Modified Roland PG1000

John Talbert, 1994
The Roland PG1000

The PG-1000 is a dedicated programmer for the Roland D-50 keyboard synthesizer and the D-550 rack mounted version. It works with MIDI system exclusive, and requires an external 9V power supply. It has an impressive 56 faders used to set the four partials, two tones and one common parameter block for each D-50 patch. Programming was made easier by displaying parameter values on a backlit LCD screen along with dedicated partial/tone select buttons.

Its impressive number of faders and MIDI output immediately suggest a possible use as a generic MIDI controller device. When it came out in the late 1980’s Midi Controller devices were rare. However, the MIDI output of the PG1000 is in System Exclusive form and each fader is tied to a specific D-50 function with a specific range of values; all of which does not easily translate to any use as a generic MIDI controller.

Upon opening up the PG1000, the internal circuitry turned out to be very simple and straightforward. It consists mainly of a NEC 78C10 Processor running code from a 32K PROM memory chip. The Processor includes eight 8-bit Analog to Digital Converters which are used to convert the 0 to 5 volt output from each slider. Each of the 8 ADCs handles 8 sliders routed through a 4051 CMOS Data Selector chip. Note that this scheme allows for a total of 64 sliders and the unit has only 56. One of the processor’s ADC is not being used – a point that will turn out to be useful later.

The NEC 78C10 Processor, better described as a single chip microcomputer, has a number of other useful features put to good use in the PG1000: an internal serial I/O engine used for the MIDI I/O, internal clocks and counters used to set the MIDI and serial data rates, a small amount of RAM memory for storing setup data, and extra I/O lines dedicated to the LCD controller and several pushbuttons.

The Task

Seeing how simple and traceable the internal circuitry was, the task of reprogramming the PROM memory to turn the PG1000 into a MIDI slide controller became a viable possibility. To avoid as much assembly language programming as possible, I chose to set up a Forth Language operating system on the memory chip. This route was made possible by the development of the eForth system by C. H. Ting. The eForth system is a complete Forth system, designed to be small enough to fit on a memory chip. It requires that only 31 simple code words be built from assembly code for any particular processor. The remaining higher level eForth code words are then built from these 31 base words. With eForth, after programming the base words, I then had a complete higher-level programming language to facilitate building the main code that will turn the PG1000 into a MIDI Controller.
The rest of this paper will document how this was done. First, here is a description of the end product.

**Modified Roland PG1000**

This Roland PG1000 has been reprogrammed to put out conventional Midi command instead of System Exclusive.

The unit has 56 sliders numbered from 0 to 55. (There is also the capability for 8 external, zero to five volt, control voltage inputs numbered as sliders 56 to 63.) It also has 10 pushbuttons, 8 of which have taken on new functions as Cursor Left, Cursor Right, Field increment, Field Decrement, Slide increment, Slide Decrement, ENTER, and MIDI.

The unit has four modes of operation:

**(1) Edit Mode.**

Each Slider has an Edit Window on the LCD display with the following fields: Slider number, slider on/off, Midi channel number, Midi operation, Midi operation data, and Slider value.

An LCD cursor can be moved to any of the above fields using the Left/Right buttons. The selected field can then be edited using the Field Increment/Decrement buttons. The edited Slider Window is only loaded into memory when the Enter button is pressed. The Slide Increment/Decrement buttons enable you to step through the Slider Windows without moving the cursor.

No Midi data is sent while in the Edit Mode. The Slider value field provides a running display of the slider value.

The unit supports the following Midi Operations:

**Key#**

Midi Key On is sent when a slider movement up from zero peaks out. The key value sent is programmed in the Midi data field and the key velocity sent is the peak value of the slider movement. A note off is sent when the slider returns to zero.

**Key# AT**

Midi Key On with After-touch. Midi Key On/Off values are sent as described above in Key#. In addition, a continuous Midi
After-touch value is sent with any slider movement until it is returned to zero.

**Control#**
Midi Controller. A continuous controller value is sent with any slider movement. The Controller number is set in the Midi data field.

**Program#**
Midi Program Change. When the slider goes above a certain threshold value, the program change number as set in the midi data field is sent once. The data is not sent again unless the slider is returned below the threshold value.

**Ch Press**
Midi Channel Pressure. A continuous channel pressure value is sent with any slider movement. The midi data field is not used.

**Ptch Whl**
Pitch Wheel. A continuous pitch wheel value is sent with any slider movement. The midi data field is not used so that only a 7 bit value is sent.

(2) Midi Run Mode.

When the Midi button is pressed, the display will change to "Midi Running" and the enabled Sliders will start sending Midi data. To get back to Edit Mode press Enter, Increment, or Decrement.

(3) Setup Mode.

One problem with the unit is that when powered off, all the slider setting you set in the Edit Mode are lost. To help alleviate this inconvenience, a “Setup Mode” was programmed to allow you to easily setup the sliders with a number of fixed settings stored away in some available EPROM space.

To enter the Setup Mode move the cursor under the Edit field labeled "Slider#" and then press Up or Down. The display will change to read "Setup# nn". There are a total of 64 possible slider setups stored in ROM memory. To load a particular setup use the Up/Down buttons to select the setup number and then press the Enter Button. The display will then go back to Edit Mode.
Setup# 00 disables all 64 sliders. Setup# 01 was designed for a class. Most of the remaining setups act like Setup#00 but are empty, available for future customizing.

(4) Forth Mode.

The unit can have a serial input/output port which connects to any computer's serial I/O. With a terminal emulation program set for 9600 baud, 8-bits, 1 stop bit, you can access the Forth language operating system used in the unit. There is a small amount of Ram memory available on the processor chip for implementing your own programs.

The unit will exit its Edit program loop and enter the "Forth Mode" with any key action on the computer terminal when the serial port is connected. To re-enter the Edit program just type EDIT and return.

Availability

If you would like one of these units for your own, your first problem is finding an original Roland PG1000. They are a pretty scarce item. Once you manage to find one, you will then need to burn a 27C128 EPROM memory chip with the available code and then install it in place of the unit’s original PROM memory.
Setups

0 All off except slider 0 with pitch wheel, and slider 1 with channel pressure.

1 Combination of Note On with aftertouch, and midi control.

2 Midi Control 0 through 63, Midi channel 15.

3 Midi Program changes 0 through 63, Midi channel 0
4 Midi Program changes 64 through 127, Midi channel 0

5 Key On 0 through 63, Midi channel 15
6 Key On 64 through 127, Midi channel 15

7 Key On with Aftertouch 0 through 63, Midi channel 15
8 Key On with Aftertouch 64 through 127, Midi channel 15
Modified PG1000 Disassembly

Once you have an EPROM chip burned with the new code, you will have to install it in place of the original PROM. The following instructions detail how this is done.

1. Remove power connector. Pull off all the slider knobs.

2. Remove 6 screws on the bottom.

3. Open up the back.

4. Remove all the screws holding the main circuit board, including the bar across the middle.

5. Gently lift up the circuit board and flip it over. Be careful not to pull any of the connections to the other smaller boards.

6. Look for the socketed PROM chip (28-pin chip). One end of the chip will have a small half-circle indentation. Make a note of how the indentation is oriented.

7. Using a small flat screw driver, work the chip out of its socket by prying up each end. Take your time, pry up a little at a time.

8. Replace the original PROM with the new one.

**IMPORTANT!**

- The chips can be destroyed with static electricity. Touch something metal before handling.
- Make sure the chip orientation (indentation) is the same as the original.
- Make sure all the pins are set in the socket before pushing.
- After pushing, make sure all pins have gone in straight.

9. Reassemble. It may take some gentle jiggling around to get the main board back in place. Don't start screwing until it is in place. Remember that the bottom two screws to the circuit board go in from the back panel.

