COMPUTER INTERFACE FOR A
PASSIVE SELF-CONTAINED MICROCOMPUTER CONTROLLED
ABOVE-KNEE PROSTHESIS

by

Erik J. Heels

Submitted to the Department of
Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 1988

© Erik J. Heels 1988

The author hereby grants to M.I.T. permission to reproduce
and to distribute copies of this document in whole or in part.

Author__________________________________________________________
Department of Electrical Engineering and Computer Science
May 16, 1983

Certified by_____________________________________________________
Woodie C. Flowers
Thesis Supervisor

Accepted by_____________________________________________________
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses
COMPUTER INTERFACE FOR A
PASSIVE SELF-CONTAINED MICROCOMPUTER CONTROLLED
ABOVE-KNEE PROSTHESIS

by

Erik J. Heels

Submitted to the
Department of Electrical Engineering and Computer Science

May 16, 1988

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering

ABSTRACT

The feasibility of a passive self-contained microcomputer controlled
above-knee prosthesis has been demonstrated by past research. The three
goals of this thesis were to revise the algorithm which controls the
damping profile of the knee, to fix the low frequency discontinuous
behavior of the knee, and to write the software for interfacing the
entire unit with a Macintosh computer. The control algorithm is written
in assembly language, and the limiting factor on its accuracy is the
 truncation of significant figures caused by mathematical manipulation of
digital data. The low frequency discontinuous behavior demonstrated by
the knee was partially caused by inaccurate analog differentiation. The
computer interface for the Macintosh computer is very simple and easily
modifiable to future control algorithms. The algorithm itself can
easily be modified by adding additional hardware to the design such as
torque transducers. As it now stands, no knowledge of computers is
necessary for a therapist to interactively change the parameters of the
control algorithm.

Thesis Supervisor: Woodie C. Flowers
Title: Professor of Mechanical Engineering
ACKNOWLEDGEMENTS

First and foremost, thank you Mom and Dad for your prayers, your constant support and worrying, and your unwavering faith in me. Do you remember when I swore I'd never buy a computer? My, how things have changed.

Thank you Woodie for introducing me to this project and for telling me that I could do it. Thank you Stuart Schechter for your advice. Thank you Bill Murray for the use of your personal lab equipment and for helping me in the lab.

Thank you Ted for showing me the inside of a Macintosh. Thank you Ed, Mike, and Leslie for your constant support around the dorm. Thank you Dave and Dave, without whom I never would have graduated from M.I.T. in four years. Thank you Peter for helping me print out the final copies of this thesis.

Finally, I owe a special thank you to Karl Lindstrom, who worked on this project last year. Thank you for flying out from California to help me, for the many long distance phone calls, and for many late nights with David Letterknee. Karl, you warned me that this project would get me as it got you. You told me it would wrap me up body and soul, that it would dominate my every waking thought. Indeed it has, but it has been well worth it. And Karl, I think you can take care of plants.
DEDICATION

To my great, great, great grandfather, James Heels, who brought my family to this continent in 1841, and who later lost his foot to a bone disease.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>2</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>3</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>4</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>5</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>6</td>
</tr>
<tr>
<td>Chapter 1 - INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>Overview and History of the Project</td>
<td>7</td>
</tr>
<tr>
<td>Introduction to the System</td>
<td>8</td>
</tr>
<tr>
<td>Goals of this Thesis</td>
<td>10</td>
</tr>
<tr>
<td>Chapter 2 - HARDWARE</td>
<td></td>
</tr>
<tr>
<td>Overview of Hardware</td>
<td>16</td>
</tr>
<tr>
<td>Hardware Modifications</td>
<td>18</td>
</tr>
<tr>
<td>Summary of Hardware Features</td>
<td>20</td>
</tr>
<tr>
<td>Chapter 3 - SOFTWARE</td>
<td></td>
</tr>
<tr>
<td>Overview of Software</td>
<td>25</td>
</tr>
<tr>
<td>Software Modifications</td>
<td>26</td>
</tr>
<tr>
<td>Summary of Software Features</td>
<td>28</td>
</tr>
<tr>
<td>Chapter 4 - COMPUTER CONTROL INTERFACE</td>
<td></td>
</tr>
<tr>
<td>Overview of Computer Control Interface</td>
<td>31</td>
</tr>
<tr>
<td>The Macintosh Software</td>
<td>32</td>
</tr>
<tr>
<td>Summary of Control Features</td>
<td>34</td>
</tr>
<tr>
<td>Chapter 5 - EVALUATIONS AND RECOMMENDATIONS</td>
<td></td>
</tr>
<tr>
<td>System Evaluation</td>
<td>38</td>
</tr>
<tr>
<td>A Blueprint for the Future</td>
<td>40</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A - µPD78C10 Data Sheet</td>
<td>45</td>
</tr>
<tr>
<td>B - Computer Board Schematics</td>
<td>56</td>
</tr>
<tr>
<td>C - Software Listings</td>
<td>59</td>
</tr>
<tr>
<td>Figure Number</td>
<td>Title</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>1-1</td>
<td>MAGNETIC PARTICLE BRAKE TRANSMISSION UNIT</td>
</tr>
<tr>
<td>1-2</td>
<td>INPUT/OUTPUT CHARACTERISTICS OF THE MPB</td>
</tr>
<tr>
<td>1-3</td>
<td>COMPLETE KNEE UNIT</td>
</tr>
<tr>
<td>1-4</td>
<td>MICROCOMPUTER SYSTEM BLOCK DIAGRAM</td>
</tr>
<tr>
<td>2-1</td>
<td>MEMORY MAP</td>
</tr>
<tr>
<td>2-2</td>
<td>SERIAL PORT DETAILS</td>
</tr>
<tr>
<td>2-3</td>
<td>CHARACTERISTICS OF A SIMPLE DIFFERENTIATOR, A PRACTICAL DIFFERENTIATOR, AND A BANDPASS FILTER</td>
</tr>
<tr>
<td>3-1</td>
<td>A PROGRAMMABLE SQUARE WAVE</td>
</tr>
<tr>
<td>4-1</td>
<td>MONITOR PROGRAM COMMANDS</td>
</tr>
<tr>
<td>4-2</td>
<td>FESTUS PROGRAM SCREEN</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

Overview and History of the Project

An above-the-knee (A/K) prosthesis is a device designed to help someone who has lost their leg above the knee. An artificial limb can never totally replace the lost limb, but such devices allow the amputee to overcome some of the barriers to living a "normal" life. A/K prostheses are designed to help amputees with the very complex task of walking, a major barrier for someone who has lost a leg. Typical existing prostheses, however, do not allow adequate control of the swing and stance phases of gait. This prosthesis was originally designed to be self contained, to be passive, to provide control of the swing and stance phases of gait with an onboard microcomputer, and to provide a personal computer interface so that control algorithms could be "fine-tuned" for each amputee. This thesis is part of ongoing prosthesis research at the Eric P. and Evelyn E. Newman Laboratory for Biomechanics and Human Rehabilitation at M.I.T.

Control of the swing and stance phases is necessary because typical prostheses do not come very close to matching the many different parameters of the walking cycle. For a detailed description of the walking cycle see Schechter (10). There are essentially two methods for
controlling the operation of an A/K prosthesis: active control and passive control. This knee unit was developed by Schechter, and it employs passive control. This method was chosen so that the prosthesis would be self-contained. Active systems are more complex than passive systems, and they have not yet been developed to the point that they are self-contained. The passive element used to control this prosthesis is a Magnetic Particle Brake (MPB). See Figure 1-1. The MPB supplies torque to the knee joint and was designed to run off of standard commercially available batteries. With this compact power supply, the prosthesis is self-contained. See Figure 1-2 for the input/output characteristics of the MPB. The onboard microcomputer, also powered by batteries, allows for control of the entire unit. The personal computer interface, part of this thesis, allows the prosthesis to be "fine-tuned" to each user. The complete knee unit is shown in Figure 1-3.

Introduction to the System

The overall microcomputer system block diagram is shown in Figure 1-4. The inputs to the system consist of knee joint velocity and position information provided by a goniometer (a potentiometer whose resistance varies linearly with angular displacement). See Figure 1-5. Other information can also be made available to the system. Torque information, for example, can be provided by adding a torque transducer. See Schechter (46). Also, sensors could be placed on the foot to detect heel contact, toe contact, and other parameters of gait. Control input is available from either a control box or a monitor. The advantage of the monitor over the control box is that the monitor provides feedback
and is very user-friendly. The monitor displays values of any numbers stored in memory. The purpose of the control is to change the parameters in the control algorithm being used for each specific user. For example, in the viscous damping program written by Lindstrom (74),

\[ PB \sim AV + B \] (1-1)

where \( PB \) is the power delivered to the brake, \( V \) is the velocity of the knee joint, and \( A \) and \( B \) are user-tunable constants. In this way, the prosthesis can be "fine-tuned" for each user. The outputs from the system include the monitor output (as discussed above) and the control signal to the brake. In Chapter 3, I will discuss the specifications of this signal and how it is controlled.

The knee unit consists the MPB and the microcomputer electronics board mounted on an aluminum housing. The MPB acts as an energy dissipater and is manufactured by Force Limited. The microcomputer is an NEC \( \mu \)PD78C10. The \( \mu \)PD78C10 is the Central Processing Unit (CPU) for the system, and it is a very powerful microcomputer with such features as 256 bytes of onboard Random Access Memory (RAM) and its own Arithmetic Logic Unit (ALU). See Appendix A for the \( \mu \)PD78C10 data sheet and a complete listing of all of its features. For a complete listing of all parts and supplier acknowledgments see Lindstrom (79). The built in RAM and ALU saves space on the computer board for other components. This will be an important issue as this project progresses and more features are added.
Goals of this Thesis

At the start of this thesis, the knee had been assembled and successfully tested with a viscous (velocity dependant) damping control algorithm. However, two problems existed with the prosthesis. First, the viscous damping algorithm was not very accurate. Second, the knee demonstrated a low frequency discontinuous behavior. The knee joint did not smoothly resist externally applied forces. Since the microprocessor runs off of a 1 [MHz] clock, one would expect that the output to the magnetic particle brake could be changed often enough to provide smooth operation. The cause of this behavior was not clear. In addition, the computer control interface had not been successfully written. The final goals of this thesis were to revise the algorithm which controls the damping profile of the knee, to fix the low frequency discontinuous behavior of the knee, and to write the software for interfacing the entire unit with a Macintosh computer. In order to meet these goals, this involved:

1. Learning the CM/M operating system and the necessary software used to modify the assembly language.
2. Learning 7800 series assembly language.
3. Developing a more accurate digital control algorithm for the viscous damping program and writing the new routine.
4. Identifying the cause of the the low frequency discontinuous behavior of the knee.
5. Understanding the system analog and digital hardware.
6. Implementing serial communications with the Macintosh computer emulating a terminal.
7. Learning the "C" programming language.

8. Writing the control programs for the Macintosh computer in the C programming language and making this program communicate with the knee via the serial port.

9. Documenting all changes to the system.
Figure 1-1  MAGNETIC PARTICLE BRAKE TRANSMISSION UNIT
Figure 1-2  INPUT/OUTPUT CHARACTERISTICS OF THE MPB
Figure 1-2  COMPLETE KNEE UNIT
Figure 1-4  MICROCOMPUTER SYSTEM BLOCK DIAGRAM
Chapter 2

HARDWARE

The hardware was successfully modified and tested. The low frequency discontinuous behavior of the knee was caused by a combination of hardware and software problems. The modifications made to the analog differentiation circuitry solved the hardware problem. Hardware schematics are shown in Appendix B.

Overview of Hardware

The hardware consists of the CPU, 12K bytes of usable memory, input subsystems, output subsystems, and the power supplies. All of these components are mounted on a wire-wrapped computer board which is mounted to the aluminum housing. The power supply system consists of four 1.5 [Volt] AA batteries, one 9 [Volt] battery, and one 3 [Volt] lithium battery. The AA batteries power the CPU and, when connected in series with the 9 [Volt] battery, they also partially power the MPB. The 3 [Volt] lithium battery is meant to preserve the state of the Static Random Access Memory (SRAM) at power-off, but this feature is not implemented in this version.

As mentioned earlier, the \( \mu P D 7 8 C 1 0 \) is a very powerful microcomputer. The \( \mu P D 7 8 C 1 0 \) has a 16 bit memory address and is capable of addressing over 65K \( (2^{16}) \) bytes of memory. For this application, it
only addresses 12K bytes of memory. Locations 0000H to 0FFFH are the 4K bytes of external ROM. This is where the control algorithms are stored. Locations 2000H to 3FFFH are the 8K bytes of external RAM. This is primarily used as a stack. Locations FF00H to FFFFH are the 256K of internal RAM. This is where many important variables are stored. The complete memory map is shown in Figure 2-1.

The serial communications subsystem consists of three lines: transmit, receive, and ground. The settings on the terminals and on the µPD78C10 were 9600 baud, 8 data bits, no parity, and 2 stop bits. The connector for the VT100 terminal is a standard 25 pin RS232 cable, but only three of the lines are actually being used. The connector for the Macintosh is an eight pin circular connector, but again only three of these pins are being used. See Figure 2-2 for serial port details.

The A/D buffer subsystem is designed to translate the resistance information from the goniometer into velocity proportional voltage signals. I will discuss this subsystem in more detail in the next section.

The D/A buffer subsystem is designed to power the MPB. It consists of an operational amplifier (op-amp) in a Schmitt-trigger configuration followed by three cascaded Metal Oxide Semiconductor Field Effect Transistors (MOSFETs). For a detailed analysis of this circuit, see Lindstrom (21).
Hardware Modifications

The A/D subsystem required modification because it was not behaving in the manner for which it was designed. The circuit has three stages comprised of op-amps, resistors, and capacitors.

The first stage is a voltage follower connected to a voltage divider. This stage is necessary to provide continuous voltage output proportional to angle because the goniometer is calibrated in small discrete increments.

With \( V_1 \) as the first stage input and \( V_2 \) as the first stage output, the output equation for the first stage is

\[
V_2 = V_1 = \frac{5(R_{13})}{(R_{13} + R_{12})} \text{(Volts)} \quad \text{for } R_{11} = 0 \text{ [Ohms]} \quad (2-1)
\]

and

\[
V_2 = V_1 = \frac{5(R_{13} + R_{11})}{(R_{13} + R_{11} + R_{12})} \text{(Volts)} \quad \text{for } R_{11} = 20K \text{ [Ohms]} \quad (2-2)
\]

For \( 0 \leq R_{11} \leq 20K \text{ [Ohms]} \), \( R_{12} = 560 \text{ [Ohms]} \), and \( R_{13} = 220 \text{ [Ohms]} \), \( V_2 \) will follow \( V_1 \) and will range from \( 1.9 \leq V_2 \leq 4.9 \text{ [Volts]} \).

The second stage is a practical analog differentiator. This circuit has a 6dB per octave AC gain and is extremely susceptible to high frequency noise. Also, because of the capacitor, there is a 90 degree phase shift which tends to cause instability. The practical differentiator corrects these problems by the addition of a capacitor and a resistor. This circuit is now a specialized bandpass filter. A simple differentiator, a practical differentiator, a bandpass filter, and their characteristics are shown in Figure 2-3. The difference
between these last two circuits is that the practical differentiator has
two poles in the pole zero diagram, but, unlike the bandpass filter,
these two poles occur at the same place. In other words, a bandpass
filter is a differentiator at low frequencies, a simple amplifier at
midrange frequencies, and an integrator at high frequencies. A
practical differentiator is also a differentiator at low frequencies and
an integrator at high frequencies, but the bandpass region is equal to
zero, and the circuit will behave as a differentiator at frequencies
below \( f_h \).

The problem with the old circuitry was that is was not a practical
differentiator; it was a second order bandpass filter with a very poor
frequency response. The practical differentiator should not have a
bandpass region. The desired cutoff frequency is 100 [Hz] because most
gait parameters occur at frequencies below this. See Schechter (68).
The frequency limitations are also shown in Figure 2-3. In addition, an
offset resistor must be placed between the positive terminal of the op-
amp and ground. The ideal op-amp model describes \( i_+ = 0 \), but this is a
model, and, in fact, \( i_+ \neq 0 \). This current must have a load. To account
for this fact, add \( R_{16} \) between the positive terminal and ground. More
specifically,

\[
R_{16} = R_{14} || R_{15}
\]  

(2-3)
in order to minimize offset error due to bias current.

The third stage is a voltage adder. The purpose of this stage is
to achieve the desired output voltage range for the \( \mu P D 7 8 C 1 0 \).
With V3 as the third stage input and V4 as the third stage output, the output equation for the third stage is

\[ V4 = -\frac{R19}{R17 + 5/R18} \text{ [Volts]} \quad (2-4) \]

The output voltage now delivered to the \( \mu P D78C10 \) ranges from 0 to 5 [Volts] which is the desired range for digital data.

**Summary of Hardware Features**

The hardware was designed so that additional transducers could be added. Any hardware additions to the present system would consist entirely of analog circuitry: op-amps, resistors, and capacitors. There is room on the computer board now for several more devices. Beyond this, a second layer may need to be added to the board. There are presently eight op-amps on the computer board (2 quadruple op-amp packages). Three of these are unused. The addition of the torque transducer would require several op-amps. Other binary information (such as heel contact) would probably only require a single op-amp.

