   OC3DOWN
      LDD   $08
      BRA   OC3END
OC3DOWN
      LDD   #28960
OC3END
      ADDD  TOC3,X
      STD   TOC3,X
      RTI

HANDLE_OC2:
      LDX   #BASE
      BCLR  TFLG1,X     %10111111
      LDAA  0,X
      ANDA  #%01000000
      BEQ   OC2DOWN
      LDD   $08
      BRA   OC2END
OC2DOWN
      LDD   #28960
OC2END
      ADDD  TOC2,X
      STD   TOC2,X
      RTI
```

Appendix C – Vector Processing Unit Code

```
*********************************************************************************
*                             Mr. T Processor 2                                 *
*                                                                               *
*       Mr. T Processor 2 is called the VPU, which stands for Vector Processing *
*       Unit.                                                                   *
*       This uses the floating point package provided by Motorola (Written by   *
*       Gordon                                                                  *
*       Doughman).  I have defined below which portion of the code is mine, and *
*       which                                                                   *
*       portion is his.  Full documentation of his code and what it does can be *
*       found                                                                   *
*       on the Motorola website.  My code will do the following:                *
*       *       Interact with the Master Processor using SPI                    *
*       *       Act based on instructions sent from Master                      *
*       *       Initialize and read input from a mouse through the PAK-IV       *
*       *       Analyze this data to compute positioning relative to the start  *
*       *       Compute the release time of the device to strike the target     *
*                                                                               *
*********************************************************************************

*********************************
*       HC11FP Constants      *
*********************************

               ORG    $0000

FPACC1EX       RMB    1       FLOATING POINT ACCUMULATOR #1..
FPACC1MN       RMB    3
MANTSGN1       RMB    1        MANTISSA SIGN FOR FPACC1 (0=+, FF=-).
FPACC2EX       RMB    1        FLOATING POINT ACCUMULATOR #2.
FPACC2MN       RMB    3
MANTSGN2       RMB    1        MANTISSA SIGN FOR FPACC2 (0=+, FF=-).


FLTFMTER       EQU    1        floating point format error in ASCFLT
OVFERR         EQU    2        floating point overflow error
UNFERR         EQU    3        floating point underflow error
DIV0ERR        EQU    4        division by 0 error
TOLGSMER       EQU    5        number too large or small to convert to int.
NSQRTERR       EQU    6        tried to take the square root of negative #
TAN90ERR       EQU    7        TANgent of 90 degrees attempted

*****************************************
*       My Constants and Variables    *
*****************************************

               ORG     $2000

DELTA_X_16     RMB     2
DELTA_Y_16     RMB     2
DELTA_X_FP     RMB     4
DELTA_Y_FP     RMB     4
RET0           RMB     1
RET1           RMB     1
RET2           RMB     1
ABSOLUTE_X     RMB     4
ABSOLUTE_Y     RMB     4
ABSOLUTE_THETA RMB     4
TEMP_X_FP      RMB     4
TEMP_Y_FP      RMB     4
TEMP_THETA     RMB     4
MOUSE_ENABLE   RMB     1
DIVISOR        RMB     4
MAX            RMB     4
MIN            RMB     4

DISTANCE       EQU     $0230
V_GET_X        EQU     $01
```

```
V_GET_Y         EQU     $02
V_GET_THETA     EQU     $03
V_START_MOUSE   EQU     $04
V_STOP_MOUSE    EQU     $05
V_RESET_M_CNT   EQU     $06
V_RESET_ANGLE   EQU     $07
V_COMP_DIST     EQU     $08

BASE            EQU     $1000
STACK           EQU     $7000
PORTA           EQU     $00
PIOC            EQU     $02
PORTC           EQU     $03
PORTB           EQU     $04
PORTCL          EQU     $05
DDRC            EQU     $07
PORTD           EQU     $08
DDRD            EQU     $09
PORTE           EQU     $0A
CFORC           EQU     $0B
OC1M            EQU     $0C
OC1D            EQU     $0D
TCNT            EQU     $0E
TIC1            EQU     $10
TIC2            EQU     $12
TIC3            EQU     $14
TOC1            EQU     $16
TOC2            EQU     $18
TOC3            EQU     $1A
TOC4            EQU     $1C
TI4/O5          EQU     $1E
TCTL1           EQU     $20
TCTL2           EQU     $21
TMSK1           EQU     $22
TFLG1           EQU     $23
TMSK2           EQU     $24
TFLG2           EQU     $25
PACTL           EQU     $26
PACNT           EQU     $27
SPCR            EQU     $28
SPSR            EQU     $29
SPDR            EQU     $2A
BAUD            EQU     $2B
SCCR1           EQU     $2C
SCCR2           EQU     $2D
SCSR            EQU     $2E
SCDR            EQU     $2F
ADCTL           EQU     $30
ADR1            EQU     $31
ADR2            EQU     $32
ADR3            EQU     $33
ADR4            EQU     $34
BPROT           EQU     $35
EPROG           EQU     $36
OPTION          EQU     $39
COPRST          EQU     $3A
PPROG           EQU     $3B
HPRIO           EQU     $3C
INIT            EQU     $3D
TEST            EQU     $3E
CONFIG          EQU     $3F

BIT0            EQU     %00000001
BIT1            EQU     %00000010
BIT2            EQU     %00000100
BIT3            EQU     %00001000
BIT4            EQU     %00010000
BIT5            EQU     %00100000
BIT6            EQU     %01000000
BIT7            EQU     %10000000

BIT543          EQU     %00111000
```

```
***
* SCI Int Vector
***
       ORG     $00C7
       JMP     SPI_ISR

*********************************
*      My Main Function       *
*********************************

       ORG     $8000

******************
* INITIALIZATION *
******************

       LDS     #STACK

       LDX     #DELTA_X_16
       LDAA    #52

***
*      Initialize Memory
***

LOOP1  CLR     0,X
       INX
       DECA
       BNE     LOOP1
***
*      Set The Divisor
***
       LDD     #DISTANCE
       JSR     Convert_D
       JSR     TFR1TO2

       LDD     #$0000
       JSR     Convert_D
       JSR     FLTDIV
       LDX     #DIVISOR
       JSR     PUTFPAC1
***
*      Set Max and Min
***
       LDD     #$0002
       JSR     Convert_D
       LDX     #MAX
       JSR     PUTFPAC1

       LDX     #FPACC1EX
       LDAA    #$FF
       STAA    4,X
       LDX     #MIN
       JSR     PUTFPAC1
***
*      Init Systems
***
       JSR     Init_SCI
       JSR     Init_SPI
***
*      Main Loop
***

       WAI
MAIN_LOOP
       LDAA    MOUSE_ENABLE
       CMPA    #$FF
       BNE     MAIN_LOOP
       JSR     Get_Local
       JSR     Compute_Vectors
       BRA     MAIN_LOOP

*****************************************
```

```
Calc_Dist:
       LDD     #$AA
       RTS

*******************************************

Check_Status:
       LDX     #BASE
       LDAA    #$E9
       JSR     Send_Rcv_3
       LDAA    #$60
       CMPA    RET0
       BNE     C_S_1
       LDAA    #8
       CMPA    RET1
       BNE     C_S_1
       LDAA    #200
       CMPA    RET2
       BNE     C_S_1
       CLRA
       RTS
C_S_1
       LDAA    #$FF
       RTS

*******************************************

Clear_Angle:
       PSHA
       CLRA
       LDX     #TEMP_THETA
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       LDS     #ABSOLUTE_THETA
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       PULA
       RTS

*******************************************

Clear_Counter:
       PSHA
       CLRA
       LDX     #TEMP_X_FP
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       LDX     #TEMP_Y_FP
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       LDX     #ABSOLUTE_X
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       LDX     #ABSOLUTE_Y
       STAA    0,X
       STAA    1,X
       STAA    2,X
       STAA    3,X
       PULA
       RTS

*******************************************
```

```
Compute_Vectors:
        LDX     #DELTA_X_FP
        JSR     GETFPAC1
        LDX     #DIVISOR
        JSR     FLTMUL
        LDX     #ABSOLUTE_THETA
        JSR     FLTADD
        LDX     #MAX
        JSR     GETFPAC2
        JSR     FLTCMP
        BLE     C_V_1
        JSR     FLTSUB
C_V_1
        LDX     #MIN
        JSR     GETFPAC2
        JSR     FLTCMP
        BLE     C_V_2
        JSR     FLTSUB
C_V_2
        LDX     #TEMP_THETA
        JSR     PUTFPAC1

        LDX     #DELTA_Y_FP
        JSR     GETFPAC2
        JSR     FLTCOS
        JSR     FLTMUL
        LDX     #TEMP_X_FP
        JSR     PUTFPAC1

        LDX     #TEMP_THETA
        JSR     GETFPAC1
        LDX     #DELTA_Y_FP
        JSR     GETFPAC2
        JSR     FLTSIN
        JSR     FLTMUL
        LDX     #TEMP_Y_FP
        JSR     PUTFPAC1

        LDX     #ABSOLUTE_X
        JSR     GETFPAC1
        LDX     #TEMP_X_FP
        JSR     GETFPAC2
        JSR     FLTADD
        JSR     PUTFPAC1

        LDX     #ABSOLUTE_Y
        JSR     GETFPAC1
        LDX     #TEMP_Y_FP
        JSR     GETFPAC2
        JSR     FLTADD
        JSR     PUTFPAC1

        PSHA
        PSHB
        LDX     #TEMP_X_FP
        LDY     #ABSOLUTE_X
        LDD     0,X
        STD     0,Y
        LDD     2,X
        STD     2,Y
        LDX     #TEMP_Y_FP
        LDY     #ABSOLUTE_Y
        LDD     0,X
        STD     0,Y
        LDD     2,X
        STD     2,Y
        LDX     #TEMP_THETA
        LDY     #ABSOLUTE_THETA
        LDD     0,X
        STD     0,Y
        LDD     2,X
        STD     2,Y
```

```
        PULB
        PULA
        RTS

****************************************

Conv_16_32:
        PSHA
        PSHB
        LDX     #FPACC1EX
        LDD     DELTA_X_16
        STD     2,X
        JSR     SINT2FLT
        LDX     #DELTA_X_FP
        JSR     PUTFPAC1
        LDX     #FPACC1EX
        LDD     DELTA_Y_16
        STD     2,X
        JSR     SINT2FLT
        LDX     #DELTA_X_FP
        JSR     PUTFPAC1
        PULB
        PULA
        RTS

****************************************

Convert_D:
        LDX     #FPACC1EX
        STD     2,X
        LDD     #$00
        STD     0,X
        STAA    4,X
        JSR     UINT2FLT
        RTS

****************************************

Disable_Mouse:
        PSHA
        LDAA    #$00
        STAA    MOUSE_ENABLE
        PULA
        RTS

****************************************

Get_Local:
        LDAA    #$EB
        JSR     Send_Rcv_3
        LDD     #$00
        BRCLR   RET0    BIT4    S_L_1
        LDAA    #$FF
S_L_1
        LDAB    RET1
        STD     DELTA_X_16
        LDD     #$00
        BRCLR   RET0    BIT5    S_L_2
        LDAA    #$FF
S_L_2
        LDAB    RET2
        STD     DELTA_Y_16
        RTS

****************************************

Init_Mouse:
        PSHA
        LDAA    #$FF
        JSR     Send_PAK
        LDAA    #$02
        JSR     Send_PAK
        LDAA    #$F0
```

```
        JSR     Send_Mouse
        LDAA    #$E8
        JSR     Send_Mouse
        LDAA    #$03
        JSR     Send_Mouse
        LDAA    #$F3
        JSR     Send_Mouse
        LDAA    #$C8
        JSR     Send_Mouse
        JSR     Check_Status
        CMPA    #$00
        BNE     Init_Mouse
        LDAA    #$FF
        STAA    MOUSE_ENABLE
        PULA
        RTS

*******************************************

Init_SPI:
        LDX     #BASE
        BSET    DDRD,X  BIT2
        BSET    DDRD,X  BIT543
        LDAA    #$C7
        STAA    SPCR,X
        CLI
        RTS

*******************************************

Init_SCI:
        PSHA
        LDX     #BASE
        LDAA    #$30
        STAA    BAUD,X
        LDAA    #$00
        STAA    SCCR1,X
        LDAA    #$00
        STAA    SCCR2,X
        PULA
        RTS

*******************************************

Send_Master_8:
        LDX     #BASE
        STAA    SPDR,X
S_M_8_1
        BRCLR   SPSR,X  BIT7    S_M_8_1
        RTS

*******************************************

Send_Master_32:
        PSHA
        LDAA    0,X
        JSR     Send_Master_8
        LDAA    1,X
        JSR     Send_Master_8
        LDAA    2,X
        JSR     Send_Master_8
        LDAA    3,X
        JSR     Send_Master_8
        PULA
        RTS

*******************************************

Send_Mouse:
        LDX     #BASE
        PSHB
        LDAB    #$0B
        STAB    SCDR,X
```

```
S_M_1
       BRCLR   SCSR,X  BIT5    S_M_1
       STAA    SCDR,X
S_M_2
       BRCLR   SCSR,X  BIT5    S_M_2
       PULB
       RTS