10. Plug in and power on.
PG1000 Circuit Revisions

Beyond changing the operation of the PG1000 by replacing its EPROM instructions, a couple circuit changes can also be made to the PG1000.

First of all, the new EPROM contains a complete FORTH language operating system which can be accessed through a standard RS232 serial connection (which was how the new code was built). There is a small amount of RAM memory available on the processor chip to allow some additional programming.

Secondly, the 78C10 processor has available 8 Analog to Digital Converters. One of them is not being used. This unused ADC can be revived to provide an additional eight external control voltages. These 0 to 5 volt control voltages can be generated by any number of devices, several of which will be shown here.

What follows are instructions for making these changes.

General

These changes require a small circuit board mounted to the inside of the PG1000 with connections to the main computer chip on the large circuit board and to the smaller MIDI/Power board. You will need the following:

- MAX 232 chip by Dallas Semiconductors (RS232 Serial Interface)
- 4051 CMOS chip (8 in to 1 out multiplexer)
- a small circuit board and two 16-pin chip sockets
- 9 pin D plug (male and female)
- 47K Resistor
- 4 Capacitors - 10 μF

A circuit diagram is shown on the following page. I have mounted the board onto the back end of the PG1000 using a 9-pin D plug. Solder 2-inch lengths of solid wire (heavy gauge, about 20) to the 9 pins on the plug. Insert the other wire ends into the edge holes of the board and then solder to the 4051 chip socket. Bend the 9 wires until the D-plug is at a right angle to the board. Drill holes for the D-plug on the back of the PG1000 where the Roland name appears. The board will sit under the LCD panel.

Ribbon cable makes the connections to the main board a little easier. Most of the connections are to the Main processor chip. This chip has 64 pins. A figure shows its orientation and pin numbering scheme.
Too much heat to any chip's pins can destroy the internal connection; therefore, use a low wattage soldering iron, clean the tip on a wet sponge till it is shiny, and don't contact the pin for more than 2 or 3 seconds. In most cases you won't need to solder directly to the processor's pins. I suggest that whenever possible, you trace the connection to another pad and solder it there.

Serial Input/Output

This RS232 Serial I/O will allow you to explore the Forth Language operating system on the new EPROM memory chip. A small amount of RAM memory is available for creating your own programs. I used this interface to reprogram the box.

The serial interface requires only three lines - send, receive and ground. They connect to another computer running a terminal emulation program. I have used the 5-pin DIN plug labeled "parameter in" on the PG1000 to get these three lines plus a Reset line out of the box. You will then need to make a cable to go from the 5-pin DIN to your own computer’s RS232 connector.

See the "Modified PG1000" description sheet for more information on this "Forth Mode" of operation. If you are not interested in this mode, ignore the MAX232 part of the circuit diagram.

External Control Voltages

The internal processor has 8 Analog to Digital Converters each of which takes care of 8 sliders. The PG1000 uses only 7 of these ADC's for a total of 56 sliders. The eighth one (AN7) is grounded. So by disconnecting the ground to AN7 and adding another 4051 chip you can get 8 more zero to 5 volt control inputs numbered 55 through 63.

The 8 control voltage inputs and circuit ground are brought out of the box using a 9-pin D plug. Each of the 8 inputs can then be connected to any 0 to 5 volt source. Zero volts will read as a 0, and 5 volts will read as 127. Be careful to limit the voltage on these inputs to the range 0 to +5 volts.
Simple Controllers

The external inputs can come from a wide variety of devices - switches, pedals, pots, light sensors, motion sensors, pressure sensors. A few simple devices are shown in the accompanying sheets.

For a light sensor use Cadmium Cells (source - Radio Shack), also known as Photoresistors. A simple pressure sensitive resistance is skin resistance accomplished by bridging across two metal poles with your finger. The harder you press the smaller the skin resistance. Another simple pressure sensitive substance is the black foam that distributors sometime use to protect chips from static electricity. This foam changes its resistance when pressed. Just sandwich it between two metal plates for a variable resistance when pressed. Piezoelectric disks can be directly connected to the inputs with some success. You can get better response, though, by electrically buffering them. Piezo's, often used in drum pads, put out a voltage which varies with changing pressure applied to it.

One useful way of distributing your 8 voltage inputs is by connecting the 9-pin D connector to a small box with 8 stereo phone jacks. Also inside the box is a simple 5 volt power supply using a 9 volt DC power wart and a 5 volt regulator, shown on an accompanying sheet. The 5 volt power source is connected to the ring of each stereo jack which is a convenient way to send power to each of your eight controller devices. I prefer this method to taking 5 volts from inside the PG1000. Each of the simple circuits shown on the sheets assumes you are using stereo plugs for the connections with a 5 volt power source on the ring.
PG1000 Revisions
Serial I/O and 8 External 0-5v Control Inputs

Numbers in parenthesis are NEC 78C10 pin numbers

Connects to box with 8 Stereo Phone Jacks. Signal on tips, separate 5 volt power supply on rings, Ground on sleeves.

5 pin DIN jack
"Parameter IN" on the PG1000
Cut traces to pins 4, 5 connect 2 to Ground
5 volt power supply for 8 external control voltages

Continuous controller / Pedal

Switching (0, 5v) Controller
Light Sensor Control Voltage

both elements are Cadmium Cells

Light Wand

Pressure Sensor Controller
Modified Roland PG1000

NEC78C10 Microcomputer
NEC 78C10 Microcomputer

At the heart of the Roland PG1000 is a NEC 78C10 single-chip microcomputer. It integrates on-chip functions that are normally provided by external components. These functions include a 16-bit ALU, a 256-byte RAM memory (used for program variables), an eight channel Analog to Digital converter (used to convert 56 slider voltages), a 16-bit timer/event counter, two 8-bit timers, a USART serial interface (used to send MIDI) and a total 44 input/output lines (used for the LCD screen, pushbuttons, LEDs, serial I/O, and other uses).