The present control algorithm takes up less than 50K bytes of memory in the 4K external ROM. There is more than adequate space for additional algorithms to be added to the ROM. The 8K bytes of RAM is being used primarily a stack. Future algorithms should take full advantage of this stack architecture. If more memory is needed, the \( \mu P D78C10 \) can be wired to handle more than 65K bytes of memory. The easiest way to do this would be to change the ROM from 4K to 8K.
The µPD78C10 is a very powerful microcomputer and is ideal for this specialized application. The complexity of the analog circuitry will be the limiting factor for space on the computer board. Since the microcomputer has a built in serial interface, all communications with the microcomputer can occur via the 3 conductor serial connector cable.
Figure 2-2 SERIAL PORT DETAILS
Figure 2-3 CHARACTERISTICS OF A SIMPLE DIFFERENTIATOR, A PRACTICAL DIFFERENTIATOR, AND A BANDPASS FILTER
Chapter 3

SOFTWARE

The software was successfully modified and tested. The complexity of mathematical manipulation of digital data causes accuracy problems in control algorithms. The new algorithm is more realistic and more flexible. It demonstrates all the features of the knee unit.

Overview of Software

The software is NEC 7800 series assembly language. It was written on an Intel 8085 CPU based computer supporting the CP/M 80 operating system. The software was assembled on the same computer with NEC 7800 series assembler. PROM programming is also supported by the 8085 computer. All files are stored on eight inch floppy disks.

The assembly level software is intended to implement control algorithms for the knee unit. Many algorithms exist, and the implemented version supplies power to the MPB according to the following equation:

\[ P_b \sim AV^2 + BV + C \] (3-1)

The \( V^2 \) algorithm was chosen because it was used as a control
algorithm for previous prostheses. For a complete history of algorithm
design see Schechter (34). The advantage of this prosthesis is that it
is not limited to one control algorithm.

More specifically, the MPB is controlled by a programmable square
wave delivered from the μPD78C10. The width of the square wave, the
duty cycle, is controlled directly by the damping algorithm. As shown
in Figure 3-1, the duty cycle can vary from 0 [μsec] to 1024 [μsec] for
a maximum value of about 1 [msec]. It is convenient to think of the
output in increment of microseconds since this is the clock rate of the
microcomputer. The output from the control algorithm is a number
between 0 and 1024. This value sets the point at which the square wave
inverts. The square wave then resets at 1024, and the counting starts
over again. Two square wave outputs, CO0(PC6) and CO1(PC7), are
available. These can be combined in various ways with control logic,
but for this implementation, only one of these outputs (CO1) was
necessary.

Software Modifications

The two main problems with the old algorithm were its
inflexibility and the manner with which it handled digital data. The
microcomputer supports eight bit multiply and divide commands. The
results of these commands is a 16 bit answer. However, often only the
most significant eight bits were being used. By throwing away the least
significant eight bits, accuracy was lost. For example, an eight bit
number ranges from 0 to 255. When this number is multiplied by another
eight bit number, the result is a 16 bit number ranging from 0 to
65,535. By using only the most significant eight bits, this number is effectively divided by 255. In this way, accuracy was being lost. See Appendix 3 for a complete listing of the software. Complete documentation of the algorithm is included in the code.

However, there is a tradeoff involved. The final value of the duty cycle is \( \leq 1024 \), and the velocity data value is \( \leq 255 \). In order to maintain all of the unique values, one could only safely multiply the velocity by a value \( \leq 4 \). The algorithm is limited to one such as

\[
P = KV \quad \text{where} \quad 0 \leq K \leq 4 \quad (3-2)
\]

unless some approximations are made or unless the frequency of the pulse width is increased. Increasing the frequency might help, but lower frequencies provide smoother average voltages to the MPB. Also, higher frequencies are audible. Unfortunately, some significant figures will be lost in the final calculations.

The control algorithm implemented is a combination of three separate algorithms. By setting the constants or combinations of the constants A, B, or C equal to zero, different algorithms can be chosen. For example, setting A and B equal to zero provides a constant damping algorithm. The following are the ranges of values for the constants:

\[
\begin{align*}
0 & \leq A \leq 8 \quad (3-3) \\
0 & \leq B \leq 256 \quad (3-4) \\
0 & \leq C \leq 256 \quad (3-5)
\end{align*}
\]
Two separate strategies for retaining accuracy are used here. The first, the $AV^2$ term, is calculated by retaining as much as possible of the original velocity information. The second, the BV term, is calculated by retaining less of the original velocity information. However, since the $AV^2$ term is the product of three terms (as opposed to two terms in BV), some accuracy will also be lost in computation. The tradeoff is between range of values for the constants and accurate computation of the final results. Therefore, higher order terms in control algorithms will have less accuracy than lower order terms.

Summary of Software Features

The present algorithm is a second order viscous damping control scheme implemented via square wave pulse width modulation. All control algorithms are written at the assembly language level. The truncation of significant figures due to mathematical manipulation of digital data limits the accuracy of higher order terms in these equations.

More control can be gained from adding simple binary sensors to the foot. These sensors would detect different parameters in the gait cycle. These parameters are heel contact, toe contact, heel off, and toe off. See Schechter (12). These two sensors could be added to control the knee. These sensors would send signals to the microcomputer which, in turn, would signal the control algorithm to jump to different subroutines. For example, once heel contact was detected, the control algorithm could jump to a maximum damping subroutine. Final implementation of such a control scheme involves understanding the
characteristics of both swing and stance control and how control signals from external sensors determine which subroutine is activated.
Figure 3-1  A PROGRAMMABLE SQUARE WAVE
Chapter 4

COMPUTER CONTROL INTERFACE

Initial attempts to interface the knee unit with a personal computer proved unsuccessful partially because the approach taken was a "top down" approach and partially because the format of the control data was not finalized. The computer control implemented here is designed in a "bottom up" approach and utilizes the user-friendly nature of the Macintosh computer.

Overview of Computer Control Interface

This interface was designed to be user friendly and to allow the user to control the damping profile of the prosthesis. Ideally, a therapist with no knowledge of computers sits down at a personal computer and interactively "fine-tunes" the prostheses for the amputee. The computer system must be easy for the therapist to use, it must support the existing serial interface, and it must control the knee from a "bottom up" program built on existing software.

The Macintosh was chosen over other personal computers because it is very easy for the user to understand. Ideally, the user needs only to move and click the Macintosh "mouse" to implement all commands. The Macintosh software could even train the user how to use the program.
Existing software written by Lindstrom allows the user to change the constants (in this version, A, B, and C) by entering hexadecimal format characters on a terminal. The constants which describe the control algorithm are stored in RAM. By changing these constants, the knee can be "fine-tuned" to each user. Therefore, the Macintosh must be able to emulate a monitor to support this software. The monitor program is part of the assembly language software in ROM. The monitor program originally supported several different commands to allow for various different choices for future development of this software. The design path chosen here represents the most realistic approach to control of the prosthesis. I will discuss this choice in more detail in the next chapter. The format of the control data for the monitor program is shown in Figure 4-1.

In order to write an effective control program on the Macintosh, the programmer must adhere to the strict Macintosh user interface guideline. For example, all Macintosh programs must support the existing "desk accessory" programs. The control program is written in LightSpeedC (© 1986 THINK Technologies, Inc.), an efficient version of the popular "C" programming language. The control program also uses SimpleTools2 (© 1985, 1987 Erik Kilk), a programming tool which provides the framework for the Macintosh user interface.

The Macintosh Software

The terminal emulation application used is Kermit (© 1986 Columbia University), a public domain terminal emulation program. This part of the interface emulates the VT100 terminal used in the lab. The
prosthesis can be attached to either the VT100 with the 25 pin connector or to the Macintosh with the eight pin connector. Kermit allows for all standard terminal controls. With the terminal program, one can change data in RAM, read data from RAM to the screen, and display the command list. With Kermit, the Macintosh effectively emulated a monitor and demonstrated all the features of the monitor program.

The Macintosh control program groups these commands into "macro" commands and sends them over the serial port with one click of the "mouse." The format of this program, called FESTUS, directly follows the function of the control algorithm implemented in ROM. With FESTUS, the commands are selected from three menus, and the appropriate information is sent out the serial port. No key strokes are required. A typical command for the terminal program is

\[ C<\text{sp}>FF60<\text{sp}>3B<\text{cr}> \] (4-1)

where \(<\text{sp}>\) is a space and \(<\text{cr}>\) is a carriage return. This command changes the value of hexidecimal memory location FF60 to the hexidecimal value 3B. This command is assigned to a command under the "Control C" menu. See Figure 4-2. Each constant has a range of values, and each menu, in theory, can support an infinite amount of commands. However, implementing any more than 100 commands under any one menu tends to slow the program down considerably. This problem can easily be fixed by adding more menus. FESTUS does not allow the user to save data to a file, but the information can be drawn on the screen with the mouse and subsequently printed out with the print screen command. See Appendix C
for more details. FESTUS allows the user to change A from 0 to 100% in increments of 25%. B and C may be changed from 0 to 100% in increments of 10%. FESTUS is very easy to change. Different commands are implemented by grouping character strings together. FESTUS demonstrated all of the features of the monitor program with the ease of the Macintosh interface and without the difficulty of entering hexadecimal characters on a monitor.

**Summary of Control Features**

The control application was written in LightSpeedC, an efficient compiler. "C" was chosen because it is popular and efficient. The complexity of writing the user interface routines for the Macintosh was minimized by using a transportable application skeleton, SimpleTools2. FESTUS allows the user to control the constants of the control algorithm and supports all Macintosh "desk accessories."

The format of the control application follows the function of the control algorithm. For example, if a different parameter were added, the only change that would need to be made to FESTUS would be the addition of a menu and the associated commands. These commands are groups of characters in the format required by the monitor program. All changes to the algorithm happen via the monitor program, and FESTUS can only be as fast as the monitor program. FESTUS is very easy to update. The code which sends the information over the serial port is the heart of the program. This code can be changed in any way the programmer desires. With the addition of this level of abstraction, FESTUS allows
a user with no knowledge of computers to "fine tune" the control algorithm for the prosthesis.
A --------------------- DISPLAY COMMAND LIST

C <XXXX> <DD>----------- CHANGE BYTE AT MEMORY LOCATION XXXX TO DD

D <XXXX> <YYYY>-------- DISPLAY DATA BLOCK FROM XXXX TO YYY

<XXXX> and <YYYY> are four digit hexadecimal memory locations.

<DD> is a two digit hexadecimal number.

Figure 4-1 MONITOR PROGRAM COMMANDS
The braking power delivered to the knee is proportional to $A*U*V + B*V + C$. Select $A$, $B$, or $C$ from the Control A, B, or C menus.

Figure 4-2  FESTUS PROGRAM SCREEN
Chapter 5

EVALUATION AND RECOMMENDATIONS

The goals of this thesis were met! The assembly language control algorithm was updated to demonstrate second and first order viscous damping. Redundant procedure calls and subroutines were eliminated. The low frequency discontinuous behavior of the prosthesis was corrected by changing the analog differentiation circuitry. FESTUS, the control application for the Macintosh, was written and successfully controlled the prosthesis. In addition, the Macintosh supported the terminal program directly with Kermit.

System Evaluation

The hardware changes made to the prosthesis provided smooth operation of the knee joint. However, with a 1024 [μsec] period, the minimum duty cycle must be set to 80 [μsec]. At values below this, the square wave output is still nonlinear and unstable. The MPB is designed to be current controlled, but here it is actually being controlled with a variable voltage square wave. Since the 6 [Volt] battery controls both the MPB and the μPD78C10, overloading could be a problem. The MPB should have a separate power supply.
The assembly language software was reduced in size considerably by removing subroutines. The monitor program originally supported commands to enter "programs" one byte at a time into RAM, to run these "programs," and to move blocks of data. Entering data one byte at a time is tedious and not necessary. Programs stored in RAM are volatile. All control algorithms should be placed permanently in ROM. The two commands I left, change and display (write and read), represent all that can be done to memory locations. All of the commands I eliminated can be duplicated by these two commands. In addition, the truncation of significant figures will be unavoidable in complex computations with digital data. In this version, many eight bit multiplications produced 16 bit results, but the desired result was only a ten bit number. Some accuracy will be lost.

The speed of FESTUS is limited by the speed of the monitor program. Originally I tried to send the entire character string at once over the serial port. The baud rates of the microcomputer and of FESTUS were both set to 9600, so I did not think that timing would be a problem. However, the monitor program runs concurrently with the viscous damping program. The two routines share processing time, and the microcomputer is forced to parallel process. The constants of the control algorithm can be changed with the terminal program because the user enters the characters on the terminal one at a time. There is an inherent delay in this process because users cannot type very fast compared to 9600 baud. When an entire string is sent over the serial port, this delay does not exist. To solve this flow control problem, I
implemented a delay routine and sent the characters over the serial port one at a time. This solved the flow control problem.

**A Blueprint for the Future**

This prosthesis is very complex and could easily support multiple research projects. When I started this thesis, there were many problems with the prosthesis. The terminal connectors had been lost, the wiring diagrams were inaccurate, and the prosthesis was not operating smoothly. Other problems arose which were not as obvious. For example, the flow control problem was not one I had anticipated. This project bridges the gap between software and hardware, and often the distinction between the two is vague. The development of this prosthesis involves many tradeoffs. Below, I list, in my perceived order of importance, goals for the future and the tradeoffs involved (if any). Some goals include:

1. Add transducers, add sensors, and develop better control algorithms. This prosthesis has the potential to support much more powerful control algorithms. By adding a torque transducer, a heel contact sensor, and a toe contact sensor, these algorithms can be realized. The addition of the torque transducer may cause some power problems.

2. Test on a patient. This would be the best way to get feedback about how the prosthesis feels.

3. Redesign the power supply. The charge pumped capacitors which deliver the -5 [Volt] supply cannot fully replace a "real" power supply. Also, a "real" power supply might prevent more of the low frequency
discontinuous behavior at duty cycles below 80 [μsec]. More batteries may need to be added to power the MPB separately, but this adds weight to the prosthesis.

4. Add offset resistors to all stages of A/D subsystem. This may be necessary for the same reason that an offset resistor was needed in the analog differentiation circuitry. However, the more precise the circuit is, the more room it takes up on the computer board.

5. Redesign D/A subsystem. This system powers the MPB. High speed switching is a difficult task, and this circuit may need to be designed differently.

6. Get a "desk accessory" terminal program for the Macintosh. In this way, one would not have to quit out of FESTUS in order to run the monitor program. These programs exist, but most don't support 9600 baud.

7. Write the "C" subroutine to send characters over the serial port. In FESTUS, each character string is a separate command. FESTUS works without this subroutine, but it would be nice to have some procedural abstraction.

8. Incorporate feedback directly into FESTUS. Information is being sent to the Macintosh as it sends the character string to the prosthesis. This information is the same that the monitor program receives. Routines could be written to display knee velocity and angle information on the screen. Writing this program would be fairly lengthy endeavor. The format of FESTUS can be altered in many ways. Control could be implemented with "scroll bars" or "buttons" instead of menu commands. Feedback could be displayed graphically or otherwise. The
tradeoff here is one between complexity of the format of FESTUS and speed of operation. Already, there is a significant flow control problem.

9. Change the monitor program to eliminate output from the microcomputer. The changes could be made much faster if the monitor program did not send information to the screen. Communications would only occur in one direction - from the monitor or Macintosh to the microcomputer. The tradeoff here is the same as above. The more simple the output to the terminal, the quicker information can be passed to the microcomputer.

10. Write a 7800 series assembler and configure a PROM programmer for the Macintosh. The Intel 8085 CPU based computer is very slow. Assemblers are not trivial, but it would be very possible to write one for the Macintosh. The final goal would be to have a totally Macintosh-based development system. The assembly language for the microcomputer would be written and assembled on the Macintosh. Then, the prosthesis could be monitored and controlled with the Macintosh. This is also not urgent, but it would be nice.

11. Fix goniometer bracket. The bracket which holds the goniometer to the housing is cracked. This will need to be fixed before the prosthesis is fully tested.

This passive, self-contained, microcomputer controlled, above-the-knee prosthesis is now programmable from a Macintosh personal computer. The prosthesis incorporates the latest hardware and software
technologies, and it is the next step towards total replacement of a lost limb.
REFERENCES


Appendix A

μPD78C10 Data Sheet
μPD78C10/78C11

CMOS Single-Chip 8-Bit Microcomputer with A/D Converter
Description

The NEC uPD78C10/uPD78C11 is a high performance, single chip microcomputer integrating sophisticated on-chip peripheral functionality normally provided by external components. The device's internal 16-bit ALU and data paths, combined with a powerful instruction set and addressing, make the uPD78C10/uPD78C11 appropriate in data processing as well as control applications. The device integrates a 16-bit ALU, 4K-ROM, 256-byte RAM with an 8-channel A/D converter, a multifunctional 16-bit timer/event counter, two 8-bit timers, a USART and two zero-cross detect inputs on a single die, allowing its use in fast, high end processing applications. This involves analog signal interface and processing.

The uPD78C11 is the mask-ROM high volume production device embedded with custom customer program. The uPD78C10 is a ROM-less version for prototyping and small volume production.