******************************************

Send_PAK:
       LDX     #BASE
       STAA    SCDR,X
S_P_1
       BRCLR   SCSR,X  BIT6    S_P_1
       RTS

******************************************

Send_Rcv_3:
       LDX     #BASE
       PSHA
       LDAA    #$0B
       STAA    SCDR,X
S_R_3_1
       BRCLR   SCSR,X  BIT6    S_R_3_1;       BYTE SENT?
       PULA
       STAA    SCDR,X;                        SEND BYTE
S_R_3_2
       BRCLR   SCSR,X  BIT5    S_R_3_2;       BYTE RECEIVED?
       LDAA    SCDR,X
       STAA    RET0
S_R_3_3
       BRCLR   SCSR,X  BIT5    S_R_3_3;       BYTE RECEIVED?
       LDAA    SCDR,X
       STAA    RET1
S_R_3_4
       BRCLR   SCSR,X  BIT5    S_R_3_4;       BYTE RECEIVED?
       LDAA    SCDR,X
       STAA    RET2
       RTS

******************************************

SPI_ISR:
       LDX     #BASE
       BRCLR   SPSR,X  BIT7    RT_SPI

       LDAA    SPDR,X
       CMPA    #V_GET_X
       BEQ     GET_X
       CMPA    #V_GET_Y
       BEQ     GET_Y
       CMPA    #V_GET_THETA
       BEQ     GET_THETA
       CMPA    #V_START_MOUSE
       BEQ     START_MOUSE
       CMPA    #V_STOP_MOUSE
       BEQ     STOP_MOUSE
       CMPA    #V_RESET_M_CNT
       BEQ     RESET_M_CNT
       CMPA    #V_RESET_ANGLE
       BEQ     RESET_ANGLE
       CMPA    #V_COMP_DIST
       BEQ     COMP_DIST
       BRA     SPI_ISR_ERROR

GET_X
       LDX     #ABSOLUTE_X
       JSR     Send_Master_8
       JSR     Send_Master_32
       BRA     RT_SPI
```

```
GET_Y
       LDX     #ABSOLUTE_Y
       JSR     Send_Master_8
       JSR     Send_Master_32
       BRA     RT_SPI

GET_THETA
       LDX     #ABSOLUTE_THETA
       JSR     Send_Master_8
       JSR     Send_Master_32
       BRA     RT_SPI

START_MOUSE
       JSR     Init_Mouse
       JSR     Send_Master_8
       BRA     RT_SPI

STOP_MOUSE
       JSR     Disable_Mouse
       JSR     Send_Master_8
       BRA     RT_SPI

RESET_M_CNT
       JSR     Init_Mouse
       JSR     Clear_Counter
       JSR     Send_Master_8
       BRA     RT_SPI

RESET_ANGLE
       JSR     Clear_Angle
       JSR     Send_Master_8
       BRA     RT_SPI

COMP_DIST
       JSR     Send_Master_8
       JSR     Calc_Dist
       JSR     Send_Master_8
       TBA
       JSR     Send_Master_8
       BRA     RT_SPI

SPI_ISR_ERROR
       LDAA    $#FF
       JSR     Send_Master_8

RT_SPI
       RTI


***********************
*      End My Code    *
***********************


******************************************************************************
*                                                                            *
*                       ASCII TO FLOATING POINT ROUTINE                      *
*                                                                            *
*       This routine will accept most any ASCII floating point format        *
*       and return a 32-bit floating point number.  The following are        *
*       some examples of legal ASCII floating point numbers.                 *
*                                                                            *
*       20.095                                                               *
*       0.125                                                                *
*       7.2984E10                                                            *
*       167.824E5                                                            *
*       5.9357E-7                                                            *
*       500                                                                  *
*                                                                            *
*       The floating point number returned is in "FPACC1".                   *
*                                                                            *
*                                                                            *
*       The exponent is biased by 128 to facilitate floating point           *
```

```
*       comparisons.  A pointer to the ASCII string is passed to the         *
*       routine in the D-register.                                            *
*                                                                             *
*                                                                             *
*******************************************************************************
*
*
*        ORG    $0000
*
*        FPACC1EX RMB    1            FLOATING POINT ACCUMULATOR #1..
*        FPACC1MN RMB    3
*        MANTSGN1 RMB    1            MANTISSA SIGN FOR FPACC1 (0=+, FF=-).
*        FPACC2EX RMB    1            FLOATING POINT ACCUMULATOR #2.
*        FPACC2MN RMB    3
*        MANTSGN2 RMB    1            MANTISSA SIGN FOR FPACC2 (0=+, FF=-).
*
*
*        FLTFMTER EQU    1
*
*
*        LOCAL VARIABLES (ON STACK POINTED TO BY Y)
*
EXPSIGN  EQU    0            EXPONENT SIGN (0=+, FF=-).
PWR10EXP EQU    1            POWER 10 EXPONENT.
*
*
         ORG    $C000        (TEST FOR EVB)
*
ASCFLT   EQU    *
         PSHX                SAVE POINTER TO ASCII STRING.
         JSR    PSHFPAC2     SAVE FPACC2.
         LDX    #0           PUSH ZEROS ON STACK TO INITIALIZE LOCALS.
         PSHX                ALLOCATE 2 BYTES FOR LOCALS.
         STX    FPACC1EX     CLEAR FPACC1.
         STX    FPACC1EX+2
         CLR    MANTSGN1     MAKE THE MANTISSA SIGN POSITIVE INITIALLY.
         TSY                 POINT TO LOCALS.
         LDX    6,Y          GET POINTER TO ASCII STRING.
ASCFLT1  LDAA   0,X          GET 1ST CHARACTER IN STRING.
         JSR    NUMERIC      IS IT A NUMBER.
         BCS    ASCFLT4      YES. GO PROCESS IT.
*
*        LEADING MINUS SIGN ENCOUNTERED?
*
ASCFLT2  CMPA   #'-          NO. IS IT A MINUS SIGN?
         BNE    ASCFLT3      NO. GO CHECK FOR DECIMAL POINT.
         COM    MANTSGN1     YES. SET MANTISSA SIGN. LEADING MINUS BEFORE?
         INX                 POINT TO NEXT CHARACTER.
         LDAA   0,X          GET IT.
         JSR    NUMERIC      IS IT A NUMBER?
         BCS    ASCFLT4      YES. GO PROCESS IT.
*
*        LEADING DECIMAL POINT?
*

ASCFLT3  CMPA   #'.          IS IT A DECIMAL POINT?
         BNE    ASCFLT5      NO. FORMAT ERROR.
         INX                 YES. POINT TO NEXT CHARACTER.
         LDAA   0,X          GET IT.
         JSR    NUMERIC      MUST HAVE AT LEAST ONE DIGIT AFTER D.P.
         BCC    ASCFLT5      GO REPORT ERROR.
         JMP    ASCFLT11     GO BUILD FRACTION.
*
*        FLOATING POINT FORMAT ERROR
*
ASCFLT5  INS                 DE-ALLOCATE LOCALS.
         INS
         JSR    PULFPAC2     RESTORE FPACC2.
         PULX                GET POINTER TO TERMINATING CHARACTER IN STRING.
         LDAA   #FLTFMTER    FORMAT ERROR.
         SEC                 SET ERROR FLAG.
         RTS                 RETURN.
*
```

```
*        PRE DECIMAL POINT MANTISSA BUILD
*
ASCFLT4  LDAA   0,X
         JSR    NUMERIC
         BCC    ASCFLT10
         JSR    ADDNXTD
         INX
         BCC    ASCFLT4
*
*        PRE DECIMAL POINT MANTISSA OVERFLOW
*
ASCFLT6  INC    FPACC1EX     INC FOR EACH DIGIT ENCOUNTERED PRIOR TO D.P.
         LDAA   0,X          GET NEXT CHARACTER.
         INX                 POINT TO NEXT.
         JSR    NUMERIC      IS IT S DIGIT?
         BCS    ASCFLT6      YES. KEEP BUILDING POWER 10 MANTISSA.
         CMPA   #'.          NO. IS IT A DECIMAL POINT?
         BNE    ASCFLT7      NO. GO CHECK FOR THE EXPONENT.
*
*        ANY FRACTIONAL DIGITS ARE NOT SIGNIFIGANT
*
ASCFLT8  LDAA   0,X          GET THE NEXT CHARACTER.
         JSR    NUMERIC      IS IT A DIGIT?
         BCC    ASCFLT7      NO. GO CHECK FOR AN EXPONENT.
         INX                 POINT TO THE NEXT CHARACTER.
         BRA    ASCFLT8      FLUSH REMAINING DIGITS.
ASCFLT7  CMPA   #'E          NO. IS IT THE EXPONENT?
         BEQ    ASCFLT13     YES. GO PROCESS IT.
         JMP    FINISH       NO. GO FINISH THE CONVERSION.
*
*        PROCESS THE EXPONENT
*
ASCFLT13 INX                 POINT TO NEXT CHARACTER.
         LDAA   0,X          GET THE NEXT CHARACTER.
         JSR    NUMERIC      SEE IF IT'S A DIGIT.
         BCS    ASCFLT9      YES. GET THE EXPONENT.
         CMPA   #'-          NO. IS IT A MINUS SIGN?
         BEQ    ASCFLT15     YES. GO FLAG A NEGATIVE EXPONENT.
         CMPA   #'+          NO. IS IT A PLUS SIGN?
         BEQ    ASCFLT16     YES. JUST IGNORE IT.
         BRA    ASCFLT5      NO. FORMAT ERROR.
ASCFLT15 COM    EXPSIGN,Y    FLAG A NEGATIVE EXPONENT. IS IT 1ST?
ASCFLT16 INX                 POINT TO NEXT CHARACTER.
         LDAA   0,X          GET NEXT CHARACTER.
         JSR    NUMERIC      IS IT A NUMBER?
         BCC    ASCFLT5      NO. FORMAT ERROR.
ASCFLT9  SUBA   #$30         MAKE IT BINARY.
         STAA   PWR10EXP,Y   BUILD THE POWER 10 EXPONENT.
         INX                 POINT TO NEXT CHARACTER.
         LDAA   0,X          GET IT.
         JSR    NUMERIC      IS IT NUMERIC?
         BCC    ASCFLT14     NO. GO FINISH UP THE CONVERSION.
         LDAB   PWR10EXP,Y   YES. GET PREVIOUS DIGIT.
         LSLB                MULT. BY 2.
         LSLB                NOW BY 4.
         ADDB   PWR10EXP,Y   BY 5.
         LSLB                BY 10.
         SUBA   #$30         MAKE SECOND DIGIT BINARY.
         ABA                 ADD IT TO FIRST DIGIT.
         STAA   PWR10EXP,Y
         CMPA   #38          IS THE EXPONENT OUT OF RANGE?
         BHI    ASCFLT5      YES. REPORT ERROR.
ASCFLT14 LDAA   PWR10EXP,Y   GET POWER 10 EXPONENT.
         TST    EXPSIGN,Y    WAS IT NEGATIVE?
         BPL    ASCFLT12     NO. GO ADD IT TO BUILT 10 PWR EXPONENT.
         NEGA
ASCFLT12 ADDA   FPACC1EX     FINAL TOTAL PWR 10 EXPONENT.
         STAA   FPACC1EX     SAVE RESULT.
         BRA    FINISH       GO FINISH UP CONVERSION.
*
*        PRE-DECIMAL POINT NON-DIGIT FOUND, IS IT A DECIMAL POINT?
*
ASCFLT10 CMPA   #'.          IS IT A DECIMAL POINT?
```