What follows is the pin configuration for the 64-pin chip, and a table of its instruction set. More detailed info can be found in the NEC Single-Chip Microcontroller Data Book (1990, NEC Electronics Inc).
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOP</td>
<td></td>
<td>No operation</td>
</tr>
<tr>
<td>01</td>
<td>LDAW wa</td>
<td>2</td>
<td>A &lt; (V / offset)</td>
</tr>
<tr>
<td>04</td>
<td>LXI rp2,word</td>
<td>3</td>
<td>SP &lt; word</td>
</tr>
<tr>
<td>05</td>
<td>ANIW wa,byte</td>
<td>3</td>
<td>(V / offset) &lt; (V / offset) AND byte</td>
</tr>
<tr>
<td>07</td>
<td>ANI A,byte</td>
<td>2</td>
<td>A &lt; A AND byte</td>
</tr>
<tr>
<td>08</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; EAH</td>
</tr>
<tr>
<td>09</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; EAL</td>
</tr>
<tr>
<td>0A</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>0B</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; C</td>
</tr>
<tr>
<td>0C</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; D</td>
</tr>
<tr>
<td>0D</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; E</td>
</tr>
<tr>
<td>0E</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; H</td>
</tr>
<tr>
<td>0F</td>
<td>MOV A, r1</td>
<td>1</td>
<td>A &lt; L</td>
</tr>
<tr>
<td>10</td>
<td>EXA</td>
<td>1</td>
<td>Alternate V, A, EA register sets</td>
</tr>
<tr>
<td>11</td>
<td>EXX</td>
<td>1</td>
<td>Alternate B, C, D, E, H, L register sets</td>
</tr>
<tr>
<td>12</td>
<td>INX rp</td>
<td>1</td>
<td>BC &lt; BC +1</td>
</tr>
<tr>
<td>13</td>
<td>DCX rp</td>
<td>1</td>
<td>BC &lt; BC - 1</td>
</tr>
<tr>
<td>14</td>
<td>LXI rp2,word</td>
<td>3</td>
<td>BC &lt; word</td>
</tr>
<tr>
<td>15</td>
<td>ORIW wa,byte</td>
<td>3</td>
<td>(V / offset) &lt; (V / offset) OR byte</td>
</tr>
<tr>
<td>16</td>
<td>XRI A,byte</td>
<td>2</td>
<td>A &lt; A EX-OR byte</td>
</tr>
<tr>
<td>17</td>
<td>ORI A,byte</td>
<td>2</td>
<td>A &lt; A OR byte</td>
</tr>
<tr>
<td>18</td>
<td>MOV r1,A</td>
<td>1</td>
<td>EAH &lt; A</td>
</tr>
<tr>
<td>19</td>
<td>MOV r1,A</td>
<td>1</td>
<td>EAL &lt; A</td>
</tr>
<tr>
<td>1A</td>
<td>MOV r1,A</td>
<td>1</td>
<td>B &lt; A</td>
</tr>
<tr>
<td>1B</td>
<td>MOV r1,A</td>
<td>1</td>
<td>C &lt; A</td>
</tr>
<tr>
<td>1C</td>
<td>MOV r1,A</td>
<td>1</td>
<td>D &lt; A</td>
</tr>
<tr>
<td>1D</td>
<td>MOV r1,A</td>
<td>1</td>
<td>E &lt; A</td>
</tr>
<tr>
<td>1E</td>
<td>MOV r1,A</td>
<td>1</td>
<td>H &lt; A</td>
</tr>
<tr>
<td>1F</td>
<td>MOV r1,A</td>
<td>1</td>
<td>L &lt; A</td>
</tr>
<tr>
<td>20</td>
<td>INRW wa</td>
<td>2</td>
<td>(V / offset) &lt; (V / offset) + 1, Skip if carry</td>
</tr>
<tr>
<td>21</td>
<td>JB</td>
<td>1</td>
<td>PC(hi) &lt; B, PC(lo) &lt; C</td>
</tr>
<tr>
<td>22</td>
<td>INX rp</td>
<td>1</td>
<td>DE &lt; DE +1</td>
</tr>
<tr>
<td>23</td>
<td>DCX rp</td>
<td>1</td>
<td>DE &lt; DE - 1</td>
</tr>
<tr>
<td>24</td>
<td>LXI rp2,word</td>
<td>3</td>
<td>DE &lt; word</td>
</tr>
<tr>
<td>25</td>
<td>GTIW wa,byte</td>
<td>3</td>
<td>(V / offset) &lt; (V / offset) - byte - 1, Skip if no borrow</td>
</tr>
<tr>
<td>26</td>
<td>ADINC A,byte</td>
<td>2</td>
<td>A &lt; A + byte, Skip if no carry</td>
</tr>
<tr>
<td>27</td>
<td>GTI A,byte</td>
<td>2</td>
<td>A &lt; A - byte - 1, Skip if no borrow</td>
</tr>
<tr>
<td>29</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (BC)</td>
</tr>
<tr>
<td>2A</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (DE)</td>
</tr>
<tr>
<td>2B</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (HL)</td>
</tr>
<tr>
<td>2C</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (DE)+</td>
</tr>
<tr>
<td>2D</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (HL)+</td>
</tr>
<tr>
<td>2E</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (DE)-</td>
</tr>
<tr>
<td>2F</td>
<td>LDAX rpa2</td>
<td>1</td>
<td>A &lt; (HL)-</td>
</tr>
<tr>
<td>30</td>
<td>DCRW wa</td>
<td>2</td>
<td>(V / offset) &lt; (V / offset) - 1, Skip if borrow</td>
</tr>
<tr>
<td>31</td>
<td>BLOCK</td>
<td>1</td>
<td>Block transfer (HL)+ to (DE)+ for C counts</td>
</tr>
<tr>
<td>32</td>
<td>INX rp</td>
<td>1</td>
<td>HL &lt; HL + 1</td>
</tr>
<tr>
<td>33</td>
<td>DCX rp</td>
<td>1</td>
<td>HL &lt; HL - 1</td>
</tr>
<tr>
<td>34</td>
<td>LXI rp2,word</td>
<td>3</td>
<td>HL &lt; word</td>
</tr>
<tr>
<td>35</td>
<td>LTIW wa,byte</td>
<td>3</td>
<td>(V / offset) - byte, Skip if borrow</td>
</tr>
<tr>
<td>36</td>
<td>SUINB A,byte</td>
<td>2</td>
<td>A &lt; A - byte, Skip if no borrow</td>
</tr>
<tr>
<td>37</td>
<td>LTI A,byte</td>
<td>2</td>
<td>A - byte, Skip if borrow</td>
</tr>
<tr>
<td>39</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(BC) &lt; A</td>
</tr>
<tr>
<td></td>
<td>Instruction</td>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>-----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>3A</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(DE) &lt; A</td>
</tr>
<tr>
<td>3B</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(HL) &lt; A</td>
</tr>
<tr>
<td>3C</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(DE)+ &lt; A</td>
</tr>
<tr>
<td>3D</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(HL)+ &lt; A</td>
</tr>
<tr>
<td>3E</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(DE)- &lt; A</td>
</tr>
<tr>
<td>3F</td>
<td>STAX rpa2</td>
<td>1</td>
<td>(HL)- &lt; A</td>
</tr>
<tr>
<td>40</td>
<td>CALL word</td>
<td>3</td>
<td>Subroutine call, PC &lt; word</td>
</tr>
<tr>
<td>41</td>
<td>INR r2</td>
<td>1</td>
<td>A &lt; A + 1, Skip if carry</td>
</tr>
<tr>
<td>42</td>
<td>INR r2</td>
<td>1</td>
<td>B &lt; B + 1, Skip if carry</td>
</tr>
<tr>
<td>43</td>
<td>INR r2</td>
<td>1</td>
<td>C &lt; C + 1, Skip if carry</td>
</tr>
<tr>
<td>44</td>
<td>LXI rp2,word</td>
<td>3</td>
<td>EA &lt; word</td>
</tr>
<tr>
<td>45</td>
<td>ONIW wa,byte</td>
<td>3</td>
<td>(V / offset) AND byte, Skip if no zero</td>
</tr>
<tr>
<td>46</td>
<td>ADI A,byte</td>
<td>2</td>
<td>A &lt; A + byte</td>
</tr>
<tr>
<td>47</td>
<td>ONI A,byte</td>
<td>2</td>
<td>A AND byte, Skip if no zero</td>
</tr>
<tr>
<td>4801</td>
<td>SLRC r2</td>
<td>2</td>
<td>A shift logical right, Skip if carry</td>
</tr>
<tr>
<td>4802</td>
<td>SLRC r2</td>
<td>2</td>
<td>B shift logical right, Skip if carry</td>
</tr>
<tr>
<td>4803</td>
<td>SLRC r2</td>
<td>2</td>
<td>C shift logical right, Skip if carry</td>
</tr>
<tr>
<td>4805</td>
<td>SLLC r2</td>
<td>2</td>
<td>A shift logical left, Skip if carry</td>
</tr>
<tr>
<td>4806</td>
<td>SLLC r2</td>
<td>2</td>
<td>B shift logical left, Skip if carry</td>
</tr>
<tr>
<td>4807</td>
<td>SLLC r2</td>
<td>2</td>
<td>C shift logical left, Skip if carry</td>
</tr>
<tr>
<td>480A</td>
<td>SK f</td>
<td>2</td>
<td>Skip if CY = 1,</td>
</tr>
<tr>
<td>480B</td>
<td>SK f</td>
<td>2</td>
<td>Skip if HC = 1,</td>
</tr>
<tr>
<td>480C</td>
<td>SK f</td>
<td>2</td>
<td>Skip if Z = 1,</td>
</tr>
<tr>
<td>480A</td>
<td>SK f</td>
<td>2</td>
<td>Skip if CY = 1,</td>
</tr>
<tr>
<td>480B</td>
<td>SK f</td>
<td>2</td>
<td>Skip if HC = 1,</td>
</tr>
<tr>
<td>4821</td>
<td>SLR r2</td>
<td>2</td>
<td>A shift logical right</td>
</tr>
<tr>
<td>4822</td>
<td>SLR r2</td>
<td>2</td>
<td>B shift logical right</td>
</tr>
<tr>
<td>4823</td>
<td>SLR r2</td>
<td>2</td>
<td>C shift logical right</td>
</tr>
<tr>
<td>4825</td>
<td>SLL r2</td>
<td>2</td>
<td>A shift logical left</td>
</tr>
<tr>
<td>4826</td>
<td>SLL r2</td>
<td>2</td>
<td>B shift logical left</td>
</tr>
<tr>
<td>4827</td>
<td>SLL r2</td>
<td>2</td>
<td>C shift logical left</td>