Features

- CMOS technology
  - 2.5 TO 6.0 V operating range
  - 17 mA operating supply current
- Complete single chip microcomputer
  - 16-bit ALU
  - 4K-ROM
  - 256-byte RAM
- 44 I/O lines
- Two zero-cross detect inputs
- Two 8-bit timers
- Multifunction 16-bit timer/event counter
- Expansion capabilities
  - 8085A bus-compatible
  - 60K-byte external memory address range
- 8-channel, 8-bit A/D converter
  - Autoscan mode
  - Channel select mode
- Full duplex USART
  - Synchronous and asynchronous
- 153 instruction set
  - 16-bit arithmetic, multiply and divide
- 1 us instruction cycle time (12 MHz operation)
- Prioritized interrupt structure
  - 3 external
  - 8 internal
- 2 Standby modes
  - HALT mode uses 9 mA supply current
  - STOP mode uses less than 1 uA supply current
- On-chip clock generator
<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Flat</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>1, 2</td>
<td>PA0-PA7</td>
<td>Port A: (Three-state input/output) 8-bit programmable I/O port. Each line independently programmable as an input or output. Reset places all lines of Port A in input mode.</td>
</tr>
<tr>
<td></td>
<td>59-64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-16</td>
<td>3-10</td>
<td>PB0-PB7</td>
<td>Port B: (Three-state input/output) 8-bit programmable I/O port. Each line independently programmable as an input or output. Reset places all lines of Port B in input mode.</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>PC0</td>
<td>Transmit Data (TxD): Serial data output terminal.</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>PC1</td>
<td>Receive Data (RxD): Serial data input terminal.</td>
</tr>
<tr>
<td>Pin No.</td>
<td>DIP/QUIP</td>
<td>Flat</td>
<td>Symbol</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>PC2</td>
<td>Port C: (Three-state input/output) 8-bit programmable I/O port. Each line independently programmable as an input or output. Alternatively Port C may be used as control lines for USART and timer. Reset puts Port C in port mode and all lines in input mode.</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>PC3</td>
<td>Serial Clock (SCK): Serial clock input/output terminal. When internal clock is used, the output can be selected; when an external clock is used, the input can be selected.</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>PC4</td>
<td>Timer Input (TI) /interrupt request input (INT2): Timer clock input terminal; can also be used as falling edge, maskable interrupt input terminal and AC input zero-cross detection terminal.</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>PC5</td>
<td>Timer Output (TO): This output signal is a square wave whose frequency is determined by the timer /counter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pin No.</td>
<td>Flat</td>
<td>Symbol</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>DIP/QUIP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23,24</td>
<td>17,18</td>
<td>PC6,PC7</td>
<td>Counter Outputs 0,1 (CO0-COL): Programmable rectangular wave output terminal based on timer/event counter.</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>NMI</td>
<td>Falling edge, nonmaskable interrupt (NMI) input.</td>
</tr>
<tr>
<td>26</td>
<td>20</td>
<td>INT1</td>
<td>This signal is a rising edge, maskable interrupt input. This input is also used to make the zero-cross detection AC input.</td>
</tr>
<tr>
<td>27</td>
<td>21</td>
<td>MODE1</td>
<td>Used as input in conjunction with MODE0 to select appropriate memory expansion mode. Also outputs M1 signal during each opcode fetch.</td>
</tr>
<tr>
<td>28</td>
<td>22</td>
<td>RESET</td>
<td>(Input, active low), RESET initializes the uPD7811.</td>
</tr>
<tr>
<td>29</td>
<td>23</td>
<td>MODE0</td>
<td>Used as input in conjunction with MODE1 to select appropriate memory expansion mode. Also used to output IO/1.</td>
</tr>
<tr>
<td>30,31</td>
<td>24,25</td>
<td>X2,X1</td>
<td>This is a crystal connection terminal for system clock oscillation. When an external clock is supplied X1 is the input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(crystal)</td>
</tr>
<tr>
<td>32</td>
<td>26</td>
<td>VSS</td>
<td>Power supply ground potential.</td>
</tr>
<tr>
<td>33</td>
<td>27</td>
<td>AVSS</td>
<td>A/D converter power supply ground potential.</td>
</tr>
<tr>
<td>34-41</td>
<td>28-35</td>
<td>AN0-AN7</td>
<td>Eight analog inputs to the A/D converter. AN7-AN4 can also be used as a digital input port for falling edge detection.</td>
</tr>
<tr>
<td>42</td>
<td>36</td>
<td>VAREF</td>
<td>Reference voltage for A/D converter. Sets conversion range upper limit.</td>
</tr>
<tr>
<td>Pin No.</td>
<td>Dip/Quip</td>
<td>Flat</td>
<td>Symbol</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>37</td>
<td>AVCC</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>38</td>
<td>RD</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>39</td>
<td>WR</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>40</td>
<td>ALE</td>
</tr>
<tr>
<td>47-54</td>
<td></td>
<td>41-48</td>
<td>PF0-PF7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55-62</td>
<td></td>
<td>49-56</td>
<td>PD0-PD7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>57</td>
<td>STOP</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>58</td>
<td>VCC</td>
</tr>
</tbody>
</table>
uPD78C10/78C11
Block Diagram
Standby Function

The 78C10/78C11 has two standby modes: HALT and STOP.

The HALT mode reduces power consumption to less than 50% of normal operating requirements, while maintaining the contents of on-chip registers, RAM, and control status. The system clock and on-board peripherals continue to operate, but the CPU stops executing instructions. The HALT mode is initiated by executing the HLT instruction. The HALT mode can be released by any non-masked interrupt or by RESET.

The STOP mode reduces power consumption to less than one tenth of 1% of normal operating requirements. There are two types of the STOP mode.

Type A is initiated by executing a STOP instruction. If VCC is maintained within the operating range (2.5 to 6.0 V), on-board RAM and CPU register contents are saved. If VCC is held above 2.0 V (but less than 2.5 V), only on-board RAM is saved. The oscillator is stopped. The STOP mode can be released by an input on NMI or RESET. The user can program oscillator stabilization time via Timer 1. By checking the standby flag (SB), the user can determine whether the processor has been in the Standby mode.

Type B is initiated by inputting a low level on the STOP input. Only RAM contents are saved, not the CPU register contents. The oscillator is stopped. The STOP mode is released by raising STOP to a high level. The oscillator stabilization time is fixed at 65 ms; 65 ms after STOP is raised, instruction execution will begin at location 0. You can increase the stabilization time by holding RESET low for the required time period.
### Standby Conditions

**Parameter**

<table>
<thead>
<tr>
<th></th>
<th><strong>HALT MODE</strong></th>
<th><strong>STOP MODE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>By Instruction</strong></td>
<td></td>
<td><strong>By STOP (Input)</strong></td>
</tr>
<tr>
<td>Oscillator</td>
<td>Run</td>
<td>Stop</td>
</tr>
<tr>
<td>PA, PB, PC</td>
<td>Stop</td>
<td>Stop</td>
</tr>
<tr>
<td>PD Single-chip Mode</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Expand Mode</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>PF Single-chip Mode</td>
<td>HIGH-Z</td>
<td>HIGH-Z</td>
</tr>
<tr>
<td>Address Bus</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Expand Mode</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Port</td>
<td>(Next Addr)</td>
<td>(Next Addr)</td>
</tr>
<tr>
<td>ALE, RD, WR</td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>On-chip RAM</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Register (MM)</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Program Counter (PC)</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Stack Pointer (SP)</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>General Register</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Program Status Word (PSW)</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Timer Upcounter</td>
<td>Run</td>
<td>00H</td>
</tr>
<tr>
<td>Timer Mode Register(TMM)</td>
<td>Hold</td>
<td>*1</td>
</tr>
<tr>
<td>TM0, TM1</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>Interrupt Control Circuit</td>
<td>Run</td>
<td>Stop</td>
</tr>
<tr>
<td>Pending Interrupts (INTFX)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>Interrupt Mask Register</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>NMI (Input)</td>
<td>Active</td>
<td>Active</td>
</tr>
<tr>
<td>INT1, INT2, (Input)</td>
<td>Active</td>
<td>Inactive</td>
</tr>
<tr>
<td>Timer/Event Counter Circuit</td>
<td>Run</td>
<td>Stop</td>
</tr>
<tr>
<td>Mode Register (EOM,ETMM)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>ECNT, ETM0,ETM1</td>
<td>Run</td>
<td>Hold</td>
</tr>
<tr>
<td>Serial Interface Circuit</td>
<td>Run</td>
<td>Stop</td>
</tr>
<tr>
<td>Mode Register (SMH, SML)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>RxB, TxB Serial Registers</td>
<td>Run</td>
<td>Undefined</td>
</tr>
<tr>
<td>A/D Converter Circuit</td>
<td>Run</td>
<td>Stop</td>
</tr>
<tr>
<td>Mode Register (ANM)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>CR0, CR1, CR2, CR3</td>
<td>Active</td>
<td>Undefined</td>
</tr>
<tr>
<td>STANDBY Flag (SB)</td>
<td>Hold</td>
<td>Hold</td>
</tr>
<tr>
<td>STOP Input</td>
<td>Active</td>
<td>Active</td>
</tr>
<tr>
<td>Zero Cross Mode Reg. (ZCM)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>Text Flags (Except SB)</td>
<td>Hold</td>
<td>Undefined</td>
</tr>
<tr>
<td>Reset Input</td>
<td>Active</td>
<td>Active</td>
</tr>
</tbody>
</table>

**Note**  
*1: During Stop Mode : 011000xxB  
After Release : 11111111B
Appendix B

Computer Board Schematics
Appendix C

Software Listings
FESTUS 1.0
13 May 88
Erik J. Heels, '88

Modification History:
13 May 88  Erik J. Heels  First draft written as FESTUS 1.0.

FESTUS 1.0 is written in LightspeedC (version 2.13, copyright THINK
Technologies, Inc., 1986) with the aid of SimpleTools2 (copyright Erik
Kilk 1985, 1987). FESTUS 1.0 supports the control algorithm \( A^*V^*V + B^*V + C \). See my thesis for a complete discussion of this. The
constants can be changed by selecting different values from different
menus. When a change is made, the program sends the appropriate
information one character at a time over the serial port b (modem port).
Information cannot be saved, but the user can draw notes on the screen
which can be printed out using the built in print screen command (apple-
shift-caps lock-4). This version does not involve procedural
abstraction, and the appropriate limits for the flow control have yet to
be determined. Again, see my thesis for a complete discussion of this.
To clear the screen of your scribbles, click the zoom box in the upper
right hand corner of the window. FESTUS 1.0 supports all desk
accessories as well.

The final application, FESTUS 1.0, includes all the following files:

<table>
<thead>
<tr>
<th>program files</th>
<th>header files</th>
<th>libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.c files)</td>
<td>(.h files)</td>
<td>(standard project tools)</td>
</tr>
</tbody>
</table>

FESTUS 1.0.c
SimpleTools2.c
DialogMgr.h
EventMgr.h
MacTypes.h
MemoryMgr.h
MenuMgr.h
PackageManager.h
pascal.h
Quickdraw.h
ResourceMgr.h
simple.h
StdFilePkg.h
stdio.h
strings.h
TextEdit.h
ToolboxUtil.h
WindowMgr.h

After the summer of 1988, I may be reached at the following address:

Erik J. Heels
9 Rocky Knoll Rd.
Cape Elizabeth, ME 04107

Feel free to ask me any questions.

*/
```c
#include "simple.h"  /* SimpleTools header file */
#include <Quickdraw.h> /* Quickdraw header file */
#define LIMIT 10

a0proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", \"\r\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \"C\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \" ");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \"F\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \"F\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \"6\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", \"4\");
    erik = strlen(output_string);
    FWWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);
```
printf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

printf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

printf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}
a25proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    printf(output_string, "%s", " \r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    printf(output_string, "%s", "C");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    printf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)

SysBeep (10);

sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "4");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "0");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "2");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

)

a50proc ()
{  
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "4");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);
sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "4");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

a75proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "4");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

a100proc ()
{
  int out_num = -7;
  int in_num = -6;
  long erik = 0;
  int loop, one =1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "4");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "8");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

b0proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

b10proc ()
{
int out_num = -7;
int in_num = -6;
long erik = 0;
int loop, one = 1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);
```c
sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\n");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "1");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "A");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
```
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

b20proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

    sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

    sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

    sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

    sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

    sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s","6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s"," ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s","3");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s","3");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s","\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

b30proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 3;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s","\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);
sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "4");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "D");
erik = strlen(output_string);
FSSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

b40proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "\%s", "\r");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "\%s", "C");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "\%s", " ");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "\%s", "F");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "\%s", "F");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "\%s", "6");
erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);
sprintf(output_string, "%s", "6");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", \"r\");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

b50proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", \"r\");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

    sprintf(output_string, "%s", "C");
erik = strlen(output_string);
Ffwrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "8");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
ernk = strlen(output_string);
FWrite(out_num, &ernk, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
}

b60proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++);
    SysBeep (10);
```c
b70proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    printf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    printf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    printf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    printf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
```
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "B");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "3");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
b80proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++)
        SysBeep (10);
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

b90proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "E");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
}

b100proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    

int loop, one = 1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
 SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

0proc ()
{
int out_num = -7;
int in_num = -6;
long erik = 0;
int loop,one =1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
F$\text{SWrite}(\text{out_num}, \&\text{erik}, \text{output_string});
for (\text{loop} = 1; \text{loop} < \text{LIMIT}; \text{loop}++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}
c10proc ()
{
    int out_num = -7;
in_num = -6;
long erik = 0;
int loop, one =1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "\%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "0");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "1");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "\%s", "A");
erik = strlen(output_string);
c20proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

    sprintf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num,&erik,output_string);
    for (loop = 1; loop < LIMIT; loop++); SysBeep (10);
```c
sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "3");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "3");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
}

c30proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
```
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "4");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "D");
erik = strlen(output_string);
FSIZEWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);
CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
}
c40proc()
{
int out_num = -7;
in_num = -6;
long erik = 0;
int loop, one = 1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);
sprintf(output_string, "%s", "0");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
    printf(output_string, "%s", " ");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
    sprintf(output_string, "%s", "6");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
    sprintf(output_string, "%s", "6");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
    sprintf(output_string, "%s", "\r");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
CloseDriver(out_num);  
CloseDriver(in_num);  
SysBeep (10);  

}  
c50proc ()  
{  
    int out_num = -7;  
    int in_num = -6;  
    long erik = 0;  
    int loop,one =1;  
    char output_string[200];  
    
    OpenDriver("\p.Aout", &out_num);  
    OpenDriver("\p.Ain", &in_num);  
    
    sprintf(output_string, "%s", "\r");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);  
    for (loop = 1; loop < LIMIT; loop++)  
        SysBeep (10);  
        
    sprintf(output_string, "%s", "C");  
    erik = strlen(output_string);  
    FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "8");  
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);
CloseDriver(out_num);
CloseDriver(in_num);
SysBeep(10);
}
c60proc()
{
int out_num = -7;
int in_num = -6;
long erik = 0;
int loop, one = 1;
char output_string[200];
OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep(10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)

SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "9");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "9");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}

c70proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one =1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "C");
erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", " ");
erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", " "");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "B");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "3");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);
c80proc ()
{
    int out_num = -7;
    int in_num = -6;
    long erik = 0;
    int loop, one = 1;
    char output_string[200];

    OpenDriver("\p.Aout", &out_num);
    OpenDriver("\p.Ain", &in_num);

    sprintf(output_string, "%s", "\r");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "C");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "F");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "6");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", "0");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
    for (loop = 1; loop < LIMIT; loop++)
    SysBeep (10);

    sprintf(output_string, "%s", " ");
    erik = strlen(output_string);
    FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++); SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}
c90proc ()
{
    int out_num = 7;
    int in_num = 6;
    long erik = 0;
    int loop,one =1;
    char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++);
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "E");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num,&erik,output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

}
c100proc ()
{
    int out_num = -7;
    int in_num = -6;
long erik = 0;
int loop, one = 1;
char output_string[200];

OpenDriver("\p.Aout", &out_num);
OpenDriver("\p.Ain", &in_num);

sprintf(output_string, "%s", "\r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "C");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F";
ern = strlen(output_string);
FSWrite(out_num, &ern, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "6");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "0");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", " ");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", "F");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

sprintf(output_string, "%s", \r");
erik = strlen(output_string);
FSWrite(out_num, &erik, output_string);
for (loop = 1; loop < LIMIT; loop++)
SysBeep (10);

CloseDriver(out_num);
CloseDriver(in_num);
SysBeep (10);

inwindow (x, y) /* executed when click in our window */
int x, y;
{

Point m, lm;
MoveTo (x, y); /* draw a Point where the mouse is */
LineTo (x, y);

lm.h = x; lm.v = y;
while (StillDown()) { /* and keep drawing like a pencil */

GetMouse (&m); /* but only when the mouse moves */
if ((m.h != lm.h) || (m.v != lm.v)) {
    LineTo (m.h, m.v);
    lm = m;
}
}

redraw ()
{
}

gotbeep () /* to be executed when beep is picked */
{
    SysBeep (10);
}

no_edit () /* turn off edit menu (on activation) */
{
    menu ("Edit", ",", itemdisable);
}
yes_edit () /* turn on edit menu (on deactivation) */
{
    menu ("Edit", ",", itemenable);
}

aboutme() /* About message */
{
    message ("Written with the aid of SimpleTools ©1986\rBy Erik Kilk,\nFESTUS, a prosthesis controller,\rwritten by Erik J. Heels ('88). Hi\nMom!\r");
}

help_info() /* About message */
{
    message ("The braking power delivered to the knee is proportional to\nA*V*V + B*V + C. Select A, B, \ror C from the Control A, B, or C\menus.\r");
}

setup () /* Setup the menus and windows */
{
    menu (applestring, "About FESTUS...", aboutme);
    /* Default About is disabled */
    menu (applestring, "About FESTUS...", itemenable);
    menu ("File", "Beep", gotbeep);
    menu ("File", "Help", help_info);
    window ("FESTUS 1.0", 20, 50, 490, 325,
        no_edit, yes_edit, redraw, inwindow);
    menu ("Control A", "A = 0%", a0proc);
    menu ("Control A", "A = 25%", a25proc);
    menu ("Control A", "A = 50%", a50proc);
    menu ("Control A", "A = 75%", a75proc);
    menu ("Control A", "A = 100%", a100proc);
    menu ("Control B", "B = 0%", b0proc);
    menu ("Control B", "B = 10%", b10proc);
    menu ("Control B", "B = 20%", b20proc);
    menu ("Control B", "B = 30%", b30proc);
    menu ("Control B", "B = 40%", b40proc);
    menu ("Control B", "B = 50%", b50proc);
    menu ("Control B", "B = 60%", b60proc);
    menu ("Control B", "B = 70%", b70proc);
    menu ("Control B", "B = 80%", b80proc);
    menu ("Control B", "B = 90%", b90proc);
    menu ("Control B", "B = 100%", b100proc);
    menu ("Control C", "C = 0%", c0proc);
    menu ("Control C", "C = 10%", c10proc);
    menu ("Control C", "C = 20%", c20proc);
    menu ("Control C", "C = 30%", c30proc);
    menu ("Control C", "C = 40%", c40proc);
    menu ("Control C", "C = 50%", c50proc);
    menu ("Control C", "C = 60%", c60proc);
    menu ("Control C", "C = 70%", c70proc);
    menu ("Control C", "C = 80%", c80proc);
    menu ("Control C", "C = 90%", c90proc);
menu ("Control C", "C = 100%", c100proc);

simplequits ();
}

main ()
{
    simpletools ("About FESTUS..."); /* Initialize SimpleTools */
    setup (); /* Install our menus and window */
    for (;;) simpleevents (); /* Handle all events */
}
/
End of code.
*/
Title: SimpleTools2.c
Author: Erik Kilk  Copyright 1985, 1987
Dates: June 7, 1985, June 3, 1986, November 8, 1986

SimpleTools is a collection of routines to aid programming
simple "Macintosh looking" programs. SimpleTools initializes
the toolbox, monitors & acts upon events, and provides generic
i/o routines for your application. You initialize your program
by letting SimpleTools know what windows and menus you want along
with what functions SimpleTools should call when they are
selected.