```
         BNE    ASCFLT7       NO. GO CHECK FOR THE EXPONENT.
         INX                  YES. POINT TO NEXT CHARACTER.
*
*        POST DECIMAL POINT PROCESSING
*
ASCFLT11 LDAA   0,X           GET NEXT CHARACTER.
         JSR    NUMERIC       IS IT NUMERIC?
         BCC    ASCFLT7       NO. GO CHECK FOR EXPONENT.
         BSR    ADDNXTD       YES. ADD IN THE DIGIT.
         INX                  POINT TO THE NEXT CHARACTER.
         BCS    ASCFLT8       IF OVER FLOW, FLUSH REMAINING DIGITS.
         DEC    FPACC1EX      ADJUST THE 10 POWER EXPONENT.
         BRA    ASCFLT11      PROCESS ALL FRACTIONAL DIGITS.
*
*
*
ADDNXTD  LDAA   FPACC1MN      GET UPPER 8 BITS.
         STAA   FPACC2MN      COPY INTO FPAC2.
         LDD    FPACC1MN+1    GET LOWER 16 BITS OF MANTISSA.
         STD    FPACC2MN+1    COPY INTO FPACC2.
         LSLD                 MULT. BY 2.
         ROL    FPACC1MN      OVERFLOW?
         BCS    ADDNXTD1      YES. DON'T ADD THE DIGIT IN.
         LSLD                 MULT BY 4.
         ROL    FPACC1MN      OVERFLOW?
         BCS    ADDNXTD1      YES. DON'T ADD THE DIGIT IN.
         ADDD   FPACC2MN+1    BY 5.
         PSHA                 SAVE A.
         LDAA   FPACC1MN      GET UPPER 8 BITS.
         ADCA   #0            ADDIN POSSABLE CARRY FROM LOWER 16 BITS.
         ADDA   FPACC2MN      ADD IN UPPER 8 BITS.
         STAA   FPACC1MN      SAVE IT.
         PULA                 RESTORE A.
         BCS    ADDNXTD1      OVERFLOW? IF SO DON'T ADD IT IN.
         LSLD                 BY 10.
         ROL    FPACC1MN
         STD    FPACC1MN+1    SAVE THE LOWER 16 BITS.
         BCS    ADDNXTD1      OVERFLOW? IF SO DON'T ADD IT IN.
         LDAB   0,X           GET CURRENT DIGIT.
         SUBB   #$30          MAKE IT BINARY.
         CLRA                 16-BIT.
         ADDD   FPACC1MN+1    ADD IT IN TO TOTAL.
         STD    FPACC1MN+1    SAVE THE RESULT.
         LDAA   FPACC1MN      GET UPPER 8 BITS.
         ADCA   #0            ADD IN POSSIBLE CARRY. OVERFLOW?
         BCS    ADDNXTD1      YES. COPY OLD MANTISSA FROM FPACC2.
         STAA   FPACC1MN      NO. EVERYHING OK.
         RTS                  RETURN.
ADDNXTD1 LDD    FPACC2MN+1    RESTORE THE ORIGINAL MANTISSA BECAUSE
         STD    FPACC1MN+1    OF OVERFLOW.
         LDAA   FPACC2MN
         STAA   FPACC1MN
         RTS                  RETURN.
*
*
*
*        NOW FINISH UP CONVERSION BY MULTIPLYING THE RESULTANT MANTISSA
*        BY 10 FOR EACH POSITIVE POWER OF 10 EXPONENT RECIEVED OR BY .1
*        (DIVIDE BY 10) FOR EACH NEGATIVE POWER OF 10 EXPONENT RECIEVED.
*
*
FINISH   EQU    *
         STX    6,Y           SAVE POINTER TO TERMINATING CHARACTER IN STRING.
         LDX    #FPACC1EX     POINT TO FPACC1.
         JSR    CHCK0         SEE IF THE NUMBER IS ZERO.
         BEQ    FINISH3       QUIT IF IT IS.
         LDAA   FPACC1EX      GET THE POWER 10 EXPONENT.
         STAA   PWR10EXP,Y    SAVE IT.
         LDAA   #$80+24       SET UP INITIAL EXPONENT (# OF BITS + BIAS).
         STAA   FPACC1EX
         JSR    FPNORM        GO NORMALIZE THE MANTISSA.
         TST    PWR10EXP,Y    IS THE POWER 10 EXPONENT POSITIVE OR ZERO?
         BEQ    FINISH3       IT'S ZERO, WE'RE DONE.
```

```
        BPL    FINISH1       IT'S POSITIVE MULTIPLY BY 10.
        LDX    #CONSTP1      NO. GET CONSTANT .1 (DIVIDE BY 10).
        JSR    GETFPAC2      GET CONSTANT INTO FPACC2.
        NEG    PWR10EXP,Y    MAKE THE POWER 10 EXPONENT POSITIVE.
        BRA    FINISH2       GO DO THE MULTIPLIES.
FINISH1 LDX    #CONST10      GET CONSTANT '10' TO MULTIPLY BY.
        JSR    GETFPAC2      GET CONSTANT INTO FPACC2.
FINISH2 JSR    FLTMUL        GO MULTIPLY FPACC1 BY FPACC2, RESULT IN FPACC1.
        DEC    PWR10EXP,Y    DECREMENT THE POWER 10 EXPONENT.
        BNE    FINISH2       GO CHECK TO SEE IF WE'RE DONE.
FINISH3 INS                  DE-ALLOCATE LOCALS.
        INS
        JSR    PULFPAC2      RESTORE FPACC2.
        PULX                 GET POINTER TO TERMINATING CHARACTER IN STRING.
        RTS                  RETURN WITH NUMBER IN FPACC1.
*
*
NUMERIC EQU    *
        CMPA   #'0           IS IT LESS THAN AN ASCII 0?
        BLO    NUMERIC1      YES. NOT NUMERIC.
        CMPA   #'9           IS IT GREATER THAN AN ASCII 9?
        BHI    NUMERIC1      YES. NOT NUMERIC.
        SEC                  IT WAS NUMERIC. SET THE CARRY.
        RTS                  RETURN.
NUMERIC1 CLC                 NON-NUMERIC CHARACTER. CLEAR THE CARRY.
        RTS                  RETURN.
*
FPNORM  EQU    *
        LDX    #FPACC1EX     POINT TO FPACC1.
        BSR    CHCK0         CHECK TO SEE IF IT'S 0.
        BEQ    FPNORM3       YES. JUST RETURN.
        TST    FPACC1MN      IS THE NUMBER ALREADY NORMALIZED?
        BMI    FPNORM3       YES. JUST RETURN..
FPNORM1 LDD    FPACC1MN+1    GET THE LOWER 16 BITS OF THE MANTISSA.
FPNORM2 DEC    FPACC1EX      DECREMENT THE EXPONENT FOR EACH SHIFT.
        BEQ    FPNORM4       EXPONENT WENT TO 0. UNDERFLOW.
        LSLD                 SHIFT THE LOWER 16 BITS.
        ROL    FPACC1MN      ROTATE THE UPPER 8 BITS. NUMBER NORMALIZED?
        BPL    FPNORM2       NO. KEEP SHIFTING TO THE LEFT.
        STD    FPACC1MN+1    PUT THE LOWER 16 BITS BACK INTO FPACC1.
FPNORM3 CLC                  SHOW NO ERRORS.
        RTS                  YES. RETURN.
FPNORM4 SEC                  FLAG ERROR.
        RTS                  RETURN.
*
CHCK0   EQU    *             CHECKS FOR ZERO IN FPACC POINTED TO BY X.
        PSHB                 SAVE D.
        PSHA
        LDD    0,X           GET FPACC EXPONENT & HIGH 8 BITS.
        BNE    CHCK01        NOT ZERO. RETURN.
        LDD    2,X           CHECK LOWER 16 BITS.
CHCK01  PULA                 RESTORE D.
        PULB
        RTS                  RETURN WITH CC SET.
*
CONSTP1 FCB    $7D,$4C,$CC,$CD          0.1 DECIMAL
CONST10 FCB    $84,$20,$00,$00          10.0 DECIMAL
*
*
****************************************************************************
*                                                                          *
*                      FPMULT: FLOATING POINT MULTIPLY                     *
*                                                                          *
*       THIS FLOATING POINT MULTIPLY ROUTINE MULTIPLIES "FPACC1" BY        *
*       "FPACC2" AND PLACES THE RESULT IN TO FPACC1. FPACC2 REMAINS        *
*       UNCHANGED.                                                         *
*                        WORSE CASE = 2319 CYCLES = 1159 uS @ 2MHz         *
*                                                                          *
****************************************************************************
*
*
FLTMUL  EQU    *
        JSR    PSHFPAC2      SAVE FPACC2.
```

```
         LDX    #FPACC1EX    POINT TO FPACC1
         JSR    CHCK0        CHECK TO SEE IF FPACC1 IS ZERO.
         BEQ    FPMULT3      IT IS. ANSWER IS 0.
         LDX    #FPACC2EX    POINT TO FPACC2.
         JSR    CHCK0        IS IT 0?
         BNE    FPMULT4      NO. CONTINUE.
         CLRA                CLEAR D.
         CLRB
         STD    FPACC1EX     MAKE FPACC1 0.
         STD    FPACC1MN+1
         BRA    FPMULT3      RETURN.
FPMULT4  LDAA   MANTSGN1     GET FPACC1 EXPONENT.
         EORA   MANTSGN2     SET THE SIGN OF THE RESULT.
         STAA   MANTSGN1     SAVE THE SIGN OF THE RESULT.
         LDAA   FPACC1EX     GET FPACC1 EXPONENT.
         ADDA   FPACC2EX     ADD IT TO FPACC2 EXPONENT.
         BPL    FPMULT1      IF RESULT IS MINUS AND
         BCC    FPMULT2      THE CARRY IS SET THEN:
FPMULT5  LDAA   #OVFERR      OVERFLOW ERROR.
         SEC                 SET ERROR FLAG.
         BRA    FPMULT6      RETURN.
FPMULT1  BCS    FPMULT2      IF RESULT IS PLUS & THE CARRY IS SET THEN ALL OK.
         LDAA   #UNFERR      ELSE UNDERFLOW ERROR OCCURED.
         SEC                 FLAG ERROR.
         BRA    FPMULT6      RETURN.
FPMULT2  ADDA   #$80         ADD 128 BIAS BACK IN THAT WE LOST.
         STAA   FPACC1EX     SAVE THE NEW EXPONENT.
         JSR    UMULT        GO MULTIPLY THE "INTEGER" MANTISSAS.
FPMULT3  TST    FPACC1EX     WAS THERE AN OVERFLOW ERROR FROM ROUNDING?
         BEQ    FPMULT5      YES. RETURN ERROR.
         CLC                 SHOW NO ERRORS.
FPMULT6  JSR    PULFPAC2     RESTORE FPACC2.
         RTS
*
*
UMULT    EQU    *
         LDX    #0
         PSHX                CREATE PARTIAL PRODUCT REGISTER AND COUNTER.
         PSHX
         TSX                 POINT TO THE VARIABLES.
         LDAA   #24          SET COUNT TO THE NUMBER OF BITS.
         STAA   0,X
UMULT1   LDAA   FPACC2MN+2   GET THE L.S. BYTE OF THE MULTIPLIER.
         LSRA                PUT L.S. BIT IN CARRY.
         BCC    UMULT2       IF CARRY CLEAR, DON'T ADD MULTIPLICAND TO P.P.
         LDD    FPACC1MN+1   GET MULTIPLICAND L.S. 16 BITS.
         ADDD   2,X          ADD TO PARTIAL PRODUCT.
         STD    2,X          SAVE IN P.P.
         LDAA   FPACC1MN     GET UPPER 8 BITS OF MULTIPLICAND.
         ADCA   1,X          ADD IT W/ CARRY TO P.P.
         STAA   1,X          SAVE TO PARTIAL PRODUCT.
UMULT2   ROR    1,X          ROTATE PARTIAL PRODUCT TO THE RIGHT.
         ROR    2,X
         ROR    3,X
         ROR    FPACC2MN     SHIFT THE MULTIPLIER TO THE RIGHT 1 BIT.
         ROR    FPACC2MN+1
         ROR    FPACC2MN+2
         DEC    0,X          DONE YET?
         BNE    UMULT1       NO. KEEP GOING.
         TST    1,X          DOES PARTIAL PRODUCT NEED TO BE NORMALIZED?
         BMI    UMULT3       NO. GET ANSWER & RETURN.
         LSL    FPACC2MN     GET BIT THAT WAS SHIFTED OUT OF P.P REGISTER.
         ROL    3,X          PUT IT BACK INTO THE PARTIAL PRODUCT.
         ROL    2,X
         ROL    1,X
         DEC    FPACC1EX     FIX EXPONENT.
UMULT3   TST    FPACC2MN     DO WE NEED TO ROUND THE PARTIAL PRODUCT?
         BPL    UMULT4       NO. JUST RETURN.
         LDD    2,X          YES. GET THE LEAST SIGNIFIGANT 16 BITS.
         ADDD   #1           ADD 1.
         STD    2,X          SAVE RESULT.
         LDAA   1,X          PROPIGATE THROUGH.
         ADCA   #0
```