</tr>
<tr>
<td>4828</td>
<td>JEA</td>
<td>2</td>
<td>PC &lt; EA</td>
</tr>
<tr>
<td>4829</td>
<td>CALB</td>
<td>2</td>
<td>Subroutine call, PC &lt; BC</td>
</tr>
<tr>
<td>482A</td>
<td>CLC</td>
<td>2</td>
<td>CY &lt; 0</td>
</tr>
<tr>
<td>482B</td>
<td>STC</td>
<td>2</td>
<td>CY &lt; 1</td>
</tr>
<tr>
<td>482D</td>
<td>MUL r2</td>
<td>2</td>
<td>EA &lt; EA x A,</td>
</tr>
<tr>
<td>482E</td>
<td>MUL r2</td>
<td>2</td>
<td>EA &lt; EA x B,</td>
</tr>
<tr>
<td>482F</td>
<td>MUL r2</td>
<td>2</td>
<td>EA &lt; EA x C,</td>
</tr>
<tr>
<td>4831</td>
<td>RLR r2</td>
<td>2</td>
<td>A rotate logical right</td>
</tr>
<tr>
<td>4832</td>
<td>RLR r2</td>
<td>2</td>
<td>B rotate logical right</td>
</tr>
<tr>
<td>4833</td>
<td>RLR r2</td>
<td>2</td>
<td>C rotate logical right</td>
</tr>
<tr>
<td>4835</td>
<td>RLL r2</td>
<td>2</td>
<td>A rotate logical left</td>
</tr>
<tr>
<td>4836</td>
<td>RLL r2</td>
<td>2</td>
<td>B rotate logical left</td>
</tr>
<tr>
<td>4837</td>
<td>RLL r2</td>
<td>2</td>
<td>C rotate logical left</td>
</tr>
<tr>
<td>4838</td>
<td>RLD</td>
<td>2</td>
<td>A, (HL) rotate left digit</td>
</tr>
<tr>
<td>4839</td>
<td>RRD</td>
<td>2</td>
<td>A, (HL) rotate right digit</td>
</tr>
<tr>
<td>483A</td>
<td>NEGA</td>
<td>2</td>
<td>Negate A, add 1 (two's complement)</td>
</tr>
<tr>
<td>483B</td>
<td>HLT</td>
<td>1</td>
<td>Set Halt mode</td>
</tr>
<tr>
<td>483D</td>
<td>DIV r2</td>
<td>2</td>
<td>EA &lt; EA div A, A &lt; remainder</td>
</tr>
<tr>
<td>483E</td>
<td>DIV r2</td>
<td>2</td>
<td>EA &lt; EA div B, B &lt; remainder</td>
</tr>
<tr>
<td>483F</td>
<td>DIV r2</td>
<td>2</td>
<td>EA &lt; EA div C, C &lt; remainder</td>
</tr>
<tr>
<td>4840</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if NMI = 1,</td>
</tr>
<tr>
<td>Line</td>
<td>Instruction</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>48 41</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FT0 = 1,</td>
</tr>
<tr>
<td>48 42</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FT1 = 1,</td>
</tr>
<tr>
<td>48 43</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if F1 = 1,</td>
</tr>
<tr>
<td>48 44</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if F2 = 1,</td>
</tr>
<tr>
<td>48 45</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FE0 = 1,</td>
</tr>
<tr>
<td>48 46</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FE1 = 1,</td>
</tr>
<tr>
<td>48 47</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FEIN = 1,</td>
</tr>
<tr>
<td>48 48</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FAD = 1,</td>
</tr>
<tr>
<td>48 49</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FSR = 1,</td>
</tr>
<tr>
<td>48 4A</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if FST = 1,</td>
</tr>
<tr>
<td>48 4B</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if ER = 1,</td>
</tr>
<tr>
<td>48 4C</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if OV = 1,</td>
</tr>
<tr>
<td>48 50</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if AN4 = 1,</td>
</tr>
<tr>
<td>48 51</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if AN5 = 1,</td>
</tr>
<tr>
<td>48 52</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if AN6 = 1,</td>
</tr>
<tr>
<td>48 53</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if AN7 = 1,</td>
</tr>
<tr>
<td>48 54</td>
<td>SKIT irf</td>
<td>2</td>
<td>Skip if SB = 1,</td>
</tr>
<tr>
<td>48 60</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if NMI = 0,</td>
</tr>
<tr>
<td>48 61</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FT0 = 0,</td>
</tr>
<tr>
<td>48 62</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FT1 = 0,</td>
</tr>
<tr>
<td>48 63</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if F1 = 0,</td>
</tr>
<tr>
<td>48 64</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if F2 = 0,</td>
</tr>
<tr>
<td>48 65</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FE0 = 0,</td>
</tr>
<tr>
<td>48 66</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FE1 = 0,</td>
</tr>
<tr>
<td>48 67</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FEIN = 0,</td>
</tr>
<tr>
<td>48 68</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FAD = 0,</td>
</tr>
<tr>
<td>48 69</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FSR = 0,</td>
</tr>
<tr>
<td>48 6A</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if FST = 0,</td>
</tr>
<tr>
<td>48 6B</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if ER = 0,</td>
</tr>
<tr>
<td>48 6C</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if OV = 0,</td>
</tr>
<tr>
<td>48 70</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if AN4 = 0,</td>
</tr>
<tr>
<td>48 71</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if AN5 = 0,</td>
</tr>
<tr>
<td>48 72</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if AN6 = 0,</td>
</tr>
<tr>
<td>48 73</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if AN7 = 0,</td>
</tr>
<tr>
<td>48 74</td>
<td>SKNIT irf</td>
<td>2</td>
<td>Skip if SB = 0,</td>
</tr>
<tr>
<td>48 82</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(DE), EAH&lt;(DE+1)</td>
</tr>
<tr>
<td>48 83</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(HL), EAH&lt;(HL+1)</td>
</tr>
<tr>
<td>48 84</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(DE+++), EAH&lt;(DE+++ +1)</td>
</tr>
<tr>
<td>48 85</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(HL+++), EAH&lt;(HL+++ +1)</td>
</tr>
<tr>
<td>48 88</td>
<td>LDEAX rpa3</td>
<td>3</td>
<td>EAL&lt;(DE+byte), EAH&lt;(DE+byte+1)</td>
</tr>
<tr>
<td>48 8C</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(HL+A), EAH&lt;(HL+A+1)</td>
</tr>
<tr>
<td>48 8D</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(HL+B), EAH&lt;(HL+B+1)</td>
</tr>
<tr>
<td>48 8E</td>
<td>LDEAX rpa3</td>
<td>2</td>
<td>EAL&lt;(HL+EA), EAH&lt;(HL+EA+1)</td>
</tr>
<tr>
<td>48 8F</td>
<td>LDEAX rpa3</td>
<td>3</td>
<td>EAL&lt;(HL+byte), EAH&lt;(HL+byte+1)</td>
</tr>
<tr>
<td>48 92</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(DE)&lt;EAL, (DE+1)&lt;EAH</td>
</tr>
<tr>
<td>48 93</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(HL)&lt;EAL, (HL+1)&lt;EAH</td>
</tr>
<tr>
<td>48 94</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(DE+++)&lt;EAL, (DE+++ +1)&lt;EAH</td>
</tr>
<tr>
<td>48 95</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(HL+++)&lt;EAL, (HL+++ +1)&lt;EAH</td>
</tr>
<tr>
<td>48 9B</td>
<td>STEAX rpa3</td>
<td>3</td>
<td>(DE+byte)&lt;EAL, (DE+byte+1)&lt;EAH</td>
</tr>
<tr>
<td>48 9C</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(HL+A)&lt;EAL, (HL+A+1)&lt;EAH</td>
</tr>
<tr>
<td>48 9D</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(HL+B)&lt;EAL, (HL+B+1)&lt;EAH</td>
</tr>
<tr>
<td>48 9E</td>
<td>STEAX rpa3</td>
<td>2</td>
<td>(HL+EA)&lt;EAL, (HL+EA+1)&lt;EAH</td>
</tr>
<tr>
<td>48 9F</td>
<td>STEAX rpa3</td>
<td>3</td>
<td>(HL+byte)&lt;EAL, (HL+byte+1)&lt;EAH</td>
</tr>
</tbody>
</table>
48 A0  DSLR  EA  2  EA shift logical right
48 A4  DSLL  EA  2  EA shift logical left
48 A8  TABLE  2  C < (PC+3+A),  B < (PC+3+A+1)
48 B0  DRLR  EA  2  EA rotate logical right
48 B4  DRLL  EA  2  EA rotate logical left
48 BB  STOP  1  Set software Stop mode
48 C0  DMOV  EA,sr4  2  EA < ECNT
48 C1  DMOV  EA,sr4  2  EA < ECPT
48 D2  DMOV  sr3,EA  2  ETM0 < EA
48 D3  DMOV  sr3,EA  2  ETM1 < EA
49 MOV  A,sr1  2  A < sr1  (1 1 s5 s4 s3 s2 s1)
4D  MOV  sr,A  2  sr < A  (1 1 s5 s4 s3 s2 s1)
4F  JRE  2  PC < PC + 2 + disp forward
50  EXH  1  Alternate H, L register sets