The purpose of SimpleTools is to encourage you to program those
simple programs or to pilot larger programs which you may not
do due to the enormous effort required to use the Macintosh
toolbox. My goal was to study Inside Macintosh once to
Create SimpleTools and then be able to forget most of the usages
of the Toolbox routines. Instead of thumbing through hundreds of
pages of Inside Macintosh just to get something up and running,
one need only remember a dozen generic calls.

SimpleTools is very powerful, yet also very simple to use. One
can get a program up and running with desk accessories, windows,
and menus in only a few minutes. Advance features of SimpleTools
allow you to retrieve enough information from SimpleTools to call
any of the toolbox routines manually if need be.

-----------------------------------------------------------------------------------

You may use, study, copy, and freely distribute SimpleTools if:
1) You mention "Programmed with the aid of SimpleTools
   (c) Erik Kilk 1986" in your About... window of all
   programs distributed free, share, or marketed.
2) You register by sending $20 or more to:
   Erik Kilk
   4949 Snyder Lane, #247
   Rohnert Park, CA  94928
   (707) 794-2424 weekday afternoons
   to encourage me to maintain and improve SimpleTools.

=> For a diskette including the most recent version of SimpleTools,
several detailed examples of using SimpleTools, and a MacWrite
file
describing SimpleTools and its use in more detail, send me a
diskette with enough stamps to mail it back to you. Make sure you
have registered as stated above.

This is 128K, MFS, HFS, old ROM, new ROM, Mac+, & TMON compatible.
This file compiles and executes with Megamax 3.0 beta and
LightSpeed 2.1. Adjust the definition in simple.h for your
compiler. When porting to other compilers, pay particular
attention to where the Lightspeed and Megamax code is specified
since these places are the likely problem areas.
LIGHTSPEED NOTE:

Drag SimpleTools out of the main segment in your project window. You do this by dragging it below the dotted horizontal Line in the project window.

SimpleTools requires the MacTraps library and stringsasm.c or strings.c. If you load the unix library, your project will be larger than needed (unless you need unix for your own program.)

MEGAMAX NOTE:

Use Megamax's convert utility to convert all Mac names to all lower case. If you send me suggestions and/or new code for SimpleTools, please convert back to mixed case first.

SimpleTools provides the following functions for your application to call. Note that your application need not call any Toolbox routines directly. The entire C program (including the standard desk accessory support):

```c
main ()
{
    simpletools ("Skel");
    simplequits ();
    runsimpletools ();
}
```

will execute as a Macintosh program, terminating upon the user selecting Quit. SimpleTools includes:

- simpletools () - init Toolbox and SimpleTools
- simplequits () - add Transfer & Quit menus
- simpleevents () - process next Mac event
- runsimpletools () - continuously process events
- menu ("File","New",new) - install a menu
- window ("My Window",..) - install a window
- run (function) - install a periodic function
- stop (function) - remove a periodic function
- havenewrom () - test for new 128K ROM
- withwindow("My Window") - set output to a window
- stgotoxy (x, y) - set pen to text position x, y
- home () - clear window, set pen to home
- getline (deflt, dest) - with TE editing, get a Line
- prompt (question, dest) - with dialog box & TE, get a Line
- message (string) - with dialog box, print a string
- getfile ("TEXT", name) - with list, select a filename
- putfile (orig, name) - with list, select a filename
A complete Macintosh Style application (including windowing and menus) is given in the following trivial example...

```c
#include <simple.h>

char name[50];

got_beep()
{
    SysBeep (10);
}

got_getname()
{
    char newname[50];
    *newname = 0;
    if (prompt ("What is your name?", newname)) {
        strcpy (name, newname);
        withwindow ("My Window");
        home();
        update();
    }
}

in_content(x, y)
int x, y;
{
    MoveTo (x, y); LineTo (x, y);
}

update()
{
    char outstring[100];
    sgotoxy (1, 5);
    sprintf (outstring,"Hello, %s", name);
#ifndef MEGAMAX
    CtoPstr (outstring);
#endif
    DrawString (outstring);
}

main ()
{
    simpletools ("Sample Program");
    simplequits ();
    menu ("Commands", "Beep", got_beep);
    menu ("Commands", "Get Name", got_getname);
    strcpy (name, "World");
    window ("My Window", 0,0,0,0,0L,0L,update,
           in_content);
    runsimpletools ();
}
ROUTINES YOUR APPLICATION MAY CALL:

simpleevents()

To be called repeatedly by your program's main routine. In most SimpleTools programs, your main routine will initialize and install SimpleTools followed by an loop such as:

    for (;;) simpleevents();

This routine handles all window changes, menu requests, and other Macintosh events. There is also a routine called with runsimpletools() which does not return. It simply performs the above loop. Program exit is accomplished by assigning an exiting routine to a menu, usually this is File/Quit.

simpletools (about_string)
char *about_string;

To be called once at the very beginning of you: main routine. This routine initializes all the Macintosh software managers and installs the basic Apple, File, and Edit menus. The about_string is the name of the menu Item to appear first under the Apple menu.

simplequits()

Installs a simple File/Quit and File/Transfer menu. You only want to installs these if no application dependent processing must be done when the user selects Quit or Transfer.

menu (name, Item, routine)
char *name;
char *Item;
ProcPtr routine;

To be called when a new menu is to be installed on the menu bar or when the characteristics of that menu are to be modified. Name is the name of the menu to appear on the menu bar. Item is the name of the Item to appear under the menu name. Routine is the name of the routine to be executed when the stated menu/Item has been selected. The characteristics of the menu may be changed by passing one of the constants itemdisable, itemenable, itemcheck, or itemuncheck in place of the routine.

window (name, xtop, ytop, xbot, ybot, act, deact, update, content)
char *name;
int xtop, ytop;
int xbot, ybot;
ProcPtr act, deact, update, content;
To be called when a new window is to be installed on the screen or when the characteristics of that window are to be modified. Name is the name of the window. Xtop, ytop, xbot, and ybot are the initial coordinates of the new window. Act is the name of the procedure to execute when the window becomes the top or active window. Deact is the name of the procedure to execute when the window ceases being the top window and deactivates. Update is the name of the procedure called when the Macintosh needs to redraw the window's contents. Content is the name of the procedure called when the mouse is pressed within the window. The procedure specified as content will be passed an x and y integer value representing the local mouse coordinates.

```c
withwindow (name)
char *name;
```

To be called when you want to select which window will receive output and drawings. In most cases, SimpleTools will select the appropriate window before calling your specified act, deact, update, or content procedures. Use this at other times.

```c
run (routine)
ProcPtr routine;
```

To be called when you want a routine to be continuously executed once each time simplesevents() is called. Small, quickly running routines should be used so as not to delay the next event processing. Pseudo multiprocessing with each routine running in its own window can be accomplished by making sure a run routine uses withwindow() to direct its output to the proper window.

```c
stop (routine)
ProcPtr routine;
```

To be called when you want to remove a previously run() routine from the list of routines to run. 50 routines can fit into the run list.

```c
home ()
```

Clears the current window and positions the pen such that any following text will appear in the upper left corner of the window.