```
        STAA   1,X
        BCC    UMULT4       IF CARRY CLEAR ALL IS OK.
        ROR    1,X          IF NOT OVERFLOW. ROTATE CARRY INTO P.P.
        ROR    2,X
        ROR    3,X
        INC    FPACC1EX     UP THE EXPONENT.
UMULT4  INS                 TAKE COUNTER OFF STACK.
        PULX                GET M.S. 16 BITS OF PARTIAL PRODUCT.
        STX    FPACC1MN     PUT IT IN FPACC1.
        PULA                GET L.S. 8 BITS OF PARTIAL PRODUCT.
        STAA   FPACC1MN+2   PUT IT IN FPACC1.
        RTS                 RETURN.
*
*
*
*****************************************************************************
*                                                                           *
*                   FLOATING POINT ADDITION                                 *
*                                                                           *
*     This subroutine performs floating point addition of the two numbers  *
*     in FPACC1 and FPACC2.  The result of the addition is placed in        *
*     FPACC1 while FPACC2 remains unchanged.  This subroutine performs      *
*     full signed addition so either number may be of the same or opposite *
*     sign.                                                                 *
*                       WORSE CASE = 1030 CYCLES = 515 uS @ 2MHz            *
*                                                                           *
*****************************************************************************
*
*
FLTADD  EQU    *
        JSR    PSHFPAC2     SAVE FPACC2.
        LDX    #FPACC2EX    POINT TO FPACC2
        JSR    CHCK0        IS IT ZERO?
        BNE    FLTADD1      NO. GO CHECK FOR 0 IN FPACC1.
FLTADD6 CLC                 NO ERRORS.
FLTADD10 JSR   PULFPAC2     RESTORE FPACC2.
        RTS                 ANSWER IN FPACC1. RETURN.
FLTADD1 LDX    #FPACC1EX    POINT TO FPACC1.
        JSR    CHCK0        IS IT ZERO?
        BNE    FLTADD2      NO. GO ADD THE NUMBER.
FLTADD4 LDD    FPACC2EX     ANSWER IS IN FPACC2. MOVE IT INTO FPACC1.
        STD    FPACC1EX
        LDD    FPACC2MN+1   MOVE LOWER 16 BITS OF MANTISSA.
        STD    FPACC1MN+1
        LDAA   MANTSGN2     MOVE FPACC2 MANTISSA SIGN INTO FPACC1.
        STAA   MANTSGN1
        BRA    FLTADD6      RETURN.
FLTADD2 LDAA   FPACC1EX     GET FPACC1 EXPONENT.
        CMPA   FPACC2EX     ARE THE EXPONENTS THE SAME?
        BEQ    FLTADD7      YES. GO ADD THE MANTISSA'S.
        SUBA   FPACC2EX     NO. FPACC1EX-FPACC2EX. IS FPACC1 > FPACC2?
        BPL    FLTADD3      YES. GO CHECK RANGE.
        NEGA                NO. FPACC1 < FPACC2. MAKE DIFFERENCE POSITIVE.
        CMPA   #23          ARE THE NUMBERS WITHIN RANGE?
        BHI    FLTADD4      NO. FPACC2 IS LARGER. GO MOVE IT INTO FPACC1.
        TAB                 PUT DIFFERENCE IN B.
        ADDB   FPACC1EX     CORRECT FPACC1 EXPONENT.
        STAB   FPACC1EX     SAVE THE RESULT.
        LDX    #FPACC1MN    POINT TO FPACC1 MANTISSA.
        BRA    FLTADD5      GO DENORMALIZE FPACC1 FOR THE ADD.
FLTADD3 CMPA   #23          FPACC1 > FPACC2. ARE THE NUMBERS WITHIN RANGE?
        BHI    FLTADD6      NO. ANSWER ALREADY IN FPACC1. JUST RETURN.
        LDX    #FPACC2MN    POINT TO THE MANTISSA TO DENORMALIZE.
FLTADD5 LSR    0,X          SHIFT THE FIRST BYTE OF THE MANTISSA.
        ROR    1,X          THE SECOND.
        ROR    2,X          AND THE THIRD.
        DECA                DONE YET?
        BNE    FLTADD5      NO. KEEP SHIFTING.
FLTADD7 LDAA   MANTSGN1     GET FPACC1 MANTISSA SIGN.
        CMPA   MANTSGN2     ARE THE SIGNS THE SAME?
        BEQ    FLTADD11     YES. JUST GO ADD THE TWO MANTISSAS.
        TST    MANTSGN1     NO. IS FPACC1 THE NEGATIVE NUMBER?
        BPL    FLTADD8      NO. GO DO FPACC1-FPACC2.
```

```
        LDX    FPACC2MN     YES. EXCHANGE FPACC1 & FPACC2 BEFORE THE SUB.
        PSHX                SAVE IT.
        LDX    FPACC1MN     GET PART OF FPACC1.
        STX    FPACC2MN     PUT IT IN FPACC2.
        PULX                GET SAVED PORTION OF FPACC2
        STX    FPACC1MN     PUT IT IN FPACC1.
        LDX    FPACC2MN+2   GET LOWER 8 BITS & SIGN OF FPACC2.
        PSHX                SAVE IT.
        LDX    FPACC1MN+2   GET LOWER 8 BITS & SIGN OF FPACC1.
        STX    FPACC2MN+2   PUT IT IN FPACC2.
        PULX                GET SAVED PART OF FPACC2.
        STX    FPACC1MN+2   PUT IT IN FPACC1.
FLTADD8 LDD    FPACC1MN+1   GET LOWER 16 BITS OF FPACC1.
        SUBD   FPACC2MN+1   SUBTRACT LOWER 16 BITS OF FPACC2.
        STD    FPACC1MN+1   SAVE RESULT.
        LDAA   FPACC1MN     GET HIGH 8 BITS OF FPACC1 MANTISSA.
        SBCA   FPACC2MN     SUBTRACT HIGH 8 BITS OF FPACC2.
        STAA   FPACC1MN     SAVE THE RESULT. IS THE RESULT NEGATIVE?
        BCC    FLTADD9      NO. GO NORMALIZE THE RESULT.
        LDAA   FPACC1MN     YES. NEGATE THE MANTISSA.
        COMA
        PSHA                SAVE THE RESULT.
        LDD    FPACC1MN+1   GET LOWER 16 BITS.
        COMB                FORM THE ONE'S COMPLEMENT.
        COMA
        ADDD   #1           FORM THE TWO'S COMPLEMENT.
        STD    FPACC1MN+1   SAVE THE RESULT.
        PULA                GET UPPER 8 BITS BACK.
        ADCA   #0           ADD IN POSSIBLE CARRY.
        STAA   FPACC1MN     SAVE RESULT.
        LDAA   #$FF         SHOW THAT FPACC1 IS NEGATIVE.
        STAA   MANTSGN1
FLTADD9 JSR    FPNORM       GO NORMALIZE THE RESULT.
        BCC    FLTADD12     EVERYTHING'S OK SO RETURN.
        LDAA   #UNFERR      UNDERFLOW OCCURED DURING NORMALIZATION.
        SEC                 FLAG ERROR.
        JMP    FLTADD10     RETURN.
FLTADD12 JMP   FLTADD6      CAN'T BRANCH THAT FAR FROM HERE.
*
FLTADD11 LDD   FPACC1MN+1   GET LOWER 16 BITS OF FPACC1.
        ADDD   FPACC2MN+1   ADD IT TO THE LOWER 16 BITS OF FPACC2.
        STD    FPACC1MN+1   SAVE RESULT IN FPACC1.
        LDAA   FPACC1MN     GET UPPER 8 BITS OF FPACC1.
        ADCA   FPACC2MN     ADD IT (WITH CARRY) TO UPPER 8 BITS OF FPACC2.
        STAA   FPACC1MN     SAVE THE RESULT.
        BCC    FLTADD12     NO OVERFLOW SO JUST RETURN.
        ROR    FPACC1MN     PUT THE CARRY INTO THE MANTISSA.
        ROR    FPACC1MN+1   PROPIGATE THROUGH MANTISSA.
        ROR    FPACC1MN+2
        INC    FPACC1EX     UP THE MANTISSA BY 1.
        BNE    FLTADD12     EVERYTHING'S OK JUST RETURN.
        LDAA   #OVFERR      RESULT WAS TOO LARGE. OVERFLOW.
        SEC                 FLAG ERROR.
        JMP    FLTADD10     RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                      FLOATING POINT SUBTRACT SUBROUTINE                    *
*                                                                            *
*       This subroutine performs floating point subtraction ( FPACC1-FPACC2) *
*       by inverting the sign of FPACC2 and then calling FLTADD since        *
*       FLTADD performs complete signed addition.  Upon returning from       *
*       FLTADD the sign of FPACC2 is again inverted to leave it unchanged    *
*       from its original value.                                             *
*                                                                            *
*                            WORSE CASE = 1062 CYCLES = 531 uS @ 2MHz        *
*                                                                            *
******************************************************************************
*
*
FLTSUB  EQU    *
```

```
        BSR    FLTSUB1      INVERT SIGN.
        JSR    FLTADD       GO DO FLOATING POINT ADD.
FLTSUB1 LDAA   MANTSGN2     GET FPACC2 MANTISSA SIGN.
        EORA   #$FF         INVERT THE SIGN.
        STAA   MANTSGN2     PUT BACK.
        RTS                 RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                         FLOATING POINT DIVIDE                              *
*                                                                            *
*       This subroutine performs signed floating point divide. The          *
*       operation performed is FPACC1/FPACC2.  The divisor (FPACC2) is left *
*       unaltered and the answer is placed in FPACC1.  There are several    *
*       error conditions that can be returned by this routine.  They are:   *
*       a) division by zero.  b) overflow.  c) underflow.  As with all      *
*       other routines, an error is indicated by the carry being set and    *
*       the error code being in the A-reg.                                  *
*                                                                            *
*                         WORSE CASE = 2911 CYCLES = 1455 uS @ 2MHz          *
*                                                                            *
******************************************************************************
*
*
FLTDIV  EQU    *
        LDX    #FPACC2EX    POINT TO FPACC2.
        JSR    CHCK0        IS THE DIVISOR 0?
        BNE    FLTDIV1      NO. GO SEE IF THE DIVIDEND IS ZERO.
        LDAA   #DIV0ERR     YES. RETURN A DIVIDE BY ZERO ERROR.
        SEC                 FLAG ERROR.
        RTS                 RETURN.
FLTDIV1 LDX    #FPACC1EX    POINT TO FPACC1.
        JSR    CHCK0        IS THE DIVIDEND 0?
        BNE    FLTDIV2      NO. GO PERFORM THE DIVIDE.
        CLC                 YES. ANSWER IS ZERO. NO ERRORS.
        RTS                 RETURN.
FLTDIV2 JSR    PSHFPAC2     SAVE FPACC2.
        LDAA   MANTSGN2     GET FPACC2 MANTISSA SIGN.
        EORA   MANTSGN1     SET THE SIGN OF THE RESULT.
        STAA   MANTSGN1     SAVE THE RESULT.
        LDX    #0           SET UP WORK SPACE ON THE STACK.
        PSHX
        PSHX
        PSHX
        LDAA   #24          PUT LOOP COUNT ON STACK.
        PSHA
        TSX                 SET UP POINTER TO WORK SPACE.
        LDD    FPACC1MN     COMPARE FPACC1 & FPACC2 MANTISSAS.
        CPD    FPACC2MN     ARE THE UPPER 16 BITS THE SAME?
        BNE    FLTDIV3      NO.
        LDAA   FPACC1MN+2   YES. COMPARE THE LOWER 8 BITS.
        CMPA   FPACC2MN+2
FLTDIV3 BHS    FLTDIV4      IS FPACC2 MANTISSA > FPACC1 MANTISSA? NO.
        INC    FPACC2EX     ADD 1 TO THE EXPONENT TO KEEP NUMBER THE SAME.
*                           DID OVERFLOW OCCUR?
        BNE    FLTDIV14     NO. GO SHIFT THE MANTISSA RIGHT 1 BIT.
FLTDIV8 LDAA   #OVFERR      YES. GET ERROR CODE.
        SEC                 FLAG ERROR.
FLTDIV6 PULX                REMOVE WORKSPACE FROM STACK.
        PULX
        PULX
        INS
        JSR    PULFPAC2     RESTORE FPACC2.
        RTS                 RETURN.
FLTDIV4 LDD    FPACC1MN+1   DO AN INITIAL SUBTRACT IF DIVIDEND MANTISSA IS
        SUBD   FPACC2MN+1   GREATER THAN DIVISOR MANTISSA.
        STD    FPACC1MN+1
        LDAA   FPACC1MN
        SBCA   FPACC2MN
        STAA   FPACC1MN
        DEC    0,X           SUBTRACT 1 FROM THE LOOP COUNT.
```

```
FLTDIV14 LSR    FPACC2MN      SHIFT THE DIVISOR TO THE RIGHT 1 BIT.
         ROR    FPACC2MN+1
         ROR    FPACC2MN+2
         LDAA   FPACC1EX     GET FPACC1 EXPONENT.
         LDAB   FPACC2EX     GET FPACC2 EXPONENT.
         NEGB                ADD THE TWO'S COMPLEMENT TO SET FLAGS PROPERLY.
         ABA
         BMI    FLTDIV5      IF RESULT MINUS CHECK CARRY FOR POSS. OVERFLOW.
         BCS    FLTDIV7      IF PLUS & CARRY SET ALL IS OK.
         LDAA   #UNFERR      IF NOT, UNDERFLOW ERROR.
         BRA    FLTDIV6      RETURN WITH ERROR.
FLTDIV5  BCS    FLTDIV8      IF MINUS & CARRY SET OVERFLOW ERROR.
FLTDIV7  ADDA   #$81         ADD BACK BIAS+1 (THE '1' COMPENSATES FOR ALGOR.)
         STAA   FPACC1EX     SAVE RESULT.
FLTDIV9  LDD    FPACC1MN     SAVE DIVIDEND IN CASE SUBTRACTION DOESN'T GO.
         STD    4,X
         LDAA   FPACC1MN+2
         STAA   6,X
         LDD    FPACC1MN+1   GET LOWER 16 BITS FOR SUBTRACTION.
         SUBD   FPACC2MN+1
         STD    FPACC1MN+1   SAVE RESULT.
         LDAA   FPACC1MN     GET HIGH 8 BITS.
         SBCA   FPACC2MN
         STAA   FPACC1MN
         BPL    FLTDIV10     SUBTRACTION WENT OK. GO DO SHIFTS.
         LDD    4,X          RESTORE OLD DIVIDEND.
         STD    FPACC1MN
         LDAA   6,X
         STAA   FPACC1MN+2
FLTDIV10 ROL    3,X          ROTATE CARRY INTO QUOTIENT.
         ROL    2,X
         ROL    1,X
         LSL    FPACC1MN+2   SHIFT DIVIDEND TO LEFT FOR NEXT SUBTRACT.
         ROL    FPACC1MN+1
         ROL    FPACC1MN
         DEC    0,X          DONE YET?
         BNE    FLTDIV9      NO. KEEP GOING.
         COM    1,X          RESULT MUST BE COMPLEMENTED.
         COM    2,X
         COM    3,X
         LDD    FPACC1MN+1   DO 1 MORE SUBTRACT FOR ROUNDING.
         SUBD   FPACC2MN+1   ( DON'T NEED TO SAVE THE RESULT. )
         LDAA   FPACC1MN
         SBCA   FPACC2MN     ( NO NEED TO SAVE THE RESULT. )
         LDD    2,X          GET LOW 16 BITS.
         BCC    FLTDIV11     IF IT DIDNT GO RESULT OK AS IS.
         CLC                 CLEAR THE CARRY.
         BRA    FLTDIV13     GO SAVE THE NUMBER.
FLTDIV11 ADDD   #1           ROUND UP BY 1.
FLTDIV13 STD    FPACC1MN+1   PUT IT IN FPACC1.
         LDAA   1,X          GET HIGH 8 BITS.
         ADCA   #0
         STAA   FPACC1MN     SAVE RESULT.
         BCC    FLTDIV12     IF CARRY CLEAR ANSWER OK.
         ROR    FPACC1MN     IF NOT OVERFLOW. ROTATE CARRY IN.
         ROR    FPACC1MN+1
         ROR    FPACC1MN+2
FLTDIV12 CLC                 NO ERRORS.
         JMP    FLTDIV6      RETURN.
*
*
*
***************************************************************************
*                                                                         *
*               FLOATING POINT TO ASCII CONVERSION SUBROUTINE             *
*                                                                         *
*        This subroutine performs floating point to ASCII conversion of   *
*        the number in FPACC1.  The ascii string is placed in a buffer    *
*        pointed to by the X index register.  The buffer must be at least *
*        14 bytes long to contain the ASCII conversion.  The resulting    *
*        ASCII string is terminated by a zero (0) byte.  Upon exit the    *
*        X Index register will be pointing to the first character of the  *
*        string.  FPACC1 and FPACC2 will remain unchanged.                *
```