51  DCR  r2  1  A < A - 1,  Skip if borrow
52  DCR  r2  1  B < B - 1,  Skip if borrow
53  DCR  r2  1  C < C - 1,  Skip if borrow
54  JMP word  3  PC < word
55  OFFIW  wa,byte  3  (V / offset) AND byte, Skip if zero
56  ACI  A,byte  2  A < A + byte + CY
57  OFFI  A,byte  2  A AND byte,  Skip if zero
5H  BIT bit,wa  2  Skip if (V / offset) bit (b2,b1,b0) is 1
60 0H  ANA  r,A  2  r < r AND A  (r = V, A, B, C, D, E, H, L)
60 1L  XRA  r,A  2  r < r EX-OR A  (r = V, A, B, C, D, E, H, L)
60 1H  ORA  r,A  2  r < r OR A  (r = V, A, B, C, D, E, H, L)
60 2L  ADDNC  r,A  2  r < r + A,  Skip if no carry
60 2H  GTA  r,A  2  r - A - 1,  Skip if no borrow
60 3L  SUBNB  r,A  2  r < r - A,  Skip if no carry
60 3H  LTA  r,A  2  r - A,  Skip if borrow
60 4L  ADD  r,A  2  r < r + A  (r = V, A, B, C, D, E, H, L)
60 5L  ADC  r,A  2  r < r + A + CY  (r = V, A, B, C, D, E, H, L)
60 6L  SUB  r,A  2  r < r - A  (r = V, A, B, C, D, E, H, L)
60 6H  NEA  r,A  2  r - A,  Skip if no zero
60 7L  SBB  r,A  2  r < r - A - CY  (r = V, A, B, C, D, E, H, L)
60 7H  EQA  r,A  2  r - A,  Skip if zero
60 8H  ANA  A,r  2  A < A AND r  (r = V, A, B, C, D, E, H, L)
60 9L  XRA  A,r  2  A < A EX-OR r  (r = V, A, B, C, D, E, H, L)
60 9H  ORA  A,r  2  A < A OR r  (r = V, A, B, C, D, E, H, L)
60 AH  ADDNC  A,r  2  A < A + r,  Skip if no carry
60 BH  GTA  A,r  2  A - r - 1,  Skip if no borrow
60 BL  SUBNB  A,r  2  A < A - r,  Skip if no carry
60 BH  LTA  A,r  2  A - r,  Skip if borrow
60 CL  ADD  A,r  2  A < A + r  (r = V, A, B, C, D, E, H, L)
60 CH  ONA  A,r  2  A AND r,  Skip if no zero
60 DL  ADC  A,r  2  A < A + r + CY  (r = V, A, B, C, D, E, H, L)
60 DH  OFFA  A,r  2  A AND r,  Skip if zero
60 EL  SUB  A,r  2  A < A - r  (r = V, A, B, C, D, E, H, L)
60 EH  NEA  A,r  2  A - r,  Skip if no zero
60 FL  SBB  A,r  2  A < a - r - CY  (r = V, A, B, C, D, E, H, L)
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 FH</td>
<td>EQA A,r</td>
<td>2 A - r, Skip if zero</td>
</tr>
<tr>
<td>61</td>
<td>DAA</td>
<td>1 Decimal adjust A</td>
</tr>
<tr>
<td>62</td>
<td>RETI</td>
<td>1 Return from Interrupt</td>
</tr>
<tr>
<td>63</td>
<td>STAW wa</td>
<td>2 (V / offset) &lt; A</td>
</tr>
<tr>
<td>64 0L</td>
<td>MVI sr2,byte</td>
<td>3 sr2 &lt; byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 8L</td>
<td>MVI sr2,byte</td>
<td>3 sr2 &lt; byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 0H</td>
<td>ANI sr2,byte</td>
<td>3 sr2 &lt; sr2 AND byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 8H</td>
<td>ANI sr2,byte</td>
<td>3 sr2 &lt; sr2 AND byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 1L</td>
<td>XRI sr2,byte</td>
<td>3 sr2 &lt; sr2 EX-OR byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 9L</td>
<td>XRI sr2,byte</td>
<td>3 sr2 &lt; sr2 EX-OR byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 1H</td>
<td>ORI sr2,byte</td>
<td>3 sr2 &lt; sr2 OR byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 9H</td>
<td>ORI sr2,byte</td>
<td>3 sr2 &lt; sr2 OR byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 2L</td>
<td>ADINC sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte, Skip if no carry (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 AL</td>
<td>ADINC sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte, Skip if no carry (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 2H</td>
<td>GTI sr2,byte</td>
<td>3 sr2 - byte-1, Skip if no borrow (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 AH</td>
<td>GTI sr2,byte</td>
<td>3 sr2 - byte-1, Skip if no borrow (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 3L</td>
<td>SUINB sr2,byte</td>
<td>3 sr2 &lt; sr2 - byte, Skip if no borrow (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 BL</td>
<td>SUINB sr2,byte</td>
<td>3 sr2 &lt; sr2 - byte, Skip if no borrow (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 3H</td>
<td>LTI sr2,byte</td>
<td>3 sr2 - byte, Skip if borrow (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 BH</td>
<td>LTI sr2,byte</td>
<td>3 sr2 - byte, Skip if borrow (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 4L</td>
<td>ADI sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 CL</td>
<td>ADI sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 4H</td>
<td>ONI sr2,byte</td>
<td>3 sr2 AND byte, Skip if no zero (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 CH</td>
<td>ONI sr2,byte</td>
<td>3 sr2 AND byte, Skip if no zero (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 5L</td>
<td>ACI sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte + CY (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 DL</td>
<td>ACI sr2,byte</td>
<td>3 sr2 &lt; sr2 + byte + CY (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 5H</td>
<td>OFFI sr2,byte</td>
<td>3 sr2 AND byte, Skip if zero (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 DH</td>
<td>OFFI sr2,byte</td>
<td>3 sr2 AND byte, Skip if zero (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 6L</td>
<td>SUI sr2,byte</td>
<td>3 sr2 &lt; sr2 - byte (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>64 EL</td>
<td>SUI sr2,byte</td>
<td>3 sr2 &lt; sr2 - byte (ANM,SMH,-,EOM, -,TMM,-,-)</td>
</tr>
<tr>
<td>64 6H</td>
<td>NEI sr2,byte</td>
<td>3 sr2 - byte, Skip if no zero (PA,PB,PC,PD, -,PF,MKH,MKL)</td>
</tr>
<tr>
<td>Address</td>
<td>Instruction</td>
<td>Flags</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>64 EH</td>
<td>NEI sr2,byte</td>
<td>3</td>
</tr>
<tr>
<td>64 7L</td>
<td>SBI sr2,byte</td>
<td>3</td>
</tr>
<tr>
<td>64 FL</td>
<td>SBI sr2,byte</td>
<td>3</td>
</tr>
<tr>
<td>64 7H</td>
<td>EQI sr2,byte</td>
<td>3</td>
</tr>
<tr>
<td>64 FH</td>
<td>EQI sr2,byte</td>
<td>3</td>
</tr>
<tr>
<td>65</td>
<td>NEIW wa,byte</td>
<td>3</td>
</tr>
<tr>
<td>66</td>
<td>SUI A,byte</td>
<td>2</td>
</tr>
<tr>
<td>67</td>
<td>NEI A,byte</td>
<td>2</td>
</tr>
<tr>
<td>68</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>69</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6A</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6B</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6C</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6D</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6E</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>6F</td>
<td>MVI r,byte</td>
<td>2</td>
</tr>
<tr>
<td>70 0E</td>
<td>SSPD word</td>
<td>4</td>
</tr>
<tr>
<td>70 0F</td>
<td>LSPD word</td>
<td>4</td>
</tr>
<tr>
<td>70 1E</td>
<td>SBCD word</td>
<td>4</td>
</tr>
<tr>
<td>70 1F</td>
<td>LBCD word</td>
<td>4</td>
</tr>
<tr>
<td>70 2E</td>
<td>SDED word</td>
<td>4</td>
</tr>
<tr>
<td>70 2F</td>
<td>LDDED word</td>
<td>4</td>
</tr>
<tr>
<td>70 3E</td>
<td>SHLD word</td>
<td>4</td>
</tr>
<tr>
<td>70 3F</td>
<td>LHLD word</td>
<td>4</td>
</tr>
<tr>
<td>70 41</td>
<td>EADD EA,r2</td>
<td>2</td>
</tr>
<tr>
<td>70 42</td>
<td>EADD EA,r2</td>
<td>2</td>
</tr>
<tr>
<td>70 43</td>
<td>EADD EA,r2</td>
<td>2</td>
</tr>
<tr>
<td>70 61</td>
<td>ESUB EA,rp3</td>
<td>2</td>
</tr>
<tr>
<td>70 62</td>
<td>ESUB EA,rp3</td>
<td>2</td>
</tr>
<tr>
<td>70 63</td>
<td>ESUB EA,rp3</td>
<td>2</td>
</tr>
<tr>
<td>70 6H</td>
<td>MOV r,word</td>
<td>4</td>
</tr>
<tr>
<td>70 7H</td>
<td>MOV word,r</td>
<td>4</td>
</tr>
<tr>
<td>70 8H</td>
<td>ANAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 9L</td>
<td>XRAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 9H</td>
<td>ORAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 AL</td>