```c
stgotoxy (column, row)
int column, row;
```

Positions the pen in the current window so that the next text output will appear in text row and column. This is compatible with some old text only terminals. stgotoxy (1, 1) positions the
pen in the upper left corner. Any negative coordinate leaves that axis of the pen where it currently is.

getline (default, destination)
char *default;
char *destination;

Calling this routine begins a "modal" mode where the user is required to enter a Line of text. This would be similar to using scanf() on "non-Mac", text-only terminals. This routine uses the Macintoshes built in Text-Edit routines allowing the user to edit his Line until <RETURN> is pressed. The flashing bar Cursor is positioned at the current pen location. The resulting string is placed into destination. Default contains the initial value to be displayed on the screen. You may use the null string "" for default.

prompt (question, answer)
char *question;
char *answer;

This routine places a small 3-lined Macintosh Style dialog window prompting the user with question and getting the answer in a boxed Text-Edit area. Two buttons are displayed to terminate the user entry. If Cancel is clicked upon, FALSE is returned. If okay is clicked upon, TRUE is returned. Answer must be set to a default value, "" is okay.

message (message)
char *message;

This routine is similar to prompt except no textual response is asked from the user. This is like an Alert dialog. Just like prompt, TRUE or FALSE is returned depending upon which Button the user presses.

getfile (file_type, reply)
char *file_type;
char *reply;

This routine places the standard Macintosh SFGetFile() window up with a list of files of file_type. Once the user selects a file, the answer is returned in the string reply. Also, and very important for HFS, the working volume/folder is set so that any subsequent open() with reply as the file name will open the correct selected file. The open() should be done before someone has a chance to change the working volume. This routine will return FALSE if the user selects the CANCEL Button.

putfile (origname, reply)
char *origname;
char *reply;
This routine is like getfile, except the standard putfile window is displayed with origname as the default name to save a file as. The actual name selected by the user is returned in reply. As getfile, the working volume/folder is set properly for the next open() call.

THE FOLLOWING IS THE FILE simple.h. YOU SHOULD COPY THIS PORTION INTO A NEW FILE NAMED simple.h SO YOU CAN #include IT INTO YOUR SOURCE FILES.

#define LIGHTSPEED {define either LIGHTSPEED or MEGAMAX or your own}
#include <stdio.h>
#if define MEGAMAX
#include <menu.h>
#include <win.h>
#endif
#if define LIGHTSPEED
#include <MenuMgr.h>
#include <WindowMgr.h>
#endif
#define itemdisable 0L
#define itemenable 1L
#define itemcheck 2L
#define itemuncheck 3L
extern char applestring[];
extern WindowPtr windowpoint();
extern MenuHandle mhand();
extern int windmenu;
extern int dogoaway;
extern int wprocid;
extern int show_new_window;
extern int sizeredraw;
extern int getlinecaps;
extern ProcPtr keydownproc;
extern ProcPtr autokeyproc;
extern void home();
extern void stnop();

Here begins SimpleTools.c

*/
#include "simple.h"           /* define compiler in here */

#ifdef MEGAMAX
  overlay "simpletools"       /* compiler dependent */
#endif

#include <mem.h>
#include <qd.h>
#include <qdvars.h>
#include <misc.h>
#include <event.h>
#include <res.h>
#include <win.h>
#include <dialog.h>
#include <menu.h>
#include <string.h>
#include <stdio.h>
#include <pack.h>
#include <te.h>
#include <toolbox.h>

#define ZZ &
#else

#include <MemoryMgr.h>
#include <Quickdraw.h>
#include <EventManager.h>
#include <ResourceMgr.h>
#include <WindowMgr.h>
#include <TextEdit.h>
#include <DialogMgr.h>
#include <MenuMgr.h>
#include <strings.h>
#include <stdio.h>
#include <PackageMgr.h>
#include <ToolboxUtil.h>
#include <StdFilePkg.h>
#include <pascal.h>

#define ZZ
#endif

#define TRUE (-1)             /* local definitions */
#define FALSE 0
#define maxsrns 50             /* procedure table size */
#define MESSN 30               /* array size for message dialog items */
#define QUESN 40               /* array size for prompt dialog items */
#define ROM85 0x28E             /* new rom stuff */
#define NEWROM 0x7FFF
#define inzoomout 8
#define inzoomin 7
#define zoomproc 8

typedef struct {          /* structure for an Item */
    char   itemname[40];   /* Item number within menu */
    int    itemno;
    int    menuid;        /* menu id */
    MenuHandle menuhand;  /* Item's menu's Handle */
ProcPtr menurun; /* procedure to run */
Ptr next;      /* pointer to the next Item */
}
itemdatum;

typedef struct {  /* structure for a menu */
  char menuname[20];  /* to allow reference by name */
  int menuId;  /* menu id */
  MenuHandle menuhand;  /* menu Handle to reference menu*/
  itemdatum **itemlist;  /* pointer to the list of items */
  ProcPtr next;  /* pointer to the next menu */
} menudatum;

typedef struct {  /* structure for a window */
  char windname[80];  /* window's name and reference */
  WindowPtr wpotr;  /* window's pointer reference */
  ProcPtr wact;  /* the activate procedure */
  ProcPtr wdeact;  /* the deactivate procedure */
  ProcPtr wupdate;  /* the update procedure */
  ProcPtr wcontent;  /* the content procedure */
  ProcPtr next;  /* pointer to the next window */
} windowdata;

#ifdef LIGHTSPEED
  pascal Boolean *TrackBox() = 0xA83B;
  pascal void *ZoomWindow() = 0xA83A;
#endif

WindowPtr windowpoint();

/* Local variables */

memudatum **simplemenus;  /* Handle to menu data */
char accname[80];  /* desk accessory name to open */
Rect dragrect, sizerect;  /* limits for moving windows */
Rect swholescreen;
windowdata **simpleshads;  /* Handle to window data */
int firstwind;  /* if no windows have been made */
int firstruns[maupsws], maxsize];  /* list of procedures to run */
int firstwind;  /* window pointer for debugging */
int snewrrom;
int getlinecaps = FALSE;

/**************************
/*
/* GLOBAL USER MODIFIABLE VARIABLES */
/* These are variables that you can declare extern so that you can use */
/* them to change the SimpleTools defaults */
/**************************
/*
/* wprocid = type of window to Create on next window() call */
typedef Proctype DocumentProc;

int wprocid = documentProc;

int dogoaway = TRUE;

int keydownproc, autokeyproc;

char applestring[2] = {'\024', '\0'};

int windmenu = TRUE;

int sizeredraw = FALSE;

int show_new_window = TRUE;

int messd[MESSN] = {2, 0, 0, 0x38, 0xf1, 0x4c, 0x12d, 0x402, 0x4f4b, 0, 0, 5, 5,
0x36, 0x12d, 0x800, 0, 0, 0x38, 0xac, 0x4c, 0xe8, 0x406,
0x4361, 0xe663, 0x656c};

int quesd[QUESTN] = {3, 0, 0, 0x21, 0xf0, 0x35, 0x12c, 0x402, 0x4f4b, 0, 0, 8, 8,
0x28, 0xe8, 0x600, 0, 0, 0x2b, 0x4b, 0xe8, 0x1000, 0, 0, 8, 0xf0, 0x1c, 0x12c, 0x406, 0x4361, 0xe663, 0x656c};
void stnop()                           /* a no op procedure for defaults */
{
}

char *ptoc(s)
char *s;
{
    #ifndef MEGAMAX
        return (PtoCstr(s));
    #else
        return (s);
    #endif
}

char *ctop(s)
char *s;
{
    #ifndef MEGAMAX
        return (CtoPstr(s));
    #else
        return (s);
    #endif
}

/* Given a menu name, find our data structure for it. Return a Handle
to this structure. This is a local procedure for SimpleTools use. */

/* local procedure */

menudatum **getourmenuhandle (name)
char *name;                       /* name of menu bar menu */
{
    menudatum **here, **temp;     /* hand to menu structure*/
    here = simplemenus;

    /* find the menu name or the end of out menu list */
    HLock (HERE);
    while (strcmp(name,(**here).menuname) && (**here).next )
    {
        temp = here;
        here = (**here).next;
        HUnlock (temp);
        HLock (here);
    }

    /* see if we found it or just the end of the list */
    if (!strcmp(name,(**here).menuname))
    {
        HUnlock (here);
        return (here);
    }
    else
    {
        HUnlock (here);
        return ((menudatum **)OL);
    }

    /* This takes a Handle to our personal Item record and either a
procedure name or a modifier code. If it got a procedure name, it sets it to the Item's procedure to run when the Item is chosen. If it got a modifier code, it changes the state of the menu's Item to checked, unchecked, enabled, or disabled. It especially keeps track of the standard Edit menu items so we can restore them after a desk accessory is finished.

/*/ 

/* Local procedure */

setitems ( items, routine) /* set a menu Item's routine or display */
itemdatum **items; /* if items is neg, then whole menu */
ProcPtr routine;
{
    int inumber;
    MenuHandle mhand;

    /* check to see if a procedure pointer was given to us */
    if ( (((long)items)>OL) && (routine > (ProcPtr)0x1000L) ) {
/* good procedure value */
        (**items).menurun = routine;
        return;
    }

    /* Calculate which Item number we are going to modify */
    if ( (long)items < OL ) {  /* the whole menu */
        mhand = (MenuHandle) (OL - (long)items);
        inumber = 0;
    } else {  /* just one Item */
        mhand = (**items).menuhand;
        inumber = (**items).itemno;
    }

    /* If a NULL procedure pointer, then set to a no_op routine */
    if ( (inumber > 0) && (**items).menurun == (ProcPtr)0L )
        (**items).menurun = (ProcPtr) stnop;

    /* Now change the state of a menu Item */
    switch ((int)routine) {
        case itemdisable:
            DisableItem(mhand,inumber); break;
        case itemenable:
            EnableItem(mhand, inumber); break;
        case itemcheck:
            CheckItem(mhand, inumber, TRUE); break;
        case itemuncheck:
            CheckItem(mhand, inumber, FALSE); break;
    }
    if (inumber == 0) DrawMenuBar(); /* if main menu was changed */
}

/* This routine is called by the simpletools() initial routine. It gets the pointer list of menus started, loads the desk accessories into the Apple menu, and loads up some standard menu entries. The reason menu File has a New entry, and none others, is because as this code
currently stands, a menu must have at least one Item. And since we
want File before Edit, I had to make an entry. The most commonly
used
Item under File is Quit. But we like quit to be at the end of the
list.
So, since New is usually always first when it is used, that the one
chosen to start File.
*/

/* Local procedure */

initsmenus about) /* init simpletools' menus */
char *about;
{
    itemdatum **items;

    simplemenus = (itemdatum **) NewHandle ( (long)sizeof(itemdatum));
    HLock (simplemenus);

    strcpy ( (**simplemenus).menuname, applestring);
    (**simplemenus).menuItem = 1;
    (**simplemenus).next = (Ptr) 0L;
    ctop (**simplemenus).menuhand = NewMenu (1, (**simplemenus).menuname);
    ptoc (**simplemenus).menuname);
    HUnlock (**simplemenus).menuhand);

    (**simplemenus).itemlist = (itemdatum **) NewHandle ( (long)sizeof(itemdatum));
    items = (itemdatum **) (**simplemenus).itemlist;
    HLock (items);

    strcpy (**items).itemname, about);
    (**items).itemno = 1;
    (**items).menuItem = 1;
    (**items).menuhand = (**simplemenus).menuhand;
    (**items).menurun = (ProcPtr) stnop;
    (**items).next = 0L;
    HUnlock (items);

    ctop (about);
    AppendMenu (**simplemenus).menuhand, about);
    ptoc (about);
    DisableItem (**simplemenus).menuhand, 1);
    menu (applestring, "-", (ProcPtr) itemdisable);
    #ifdef MEGAMAX
    AddResMenu (**simplemenus).menuhand, "DRVR");
    #else
    AddResMenu (**simplemenus).menuhand, 'DRVR');
    #endif
    InsertMenu (**simplemenus).menuhand, 0);
    HUnlock (simplemenus);

    menu ("File", "New", (ProcPtr)itemdisable);
    menu ("Edit", "Undo", stnop);
    menu ("Edit", "-", (ProcPtr)itemdisable);
    menu ("Edit", "Cut/X", stnop);
menu ("Edit", "Copy/C", stnop);
menu ("Edit", "Paste/V", stnop);
menu ("Edit", "Clear", stnop);

} /* Local procedure */

#ifndef LIGHTSPEED

gottrans ()
{
    char prog[80];
    char *argv[3];
    if ( getfile("APPL", prog) ) {
        argv[1] = NULL;
        execv (prog, argv);
    }
}
#endif

/* Local procedure */

gotquit ()
{
    ExitToShell();
}

/* This routine is for the Windows menu Item. The Windows menu is set up when new windows are added. It is used to bring forward and bring into view windows that may be under other windows or have been sent hiding by a click on their close box. */

/* Local procedure */

showwindow(name)     /* show the named window */
char *name;
{
    WindowPtr foo;
    foo = windowpoint(name);     /* get its window pointer */
    if ( foo ) {
        ShowWindow(foo);        /* show it on the screen */
        SetPort (foo);          /* set further output to it */
        if ( foo != FrontWindow() )  /* if it isn't active, */
            SelectWindow (foo);  /* activate it */
    }
}

/* Local procedure */

winddatum **wdatum(windpt)     /* return Handle to window data */
WindowPtr windpt;
{
    winddatum **wind, **temp;

    if (firstwind) return ((winddatum **) OL);
    wind = simplewinds;
    HLock (wind);
while ( (**wind).wptr != windpt) && (**wind).next) {
    temp = wind;
    wind = (winddatum **) (**wind).next;
    HUnlock (temp);
    HLock (wind);
}

if ( (**wind).wptr == windpt) {
    HUnlock (wind);
    return (wind);
} else {
    HUnlock (wind);
    return ((winddatum **) 0L); /* zero if not found */
}

/* Local procedure */

runruns(event) /* run all the installed run procedures */
EventRecord *event; /* returns number of routines run */
{
    int i=0;
    WindowPtr saveport;
    GetPort (&saveport);
    while ( simpleruns[i] )
    { *(simpleruns[i++]) (event);
      SetPort (saveport);
      return(i);
    }

/* Local procedure */

stdialog( question, answer, type) /* a general dialog displayer */
char *question;
char *answer;
int type; /* type: 1=prompt, 2=message */
{
    DialogPtr dialog; /* dialog reference */
    Handle Item, items; /* handles for the dialog items */
    Rect screen, box; /* rectangles for dialog/items */
    int dtype, hit, canc; /* Item type and which was hit */
    char tempanswer[255]; /* address where answer is */

    items = NewHandle (512L); /* get memory for items list */
    HLock (items); /* lock it down */
    if (type == 1)
        BlockMove (quesd, *items, (long) QUESN * 2L);
    else
        BlockMove (messd, *items, (long) MESSN * 2L);
    SetRect (&screen, 103, 50, 409, 137);

    /* For LIGHTSPEED, use a lower case d and upper case B and P */
dialog = NewDialog (OL, &screen, ",", 0, dBoxProc, -1L, 0, OL, items);
GetDItem (dialog, 2, &dtype, &Item, &box);
cTop (question);
SetIText (Item, question); /* set Item#2 text */
ptoc (question);
if (type == 1) { /* set default answer */
    GetDItem (dialog, 3, &dtype, &Item, &box);
cTop (answer);
    SetIText (Item, answer);
    ptoc (answer);
canc = 4;
} else
canc = 3;
ShowWindow (dialog); /* display the dialog */
do {
    ModalDialog (OL, &hit); /* process the dialog */
} while ((hit != 1) & (hit != canc));
if (type == 1) {
    GetDItem (dialog, 3, &dtype, &Item, &box);
    HLock (Item);
    GetIText (Item, tempanswer); /* get Item#3 text */
    ptoc (tempanswer);
    strcpy (answer, tempanswer); /* make a copy of it */
    HUnlock (Item);
}
HUnlock(items); /* unlock items memory */
HPurge(items); /* purge it */
DisposDialog (dialog); /* get rid of dialog */
return (hit==1); /* return true if ok */

/* Local procedures */

docommand (which, thisevent)
long which;
EventRecord *thisevent;
{
    int themenu, theitem;
    long size;
    char *cpoint;
    GrafPtr tempport;
    menudatum **here, **temp;
    itemdatum **items, **tempitems;
    char **myrshandle;
    Handle myhandle;

    themenu = HiWord (which);
    theitem = LoWord (which);
    if ((themenu == 1) && (theitem != 1)) {
        /* start up a desk accessory */
        HLock (simplemenus);
        GetItem (**simplemenus).menuhand, theitem, accname);
SetResLoad (FALSE);
#endif MEGAMAX
  myreshandle = GetNamedResource ("DRVR", accname);
#else
  myreshandle = GetNamedResource ("DRVR", accname);
#endif
SetResLoad (TRUE);
size = SizeResource (myreshandle);
myhandle = NewHandle (size + 3072L);
if (myhandle == 0L)
  message ("Not enough memory to do that.");
else {
  DisposeHandle (myhandle);
  GetPort (&tempport);
  OpenDeskAcc(accname);
  SetPort (tempport);
}
HUnlock (simplemenus);
return;
}
if (themenu ==3) {
  /* do any system edits */
  if (SystemEdit(theitem -1)) return;
}

/* now we run an installed menu procedure */
here = simplemenus;
HLock (here);

/* find out menu structure given the menu id */
while ( (**here).menuId != themenu) &**here).next) {
  temp = here;
  here = (menudatum **) (**here).next;
  HUnlock (temp);
  HLock (here);
}
if (**here).menuId == themenu) {
  /* now find the Item structure */
  items = (**here).itemlist;
  HUnlock (here);
  HLock (items);

  while ( (**items).itemno != theitem) &**items).next) {
    tempitems = items;
    items = (itemdatum **) (**items).next;
    HUnlock (tempitems);
    HLock (items);
  }

  /* prepare to give the Item name to the procedure */
  cpoint = (**items).itemname;
  if (**items).itemno == theitem
  /* if we found the Item, call its procedure */
    (**items).menurun) (cpoint, thisevent);
  HUnlock (items);
/* Local procedure */

domousedown(thisevent) /* respond to mouse down events */
EventRecord *thisevent; /* passed the event record */
{
    WindowPtr whichwindow;
    int code, x, y;
    char *cpoint;
    menudatum **omhand;
    windatum **thewdatum;
    long newplace;
    Point temp;
    GrafPtr saveport;

code = FindWindow(ZZ(thisevent->where), &whichwindow);
switch (code) {
    case inMenuBar:
        docommand(MenuSelect(ZZ(thisevent->where)), thisevent);
        break;
    case inSysWindow:
        SystemClick(thisevent, whichwindow); break;
    case inDrag:
        DragWindow(whichwindow, ZZ(thisevent->where),
            &dragrect); break;
    case inGrow:
        newplace= GrowWindow(whichwindow, ZZ(thisevent->where),
            &sizerect);
        SizeWindow(whichwindow, LoWord(newplace),
            HiWord(newplace), TRUE);
        if (sizeredraw) {
            GetPort (&saveport);
            SetPort (whichwindow);
            EraseRect (&swholescreen);
            InvalRect (&swholescreen);
            SetPort (saveport);
        }
        break;
    case inGoAway:
        if ( TrackGoAway(whichwindow, ZZ(thisevent->where))) {
            HideWindow (whichwindow);
        }
        break;
    case inzoomout:
    case inzoomin:
#ifdef MEGAMAX
        if ( trackbox(whichwindow, ZZ(thisevent->where), code)) {
            zoomwindow (whichwindow, code, 0);
        }
#else
        if ( TrackBox(whichwindow, ZZ(thisevent->where), code)) {
            ZoomWindow (whichwindow, code, 0);
#endif
        GetPort (&saveport);
        SetPort (whichwindow);
        EraseRect (&swholescreen);
InvalRect (&wholeescreen);
SetPort (saveport);
}
break;
case inContent:

/* make the window active if it isn't yet */
if (whichwindow != FrontWindow()) {
    SelectWindow (whichwindow);
}

/* find our window data */
thewdatum = wdatum (whichwindow);
if (thewdatum) {

    /* convert the Point of click to the window's own coordinates since this will be always more useful than the global coordinates */
    temp = thisevent->where;
    SetPort (whichwindow);
    GlobalToLocal (&temp);
    #ifdef MEGAMAX
        x = temp.a.h;
        y = temp.a.v;
    #else
        x = temp.h;
        y = temp.v;
    #endif

    /* call the window's in content routine */
    HLock (thewdatum);
    (**(**thewdatum).wcontent)) (x, y, &ichwindow, thisevent);
    HUnlock (thewdatum);
}
break;

.parseFloat

 /***************************************************************************/
 /*
 /* GLOBAL ROUTINES INTENDED TO BE USER CALLABLE PROCEDURES */
 /* THE FOLLOWING PROCEDURES HAVE BEEN WRITTEN FOR THE USER'S */
 /* APPLICATION TO CALL. */
 /* */
 /***************************************************************************/

havemewrom () /* returns true if new roms installed */
{
    return (**((int *)ROM85)) == NEWROM);
}
Menu is usually called like:

```c
    menu ("File", "Print...", got_print)
```

where the first argument is the name appearing on the menubar. The 2nd argument is the name appearing when the menu is pulled down. The 3rd argument is USUALLY the routine to be called when the user selects this particular menu. Non-existent menus are created following the last. The menu ordering may never be changed once created. Existent menus have their "routine-to-be-executed" assignment changed to the new routine. If the long values 0L, 1L, 2L, or 3L are passed instead of a procedure, the menu characteristic is set as specified by the constants itemdisable, itemenable, itemcheck, itemuncheck.

For example:

```c
    menu ("File", "Print...", itemdisable)
```

PROCEDURES ASSIGNED TO MENUS ARE CALLED WITH TWO ARGUMENTS. YOU DO NOT NEED TO DECLARE THESE IN YOUR PROCEDURE IF YOU DO NOT USE THEM. FOR EXAMPLE, GOT_PRINT MAY BE DECLARED AS:

```c
    got_print()
    char *itemname;
    EventRecord *current_event;
    {
        ...
    }
```

Itemname is a char* pointing to the Item name. This allows the same menu procedure to be used for multiple menu/Item pairs. Maybe your Size menu just as items 9 Point, 10 Point, 12 Point, etc. This way you can specify the same procedure for each and determine what to do by looking at itemname.

Current_event is a pointer to the current EventRecord that detected the menu selection. You may look at this as needed.

```
    menu (name, Item, routine) /* install or change a menu */
    char *name; /* the menu name */
    char *Item; /* the Item name */
    ProcPtr routine; /* a procedure or modifier */
    {
        menudatum **here,**temp; /* a roving Handle to our data */
        menudatum **ourmhandle; /* another Handle to our data */
        itemdatum **items,**tempitems; /* a Handle to the Item */
        int lastid, lastitem;