```
*                                                                            *
******************************************************************************
*
*
FLTASC   EQU    *
         PSHX                  SAVE THE POINTER TO THE STRING BUFFER.
         LDX    #FPACC1EX      POINT TO FPACC1.
         JSR    CHCK0          IS FPACC1 0?
         BNE    FLTASC1        NO. GO CONVERT THE NUMBER.
         PULX                  RESTORE POINTER.
         LDD    #$3000         GET ASCII CHARACTER + TERMINATING BYTE.
         STD    0,X            PUT IT IN THE BUFFER.
         RTS                   RETURN.
FLTASC1  LDX    FPACC1EX       SAVE FPACC1.
         PSHX
         LDX    FPACC1MN+1
         PSHX
         LDAA   MANTSGN1
         PSHA
         JSR    PSHFPAC2       SAVE FPACC2.
         LDX    #0
         PSHX                  ALLOCATE LOCALS.
         PSHX
         PSHX                  SAVE SPACE FOR STRING BUFFER POINTER.
         TSY                   POINT TO LOCALS.
         LDX    15,Y           GET POINTER FROM STACK.
         LDAA   #$20           PUT A SPACE IN THE BUFFER IF NUMBER NOT NEGATIVE.
         TST    MANTSGN1       IS IT NEGATIVE?
         BEQ    FLTASC2        NO. GO PUT SPACE.
         CLR    MANTSGN1       MAKE NUMBER POSITIVE FOR REST OF CONVERSION.
         LDAA   #'-            YES. PUT MINUS SIGN IN BUFFER.
FLTASC2  STAA   0,X
         INX                   POINT TO NEXT LOCATION.
         STX    0,Y            SAVE POINTER.
FLTASC5  LDX    #N9999999      POINT TO CONSTANT 9999999.
         JSR    GETFPAC2       GET INTO FPACC2.
         JSR    FLTCMP         COMPARE THE NUMBERS. IS FPACC1 > 9999999?
         BHI    FLTASC3        YES. GO DIVIDE FPACC1 BY 10.
         LDX    #P9999999      POINT TO CONSTANT 999999.9
         JSR    GETFPAC2       MOVE IT INTO FPACC2.
         JSR    FLTCMP         COMPARE NUMBERS. IS FPACC1 > 999999.9?
         BHI    FLTASC4        YES. GO CONTINUE THE CONVERSION.
         DEC    2,Y            DECREMENT THE MULT./DIV. COUNT.
         LDX    #CONST10       NO. MULTIPLY BY 10. POINT TO CONSTANT.
FLTASC6  JSR    GETFPAC2       MOVE IT INTO FPACC2.
         JSR    FLTMUL
         BRA    FLTASC5        GO DO COMPARE AGAIN.
FLTASC3  INC    2,Y            INCREMENT THE MULT./DIV. COUNT.
         LDX    #CONSTP1       POINT TO CONSTANT ".1".
         BRA    FLTASC6        GO DIVIDE FPACC1 BY 10.
FLTASC4  LDX    #CONSTP5       POINT TO CONSTANT OF ".5".
         JSR    GETFPAC2       MOVE IT INTO FPACC2.
         JSR    FLTADD         ADD .5 TO NUMBER IN FPACC1 TO ROUND IT.
         LDAB   FPACC1EX       GET FPACC1 EXPONENT.
         SUBB   #$81           TAKE OUT BIAS +1.
         NEGB                  MAKE IT NEGATIVE.
         ADDB   #23            ADD IN THE NUMBER OF MANTISSA BITS -1.
         BRA    FLTASC17       GO CHECK TO SEE IF WE NEED TO SHIFT AT ALL.
FLTASC7  LSR    FPACC1MN       SHIFT MANTISSA TO THE RIGHT BY THE RESULT (MAKE
         ROR    FPACC1MN+1     THE NUMBER AN INTEGER).
         ROR    FPACC1MN+2
         DECB                  DONE SHIFTING?
FLTASC17 BNE    FLTASC7        NO. KEEP GOING.
         LDAA   #1             GET INITIAL VALUE OF "DIGITS AFTER D.P." COUNT.
         STAA   3,Y            INITIALIZE IT.
         LDAA   2,Y            GET DECIMAL EXPONENT.
         ADDA   #8             ADD THE NUMBER OF DECIMAL +1 TO THE EXPONENT.
*                              WAS THE ORIGINAL NUMBER > 9999999?
         BMI    FLTASC8        YES. MUST BE REPRESENTED IN SCIENTIFIC NOTATION.
         CMPA   #8             WAS THE ORIGINAL NUMBER < 1?
         BHS    FLTASC8        YES. MUST BE REPRESENTED IN SCIENTIFIC NOTATION.
         DECA                  NO. NUMBER CAN BE REPRESENTED IN 7 DIGITS.
         STAA   3,Y            MAKE THE DECIMAL EXPONENT THE DIGIT COUNT BEFORE
```

```
*                               THE DECIMAL POINT.
         LDAA   #2              SETUP TO ZERO THE DECIMAL EXPONENT.
FLTASC8  SUBA   #2              SUBTRACT 2 FROM THE DECIMAL EXPONENT.
         STAA   2,Y             SAVE THE DECIMAL EXPONENT.
         TST    3,Y             DOES THE NUMBER HAVE AN INTEGER PART? (EXP. >0)
         BGT    FLTASC9         YES. GO PUT IT OUT.9
         LDAA   #'.             NO. GET DECIMAL POINT.
         LDX    0,Y             GET POINTER TO BUFFER.
         STAA   0,X             PUT THE DECIMAL POINT IN THE BUFFER.
         INX                    POINT TO NEXT BUFFER LOCATION.
         TST    3,Y             IS THE DIGIT COUNT TILL EXPONENT =0?
         BEQ    FLTASC18        NO. NUMBER IS <.1
         LDAA   #'0             YES. FORMAT NUMBER AS .0XXXXXXX
         STAA   0,X             PUT THE 0 IN THE BUFFER.
         INX                    POINT TO THE NEXT LOCATION.
FLTASC18 STX    0,Y             SAVE NEW POINTER VALUE.
FLTASC9  LDX    #DECDIG         POINT TO THE TABLE OF DECIMAL DIGITS.
         LDAA   #7              INITIALIZE THE THE NUMBER OF DIGITS COUNT.
         STAA   5,Y
FLTASC10 CLR    4,Y             CLEAR THE DECIMAL DIGIT ACCUMULATOR.
FLTASC11 LDD    FPACC1MN+1      GET LOWER 16 BITS OF MANTISSA.
         SUBD   1,X             SUBTRACT LOWER 16 BITS OF CONSTANT.
         STD    FPACC1MN+1      SAVE RESULT.
         LDAA   FPACC1MN        GET UPPER 8 BITS.
         SBCA   0,X             SUBTRACT UPPER 8 BITS.
         STAA   FPACC1MN        SAVE RESULT. UNDERFLOW?
         BCS    FLTASC12        YES. GO ADD DECIMAL NUMBER BACK IN.
         INC    4,Y             ADD 1 TO DECIMAL NUMBER.
         BRA    FLTASC11        TRY ANOTHER SUBTRACTION.
FLTASC12 LDD    FPACC1MN+1      GET FPACC1 MANTISSA LOW 16 BITS.
         ADDD   1,X             ADD LOW 16 BITS BACK IN.
         STD    FPACC1MN+1      SAVE THE RESULT.
         LDAA   FPACC1MN        GET HIGH 8 BITS.
         ADCA   0,X             ADD IN HIGH 8 BITS OF CONSTANT.
         STAA   FPACC1MN        SAVE RESULT.
         LDAA   4,Y             GET DIGIT.
         ADDA   #$30            MAKE IT ASCII.
         PSHX                   SAVE POINTER TO CONSTANTS.
         LDX    0,Y             GET POINTER TO BUFFER.
         STAA   0,X             PUT DIGIT IN BUFFER.
         INX                    POINT TO NEXT BUFFER LOCATION.
         DEC    3,Y             SHOULD WE PUT A DECIMAL POINT IN THE BUFFER YET?
         BNE    FLTASC16        NO. CONTINUE THE CONVERSION.
         LDAA   #'.             YES. GET DECIMAL POINT.
         STAA   0,X             PUT IT IN THE BUFFER.
         INX                    POINT TO THE NEXT BUFFER LOCATION.
FLTASC16 STX    0,Y             SAVE UPDATED POINTER.
         PULX                   RESTORE POINTER TO CONSTANTS.
         INX                    POINT TO NEXT CONSTANT.
         INX
         INX
         DEC    5,Y             DONE YET?
         BNE    FLTASC10        NO. CONTINUE CONVERSION OF "MANTISSA".
         LDX    0,Y             YES. POINT TO BUFFER STRING BUFFER.
FLTASC13 DEX                    POINT TO LAST CHARACTER PUT IN THE BUFFER.
         LDAA   0,X             GET IT.
         CMPA   #$30            WAS IT AN ASCII 0?
         BEQ    FLTASC13        YES. REMOVE TRAILING ZEROS.
         INX                    POINT TO NEXT AVAILABLE LOCATION IN BUFFER.
         LDAB   2,Y             DO WE NEED TO PUT OUT AN EXPONENT?
         BEQ    FLTASC15        NO. WE'RE DONE.
         LDAA   #'E             YES. PUT AN 'E' IN THE BUFFER.
         STAA   0,X
         INX                    POINT TO NEXT BUFFER LOCATION.
         LDAA   #'+             ASSUME EXPONENT IS POSITIVE.
         STAA   0,X             PUT PLUS SIGN IN THE BUFFER.
         TSTB                   IS IT REALLY MINUS?
         BPL    FLTASC14        NO. IS'S OK AS IS.
         NEGB                   YES. MAKE IT POSITIVE.
         LDAA   #'-             PUT THE MINUS SIGN IN THE BUFFER.
         STAA   0,X
FLTASC14 INX                    POINT TO NEXT BUFFER LOCATION.
         STX    0,Y             SAVE POINTER TO STRING BUFFER.
```