<td>ADDNCX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 AH</td>
<td>GTAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 BL</td>
<td>SUBNBX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 BH</td>
<td>LTAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 CL</td>
<td>ADDX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 CH</td>
<td>ONAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 DL</td>
<td>ADCX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 DH</td>
<td>OFFAX rpa</td>
<td>2</td>
</tr>
<tr>
<td>70 EL</td>
<td>SUBX rpa</td>
<td>2</td>
</tr>
<tr>
<td>Address</td>
<td>Opcode</td>
<td>Op1</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>70 EH</td>
<td>NEAX</td>
<td>rpa</td>
</tr>
<tr>
<td>70 FL</td>
<td>SBBX</td>
<td>rpa</td>
</tr>
<tr>
<td>70 FH</td>
<td>EQAX</td>
<td>rpa</td>
</tr>
<tr>
<td>71</td>
<td>MVIW</td>
<td>wa,byte</td>
</tr>
<tr>
<td>72</td>
<td>SOFTI</td>
<td></td>
</tr>
<tr>
<td>74 0L</td>
<td>ANI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 1L</td>
<td>XRI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 1H</td>
<td>ORI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 2L</td>
<td>ADINC</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 2H</td>
<td>GTI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 3L</td>
<td>SUINB</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 3H</td>
<td>LTI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 4L</td>
<td>ADI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 4H</td>
<td>ONI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 5L</td>
<td>ACI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 5H</td>
<td>OFFI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 6L</td>
<td>SU1</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 6H</td>
<td>NE1</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 7L</td>
<td>SBI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 7H</td>
<td>EQI</td>
<td>r,byte</td>
</tr>
<tr>
<td>74 88</td>
<td>ANAW</td>
<td>wa</td>
</tr>
<tr>
<td>74 8D</td>
<td>DAN</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 8E</td>
<td>DAN</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 8F</td>
<td>DAN</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 90</td>
<td>XRAW</td>
<td>wa</td>
</tr>
<tr>
<td>74 95</td>
<td>DXR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 96</td>
<td>DXR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 97</td>
<td>DXR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 98</td>
<td>ORAW</td>
<td>wa</td>
</tr>
<tr>
<td>74 9D</td>
<td>DOR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 9E</td>
<td>DOR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 9F</td>
<td>DOR</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 A0</td>
<td>ADDNCW</td>
<td>wa</td>
</tr>
<tr>
<td>74 A5</td>
<td>DADDNC</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 A6</td>
<td>DADDNC</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 A7</td>
<td>DADDNC</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 A8</td>
<td>GTAW</td>
<td>wa</td>
</tr>
<tr>
<td>74 AD</td>
<td>DGT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 AE</td>
<td>DGT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 AF</td>
<td>DGT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 B0</td>
<td>SUBNBW</td>
<td>wa</td>
</tr>
<tr>
<td>74 B5</td>
<td>DSUBNB</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 B6</td>
<td>DSUBNB</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 B7</td>
<td>DSUBNB</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 B8</td>
<td>LTAW</td>
<td>wa</td>
</tr>
<tr>
<td>74 BD</td>
<td>DLT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 BE</td>
<td>DLT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 BF</td>
<td>DLT</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 C0</td>
<td>ADDW</td>
<td>wa</td>
</tr>
<tr>
<td>74 C5</td>
<td>DADD</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 C6</td>
<td>DADD</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 C7</td>
<td>DADD</td>
<td>EA,rp3</td>
</tr>
<tr>
<td>74 C8</td>
<td>ONAW</td>
<td>wa</td>
</tr>
<tr>
<td>Code</td>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>74 CD</td>
<td>DON EA,rp3 2</td>
<td>EA AND BC, Skip if no zero</td>
</tr>
<tr>
<td>74 CE</td>
<td>DON EA,rp3 2</td>
<td>EA AND DE, Skip if no zero</td>
</tr>
<tr>
<td>74 CF</td>
<td>DON EA,rp3 2</td>
<td>EA AND HL, Skip if no zero</td>
</tr>
<tr>
<td>74 D0</td>
<td>ADCW wa 3</td>
<td>A &lt; A + (V / offset) + CY</td>
</tr>
<tr>
<td>74 D5</td>
<td>DADC EA,rp3 2</td>
<td>EA &lt; EA + BC + CY</td>
</tr>
<tr>
<td>74 D6</td>
<td>DADC EA,rp3 2</td>
<td>EA &lt; EA + DE + CY</td>
</tr>
<tr>
<td>74 D7</td>
<td>DADC EA,rp3 2</td>
<td>EA &lt; EA + HL + CY</td>
</tr>
<tr>
<td>74 D8</td>
<td>OFFAW wa 3</td>
<td>A AND (V / offset), Skip if zero</td>
</tr>
<tr>
<td>74 DD</td>
<td>DOFF EA,rp3 2</td>
<td>EA AND BC, Skip if zero</td>
</tr>
<tr>
<td>74 DE</td>
<td>DOFF EA,rp3 2</td>
<td>EA AND DE, Skip if zero</td>
</tr>
<tr>
<td>74 DF</td>
<td>DOFF EA,rp3 2</td>
<td>EA AND HL, Skip if zero</td>
</tr>
<tr>
<td>74 E0</td>
<td>SUBW wa 3</td>
<td>A &lt; A - (V / offset)</td>
</tr>
<tr>
<td>74 E5</td>
<td>DSUB EA,rp3 2</td>
<td>EA &lt; EA - BC</td>
</tr>
<tr>
<td>74 E6</td>
<td>DSUB EA,rp3 2</td>
<td>EA &lt; EA - DE</td>
</tr>
<tr>
<td>74 E7</td>
<td>DSUB EA,rp3 2</td>
<td>EA &lt; EA - HL</td>
</tr>
<tr>
<td>74 E8</td>
<td>NEAW wa 3</td>
<td>A - (V / offset), Skip if no zero</td>
</tr>
<tr>
<td>74 ED</td>
<td>DNE EA,rp3 2</td>
<td>EA - BC, Skip if no zero</td>
</tr>
<tr>
<td>74 EE</td>
<td>DNE EA,rp3 2</td>
<td>EA - DE, Skip if no zero</td>
</tr>
<tr>
<td>74 EF</td>
<td>DNE EA,rp3 2</td>
<td>EA - HL, Skip if no zero</td>
</tr>
<tr>
<td>74 F0</td>
<td>SBBW wa 3</td>
<td>A &lt; A - (V / offset) - CY</td>
</tr>
<tr>
<td>74 F5</td>
<td>DSBB EA,rp3 2</td>
<td>EA &lt; EA - BC - CY</td>
</tr>
<tr>
<td>74 F6</td>
<td>DSBB EA,rp3 2</td>
<td>EA &lt; EA - DE - CY</td>
</tr>
<tr>
<td>74 F7</td>
<td>DSBB EA,rp3 2</td>
<td>EA &lt; EA - HL - CY</td>
</tr>
<tr>
<td>74 F8</td>
<td>EQAW wa 3</td>
<td>A - (V / offset), Skip if zero</td>
</tr>
<tr>
<td>74 FD</td>
<td>DEQ EA,rp3 2</td>
<td>EA - BC, Skip if zero</td>
</tr>
<tr>
<td>74 FE</td>
<td>DEQ EA,rp3 2</td>
<td>EA - DE, Skip if zero</td>
</tr>
<tr>
<td>74 FF</td>
<td>DEQ EA,rp3 2</td>
<td>EA - HL, Skip if zero</td>
</tr>
<tr>
<td>75</td>
<td>EIQW wa,byte 3</td>
<td>(V / offset) - byte, Skip if zero</td>
</tr>
<tr>
<td>76</td>
<td>SBI A,byte 2</td>
<td>A &lt; A - byte - CY</td>
</tr>
<tr>
<td>77</td>
<td>EQI A,byte 2</td>
<td>A - byte, Skip if zero</td>
</tr>
<tr>
<td>78</td>
<td>CALF word 2</td>
<td>Subroutine call to 8xx</td>
</tr>
<tr>
<td>79</td>
<td>CALF word 2</td>
<td>Subroutine call to 9xx</td>
</tr>
<tr>
<td>7A</td>
<td>CALF word 2</td>
<td>Subroutine call to Axx</td>
</tr>
<tr>
<td>7B</td>
<td>CALF word 2</td>
<td>Subroutine call to Bxx</td>
</tr>
<tr>
<td>7C</td>
<td>CALF word 2</td>
<td>Subroutine call to Cxx</td>
</tr>
<tr>
<td>7D</td>
<td>CALF word 2</td>
<td>Subroutine call to Dxx</td>
</tr>
<tr>
<td>7E</td>
<td>CALF word 2</td>
<td>Subroutine call to Exx</td>
</tr>
<tr>
<td>7F</td>
<td>CALF word 2</td>
<td>Subroutine call to Fxx</td>
</tr>
<tr>
<td>80</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 80</td>
</tr>
<tr>
<td>81</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 82</td>
</tr>
<tr>
<td>82</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 84</td>
</tr>
<tr>