        /* get the Handle to menu named 'name' */
        if ((ourmhandle = getourmenuhandle (name)) == 0L) {
            /* make a new menu entry by finding the end of the list */
            here = simplemenus;
        }
    }
```
HLock (here);
while (**here).next) {
    temp = here;
    here = (mendatum **) (**here).next;
    HUnlock (temp);
    HLock (here);
}

/ * make a structure for our new entry */
lastid = (**here).menuId;
(**here).next = (Ptr) NewHandle ( (long) sizeof(mendatum));
temp = here;
here = (mendatum **) (**here).next;
HUnlock (temp);
HLock (here);

strcpy ( (**here).menuname, name);
(**here).menuId = ++lastid;
(**here).next = (Ptr) 0L;

/ * make a new Item structure */
(**here).itemlist = (itemdatum **) NewHandle ( 
(long) sizeof(itemdatum));

/ * make a new menu entry for the Macintosh */
cтоп (name);
(**here).menuhand = NewMenu (lastid, name);
ptoc (name);
items = (**here).itemlist;

HLock (items);
strcpy (**items).itemname, Item);
(**items).itemno = 1;
(**items).menuId = lastid;
(**items).menuhand = (**here).menuhand;
(**items).menurun = (ProcPtr) 0L;
(**items).next = 0L;
HUnlock (items);

/ * install and display the menu */
cтоп (Item);
AppendMenu (**here).menuhand, Item);
ptoc (Item);
InsertMenu (**here).menuhand,0);
HUnlock (here);

setitems (items, routine);
DrawMenuBar();
return(TRUE);
}
else {
    HLock (ourmhandle);

    if (strlen(Item) == 0) {
        / * then adjust main menu */
        setitems( 0L - (long) (**ourmhandle).menuhand), routine);
        return(FALSE);
*/ see if Item is in list */
items = (**ourmhandle).itemlist;
HLock (items);

while ( strcmp(Item,(**items).itemname) && (**items).next) {
tempitems = items;
items = (itemdatum **)(**items).next;
HUnlock (tempitems);
HLock (items);
}

if (strcmp(Item,(**items).itemname) ==0) {
setitems( items, routine);
return(FALSE);
}

else {
	/* make new Item entry */
	lastitem = (**items).itemno;
	(**items).next = *(Ptr)NewHandle((long)sizeof(itemdatum));

tempitems = items;
items = (itemdatum **)(**items).next;
HUnlock (tempitems);
HLock (items);

strcpy (**items).itemname, Item);
(**items).itemno = ++lastitem;
(**items).menuId = (**ourmhandle).menuId;
(**items).menuhand = (**ourmhandle).menuhand;
(**items).menurun = (ProcPtr) 0L;
(**items).next = 0L;
HUnlock (items);

	/* and install the Item in the menu bar */
ctop (Item);
AppendMenu (**ourmhandle).menuhand, Item);
ptoc (Item);
HUnlock (ourmhandle);
setitems (items, routine);
return(TRUE);
}

/* Given a menu name, return the real menu Handle as used by most
of the Macintosh toolbox menu manager routines.
*/

MenuHandle mhand (name) /* find MenuHandle */
char *name; /* given name of menu */
{

  menudatum **menu;
MenuHandle temp; /* a Handle to our data */

menu = getourmenuhandle(name);
if ( menu ) {

HLock (menu);
    temp = (**menu).menuhand;
HUnlock (menu);
    return ( temp ); /* return menu Handle */
} else
    return ( (MenuHandle) 0 );
}

/* Call this routine if you want these SimpleTools defined quitting */
/* procedures. You may just install your own instead. The time to */
/* call this is after you have installed all your other "File" items. */
/* By calling this last, you will place Transfer and Quit on the end */
/* of the menu list. */
simplequits ()
{
    menu ("File", "-", itemdisable);
    #ifndef LIGHTSPEED
    menu ("File", "Transfer.../T", gottrans);
    #endif
    menu ("File", "Quit/Q", gotquit);
}

/* Given a window's name, return its window pointer so that other */
/* Macintosh Window Manager routines can be called for that window. */
WindowPtr windowpoint (name) /* get window pointer */
    char *name; /* given window's name */
{
    winddatum **wind, **tempwind; /* Handle to our window data */
    WindowPtr temp;

    if (firstwind) return ((WindowPtr)0);
    wind = simplewinds; /* look for the named window */
    HLock (wind);

    while ( strcmp (**wind).windname, name ) && (**wind).next) { /*
        tempwind = wind;
        wind = (winddatum **) (**wind).next;
        HUnlock (tempwind);
        HLock (wind);
    }

    if ( strcmp (**wind).windname, name ) ==0) {
        temp = (**wind).wptr;
        HUnlock (wind);
        return ( temp ); /* return pointer */
    } else {
        HUnlock (wind);
        return ( (WindowPtr) 0 );/* or zero if it wasn't found */
    }
}
/ This routine installs a new window onto the screen. It also gives
that window an Item in the Window menu. This routine is also used
to modify a window's associated routines. The x,y positions are the
top left and bottom right corners of where the window should
originally
be placed. The coordinates are never used when this routine is
called
to update an already existing window. But the spaces must be filled,
so you can use zeros if you want. Once the window has been displayed
in
its original position, the user has complete control of its size and
placement with the mouse.

YOU MUST ASSIGN PROCEDURES TO BE CALLED WHEN SIMPLETELSS DETECTS THAT
THIS WINDOW IS BECOMING ACTIVE, DEACTIVATING, NEEDS UPDATING, OR
THE MOUSE HAS BEEN PRESSED IN ITS CONTENT. JUST LIKE THE MENU
PROCEDURE,
THESE PROCEDURES ARE PASSED SOME ARGUMENTS. YOU DO NOT HAVE TO
DECLARE THESE IF YOU DON'T WHAT TO USE THEM. IF YOU USE THE
ARGUMENTS,
YOU WOULD DECLARE THESE PROCEDURES AS FOLLOWS:

my_activate (windp, event)    same as my_update
my_deactivate (windp, event)  same as my_update

my_update (windp, event)
WindowPtr windp;
EventRecord *event;
{
   ...
}

my_inContent (x, y, windp, event)
int x, y;    mouse position in local coords
WindowPtr windp;  like above
EventRecord *event;  like above
{
   ...
}

window(name, xtop, ytop, xbot, ybot, a, d, u, c)
char *name;    /* window's name */
int xtop, ytop, xbot, ybot;    /* position if this is a new window */
ProcPtr a, d, u, c;    /* activate, deactivate, update, and */
{                            /* content procedures */
   winddatum **wind, **temp;/* Handle to our window data */
   winddatum **newentry;    /* another Handle */
   Rect newplace;            /* rectangle for the window's placement */

   if (a == (ProcPtr) 0)
a = (ProcPtr) stnop;
if (d == (ProcPtr) 0)
    d = (ProcPtr) stnop;
if (u == (ProcPtr) 0)
    u = (ProcPtr) stnop;
if (c == (ProcPtr) 0)
    c = (ProcPtr) stnop;
if ( !firstwind ) {
    /* see if window is in the list */
    wind = simplewins;
    HLock (wind);

    while ( strcmp (**wind).windname, name) && (**wind).next) {
        temp = wind;
        wind = (winddatum **) (**wind).next;
        HUnlock (temp);
        HLock (wind);
    }

    if ( strcmp (**wind).windname, name) ==0) {
        /* reset the found window's parameters */
        (**wind).wact = (ProcPtr) a;
        (**wind).wdeact = (ProcPtr) d;
        (**wind).wupdate = (ProcPtr) u;
        (**wind).wcontent = (ProcPtr) c;
        SetPort ( (**wind).wptra); 
        HUnlock (wind);

        return(FALSE);
    }
    HUnlock (wind);
}

/* make a new window entry */
newentry = (winddatum **)NewHandle ( (long) sizeof (winddatum));
if (firstwind)
    simplewins = newentry;
else
    (**wind).next = (Ptr) newentry;
firstwind = 0;
HLock (newentry);

strcpy (**newentry).windname, name);
SetRect (&newplace, xtop, ytop, xbot, ybot);
if (EmptyRect (&newplace))
    SetRect (&newplace, 10, 42, 500, 330);
cop (name);
(**newentry).wptra = NewWindow (0L, &newplace, name,
show_new_window,
    wprocid, -1L, dogoaway, newentry);
pctc (name);
(**newentry).wact = (ProcPtr) a;
(**newentry).wdeact = (ProcPtr) d;
(**newentry).wupdate = (ProcPtr) u;
(**newentry).wcontent = (ProcPtr) c;
(**newentry).next = (Ptr) 0;
if (windmenu)
    menu ("Windows", name, showawindow);
SetPort ( (**newentry).wptr);
HUnlock (newentry);

return(TRUE);
}

withwindow(name) /* set output to window by name */
char *name; /* give it the window's name */
{ /* returns if window exists */
    winndatum **wind, **temp;
    wind = simplewinds;
    if (firstwind) return(FALSE); /* search for the window's name */
    HLock (wind);
    while ( strcmp (**wind).windname, name) && (**wind).next) {
        temp = wind;
        wind = (winndatum **) (**wind).next;
        HUnlock (temp);
        HLock (wind);
    }
    if ( strcmp (**wind).windname, name) ==0) {
        SetPort ( (**wind).wptr); /* set output to it */
        HUnlock (wind);
        return(TRUE);
    } else {
        HUnlock (wind);
        return(FALSE);
    }
}

/* This run procedure is used to install routines to be occasionally
run. The routine will be run once for each call to simpleevents() which
is done repeatedly by runsimpletools().

EACH ROUTINE INSERTED INTO THE RUN LIST IS RUN MULTIPLE TIMES UNTIL
IT IS REMOVED BY CALLING STOP. THE ROUTINE IS CALLED WITH A SINGLE
ARGUMENT, A POINTER TO THE EVENT JUST RETURNED BY GETNEXTEVENT() AND
BEFORE SIMPLETOOLS PROCESSES IT.
*/

run(routine) /* install a run procedure */
ProcPtr routine; /* give it the procedure */
{ /* return TRUE if successful */
    int i;
    i = 0; /* add it to the end of the list */
    while ( simpleruns[i] != (ProcPtr) 0L) i++;
    if (i < maxsruns) {
        simpleruns[i] = routine;
        simpleruns[i+1] = (ProcPtr) 0L;
        return(TRUE);
    } else
        return(FALSE);
}

/* This routine removes a procedure from the list of run procedures */
stop(routine)        /* stop a procedure from running*/
ProcPtr routine;    /* give the procedure */
{ /* return TRUE if successful */
    int i = 0;
    while ((simpleruns[i] != routine) && simpleruns[i]) i++;
    if (simpleruns[i]) {
        while (simpleruns[i] != (ProcPtr)0) {
            simpleruns[i] = simpleruns[i+1];
            i++;
        }
        return(TRUE);
    } else {
        return(FALSE);
    }
}

void home ()        /* text-based home of the pen with
                   the window being erased. */
{
    GrafPtr port;
    GetPort (&port);
    EraseRect (&(port->portRect));
    stgotoxy (1, 1);
}

stgotoxy (x, y)     /* goto text position x, y */
int x, y;
{
    Point pt;
    int newx, newy;
    FontInfo font;

    GetFontInfo (&font);
    GetPen (&pt);
    #ifdef MEGAMAX
    if (x < 0)
        newx = pt.a.h;
    else
        newx = font.widMax * (x);
    if (y < 0)
        newy = pt.a.v;
    #else
    if (x < 0)
        newx = pt.h;
    else
        newx = font.widMax * (x);
    if (y < 0)
        newy = pt.v;
    #endif
    else
        newy = (font.ascent + font.descent + font.leading) * (y+1);
    MoveTo (newx, newy);
}

/* The getline procedure is to be called when you want to simply get a
Line
of text from the user at the current pen position on the screen. You will probably proceed this with a call to stgotoxy(x,y). You would call it like:

```c
getline ("Erik", name);
```

where name is a character array. This works MUCH better than scanf() or gets() since it uses the Macintosh TextEdit routines to allow the user to edit the Line being input.

Getline is very "modal" and no other events are handled while the user is expected to enter the Line. Getline returns ONLY when the user presses <RETURN>.

Routines scheduled to run by the run() routine are called. Make sure your run routines don't strip all <RETURNS> from the event record they get or getline will never stop.

```c

getline (default, destination) /* using TE, get a Line */
char *default, *destination; /* default string, dest */
{
    TEHandle hte;
    Rect destRect;
    Point pen,pt;
    FontInfo FInfo;
    int done, mask, code, in_already, nextcap;
    GrafPtr port, window;
    EventRecord event;
    char key;
    CursHandle c;