```
         CLRA                    SET UP FOR DIVIDE.
         LDX    #10              DIVIDE DECIMAL EXPONENT BY 10.
         IDIV
         PSHB                    SAVE REMAINDER.
         XGDX                    PUT QUOTIENT IN D.
         ADDB   #$30             MAKE IT ASCII.
         LDX    0,Y              GET POINTER.
         STAB   0,X              PUT NUMBER IN BUFFER.
         INX                     POINT TO NEXT LOCATION.
         PULB                    GET SECOND DIGIT.
         ADDB   #$30             MAKE IT ASCII.
         STAB   0,X              PUT IT IN THE BUFFER.
         INX                     POINT TO NEXT LOCATION.
FLTASC15 CLR    0,X              TERMINATE STRING WITH A ZERO BYTE.
         PULX                    CLEAR LOCALS FROM STACK.
         PULX
         PULX
         JSR    PULFPAC2         RESTORE FPACC2.
         PULA
         STAA   MANTSGN1
         PULX                    RESTORE FPACC1.
         STX    FPACC1MN+1
         PULX
         STX    FPACC1EX
         PULX                    POINT TO THE START OF THE ASCII STRING.
         RTS                     RETURN.
*
*
DECDIG   EQU    *
         FCB    $0F,$42,$40  DECIMAL 1,000,000
         FCB    $01,$86,$A0  DECIMAL   100,000
         FCB    $00,$27,$10  DECIMAL    10,000
         FCB    $00,$03,$E8  DECIMAL     1,000
         FCB    $00,$00,$64  DECIMAL       100
         FCB    $00,$00,$0A  DECIMAL        10
         FCB    $00,$00,$01  DECIMAL         1
*
*
P9999999 EQU    *            CONSTANT 999999.9
         FCB    $94,$74,$23,$FE
*
N9999999 EQU    *            CONSTANT 9999999.
         FCB    $98,$18,$96,$7F
*
CONSTP5 EQU    *            CONSTANT .5
         FCB    $80,$00,$00,$00
*
*
FLTCMP   EQU    *
         TST    MANTSGN1     IS FPACC1 NEGATIVE?
         BPL    FLTCMP2      NO. CONTINUE WITH COMPARE.
         TST    MANTSGN2     IS FPACC2 NEGATIVE?
         BPL    FLTCMP2      NO. CONTINUE WITH COMPARE.
         LDD    FPACC2EX     YES. BOTH ARE NEGATIVE SO COMPARE MUST BE DONE
         CPD    FPACC1EX     BACKWARDS. ARE THEY EQUAL SO FAR?
         BNE    FLTCMP1      NO. RETURN WITH CONDITION CODES SET.
         LDD    FPACC2MN+1   YES. COMPARE LOWER 16 BITS OF MANTISSAS.
         CPD    FPACC1MN+1
FLTCMP1  RTS                 RETURN WITH CONDITION CODES SET.
FLTCMP2  LDAA   MANTSGN1     GET FPACC1 MANTISSA SIGN.
         CMPA   MANTSGN2     BOTH POSITIVE?
         BNE    FLTCMP1      NO. RETURN WITH CONDITION CODES SET.
         LDD    FPACC1EX     GET FPACC1 EXPONENT & UPPER 8 BITS OF MANTISSA.
         CPD    FPACC2EX     SAME AS FPACC2?
         BNE    FLTCMP1      NO. RETURN WITH CONDITION CODES SET.
         LDD    FPACC1MN+1   GET FPACC1 LOWER 16 BITS OF MANTISSA.
         CPD    FPACC2MN+1   COMPARE WITH FPACC2 LOWER 16 BITS OF MANTISSA.
         RTS                 RETURN WITH CONDITION CODES SET.
*
*
*
*******************************************************************************
*                                                                             *
```

```
*                     UNSIGNED INTEGER TO FLOATING POINT                     *
*                                                                            *
*        This subroutine performs "unsigned" integer to floating point       *
*        conversion of a 16 bit word.  The 16 bit integer must be in the     *
*        lower 16 bits of FPACC1 mantissa.  The resulting floating point     *
*        number is returned in FPACC1.                                       *
*                                                                            *
******************************************************************************
*
*
UINT2FLT EQU    *
         LDX    #FPACC1EX    POINT TO FPACC1.
         JSR    CHCK0        IS IT ALREADY 0?
         BNE    UINTFLT1     NO. GO CONVERT.
         RTS                 YES. JUST RETURN.
UINTFLT1 LDAA   #$98         GET BIAS + NUMBER OF BITS IN MANTISSA.
         STAA   FPACC1EX     INITIALIZE THE EXPONENT.
         JSR    FPNORM       GO MAKE IT A NORMALIZED FLOATING POINT VALUE.
         CLC                 NO ERRORS.
         RTS                 RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                      SIGNED INTEGER TO FLOATING POINT                      *
*                                                                            *
*        This routine works just like the unsigned integer to floating       *
*        point routine except the the 16 bit integer in the FPACC1           *
*        mantissa is considered to be in two's complement format.  This      *
*        will return a floating point number in the range -32768 to +32767.  *
*                                                                            *
******************************************************************************
*
*
SINT2FLT EQU    *
         LDD    FPACC1MN+1   GET THE LOWER 16 BITS OF FPACC1 MANTISSA.
         PSHA                SAVE SIGN OF NUMBER.
         BPL    SINTFLT1     IF POSITIVE JUST GO CONVERT.
         COMA                MAKE POSITIVE.
         COMB
         ADDD   #1           TWO'S COMPLEMENT.
         STD    FPACC1MN+1   PUT IT BACK IN FPACC1 MANTISSA.
SINTFLT1 BSR    UINT2FLT     GO CONVERT.
         PULA                GET SIGN OF ORIGINAL INTEGER.
         LDAB   #$FF         GET "MINUS SIGN".
         TSTA                WAS THE NUMBER NEGATIVE?
         BPL    SINTFLT2     NO. RETURN.
         STAB   MANTSGN1     YES. SET FPACC1 SIGN BYTE.
SINTFLT2 CLC                 NO ERRORS.
         RTS                 RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                   FLOATING POINT TO INTEGER CONVERSION                     *
*                                                                            *
*        This subroutine will perform "unsigned" floating point to integer   *
*        conversion.  The floating point number if positive, will be         *
*        converted to an unsigned 16 bit integer ( 0 <= X <= 65535 ).  If    *
*        the number is negative it will be converted to a twos complement    *
*        16 bit integer.  This type of conversion will allow 16 bit          *
*        addresses to be represented as positive numbers when in floating    *
*        point format.  Any fractional number part is disguarded             *
*                                                                            *
******************************************************************************
*
*
FLT2INT  EQU    *
         LDX    #FPACC1EX    POINT TO FPACC1.
         JSR    CHCK0        IS IT 0?
         BEQ    FLT2INT3     YES. JUST RETURN.
```

```
         LDAB   FPACC1EX     GET FPACC1 EXPONENT.
         CMPB   #$81         IS THERE AN INTEGER PART?
         BLO    FLT2INT2     NO. GO PUT A 0 IN FPACC1.
         TST    MANTSGN1     IS THE NUMBER NEGATIVE?
         BMI    FLT2INT1     YES. GO CONVERT NEGATIVE NUMBER.
         CMPB   #$90         IS THE NUMBER TOO LARGE TO BE MADE AN INTEGER?
         BHI    FLT2INT4     YES. RETURN WITH AN ERROR.
         SUBB   #$98         SUBTRACT THE BIAS PLUS THE NUMBER OF BITS.
FLT2INT5 LSR    FPACC1MN     MAKE THE NUMBER AN INTEGER.
         ROR    FPACC1MN+1
         ROR    FPACC1MN+2
         INCB                DONE SHIFTING?
         BNE    FLT2INT5     NO. KEEP GOING.
         CLR    FPACC1EX     ZERO THE EXPONENT (ALSO CLEARS THE CARRY).
         RTS
FLT2INT1 CMPB   #$8F         IS THE NUMBER TOO SMALL TO BE MADE AN INTEGER?
         BHI    FLT2INT4     YES. RETURN ERROR.
         SUBB   #$98         SUBTRACT BIAS PLUS NUMBER OF BITS.
         BSR    FLT2INT5     GO DO SHIFT.
         LDD    FPACC1MN+1   GET RESULTING INTEGER.
         COMA                MAKE IT NEGATIVE.
         COMB
         ADDD   #1           TWO'S COMPLEMENT.
         STD    FPACC1MN+1   SAVE RESULT.
         CLR    MANTSGN1     CLEAR MANTISSA SIGN. (ALSO CLEARS THE CARRY)
         RTS                 RETURN.
FLT2INT4 LDAA   #TOLGSMER    NUMBER TOO LARGE OR TOO SMALL TO CONVERT TO INT.
         SEC                 FLAG ERROR.
         RTS                 RETURN.
FLT2INT2 LDD    #0
         STD    FPACC1EX     ZERO FPACC1.
         STD    FPACC1MN+1   (ALSO CLEARS THE CARRY)
FLT2INT3 RTS                 RETURN.
*
*
*
***********************************************************************
*                                                                     *
*                      SQUARE ROOT SUBROUTINE                         *
*                                                                     *
*        This routine is used to calculate the square root of the floating   *
*        point number in FPACC1.  If the number in FPACC1 is negative an     *
*        error is returned.                                           *
*                                                                     *
*                       WORSE CASE = 16354 CYCLES = 8177 uS @ 2MHz    *
*                                                                     *
***********************************************************************
*
*
FLTSQR   EQU    *
         LDX    #FPACC1EX    POINT TO FPACC1.
         JSR    CHCK0        IS IT ZERO?
         BNE    FLTSQR1      NO. CHECK FOR NEGATIVE.
         RTS                 YES. RETURN.
FLTSQR1  TST    MANTSGN1     IS THE NUMBER NEGATIVE?
         BPL    FLTSQR2      NO. GO TAKE ITS SQUARE ROOT.
         LDAA   #NSQRTERR    YES. ERROR.
         SEC                 FLAG ERROR.
         RTS                 RETURN.
FLTSQR2  JSR    PSHFPAC2     SAVE FPACC2.
         LDAA   #4           GET ITERATION LOOP COUNT.
         PSHA                SAVE IT ON THE STACK.
         LDX    FPACC1MN+1   SAVE INITIAL NUMBER.
         PSHX
         LDX    FPACC1EX
         PSHX
         TSY                 POINT TO IT.
         BSR    TFR1TO2      TRANSFER FPACC1 TO FPACC2.
         LDAA   FPACC2EX     GET FPACC1 EXPONENT.
         SUBA   #$80         REMOVE BIAS FROM EXPONENT.
         INCA                COMPENSATE FOR ODD EXPONENTS (GIVES CLOSER GUESS)
         BPL    FLTSQR3      IF NUMBER >1 DIVIDE EXPONENT BY 2 & ADD BIAS.
         LSRA                IF <1 JUST DIVIDE IT BY 2.
```

```
        BRA    FLTSQR4      GO CALCULATE THE SQUARE ROOT.
FLTSQR3 LSRA                DIVIDE EXPONENT BY 2.
        ADDA   #$80         ADD BIAS BACK IN.
FLTSQR4 STAA   FPACC2EX     SAVE EXPONENT/2.
FLTSQR5 JSR    FLTDIV       DIVIDE THE ORIGINAL NUMBER BY THE GUESS.
        JSR    FLTADD       ADD THE "GUESS" TO THE QUOTIENT.
        DEC    FPACC1EX     DIVIDE THE RESULT BY 2 TO PRODUCE A NEW GUESS.
        BSR    TFR1TO2      PUT THE NEW GUESS INTO FPACC2.
        LDD    0,Y          GET THE ORIGINAL NUMBER.
        STD    FPACC1EX     PUT IT BACK IN FPACC1.
        LDD    2,Y          GET MANTISSA LOWER 16 BITS.
        STD    FPACC1MN+1
        DEC    4,Y          BEEN THROUGH THE LOOP 4 TIMES?
        BNE    FLTSQR5      NO. KEEP GOING.
        LDD    FPACC2EX     THE FINAL GUESS IS THE ANSWER.
        STD    FPACC1EX     PUT IT IN FPACC1.
        LDD    FPACC2MN+1
        STD    FPACC1MN+1
        PULX                GET RID OF ORIGINAL NUMBER.
        PULX
        INS                 GET RID OF LOOP COUNT VARIABLE.
        JSR    PULFPAC2     RESTORE FPACC2.
        CLC                 NO ERRORS.
        RTS
*
*
TFR1TO2 EQU    *
        LDD    FPACC1EX     GET FPACC1 EXPONENT & HIGH 8 BIT OF MANTISSA.
        STD    FPACC2EX     PUT IT IN FPACC2.
        LDD    FPACC1MN+1   GET FPACC1 LOW 16 BITS OF MANTISSA.
        STD    FPACC2MN+1   PUT IT IN FPACC2.
        LDAA   MANTSGN1     TRANSFER THE SIGN.
        STAA   MANTSGN2
        RTS                 RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                        FLOATING POINT SINE                                 *
*                                                                            *
******************************************************************************
*
*
FLTSIN  EQU    *
        JSR    PSHFPAC2     SAVE FPACC2 ON THE STACK.
        JSR    ANGRED       GO REDUCE THE ANGLE TO BETWEEN +/-PI.
        PSHB                SAVE THE QUAD COUNT.
        PSHA                SAVE THE SINE/COSINE FLAG.
        JSR    DEG2RAD      CONVERT DEGREES TO RADIANS.
        PULA                RESTORE THE SINE/COSINE FLAG.
FLTSIN1 JSR    SINCOS       GO GET THE SINE OF THE ANGLE.
        PULA                RESTORE THE QUAD COUNT.
        CMPA   #2           WAS THE ANGLE IN QUADS 1 OR 2?
        BLS    FLTSIN2      YES. SIGN OF THE ANSWER IS OK.
        COM    MANTSGN1     NO. SINE IN QUADS 3 & 4 IS NEGATIVE.
FLTSIN2 CLC                 SHOW NO ERRORS.
        JSR    PULFPAC2     RESTORE FPACC2
        RTS                 RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                        FLOATING POINT COSINE                               *
*                                                                            *
******************************************************************************
*
*
FLTCOS  EQU    *
        JSR    PSHFPAC2     SAVE FPACC2 ON THE STACK.
        JSR    ANGRED       GO REDUCE THE ANGLE TO BETWEEN +/-PI.
        PSHB                SAVE THE QUAD COUNT.
```