<td>83</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 86</td>
</tr>
<tr>
<td>84</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 88</td>
</tr>
<tr>
<td>85</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 8A</td>
</tr>
<tr>
<td>86</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 8C</td>
</tr>
<tr>
<td>87</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 8E</td>
</tr>
<tr>
<td>88</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 90</td>
</tr>
<tr>
<td>89</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 92</td>
</tr>
<tr>
<td>8A</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 94</td>
</tr>
<tr>
<td>8B</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 96</td>
</tr>
<tr>
<td>8C</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 98</td>
</tr>
<tr>
<td>8D</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 9A</td>
</tr>
<tr>
<td>8E</td>
<td>CALT 1</td>
<td>Subroutine call to Jump Table 9C</td>
</tr>
<tr>
<td>8F</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>91</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>92</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>93</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>94</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>95</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>96</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>97</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>98</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9A</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9B</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9C</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9D</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9E</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>9F</td>
<td>CALT</td>
<td>1</td>
</tr>
<tr>
<td>A0</td>
<td>POP rp1</td>
<td>1</td>
</tr>
<tr>
<td>A1</td>
<td>POP rp1</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>POP rp1</td>
<td>1</td>
</tr>
<tr>
<td>A3</td>
<td>POP rp1</td>
<td>1</td>
</tr>
<tr>
<td>A4</td>
<td>POP rp1</td>
<td>1</td>
</tr>
<tr>
<td>A5</td>
<td>DMOV EA,rp3</td>
<td>1</td>
</tr>
<tr>
<td>A6</td>
<td>DMOV EA,rp3</td>
<td>1</td>
</tr>
<tr>
<td>A7</td>
<td>DMOV EA,rp3</td>
<td>1</td>
</tr>
<tr>
<td>A8</td>
<td>INX EA</td>
<td>1</td>
</tr>
<tr>
<td>A9</td>
<td>DCX EA</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>EI</td>
<td>1</td>
</tr>
<tr>
<td>AB</td>
<td>LDAX rpa2</td>
<td>2</td>
</tr>
<tr>
<td>AC</td>
<td>LDAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>AD</td>
<td>LDAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>AE</td>
<td>LDAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>AF</td>
<td>LDAX rpa2</td>
<td>2</td>
</tr>
<tr>
<td>B0</td>
<td>PUSH rp1</td>
<td>1</td>
</tr>
<tr>
<td>B1</td>
<td>PUSH rp1</td>
<td>1</td>
</tr>
<tr>
<td>B2</td>
<td>PUSH rp1</td>
<td>1</td>
</tr>
<tr>
<td>B3</td>
<td>PUSH rp1</td>
<td>1</td>
</tr>
<tr>
<td>B4</td>
<td>PUSH rp1</td>
<td>1</td>
</tr>
<tr>
<td>B5</td>
<td>DMOV rp3,EA</td>
<td>1</td>
</tr>
<tr>
<td>B6</td>
<td>DMOV rp3,EA</td>
<td>1</td>
</tr>
<tr>
<td>B7</td>
<td>DMOV rp3,EA</td>
<td>1</td>
</tr>
<tr>
<td>B8</td>
<td>RET</td>
<td>1</td>
</tr>
<tr>
<td>B9</td>
<td>RETS</td>
<td>1</td>
</tr>
<tr>
<td>BA</td>
<td>DI</td>
<td>1</td>
</tr>
<tr>
<td>BB</td>
<td>STAX rpa2</td>
<td>2</td>
</tr>
<tr>
<td>BC</td>
<td>STAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>BD</td>
<td>STAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>BE</td>
<td>STAX rpa2</td>
<td>1</td>
</tr>
<tr>
<td>BF</td>
<td>STAX rpa2</td>
<td>2</td>
</tr>
<tr>
<td>C0</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C1</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C3</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>Code</td>
<td>Location</td>
<td>Jump</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>C4</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C5</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C6</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C7</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C8</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>C9</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CA</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CB</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CC</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CD</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CE</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>CF</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D0</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D1</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D3</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D4</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D6</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D7</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D8</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>D9</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DA</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DB</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DC</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DD</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>DF</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E0</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E1</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E2</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E3</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E4</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E5</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E6</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E7</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E8</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>E9</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>EA</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>EB</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>EC</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>ED</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>EE</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>EF</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F0</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F1</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F2</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F3</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F4</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F5</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F6</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F7</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>F8</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>Character</td>
<td>Jump Value</td>
<td>Jump</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>F9</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FA</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FB</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FC</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FD</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FE</td>
<td>JR</td>
<td>1</td>
</tr>
<tr>
<td>FF</td>
<td>JR</td>
<td>1</td>
</tr>
</tbody>
</table>
Modified
Roland PG1000
eForth Programming Tool
eForth 1.0 by Bill Muench and C. H. Ting, 1990
Much of the code is derived from the following sources:
  8086 figForth by Thomas Newman, 1981 and Joe smith, 1983
  aFORTH by John Rible
  bFORTH by Bill Muench