    GetPort (&port);    /* Calculate Rect for TE */
    if (port != FrontWindow()) SelectWindow (port);
    GetPen (&pen);
    GetFontInfo (&FInfo);
    #ifdef MEGAMAX
        SetRect (&destRect, pen.a.h, pen.a.v - FInfo.ascent, 1000, pen.a.v + FInfo.descent);
    #else
        SetRect (&destRect, pen.h - 1, pen.v - FInfo.ascent, 1000, pen.v + FInfo.descent);
    #endif
    EraseRect (&destRect);
    hte = TENew (&destRect, &destRect);
    TESetText (default, (long)strlen(default), hte);
    TEActivate (hte);
    TEOUpdate (&destRect, hte);
    mask = mDownMask + keyDownMask + autoKeyMask + mUpMask;
    done = FALSE;
    #ifdef MEGAMAX
        c = GetCursor (ibeamcursor);
    #else
        c = GetCursor (ibeamCursor);
    #endif
    in_already = FALSE;
    nextcap = getlinecaps;
```
do { /* "modal" loop until <cr> */
    SystemTask();
    TEIdle (hte);
    GetNextEvent (mask, &event);
    runruns (&event);
    GetMouse (&pt); /* use I beam in TE */
    if (PtInRect (ZZ(pt), &destRect)) {
        if (! in_already ) {
            SetCursor (*c);
            in_already = TRUE;
        }
    } else {
        if (in_already ) {
            InitCursor ();
            in_already = FALSE;
        }
    }
    switch (event.what) {
    case mouseDown:
        code = FindWindow (ZZ(event.where), &window);
        if ((code == inContent) && (window == port)) {
            GlobalToLocal (event.where);
            if (PtInRect (ZZ(event.where), &destRect))
                TEClick (ZZ(event.where), 0, hte);
            else SysBeep (1);
        } else SysBeep (20);
        break;
    case keyDown:
    case autoKey:
        key = (char) (event.message & 0xFFL);
        if (nextcap && (key >= 'a') && (key <= 'z'))
            key = ' ';
        nextcap = FALSE;
        if (key == ' ') nextcap = getlinecaps;
        if (key != '\r') TEKey (key, hte);
        else done = TRUE;
        break;
    }
} while (!done);
TDEactivate (hte);

/* For LIGHTSPEED, use a lowercase te and upper case L in */
/* TElength. Megamax conversion utility does this wrong too. */

strncpy (destination, *TEGetText(hte), (*hte)->teLength);
destination[(*hte)->teLength] = 0;

TDEdispose (hte);
InitCursor ();

/* Use prompt when you want a tiny window to pop up to ask the user */
/* a question. The question is drawn and a TextEdit box is provided */
/* to get the answer. Whatever the user leaves in the answer box */
/* is returned in answer. Two buttons are also displayed: OK and */
CANCEL. Prompt returns TRUE or FALSE depending on which Button was pressed.

/**
prompt ( question, answer) /* dialog box question/answer */
char *question;
char *answer;
{
    return (stdlib (question, answer, 1));
}

/** Message is just like prompt except no answer box is displayed. An
OK and CANCEL Button works just like prompt. */

message ( message ) /* dialog box message */
char *message;
{
    return (stdlib (message, message, 2));
}

/** This routine is a simpler way to call the toolbox SFGetFile() routine. Simple call this like:

    getfile ("TEXT", filename)

where filename is a character array. Replace TEXT with whatever
file type you desire. The file manager's working directory is set
correctly so that a subsequent open() call with filename will work. */

getfile (ftype, reply)
char ftype[];
char reply[];
{
    Point where;
    SFReply frommac;

    #ifdef MEGAMAX
        where.a.h = 75; where.a.v = 50;
    #else
        where.h = 75; where.v = 50;
    #endif
    if (strlen(ftype) != 4)
        SFGetFile (ZZ(where), NULL, NULL, -1, NULL, NULL, &frommac);
    else
        SFGetFile (ZZ(where), NULL, NULL, 1, ftype, 0L, &frommac);
    if (frommac.good) {
        SetVol ("", frommac.vRefNum);
        strcpy (reply, frommac.fName);
        return (TRUE);
    }
    else return (FALSE);
}

/** This is like getfile, but may get a new file name from the user.
Origname is the default you want to present to the user. */
putfile (origname, reply)
char *origname;
SFReply *reply;
{
    Point where;
    SFReply frommac;

    #ifdef MEGAMAX
        where.a.h = 75; where.a.v = 50;
    #else
        where.h = 75; where.v = 50;
    #endif
    SFPutFile (ZZ(where), ",", origname, 0L, &frommac);
    if (frommac.good) {
        SetVol ("", frommac.vRefNum);
        strcpy (reply, frommac.fName);
        return (TRUE);
    }
    return (FALSE);
}

/* This routine initializes SimpleTools and MUST be called before */
/* most of the other SimpleTools routines are called. */
/* The passed about string is the menu Item name to appear just under */
/* the Apple menu. This will be disabled and can be enabled using */
/* a menu() call. This routine also initializes the Macintosh */
/* for application execution and desk accessory processing. */
simpletools(about)    /* to be called at the beginning of program */
char *about;
{
    #ifdef MEGAMAX
        maxapplzone();    /* allow maximum heap expansion */
    #else
        MaxApplZone();
    #endif
    FlushEvents (everyEvent,0);    /* ignore left over events */
    InitGraf (&thePort);    /* initialize the screen */
    InitFonts();
    InitWindows();
    InitMenus();
    InitCursor();        /* make the arrow Cursor */
    TEInit();
    InitDialogs(gotquit);
    snewrom = havenewrom();
    /* Ugh. For LightSpeed use a lower case d in DocumentProc. */
    /* Megamax conversion utility is at fault here. */
    wprocid = documentProc;
    if (snewrom) wprocid = zoomproc;
    SetRect ( &sizerect, 20, 50, 250, 330);
    simpleruns[0] = (ProcPtr) 0;    /* empty the run list */
    /* These are the bounds we are allowed to size a window or
Move a window to. */
swholescreen = dragrect = thePort -> portRect;
InsetRect (&dragrect, 4, 4);
SetRect (&sizerect, 20, 20, 2048, 700);
firstwind = 1;                    /* empty window table     */
keydownproc = (ProcPtr) stnop;    /* default key hit procedures */
autokeyproc = (ProcPtr) stnop;
initmenus(about);                /* install the menus */
}
simpleevents()                     /* to be called in the main loop */
{
    EventRecord newevent;
    winddatum **thewdatum;
    SystemTask();                        /* Do the system D.A. etc. stuff */
    HILiteMenu(0);
    GetNextEvent(everyEvent, &newevent);
    runruns(&newevent);                   /* Do our run procedures */
    switch (newevent.what) {
        case mouseDown:
            domousedown(&newevent); break;
        case keyDown:
            if (newevent.modifiers & cmdKey)
                docommand(MenuKey((char)(newevent.message & 0xffL)),
                        &newevent);
        /*(keydownproc))(&newevent);
            break;
        case autoKey:
            if (newevent.modifiers & cmdKey)
                docommand(MenuKey((char)(newevent.message & 0xffL)),
                        &newevent);
        /*(autokeyproc))(&newevent);
            break;
        case activateEvt:
            thewdatum = wdatum(newevent.message);
            if (thewdatum) {
                SetPort(newevent.message);
                HLock (thewdatum);
                if (newevent.modifiers & 1) {
                    *((**thewdatum).wact)) (newevent.message,
                    &newevent);
                } else {
                    *((**thewdatum).wdeact)) (newevent.message,
                    &newevent);}
                HUnlock (thewdatum);}
        break;
        case updateEvt:
            thewdatum = wdatum(newevent.message);
            if (thewdatum) {
                SetPort (newevent.message);
                BeginUpdate (newevent.message);
                HLock (thewdatum);
                *((**thewdatum).wupdate)) (newevent.message,
                &newevent);
                HUnlock (thewdatum);
                EndUpdate (newevent.message);}
            break;
    }
    runsimpletools ()
    {for (;;) simpleevents();}
ABOVE THE KNEE PROSTHESIS PROGRAM WHICH ACCEPTS
CONTROL FROM A CONTROLLER BOX OR SERIAL COMMUNICATION.

WITH CALL Routines.
WITH VECTORED INTERRUPTS.
WITH LIBRARY CALL.
WITH A CHANNEL LAYING ROUTINE.
WITH A NEW MENU OPTION.
WITH A COMMAND ACCEPTABLE CHECKER.
WHEN WAIT, THEN """
WITH A COMMAND TIMES OUT.

BOOK REFERENCES: - I.E. = REFERENCE TO VOL. I, A 5 SECTION A.2 OF BOE NY.
C: APPLICATION NOTE USING THE TMC 7200 LINE CONTROLLER, 1966.

------------------------------------------------------------------------
A. SETTING UP REGISTERS
------------------------------------------------------------------------

MEMORY MAP:
STACK POINTER INITIALLY = 0000
COMMAND REGISTER = 0000-07F
COMMAND POINTER = 0000-07F
FLAG REGISTER = 07F
ERROR REGISTER = 07F
SERIAL RECEIVE ADDRESS REGISTER = 07F
LOAD COMMAND ADDRESS REGISTER = 07F

STACK EQU 0000H
INITRAM EQU 0008H
BASE EQU 2000H
LF EQU 00H
CR EQU 0DH
DEL EQU 0AH

STACK POINTER INITIALLY EQUALS 0FFH.
FOR VECTORED INTERRUPTS, I.E. PROGRAMS.
LINE FEED, """
CARRIAGE RETURN, """
CARRIAGE RETURN, LINE FEED.
1PHYSICAL HARDWARE LOCATIONS FOR STATIC RAM ARE FROM 2000H TO 4000H BUT THERE IS SOME INBOARD RAM IN LOCATIONS FEFFH TO FFFFH.
ISO TO CLEAR RAM, MUST SPECIFY ALL LOCATIONS.
1SEE INITIALIZATION RAM SUBROUTINE.

1BEGINNING OF COMMAND REGISTER.
1END OF COMMAND REGISTER. IT IS 80 BYTES LONG.

1EXTERNAL RAM LENGTH IS 8192.
1HALF OF EXTERNAL LENGTH USED IN CLEAR AND INITIALIZING ROUTINES.

1POINTER TO THE INTERNAL LOCATIONS OF CMDREG.
1A REGISTER FOR FLAGS USED IN THE PROGRAM.
1A REGISTER USED TO SIGNAL ERRORS IN PROGRAM.
1A REGISTER USED TO INITIATE THE POWER-ON-SEND INTEGRATED ROUTINE.
1A REGISTER USED TO INITIALIZE INTERRUPT INSTRUCTION CODE FROM INT DISCONNECT.

1PERIOD EQU 0400H 1PERIOD OF 8 SECONDS, 99.99 HOURS.
1PERIOD EQU PERIOD USED BELOW IN SECTION 1.
1PERIOD EQU PERIOD 1INITIAL FUTURE CYCLE 1PERIOD = 8 SECOND.

1REGISTER FOR L CONSTANT IN -V + V + Z.
1REGISTER FOR H CONSTANT IN -V + V + Z.
1REGISTER FOR B CONSTANT IN -V + V + Z.
1REGISTER FOR E CONSTANT IN -V + V + Z.
1REGISTER FOR MODIFIER REGISTER.
1REGISTER FOR MODIFIER REGISTER.
1REGISTER FOR MODIFIER REGISTER.
1REGISTER FOR MODIFIER REGISTER.

1AND THE RESPECTIVE INITIAL VALUES.

1MINIMUM VALUE IS 0.005 DUE TO LINEARITY.
1CAN RANGE FROM 0.015 TO 0.735. SET HERE TO 0.
1INITIAL PULSE WIDTH IS 1.00. WILL BE REPLACED BY THE O-CLOCKS.
1CAN RANGE FROM 0 TO 256.
1CAN RANGE FROM 0 TO 256.
1CAN RANGE FROM 0 TO 256.
1CAN RANGE FROM 0 TO 256.

1MAXIMUM PULSE WIDTH IS 1024. NO NEED TO CHANGE THIS EVER IN THIS PROGRAM.

1C. INTERRUPT JUMP TABLE

1ORQ 0000H 1SEE P. 7-13. HIGH PRIORITY COMES FIRST.
1JMP PWRRUP 1EXTERNAL RESET: JUMP TO POWER UP.
1ORQ 0044H 1
1JMP SETPGM 1INITIAL LOW-LEVEL INTERRUPT.
1ORQ 0H 1
1JMP $+BASE 1INT0, 11 INTERNAL TIMER INT.
1ORQ 16H 1SEE SEC 4-5A.
1JMP $+BASE 1INT1, INT2 EXTERNAL INT.
1ORQ 24H 1
D. MORE INITIALIZATION

PWRUP: DI  JUST TO BE SURE, DISABLE INTERRUPTS.
        LXI  SP, STCK+1  SET STACK POINTER.
        XRA  R, R      CLEAR ACCUM. & FLG.
        MOV  NA, R    *A PORT OUTPUT PORT.
        MOV  NB, R    *B PORT OUTPUT PORT.
        MVI  A, INTRAM  SET INTERNAL RAM.
        MOV  MM, R    *MEMORAY MAP.

E. INITIALIZE PORT - AT 000H

CH(41). MVI  M #41,  (CH(41)  DIRECTIONS PORT)  SET DIRECTIONS PORT.
         MOV  NA, A  (CH(41)  MODE C CONTROL REGISTER)  SET C MODE
         MVI  M #11, A  (SET 5 REG AS DISABLED OUTPUT CHANNELS.
         MOV  MCI, A  (INITIALIZE MODE C REGISTER (RXD AS TXM)).

F. SET BAUD RATE

USE TIMERS AND INPUT TO GENERATE 1200 BAUD (SEE TABLE IN MANUAL - F. 0-30H).

MVI  M #3, A  (FOR 11.0592 MHZ ATAL).
MVI  TMM, 80H  (SET TMM REG FOR COUNTER CLOCK.)

G. INITIALIZE SERIAL MODE REGISTER

8-BIT DATA, 2 STOP BITS, ASYNCH MODE, 11.0592 MHZ CPU, NO PARITY CHECK.
SEE SEC. I-9A.

MVI  SMH, 0CH  (ENABLE THE RECEIVE, TRANSMIT, SELECT
INTERNAL CLK. "TO" OUTPUT FOR /SCK.
MVI  A, 0CEH  (SPEC AS STATED ABOVE (8-BIT, ETC.).
MOV  SML, A

H. INITIALIZE THE A TO D CONVERTER REGISTER

SCAN MODE, READING CHANNELS
A0-3 INTO REGISTERS CR0-3 RESPECTIVELY, OSCILLATIONS FOR AN 11 MHZ CLOCK.
ATOD:  MVI  ANM, 00H

I. INITIALIZE 2 PROGRAMMABLE SQUARE WAVES

***************
SOWV: MVI A, VIH ;CLEAR AND STOP ECNT COUNTER.
    MOV ETMM, A
    MVI A, 0C7H ;SET CO0 AND CO1 OF PORT C TO THE CONTROL MOI:
    MOV MCC, A ;
INITSW: MVI E0M, 0B7H ;SET CO1=1 AND CO0=1 AND SELECT INVERSION FOR
    ;CO0 AND CO1.
LDCTR: LXI EA, PERD2 ;LOAD ETM1 WITH A COUNT = 1/2 SQ. WAVE PERIOD.
    DMOV ETM1, EA
    LXI EA, FHBT ;LOAD ETM0 WITH A PHASE SHIFT.
    DMOV ETM0, EA
SOWV0: MVI A, 0FCH ;CLEAR ECNT WHEN = ETM1. CO0 OUTPUT CHANGE A
    MOV ETMN, A ;ECNT = ETM0, AND CO1 OUTPUT CHANGE AT
    ;ECNT = ETM1. COUNT BEGINS.

;********************************************************************
; J. BATTERY BACKUP ROUTINE
;********************************************************************
    SJWT V010 ;IF HAVE NOT LOST POWER FROM STANDBY, SKIP.
    SFR J01H1 ;ELSE WE HAVE BEEN ON STANDBY, DO NOT
    ;REINITIALIZE THE INTERNAL RAM BUFFER.
;********************************************************************
; INITIALIZE MEMORY CONTENTS FOR IMPORTANT VARIABLES
;********************************************************************
      MOV H, AINT ;INITIALIZE REG. VALUE GOES IN HL REGISTER.
    SINC HR ;LOAD HL INTO FREG.
    LDI H, RINT ;SAME THING FOR INTY CYCLE.
    SHLD IVROS ;THE 16 BITE VALUES BELOW CAN BE LOADED
    ;WITH ONE COMMAND AND THIS 0FFH OFFSET.
    MVI V, 0FFH
    MVIW VLEF, VLVINT
    MVIW B2EREG, B2EINT
    MVIW CEREF, CEEINT
    LDI H, NORM
    SHLD MEMREG
;********************************************************************
; SOWTV: NOP ;PART OF BATTERY BACKUP ROUTINE.
;********************************************************************
;********************************************************************
; L. INITIALIZE THE INTERNAL RAM
;********************************************************************
; CLEARING IT (1.E. FILLING IT WITH 0'S).
    LXI H, RAMST ;LOOP STARTS AT RAM'S FIRST LOCATION.
    LXI EA, 0 ;PREPARE TO PUT ZEROS IN THE RAM.
    MVI B, RAMD2-1 ;REGISTER BC IS THE COUNTER.
ZMEM: STEAX H++ ;16 BIT STORE, DOUBLE INCREMENT ADDRESS.
    LCR B ;DECCREMENT COUNTER.
    JR ZMEM ;LOOP THROUGH UNTIL B=0 (UNTIL CARRY FLAG).
;********************************************************************
; M. ERROR AND FLAGS REGISTERS FOR EASY PROGRAM USE
;********************************************************************
; DEFINED TO BE = 11111111B.
    MVI V, 0FFH ;THE WORKING REGISTER IS THE INTERNAL RAM.
    ORIW ERRREG, 0FFH ;USE WORKING REGISTER COMMAND TO MOV 1'S
    ORIW FLOREG, 0FFH ;INTO REGISTERS.
    LXI H, CM2OC ;SET THE SER. REG. ADD. REG. WITH CMOD0
N. PULSE WIDTH MODULATION

:BRACKET CONTROLLER ROUTINE: ACCEPTS A VALUE IN HL AND PLACES IT IN
THE PWM CONTROLLING REGISTER (ETM0).

:VALID RANGES ARE FROM 1 TO (PERIOD/2 - 1).

:IN THIS VERSION FROM 1 TO 3FFH.

:THE POWER DELIVERED TO THE MOTOR IS PROPORTIONAL TO THIS RESULT, BUT THE
NUMBER STORED IN THE EVENT COUNTER REGISTER IS PROPORTIONAL TO 1624 - RESULT.

:SEE F. INC.

INVERT:

LBCD
PRLREG
LIRD STREC
LMOV A,L
LMOV B,A
LMPB
LMPB
LMPB
LMPB

:LOAD B WITH 1624.

:MOVE RESULT FROM RESULT INTO HL.

:MOVE A INTO EA.

:CALL KM0.

:MOVE A INTO F.

:THE GAME RULES THE VALUE

:OF THE RESULT.

:THE DIFFERENCE IS STORED IN A.

:SEE MANUAL P. 13C.

LMOV LEH
LMOV LEH
LMOV EHI,EA
LMOV LEH
LMOV LEH
LMOV EHI,ET

:GET THE NEW PULSE WIDTH.


:RESET THE EVENT COUNTER

:TO ITS ORIGINAL STATUS.

II. MENU DRIVING SUBROUTINE

---

:SHOW MENU

:THIS IS A NEW SUBROUTINE TO SHOW THE NEW MENU FOR THE NEW MONITOR
PROGRAM. IT ACCEPTS COMMANDS INTO A COMMAND REGISTER.

:ZMENU: CALL ICLEAR

:CALL CMDLS

:CLEAR THE SCREEN.

:PRINT THE NEW MENU ON THE SCREEN.

:HOW THE USER WILL ENTER A LINE COMMAND WHICH WILL BE PLACED IN THE
:COMMAND REGISTER.

:ZMENU: LXI H,PROMPT

:CALL PMSG

:PRINT PROMPT ON SCREEN.

:ZMENU: LXI H,CMDD

:CLEAR COMMAND REGISTER BEFORE NEW COMMAND LOOP.

:ZMENU: SHLD CMPPTR

:EVENTUALLY COMES HERE. REEXCHANGE REGISTER.
; WORKING ON SPAKE SET OF REGISTERS.
; GET CHARACTER FROM SERIAL PORT.
; NOTE - NOT IMPLEMENTED IN THIS VERSION.
; IF NOT A <CR> THEN CONTINUE.
; IF <CR> END OF COMMAND, DECODE COMMAND.
; IF NOT A BACKSPACE THEN CONTINUE.
; IF A <BS> THEN DECREMENT COMMAND POINTER.
; STORE THE CHARACTER IN THE COMMAND REGISTER.
; LOAD THE EA REGISTER WITH THE VALUE OF CMDE!
; COMPARE THIS VALUE TO POINTER 'VALUE IN H.'
; IF H< EA (END) THEN SIGNAL ERROR 2.
; ELSE GET NEXT CHARACTER FOR DECODER.
; LOAD IN THE DECODING ADDRESS FROM SRAEG
; AND JUMP TO IT.