```
        PSHA                 SAVE THE SINE/COSINE FLAG.
        JSR    DEG2RAD       CONVERT TO RADIANS.
        PULA                 RESTORE THE SINE/COSINE FLAG.
        EORA   #$01          COMPLIMENT 90'S COPMLIMENT FLAG FOR COSINE.
        JSR    SINCOS        GO GET THE COSINE OF THE ANGLE.
        PULA                 RESTORE THE QUAD COUNT.
        CMPA   #1            WAS THE ORIGINAL ANGLE IN QUAD 1?
        BEQ    FLTCOS1       YES. SIGN IS OK.
        CMPA   #4            WAS IT IN QUAD 4?
        BEQ    FLTCOS1       YES. SIGN IS OK.
        COM    MANTSGN1      NO. COSINE IS NEGATIVE IN QUADS 2 & 3.
FLTCOS1 JMP    FLTSIN2       FLAG NO ERRORS, RESTORE FPACC2, & RETURN.
*
*
*
******************************************************************************
*                                                                            *
*                 FLOATING POINT SINE AND COSINE SUBROUTINE                  *
*                                                                            *
******************************************************************************
*
*
SINCOS  EQU    *
        PSHA                 SAVE SINE/COSINE FLAG ON STACK.
        LDX    FPACC1MN+1    SAVE THE VALUE OF THE ANGLE.
        PSHX
        LDX    FPACC1EX
        PSHX
        LDAA   MANTSGN1
        PSHA
        LDX    #SINFACT      POINT TO THE FACTORIAL TABLE.
        PSHX                 SAVE POINTER TO THE SINE FACTORIAL TABLE.
        PSHX                 JUST ALLOCATE ANOTHER LOCAL (VALUE NOT IMPORTANT)
        LDAA   #$4           GET INITIAL LOOP COUNT.
        PSHA                 SAVE AS LOCAL ON STACK
        TSY                  POINT TO LOCALS.
        JSR    TFR1TO2       TRANSFER FPACC1 TO FPACC2.
        JSR    FLTMUL        GET Xª2 IN FPACC1.
        TST    10,Y          ARE WE DOING THE SINE?
        BEQ    SINCOS7       YES. GO DO IT.
        LDX    #COSFACT      NO. GET POINTER TO COSINE FACTORIAL TABLE.
        STX    1,Y           SAVE IT.
        JSR    TFR1TO2       COPY Xª2 INTO FPACC2.
        BRA    SINCOS4       GENERATE EVEN POWERS OF "X" FOR COSINE.
SINCOS7 JSR    EXG1AND2      PUT Xª2 IN FPACC2 & X IN FPACC1.
SINCOS1 JSR    FLTMUL        CREATE Xª3,5,7,9 OR Xª2,4,6,8.
SINCOS4 LDX    FPACC1MN+1    SAVE EACH ONE ON THE STACK.
        PSHX
        LDX    FPACC1EX
        PSHX
        LDAA   MANTSGN1
        PSHA                 SAVE THE MANTISSA SIGN.
        DEC    0,Y           HAVE WE GENERATED ALL THE POWERS YET?
        BNE    SINCOS1       NO. GO DO SOME MORE.
        LDAA   #$4           SET UP LOOP COUNT.
        STAA   0,Y
        TSX                  POINT TO POWERS ON THE STACK.
SINCOS2 STX    3,Y           SAVE THE POINTER.
        LDX    1,Y           GET THE POINTER TO THE FACTORIAL CONSTANTS.
        JSR    GETFPAC2      PUT THE NUMBER IN FPACC2.
        INX                  POINT TO THE NEXT CONSTANT.
        INX
        INX
        INX
        STX    1,Y           SAVE THE POINTER.
        LDX    3,Y           GET POINTER TO POWERS.
        LDAA   0,X           GET NUMBER SIGN.
        STAA   MANTSGN1      PUT IN FPACC1 MANTISSA SIGN.
        LDD    1,X           GET LOWER 16-BITS OF THE MANTISSA.
        STD    FPACC1EX      PUT IN FPACC1 MANTISSA.
        LDD    3,X           GET HIGH 8 BITS OF THE MANTISSA & EXPONENT.
        STD    FPACC1MN+1    PUT IT IN FPACC1 EXPONENT & MANTISSA.
        JSR    FLTMUL        MULTIPLY THE TWO.
```

```
        LDX     3,Y           GET POINTER TO POWERS BACK.
        LDD     FPACC1MN+1    SAVE RESULT WHERE THE POWER OF X WAS.
        STD     3,X
        LDD     FPACC1EX
        STD     1,X
        LDAA    MANTSGN1      SAVE SIGN.
        STAA    0,X
        INX                   POINT TO THE NEXT POWER.
        INX
        INX
        INX
        INX
        DEC     0,Y           DONE?
        BNE     SINCOS2       NO. GO DO ANOTHER MULTIPLICATION.
        LDAA    #$3           GET LOOP COUNT.
        STAA    0,Y           SAVE IT.
SINCOS3 LDX     3,Y           POINT TO RESULTS ON THE STACK.
        DEX                   POINT TO PREVIOUS RESULT.
        DEX
        DEX
        DEX
        DEX
        STX     3,Y           SAVE THE NEW POINTER.
        LDAA    0,X           GET NUMBERS SIGN.
        STAA    MANTSGN2      PUT IT IN FPACC2.
        LDD     1,X           GET LOW 16 BITS OF THE MANTISSA.
        STD     FPACC2EX      PUT IN FPACC2.
        LDD     3,X           GET HIGH 8 BIT & EXPONENT.
        STD     FPACC2MN+1    PUT IN FPACC2.
        JSR     FLTADD        GO ADD THE TWO NUMBERS.
        DEC     0,Y           DONE?
        BNE     SINCOS3       NO. GO ADD THE NEXT TERM IN.
        TST     10,Y          ARE WE DOING THE SINE?
        BEQ     SINCOS5       YES. GO PUT THE ORIGINAL ANGLE INTO FPACC2.
        LDX     #ONE          NO. FOR COSINE PUT THE CONSTANT 1 INTO FPACC2.
        JSR     GETFPAC2
        BRA     SINCOS6       GO ADD IT TO THE SUM OF THE TERMS.
SINCOS5 LDAA    5,Y           GET THE VALUE OF THE ORIGINAL ANGLE.
        STAA    MANTSGN2      PUT IT IN FPACC2.
        LDD     6,Y
        STD     FPACC2EX
        LDD     8,Y
        STD     FPACC2MN+1
SINCOS6 JSR     FLTADD        GO ADD IT TO THE SUM OF THE TERMS.
        TSX                   NOW CLEAN UP THE STACK.
        XGDX                  PUT STACK IN D.
        ADDD    #31           CLEAR ALL THE TERMS & TEMPS OFF THE STACK.
        XGDX
        TXS                   UPDATE THE STACK POINTER.
        RTS                   RETURN.
*
*
ANGRED  EQU     *
        CLRA                  INITIALIZE THE 45'S COMPLIMENT FLAG.
        PSHA                  PUT IT ON THE STACK.
        INCA                  INITIALIZE THE QUAD COUNT TO 1.
        PSHA                  PUT IT ON THE STACK.
        TSY                   POINT TO IT.
        LDX     #THREE60      POINT TO THE CONSTANT 360.
        JSR     GETFPAC2      GET IT INTO FPACC.
        TST     MANTSGN1      IS THE INPUT ANGLE NEGATIVE:
        BPL     ANGRED1       NO. SKIP THE ADD.
        JSR     FLTADD        YES. MAKE THE ANGLE POSITIVE BY ADDING 360 DEG.
ANGRED1 DEC     FPACC2EX      MAKE THE CONSTANT IN FPACC2 90 DEGREES.
        DEC     FPACC2EX
ANGRED2 JSR     FLTCMP        IS THE ANGLE LESS THAN 90 DEGREES ALREADY?
        BLS     ANGRED3       YES. RETURN WITH QUAD COUNT.
        JSR     FLTSUB        NO. REDUCE ANGLE BY 90 DEGREES.
        INC     0,Y           INCREMENT THE QUAD COUNT.
        BRA     ANGRED2       GO SEE IF IT'S LESS THAN 90 NOW.
ANGRED3 LDAA    0,Y           GET THE QUAD COUNT.
        CMPA    #1            WAS THE ORIGINAL ANGLE IN QUAD 1?
        BEQ     ANGRED4       YES. COMPUTE TRIG FUNCTION AS IS.
```

```
         CMPA   #3             NO. WAS THE ORIGINAL ANGLE IN QUAD 3?
         BEQ    ANGRED4        YES. COMPUTE THE TRIG FUNCTION AS IF IN QUAD 1.
         LDAA   #$FF           NO. MUST COMPUTE THE TRIG FUNCTION OF THE 90'S
         STAA   MANTSGN1       COMPLIMENT ANGLE.
         JSR    FLTADD         ADD 90 DEGREES TO THE NEGATED ANGLE.
ANGRED4  DEC    FPACC2EX       MAKE THE ANGLE IN FPACC2 45 DEGREES.
         JSR    FLTCMP         IS THE ANGLE < 45 DEGREES?
         BLS    ANGRED5        YES. IT'S OK AS IT IS.
         INC    FPACC2EX       NO. MUST GET THE 90'S COMPLIMENT.
         LDAA   #$FF           MAKE FPACC1 NEGATIVE.
         STAA   MANTSGN1
         JSR    FLTADD         GET THE 90'S COMPLIMENT.
         INC    1,Y            SET THE FLAG.
ANGRED5  PULB                  GET THE QUAD COUNT.
         PULA                  GET THE COMPLIMENT FLAG.
         RTS                   RETURN WITH THE QUAD COUNT & COMPLIMENT FLAG.
*
*
EXG1AND2 EQU    *
         LDD    FPACC1EX
         LDX    FPACC2EX
         STD    FPACC2EX
         STX    FPACC1EX
         LDD    FPACC1MN+1
         LDX    FPACC2MN+1
         STD    FPACC2MN+1
         STX    FPACC1MN+1
         LDAA   MANTSGN1
         LDAB   MANTSGN2
         STAA   MANTSGN2
         STAB   MANTSGN1
         RTS                   RETURN.
*
*
SINFACT  EQU    *
         FCB    $6E,$38,$EF,$1D         +(1/9!)
         FCB    $74,$D0,$0D,$01         -(1/7!)
         FCB    $7A,$08,$88,$89         +(1/5!)
         FCB    $7E,$AA,$AA,$AB         -(1/3!)
*
*
COSFACT  EQU    *
         FCB    $71,$50,$0D,$01         +(1/8!)
         FCB    $77,$B6,$0B,$61         -(1/6!)
         FCB    $7C,$2A,$AA,$AB         +(1/4!)
         FCB    $80,$80,$00,$00         -(1/2!)
*
*
ONE      FCB    $81,$00,$00,$00         1.0
PI       FCB    $82,$49,$0F,$DB         3.1415927
THREE60  FCB    $89,$34,$00,$00         360.0
*
*
*
********************************************************************************
*                                                                              *
*                         FLOATING POINT TANGENT                               *
*                                                                              *
********************************************************************************
*
*
FLTTAN   EQU    *
         JSR    PSHFPAC2       SAVE FPACC2 ON THE STACK.
         JSR    TFR1TO2         PUT A COPY OF THE ANGLE IN FPACC2.
         JSR    FLTCOS          GET COSINE OF THE ANGLE.
         JSR    EXG1AND2        PUT RESULT IN FPACC2 & PUT ANGLE IN FPACC1.
         JSR    FLTSIN          GET SIN OF THE ANGLE.
         JSR    FLTDIV          GET TANGENT OF ANGLE BY DOING SIN/COS.
         BCC    FLTTAN1         IF CARRY CLEAR, ANSWER OK.
         LDX    #MAXNUM         TANGENT OF 90 WAS ATTEMPTED. PUT LARGEST
         JSR    GETFPAC1        NUMBER IN FPACC1.
         LDAA   #TAN90ERR       GET ERROR CODE IN A.
FLTTAN1  JSR    PULFPAC2        RESTORE FPACC2.
```

```
        RTS                  RETURN.
*
*
MAXNUM  EQU    *
        FCB    $FE,$7F,$FF,$FF         LARGEST POSITIVE NUMBER WE CAN HAVE.
*
*
*
******************************************************************************
*                                                                            *
*                             TRIG UTILITIES                                 *
*                                                                            *
*       The routines "DEG2RAD" and "RAD2DEG" are used to convert angles     *
*       from degrees-to-radians and radians-to-degrees respectively. The    *
*       routine "GETPI" will place the value of PI into FPACC1. This        *
*       routine should be used if the value of PI is needed in calculations *
*       since it is accurate to the full 24-bits of the mantissa.           *
*                                                                            *
******************************************************************************
*
*
DEG2RAD  EQU    *
         JSR    PSHFPAC2     SAVE FPACC2.
         LDX    #PIOV180     POINT TO CONVERSION CONSTANT PI/180.
DEG2RAD1 JSR    GETFPAC2     PUT IT INTO FPACC2.
         JSR    FLTMUL       CONVERT DEGREES TO RADIANS.
         JSR    PULFPAC2     RESTORE FPACC2.
         RTS                 RETURN. (NOTE! DON'T REPLACE THE "JSR/RTS" WITH
*                            A "JMP" IT WILL NOT WORK.)
*
*
RAD2DEG  EQU    *
         JSR    PSHFPAC2     SAVE FPACC2.
         LDX    #C180OVPI    POINT TO CONVERSION CONSTANT 180/PI.
         BRA    DEG2RAD1     GO DO CONVERSION & RETURN.
*
*
GETPI    EQU    *
         LDX    #PI          POINT TO CONSTANT "PI".
         JMP    GETFPAC1     PUT IT IN FPACC1 AND RETURN.
*
*
PIOV180  EQU    *
         FCB    $7B,$0E,$FA,$35
*
C180OVPI EQU    *
         FCB    $86,$65,$2E,$E1
*
*
*
******************************************************************************
*                                                                            *
*       The following two subroutines, PSHFPAC2 & PULPFAC2, push FPACC2     *
*       onto and pull FPACC2 off of the hardware stack respectively.        *
*       The number is stored in the "memory format".                        *
*                                                                            *
******************************************************************************
*
*
PSHFPAC2 EQU    *
         PULX                GET THE RETURN ADDRESS OFF OF THE STACK.
         PSHX                ALLOCATE FOUR BYTES OF STACK SPACE.
         PSHX
         XGDX                PUT THE RETURN ADDRESS IN D.
         TSX                 POINT TO THE STORAGE AREA.
         PSHB                PUT THE RETURN ADDRESS BACK ON THE STACK.
         PSHA
         JMP    PUTFPAC2     GO PUT FPACC2 ON THE STACK & RETURN.
*
*
PULFPAC2 EQU    *
         TSX                 POINT TO THE RETURN ADDRESS.
         INX                 POINT TO THE SAVED NUMBER.
```