; DECODE COMMAND REGISTER.

; STORE THE ZEROS AT H, INCREMENT H BY 2.
; DECREMENT COUNTER.
; LOOP BACK UNTIL B = 0.
; WHEN B=0 RESET H TO BEGINNING OF CMD REG.
; RETURN FROM CALL.

; THIS COMMAND LOOKS AT THE COMMAND REGISTE
; AND READS THE FIRST NON-NULL CHARACTER AND JUMPS TO THE CORRESPONDING
; COMMAND IN THE PROGRAM.

; RESET THE POINTER TO START OF CMD REG.
; IF THE COMMAND IS NOT 'A' SKIP NEXT LINE.
; 'A' COMMAND REWRITES THE MENU.
; IF NOT A 'C' SKIP NEXT LINE.
; 'C' COMMAND DISPLAYS DATA.
; IF NOT A 'D' SKIP NEXT LINE.
; 'D' COMMAND DISPLAYS DATA.

; IF THE COMMAND IS NOT ANY OF THE ABOVE
; THEN PRINT CRLF.
; AND PRINT OUT THE
; COMMAND REGISTER FOLLOWED BY A '?'.
; THIS ECHOES THE COMMAND BACK TO THE SCREEN.
; CALL XMIT

; CLEAR THE COMMAND REGISTER AND WAIT FOR A NEW
; COMMAND TO BE ENTERED.
THIS COMMAND CHANGES THE DATA IN XXXX (EA) TO DD (A) IN MEMORY
WHEN DONE IT RETURNS TO WAIT FOR NEXT COMMAND.

CCMD:
- CALL CMNN
- CALL CMWORD
- LDAX H
- CALL CMBYTE
- BIT \$ERRREG
- JMP CCMDE1
- DMOV H,EA
- STAX H
- LXI H,CDONEM
- CALL PMSG
- JMP ZMENU1

CCMDE1:
- CALL ERROR
- JMP ZMENU1

*************************************************************************
D. D COMMAND - DISPLAY DATA
*************************************************************************

THIS COMMAND DISPLAYS THE DATA FROM XXXX (DE) TO YYYYY (EA) ON THE MONITOR
SCREEN. WHEN DONE IT RETURNS TO WAIT FOR NEXT COMMAND.

DCMD:
- CALL CM RANGE
- BIT \$ERRREG
- JMP CCMDE1
- CALL DCMDE1
- JMP ZMENU1

DCMDE1:
- MVI A,CR
- CALL XMIT
- MVI A,LF
- CALL XMIT
- MOV A,D
- CALL PBYTE
- MOV A,E
- CALL PBYTE
- MVI A,'-'
- CALL XMIT
- MVI B,15

DCMD2:
- MVI A,'
- CALL XMIT
- MVI C,1

DCMDS:
- LDAX PBYTE
- CALL DCMDT
- DLT EA,D
- JMP DCMDS
- RET

DCMD4:
- DCR B
- JMP DCMDS
- DCR C
- JMP DCMDS

*************************************************************************
E. RAM MEMORY SPACE CHECK
*************************************************************************

THIS ROUTINE TESTS TO SEE IF THE VALUE THAT IS ABOUT TO BE PRINTED
IS WITHIN THE NEXT MEMORY SPACE AVAILABLE. THESE SPACES ARE FROM
WAS IMPLEMENTED TO REDUCE POWER DISSIPATION.

DCMDT: PUSH EA
LXI EA,0000H
DEQ EA,D
JMP DCMDT0
POP EA
LXI EA,0000H
LXI D,0FFFH
RET

DCMDT0: LXI EA,1000H
GET EA,;1000H THEN SKIP NEXT LINE.
JMP DCMDT1
JMP DCMDT9
RETURN.

DCMDT1: LXI EA, BASE
GET EA,;BASE ADDRESS OF RAM THEN S1 IF NEXT LINE.
JMP DCMDT
ALL DCMDT9
LXI EA,;GET NEXT MEMORY POINT FROM ADDRESS OF NEXT LINE
PRINT
LXI EA,;NEXT LINE IF POINT IS LESS THAN
LXI EA,;EXTERNAL RAM FILE END THEN JUMPTO 11
LXI EA,;ELSE GET NEXT CHECK
MDF DCMDT0
AND RETURN.

DCMDT2: LXI EA,0000H
GET EA,;IF POINT IS GREATER THAN
JMP DCMDT9
PDP EA
RET

DCMDT9: LXI D,0FFFH
PDP EA
CALL DCMDERR
POP EA
DCMDERR: MVI EA,0
RET

******************************************************************************
# E. EACHSPACE FUNCTION
******************************************************************************
CMBDS: LXI EA,CMBDS
GET EA,;GET ADDRESS OF THE COMMAND REGISTER
DLT EA,;END OF NEXT LINE IF H IS PAST START OF REG.
JMP LEBUM
CALL LEBSM
JMP LEBUM
JMP LEBUM
CALL LEBUM
RET

******************************************************************************
# G. COMMAND OVERFLOW ERROR
******************************************************************************
CMDErr1: ANIw ERRREG,1111110B;SET ERROR FLAG.
RET ;AND RETURN.

PERROR: ORIw ERRREG,0000001B; ERROR RESETS THE ERROR FLAG
LXI H,LDERR
AND THEN IT SENDS THE ERROR MESSAGE.
CALL PMSG
RET ;AND RETURNS.

******************************************************************************
# H. NEXT-NON-NULL COMMAND
******************************************************************************

* THIS FUNCTION SEARCHES THE COMMAND REGISTER FOR THE NEXT-NON-NULL VALUE.
CMN: LDA X
GET CHARACTER FROM THE COMMAND REGISTER.
KEI A,' 
GET NEXT CHARACTER IF THIS CHARACTER:
JMP CMN

NEI A,'( ')
NEI A,'( ')
NEI A,'( ')
JMP CMN
NEI A,'( ')
JMP CMN
NEI A,'( ')
JMP CMN
NEI A,'( ')
JMP CMN

DCX H
IF NOT THEN CHARACTER IS NON-NULL, RESET
RET THE POINTED NEXT READ WILL ENCOUNTER
THE NULL CHARACTER AND RETURN.

--------------------------------------------------------------------------------------------------------

MHEX S: CALL CMHEX
FIRST CALL FROM THE COMMAND REGISTER WILL

CMHEX: CMP H, Y
RETURN THE VALUE FOR 'H' FROM CHARACTER.
JMP CMHEX
IF A HEX ERROR OCCURS (CH. 00-3F) ERROR.
BEQ H, Y
IF H FITS THEN CONTINUE.
RET IF FROM 3-7 THEN RETURN VALUE.
SUITE... 
SUBTRACT THE VALUE OF H FROM CHARACTER.
JMP CMHEX
IF A BREACH OCCURS (-9 CH. 40) ERROR.
LTJ A, 0
IF A LESS THEN CONTINUE (0 TO 10 VALUE).
JMP CMHEX
IF GREATER THAN ASCII THEN ERROR.
ADI A, 10
RADE 10 TO MAKE VALUE FOR A=10.
RET $DONE.

CMHEX3: CALL CMHEX

RET $DONE.

--------------------------------------------------------------------------------------------------------

CMHEX: CALL CMHEX
SECOND CALL FROM THE COMMAND REGISTER WILL

CMHEX: CALL CMHEX
GET A HEX CHARACTER.
CALL CMHEX
PUT IT IN HIGH HALF OF REGISTER E.
ADD H, E
ADD FIRST HEX TO SECOND HEX A BYTE IN REGISTER.
POT
RETRIEVE ORIGINAL VALUE IN REGISTER E.
RET $DONE AND RETURN.

CMHEX2: MOV FF, H
PUT IT ON THE LED'S.
SLL A
SWITCH IT INTO HIGH HALF OF BYTE.
SLL A
SLL A
MOV E, A
SAVE IT IN REGISTER E.
RET $RETURN.

--------------------------------------------------------------------------------------------------------

CMHEX3: CALL CMHEX
THIRD CALL FROM THE COMMAND REGISTER WILL

CMHEX: CALL CMHEX
GET A HEX CHARACTER.
CALL CMHEX
PUT IT IN LOW HALF OF REGISTER E.
ADD H, E
ADD FIRST HEX TO SECOND HEX A BYTE IN REGISTER E.
POT
RETRIEVE ORIGINAL VALUE IN REGISTER E.
RET $DONE AND RETURN.

CMHEX2: MOV FE, H
PUT IT ON THE LED'S.
SLL A
SWITCH IT INTO LOW HALF OF BYTE.
SLL A
SLL A
MOV E, A
SAVE IT IN REGISTER E.
RET $RETURN.

--------------------------------------------------------------------------------------------------------

CMHEX3: CALL CMHEX
FOURTH CALL FROM THE COMMAND REGISTER WILL

CMHEX: CALL CMHEX
GET A HEX CHARACTER.
CALL CMHEX
PUT IT IN LOW HALF OF REGISTER E.
ADD H, E
ADD FIRST HEX TO SECOND HEX A BYTE IN REGISTER E.
POT
RETRIEVE ORIGINAL VALUE IN REGISTER E.
RET $DONE AND RETURN.

CMHEX2: MOV FE, H
PUT IT ON THE LED'S.
SLL A
SWITCH IT INTO LOW HALF OF BYTE.
SLL A
SLL A
MOV E, A
SAVE IT IN REGISTER E.
RET $RETURN.

--------------------------------------------------------------------------------------------------------

CMHEX3: CALL CMHEX
FIFTH CALL FROM THE COMMAND REGISTER WILL

CMHEX: CALL CMHEX
GET A HEX CHARACTER.
CALL CMHEX
PUT IT IN LOW HALF OF REGISTER E.
ADD H, E
ADD FIRST HEX TO SECOND HEX A BYTE IN REGISTER E.
POT
RETRIEVE ORIGINAL VALUE IN REGISTER E.
RET $DONE AND RETURN.

CMHEX2: MOV FE, H
PUT IT ON THE LED'S.
SLL A
SWITCH IT INTO LOW HALF OF BYTE.
SLL A
SLL A
MOV E, A
SAVE IT IN REGISTER E.
RET $RETURN.
-144-

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

| THIS FUNCTION RETRIEVES A HEX WORD FROM THE COMMAND REGISTER BY COMBINING TWO HEX BYTES AND PLACING THEM IN THE REGISTER PAIR EA. |

| CMWORD: CALL CMBYTE  ;GET FIRST HEX BYTE.  
| MOV EAH,A  ;AND MOVE IT INTO UPPER HALF OF EA.  
| CALL CMBYTE  ;GET NEXT HEX BYTE.  
| MOV EAL,A  ;AND MOVE IT INTO LOWER HALF OF EA.  
| RET  |

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

| L. CHECK RANGE |

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

| THIS FUNCTION WILL ACCEPT TWO HEX WORDS AS INPUT FROM THE COMMAND REGISTER. THE FIRST WORD WILL BE PLACED IN REGISTER EA. THE ORIGIN IN REGISTER ED. THE FUNCTION WILL CHECK TO MAKE SURE THE WORDS ARE WITHIN THE RANGE. |

| CMCHECK: CALL CMBYTE  ;GET NEXT WORD FROM CM.  
| CALL CMBYTE  ;GET NEXT WORD FROM CM.  |
| POP V  ;FOR THE OLD EA OR THE OLD ED.  |
| DL EAX,0  ;IF EA IS LESS THAN 0 THEN JIP NEXT LINE.  
| RET  ;IF NOT WE THEN HUP  |
| PUSH V  ;FAILED TEST V I R V  |
| CALL CHECK1  ;RETURN.  
| POP V  |

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

| M. CLEAR SCREEN |

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

| Iclear: M.I  ;LOAD COUNTER TO CLEAR 48 LINES.  
| M1 A,CR  ; |
| CALL XMIT  ;SEND CR.  
| MVI A,LF  ; |
| LFLDOO: CALL .M1  ;SEND LF.  
| CALL DELAY  ;A DELAY TO ALLOW THE LINES TO SETTLE.  
| VCR C  ;CLEAR 48 LINES.  
| JP LFLDOO  ;END-LOOP.  
| RET  ;YES-RETURN.  |

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

| N. MESSAGE PRINTING SUBROUTINES |

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

| PMSG: LDAH H+  ;GET CHARACTER FROM MEMORY AT HL.  
| NEI A,B  ;IF BYTE IS A THEN CONTINUE.  
| RET  ;ELSE END OF MESSAGE, RETURN.  
| CALL XMIT  ;TRANSMIT BYTE.  
| CALL DELAY  ;NOT IMPLEMENTED IN THIS VERSION.  
| JR PMSG  ;REPEAT UNTIL END OF MESSAGE.  |

| XMIT: SKIT FST  ;TEST XMIT FLAG.  
| JR XMIT  ;WAIT TILL SET.  
| MOV TXB,A  ;SEND THE CHARACTER THAT IS IN REGISTER A.  
| RET  ;DONE RETURN.  |
; DCR B ; I5 NUMBER OF TIMES.
; JR PLOOP ;
; RET ; RETURN.

;*******************************************************************************
;*******************************************************************************
; O. PUSH BYTE
;*******************************************************************************

;USES AND CHANGES REGISTER E.

; PBYTE: PUSH D ; SAVE "A" IN AN UNUSED REGISTER.
; MOV E, A ; AFTER SAVING A
; SLR A ; SHIFT 4 HIGH BITS
; SLR A ; INTO THE 4 LOW BITS.
; CALL, PHE ; PRINT HEX ASCII VALUE.

;*******************************************************************************
;*******************************************************************************
; H. INTERPRET CHARACTER
;*******************************************************************************

; THIS SUBROUTINE WRITES THE ASCII VALUE RECEIVED TO THE
; TRANSMIT CHANNEL OF PORT C WHEN THAT CHARACTER IS RECEIVED FROM THE MONITOR.
; LED'S NOT IMPLEMENTED IN THIS VERSION.

; INCH: NOP ; NO OPERATION.
; RXD1: SHT FSR ; TEST RECEIVE INTERRUPT FLAS.
; JR RXD1 ; JUMP BACK IF NOT SET, WAIT FOR A CHARACTER.
; MOV A, RXD ; PUT CHARACTER FROM RXD INTO A.
; MOV PB, A ; PUT IT OUT ON PORT B LED'S.

; TXD1: SHT FSI ; TEST TRANSMIT FLAG.
; JR TXD1 ; WAIT UNTIL ITS SET (READY TO TXD AGAIN).
; MOV TXB, A ;
; RET ; ONCE CHARACTER HAS BEEN ECHOED, RETURN.

;*******************************************************************************
;*******************************************************************************
; D. DELAY ROUTINE
;*******************************************************************************

; DELAY: LXI CL, DEFAB ; A DELAY TO LET LINKS SETTL.
; LXI D, 00 ;
; DELAY1: DSUBN EA, D ;
; JMP DELAY2 ;
; JMP DELAY1 ;
; DELAY2: MOV A, 00AAH ;
; MOV PB, A ;
; RET ;
A. SETUP

VISC:  NOP  ; NO OPERATION.
       SHIT FSR  ; IF RECEIVE FLAG SET, SKIP NEXT LINE.
       JR VINAD  ; ELSE JUMP TO ANALOG.
       MOV A, RXB  ; GET CHARACTER.

VTXD:  SHIT FST  ; CHECK THE FLAG UNTIL READY TO
       JP VTI  ; TRANSMIT CHARACTER BACK TO SCREEN.
       MOV TAH, A  ; TRANSMIT CHARACTER.
       CALL TMENU2
       JMP VISC

TML:  CALL TML  ; CALL TO CHARACTER LOOP IF FLAG IS NOT SET.
       VTO  ; RETURN TO END.
       RST  ; RETURN TO MACHINE LANGUAGE.

LLO:  MOV A, 0  ; MOV A, TO LOCATION.
       ORL A  ; THEN PUMP TO THE LOCATION IN THE MODIFICATION
       JMP VML  ; REGISTER. FOR CONTROL BOX.

6. ABSOLUTE VELOCITY CALCULATION

FIRST GET RELATIVE VELOCITY FROM IVLREG. THEN SUBTRACT 128 FROM THE RELATIVE VELOCITY IF THE RELATIVE VELOCITY IS > 128, OR SUBTRACT THE RELATIVE VELOCITY FROM 128 IF THE RELATIVE VELOCITY IS < 128. MULTIPLY THE ABSOLUTE VELOCITY BY 2 TO ACHIEVE A RANGE OF 0-255.

NORMAL:  LOAD A WITH ZERO VELOCITY VALUE = 128.
       MOV C, A  ; COPY THE ZERO VELOCITY VALUE TO C.
       CALL ABSV
       JMP V42

ABSV:  MOV A, 0  ; SET DEC DECODING READING RELATIVE VELOCITY.
       ADD C, A  ; IF RELATIVE VELOCITY > 128, THEN SKIP NEXT LINE.
       JMP V1  ; ELSE JUMP TO V1.
       SUB C, A  ; SUBTRACT 128 TO GET ABS VELOCITY
       JMP V2  ; AND SKIP OVER V1 SECTION.

V1:  MOV B, A  ; IF RELATIVE VEL. <= 128 THEN SUBTRACT IT FROM C.
       MOV A, C  ; 128 TO GET ABSOLUTE VELOCITY.
       SUB A, B  ;

V2:  LTI A, 128  ; NOW DOUBLE THE VELOCITY (RANGE 0-255).
       JMP V3  ; IF ABS. VEL. > 128 THEN MOVE 255 INTO A.
       MVI C, 2  ; ELSE MULTIPLY BY 2.
       MLL C  ;
       MOV A, A+L  ; THE RETRIEVE IT INTO A.
       JMP V4  ;

V3:  MVI A, 1  ; MOVE 255 INTO REGISTER A.
       RET  ;
V4:  MOV C, A  ; MOVE RESULT INTO C.
       ; C HOLD VELOCITY.

C. A(V*V) CALCULATION

***************
1. **Calculate the term proportional to V**. It is only proportional because the low order four bits are lost in the division. For convenience, let \( z \approx (\text{Absolute Velocity}) \approx \text{Velocity} \approx \text{V} \) in the comments for this section. Finally, multiply this V*V term by the A-constant.

V43:  
MOV B, A  
MUL B  

Move velocity from A into B.  
Square value. Result in EA.  
V*V is in EA.  
This is a 16-bit result, but we only need the low order 8 bits. This is V*V term.

LOAD A = A0 = A1 = A2 = A3 = A4 = A5 = A6 = A7  
MOV B, A  
MOV A, EA  
MUL B  

Load A with value of A-constant.  
Move A-constant to B.  
Move V*V/256 to A.  
Multiply A-constant by V*V, result in EA.

MOV A, EAH  

Move high 8 bits of A*V to A.

MOV A, D  
SCALE a*V up by 4.  

MOV A, EAH  

Move high 8 bits of A*V to A.

ECI A, B  

If they are \( \neq 0 \), 4B*V > 255.

JMP V6A  

Set to zero if 4B*V > 255.

MOV A, EA  

Otherwise, low bits are the answer.

V6A:  
MOV A, 255  

Set 4B*V = 255.

V6B:  
MOV C, 255  

Judge.

V6C:  
MOV B, A  

Load C-constant into A.

LOAD A = A0 = A1 = A2 = A3 = A4 = A5 = A6 = A7  

MOV A, EA  

Subtract C-constant from 255 to get the fraction that the velocity will change.

MOV A, B  

Retrieve 4B*V into A.

MUL C  

14B*V/255 - C-constant is in EA.

MOV A, EA  

This gives the fraction needed in A.  
(4B*V/255 - C-constant)/256 is in A.

D. **Summation of Terms**

NOW retrieve the value of the C-constant and add it to the two previous results.

V6D:  
LOAD A = A0 = A1 = A2 = A3 = A4 = A5 = A6 = A7  

ADD A-constant to previous term.
-148-

; THIS GETS THIS TERM IN THE RIGHT RANGE.
; RESULT IS IN EA.

; RETRIEVE V*V TERM.
; ADD V*V TERM TO V TERM.
; MOVE RESULT TO HL.
; MOVE RESULT INTO DTREG.
; DONE! THIS ACHIEVES DAMPING VALUES
; PROPORTIONAL TO A(V*V) + B*V + C.

; RATHER THAN WAIT 100 A/D CYCLES
; WAIT FOR AN FEI INTERRUPT
; BEFORE GETTING THE NEXT A/D READ.

; LOOP INFINITELY RARE UNTIL RESET.

;-----------------------------
; 1. CONTROL AND PROGRAM
;-----------------------------

; MODXEE: MOV A, CR
; MOV P, A
; STA MODXEE
; JMP MODXEE

; MODDIE: MOV A, DT
; MOV P, A
; STA MODDIE
; JMP MODDIE

; SETPGM: SKIT FAD
; JR SETPGM
; EXX
; PUSH V
; PUSH B
; PUSH D
; PUSH H
; PUSH BA

; MVI DA, 0FFH
; BIT 7, FLREG
; JMP SETPGM

; ELSE SOTO FLIP FLAG.

; AND CONTINUE.

; SETPGM: OFHH
; JR SETPGM

; EXX
; PUSH V
; PUSH B
; PUSH D
; PUSH H
; PUSH BA

; MVI DA, 0FFH
; BIT 7, FLREG
; JMP SETPGM

; ELSE SOTO FLIP FLAG.

; AND CONTINUE.
-149-

JMP SETPG3
JMP SETS

; IF THE FLAG IS ON (IM MATARY) THEN RUN NORMAL
; PROGRAM WITH NO MODIFYING REGISTERS.
; ELSE CONTINUE.

SETPG3: MOV A, CR1
GTE A, 50
JMP SET1
GTE A, 100
JMP SET2
GTE A, 150
JMP SET3
GTE A, 225
JMP SET4
MOV A, CR3
LTI A, 225
JMP SET5
JMP SET5

; THIS PART OF THE PROGRAM BLITS THE MODIFIED REGISTER CONTENT TO THE
; APPROPRIATE LOCATION

SET1: LXI H, MODH
MVI M, MODM
JMP SET3
SET2: LXI H, MODL
SHLD MODREG
JMP SET5
SET3: LXI H, MODT
SHLD MODREG
JMP SET5
SET4: LXI H, MODY
SHLD MODREG
JMP SET5
SET5: LXI H, MODY
SHLD MODREG
JMP SET5
SET6: CALL CALIB
; CALIBRATE VELOCITY, POSITION 5 ON I/O.
SET7: POP EA
POP H
POP D
POP B
POP V
EXX
; WHEN DONE EXCHANGE REGISTERS -GAIN
KETI
; AND RETURN FROM THE INTERRUPT.

;*******************************************************************************
; V. MESSAGES FOR SYSTEM
;*******************************************************************************
; NOTE - HERE I HAVE ABANDONED MY CONVENTION OF COMMENTING AFTER EACH LINE
; BECAUSE THIS SECTION IS EASIER TO READ IF THE COMMENTS COME AT THE BEGINNING
; AND AT THE END OF THE CODE.

FSFACE: DB ', '
DB 0
DELETE: DW BSSP
DW BS
DB 0
PCRLF: DW CRLF
DB 0

; CMDLST: DW CRLF
DB ' A - - - - - - - - DISPLAY COMMAND LIST ',
DW CRLF, CRLF
DB 'D (XXX),<YYY>' ; DISPLAY DATA BLOCK FROM XXXX TO <YYY>
DW CRLF,CRLF
DB 00

; END OF COMMAND LIST.

; PROMPT: DW CRLF,CRLF
; DB 'HI!'
; DB 00

; CHANGE COMMAND DONE MESSAGE.

; CCOME: DW CRLF
; DB 'CHANGE DONE.'
; DB 00

; LDERR: DW CRLF
; DISPLAY ERROR MESSAGE.
; DB 00