```
         INX
         JSR    GETFPAC2     RESTORE FPACC2.
         PULX                GET THE RETURN ADDRESS OFF THE STACK.
         INS                 REMOVE THE NUMBER FROM THE STACK.
         INS
         INS
         INS
         JMP    0,X          RETURN.
*
*
*
****************************************************************************
*                                                                          *
*                           GETFPACx SUBROUTINE                            *
*                                                                          *
*       The GETFPAC1 and GETFPAC2 subroutines get a floating point number   *
*       stored in memory and put it into either FPACC1 or FPACC2 in a format *
*       that is expected by all the floating point math routines. These     *
*       routines may easily be replaced to convert any binary floating point *
*       format (i.e. IEEE format) to the format required by the math        *
*       routines.  The "memory" format converted by these routines is shown  *
*       below:                                                             *
*                                                                          *
*       31_______24 23 22_____________________0                            *
*        exponent   s         mantissa                                     *
*                                                                          *
*       The exponent is biased by 128 to facilitate floating point          *
*       comparisons.  The sign bit is 0 for positive numbers and 1          *
*       for negative numbers.  The mantissa is stored in hidden bit         *
*       normalized format so that 24 bits of precision can be obtained.     *
*       Since a normalized floating point number always has its most        *
*       significant bit set, we can use the 24th bit to hold the mantissa   *
*       sign.  This allows us to get 24 bits of precision in the mantissa   *
*       and store the entire number in just 4 bytes.  The format required by *
*       the math routines uses a seperate byte for the sign, therfore each  *
*       floating point accumulator requires five bytes.                     *
*                                                                          *
****************************************************************************
*
*
GETFPAC1 EQU    *
         LDD    0,X          GET THE EXPONENT & HIGH BYTE OF THE MANTISSA,
         BEQ    GETFP12      IF NUMBER IS ZERO, SKIP SETTING THE MS BIT.
         CLR    MANTSGN1     SET UP FOR POSITIVE NUMBER.
         TSTB                IS NUMBER NEGATIVE?
         BPL    GETFP11      NO. LEAVE SIGN ALONE.
         COM    MANTSGN1     YES. SET SIGN TO NEGATIVE.
GETFP11  ORAB   #$80         RESTORE MOST SIGNIFICANT BIT IN MANTISSA.
GETFP12  STD    FPACC1EX     PUT IN FPACC1.
         LDD    2,X          GET LOW 16-BITS OF THE MANTISSA.
         STD    FPACC1MN+1   PUT IN FPACC1.
         RTS                 RETURN.
*
*
GETFPAC2 EQU    *
         LDD    0,X          GET THE EXPONENT & HIGH BYTE OF THE MANTISSA,
         BEQ    GETFP22      IF NUMBER IS 0, SKIP SETTING THE MS BIT.
         CLR    MANTSGN2     SET UP FOR POSITIVE NUMBER.
         TSTB                IS NUMBER NEGATIVE?
         BPL    GETFP21      NO. LEAVE SIGN ALONE.
         COM    MANTSGN2     YES. SET SIGN TO NEGATIVE.
GETFP21  ORAB   #$80         RESTORE MOST SIGNIFICANT BIT IN MANTISSA.
GETFP22  STD    FPACC2EX     PUT IN FPACC1.
         LDD    2,X          GET LOW 16-BITS OF THE MANTISSA.
         STD    FPACC2MN+1   PUT IN FPACC1.
         RTS                 RETURN.
*
*
*
****************************************************************************
*                                                                          *
*                        PUTFPACx SUBROUTINE                               *
*                                                                          *
```

```
*       These two subroutines perform to opposite function of GETFPAC1 and   *
*       GETFPAC2. Again, these routines are used to convert from the         *
*       internal format used by the floating point package to a "memory"     *
*       format. See the GETFPAC1 and GETFPAC2, documentation for a           *
*       description of the "memory" format.                                  *
*                                                                             *
*******************************************************************************
*
*
PUTFPAC1 EQU    *
         LDD    FPACC1EX     GET FPACC1 EXPONENT & UPPER 8 BITS OF MANT.
         TST    MANTSGN1     IS THE NUMBER NEGATIVE?
         BMI    PUTFP11      YES.  LEAVE THE M.S. BIT SET.
         ANDB   #$7F         NO.  CLEAR THE M.S. BIT.
PUTFP11  STD    0,X          SAVE IT IN MEMORY
         LDD    FPACC1MN+1   GET L.S. 16 BITS OF THE MANTISSA.
         STD    2,X
         RTS
*
*
PUTFPAC2 EQU    *
         LDD    FPACC2EX     GET FPACC1 EXPONENT & UPPER 8 BITS OF MANT.
         TST    MANTSGN2     IS THE NUMBER NEGATIVE?
         BMI    PUTFP21      YES.  LEAVE THE M.S. BIT SET.
         ANDB   #$7F         NO.  CLEAR THE M.S. BIT.
PUTFP21  STD    0,X          SAVE IT IN MEMORY
         LDD    FPACC2MN+1   GET L.S. 16 BITS OF THE MANTISSA.
         STD    2,X
         RTS
*
```

Appendix D – Loading and Firing Code

This is the most recent version of my code. It is not complete of functional, and I am constantly changing it. I will replace this with the final version when I have complete it.

```
****************************************
*     My Constants and Variables    *
****************************************

DISTANCE     EQU   $0230
V_GET_X            EQU   $01
V_GET_Y            EQU   $02
V_GET_THETA EQU   $03
V_START_MOUSE      EQU   $04
V_STOP_MOUSE       EQU   $05
V_RESET_M_CNT      EQU   $06
V_RESET_ANGLE      EQU   $07
V_COMP_DIST EQU   $08

BASE         EQU   $1000
STACK        EQU   $7000
PORTA        EQU   $00
PIOC         EQU   $02
PORTC        EQU   $03
PORTB        EQU   $04
PORTCL             EQU   $05
DDRC         EQU   $07
PORTD        EQU   $08
DDRD         EQU   $09
PORTE        EQU   $0A
CFORC        EQU   $0B
OC1M         EQU   $0C
OC1D         EQU   $0D
TCNT         EQU   $0E
TIC1         EQU   $10
TIC2         EQU   $12
TIC3         EQU   $14
TOC1         EQU   $16
TOC2         EQU   $18
TOC3         EQU   $1A
TOC4         EQU   $1C
TI4/O5             EQU   $1E
TOC5             EQU     $1E
TCTL1        EQU   $20
TCTL2        EQU   $21
TMSK1        EQU   $22
TFLG1        EQU   $23
TMSK2        EQU   $24
TFLG2        EQU   $25
PACTL        EQU   $26
PACNT        EQU   $27
SPCR         EQU   $28
SPSR         EQU   $29
SPDR         EQU   $2A
```

```
BAUD        EQU   $2B
SCCR1       EQU   $2C
SCCR2       EQU   $2D
SCSR        EQU   $2E
SCDR        EQU   $2F
ADCTL       EQU   $30
ADR1        EQU   $31
ADR2        EQU   $32
ADR3        EQU   $33
ADR4        EQU   $34
BPROT       EQU   $35
EPROG       EQU   $36
OPTION            EQU   $39
COPRST            EQU   $3A
PPROG       EQU   $3B
HPRIO       EQU   $3C
INIT        EQU   $3D
TEST        EQU   $3E
CONFIG            EQU   $3F

PE1   EQU   %00000001   ;Value for ADCTL:
PE2   EQU   %00000010

BIT0        EQU   %00000001
BIT1        EQU   %00000010
BIT2        EQU   %00000100
BIT3        EQU   %00001000
BIT4        EQU   %00010000
BIT5        EQU   %00100000
BIT6        EQU   %01000000
BIT7        EQU   %10000000

BIT543            EQU   %00111000

* Interupt Vectors
        ORG     $00D3
        JMP     handle_oc5
        JMP     handle_oc4
        JMP     handle_oc3
        JMP     handle_oc2
        JMP     handle_oc1

      ORG $8000
INT
      lds   #$01ff
      clra
      staa  TMSK2,x
      ldaa  #%11111000        ;the 1's corrospond to OC1-5
      staa  TFLG1,x
      staa  TMSK1,x
      bset  TCTL1,x %01010101 ; toggle at interrupt, OC2-4
      ldaa  #$80              ; special OC1 setup
      staa  PACTL,x
      staa  OC1M,x
        cli
        LDX     #$1000
```

```
      LDAA  #%10000000  ; Bit 7 (ADPU) of OPTION
      STAA  OPTION,X    ; ..turn on A/D system

* Wait 100 microsec = 200 E-cycles for Charge Pump to Stabilize
      LDAA  #40         ; 2 E cycles
WAIT1 DECA              ; 2 E cycles
      BNE   WAIT1       ; 3 E cycles [(2+3)*40=200 E's]

      rts

      ORG   $8500
LOAD  LDD   #$14B0
      STD   $2
      LDAB  $D7

      LDAA  #PE1
      JSR   SAMPLE
      CBA
      BGT   LOAD
      BEQ   DONE1
      LDD   #$04B0
      STD   $2
      BRA   LOAD
DONE1 LDD   #$0BA0
      STD   $2
      RTS

      ORG   $9000
ARM   LDD   #$14B0
      STD   $2
      LDAB  $F7

      LDAA  #PE1
      JSR   SAMPLE
      CBA
      BGT   ARM
      BEQ   DONE2
      LDD   #$04B0
      STD   $2
      BRA   ARM
DONE2   LDD     #$0BA0
      STD   $2

      LDAA  $FF
      STAA  $7000
      RTS

      ORG   $9500
LOCK  LDD   #$14A0
      STD   $4
      LDAA  #106
LOCK1 NOP
      DECA
        BNE LOCK1

      LDD   #$0BA0
      STD   $04
```

```
      LDD   #$04B0
      STD   $02
      RTS

FIRE  LDD   #$04A0
      STD   $4
      LDAA  #106
FIRE1 NOP
      DECA
      BNE FIRE1

      LDD   #$0BA0
      STD   $04
      RTS



SAMPLE  LDX     #1000
      STAA  ADCTL,X           ;    PE1=BIT0

      LDAA  #6          ; 2 E cycles
WAIT2 DECA              ; 2 E cycles
      BNE   WAIT2       ; 3 E cycles (2+3)*6=30 E's
      LDAA  ADR1,X            ; Get new data

      RTS

handle_oc5:
      ldx   #$1000            ;required for bclr
      bclr  TFLG1,x %11110111
      ldaa  0,x         ;find level of PWM now
      anda  #%00001000
      beq   oc5pwmdown  ;level is down
      ldd   $08         ;load 'on' period from low mem
      bra   oc5pwmend
oc5pwmdown
      ldd   #28960            ;load constant low period
oc5pwmend
        addd    TOC5,x          ;add loaded period (hi or low)
      std   TOC5,x            ;set next inerrupt
      rti               ;all done
handle_oc4:
      ldx   #$1000            ;a slightly different routine is needed
      bclr  TFLG1,x %11101111 ;for each line. (different masks)
      ldaa  0,x
      anda  #%00010000
      beq   oc4pwmdown
      ldd   $06
      bra   oc4pwmend
oc4pwmdown
      ldd   #28960
oc4pwmend
      addd  TOC4,x
      std   TOC4,x
      rti
handle_oc3:
      ldx   #$1000
      bclr  TFLG1,x %11011111
```

```
      ldaa  0,x
      anda  #%00100000
      beq   oc3pwmdown
      ldd   $04
      bra   oc3pwmend
oc3pwmdown
      ldd   #28960
oc3pwmend
      addd  TOC3,x
      std   TOC3,x
      rti
handle_oc2:
      ldx   #$1000
      bclr  TFLG1,x %10111111
      ldaa  0,x
      anda  #%01000000
      beq   oc2pwmdown
      ldd   $02
      bra   oc2pwmend
oc2pwmdown
      ldd   #28960
oc2pwmend
      addd  TOC2,x
      std   TOC2,x
      rti
handle_oc1:
      ldx   #$1000
      bclr  TFLG1,x %01111111
      ldaa  0,x
      anda  #%10000000
      beq   oc1pwmdown
      ldd   $00
      bra   oc1pwmend
oc1pwmdown
      ldd   #28960
oc1pwmend
      addd  TOC1,x
      std   TOC1,x
      ldaa  OC1D,x
      eora  #$80
      staa  OC1D,x
      rti
```