eForth is a small portable Forth design for a wide range of
microprocessors.

The goal of this implementation is to provide a simple eForth Model
which can be ported easily to many 8, 16, 24 and 32 bit CPU's.
The following attributes make it suitable for CPU's of the '90:

  small machine dependent kernel and portable high level code
  source code in the MASM format
  direct threaded code
  separated code and name dictionaries
  simple vectored terminal and file interface to host computer
  aligned with the proposed ANS Forth Standard
  easy upgrade path to optimize for specific CPU

You are invited to implement this Model on your favorite CPU and
contribute it to the eForth Library for public use. You may use
a portable implementation to advertise more sophisticated and
optimized version for commercial purposes. However, you are
expected to implement the Model faithfully. The eForth Working
Group reserves the right to reject implementation which deviates
significantly from this Model.

As the ANS Forth Standard is still evolving, this Model will
change accordingly. Implementations must state clearly the
version number of the Model being tracked.

Representing the eForth Working Group in the Silicon Valley FIG Chapter.
Send contributions to:

  Dr. C. H. Ting
  156 14th Avenue
  San Mateo, CA 94402
  (415) 571-7639

This disk and the companion manual 'eForth Implementation Guide'
are available from Offete enterprises, Inc., 1306 South B Street,
San Mateo, CA 94402, (415)574-8250 for $25. The other implementation
8051 eForth and its manual are also distributed by Offete for $25.
eForth Glossary

Derived from bFORTH by Bill Muench, 1990.

WARNING: Advanced information -- subject to change.

Attributes  Capitalized symbols.
  
  C the word may only be used during compilation of a colon definition.
  D the word is a defining word.
  I the word is IMMEDIATE and will execute during compilation, unless special action is taken.
  U a user value.

================================================================

Stack notes

( compile \ run \ child ;Return ;Float ; <input stream> )

( before -- after ;R before -- after ;F before -- after ; <string> )

================================================================

Glossary

!( w a -- )"store"
Store a 16-bit number at aligned address.

!CSP( -- )"set c s p"
Save the values of the current stack pointers.

!IO( -- )"store i o"
Initialize the serial I/O device.

#( d -- d )"number sign"
Convert one digit of a number using the current base. Must be used within <# and #>. 

#>( d -- b u )"number sign greater"
Terminate a numeric conversion.

#S( ud -- 0 0 )"number sign s"
Convert all digits of a number using the current base.

#TIB( -- a )"number t i b"
The system double variable which holds the size and aligned address of the terminal input buffer.

$"( -- ; <string> \ -- $ )I,C"string quote"
Used only within a definition to compile an inline packed string terminated by the " double quote character. At run-time the address of the packed string is pushed on the data stack.

$"( -- $ )C"string quote primitive"
Return the address of a compiled inline packed string. The run-time primitive compiled by " .

$,,"( -- ; <string> )"string comma quote"
Compile an inline packed character string into the code area, terminated by the " double quote character.

$,n( $ -- )"string comma n"
Create a name for a definition using string. Set the code pointer to the next free cell in the code area, no code is compiled. The name is not linked into the dictionary.

$COMPILE( $ -- )
Convert a string to a word address. Execute the word in interpreting mode or compile it if in compiling mode.

$INTERPRET( $ -- )"string interpret"
At the interactive level, if a word is defined perform its action. If not, try to convert it to a number, if
that fails, issue an error message.

`(-- ca ; <string> )"tick"
Return the code address of the word following.

`?KEY( -- a )U"tick question key"
The system input device status vector.

`BOOT( -- a )
Return the address of a system boot-up routine.

`ECHO( -- a )U"tick echo"
The system echo device vector.

`EMIT( -- a )U"tick emit"
The system output device vector.

`EVAL( -- a )U"tick eval"
The system interpret/compile vector.

`EXPECT( -- a )U"tick expect"
The system line input vector.

`NUMBER( -- a )U"tick number"
The system number conversion vector.

`PROMPT( -- a )U"tick prompt"
The system prompt vector.

`TAP( -- a )U"tick tap"
The input case function vector.

`(( -- ; <string> )I"paren"
Begin a comment. The comment is terminated by the ) character. May be used inside or outside a definition.

`(* n n -- n )"star"
Multiply two signed numbers. Return a 16-bit signed number.

`*/( n1 n2 n3 -- q )"star slash"
Multiply n1 by n2 producing the 32-bit intermediate product d. Divide d by n3 producing a 16-bit quotient.

`*/MOD( n1 n2 n3 -- r q )"star slash mod"
Multiply n1 by n2 producing the 32-bit intermediate product d. Divide d by n3 producing a 16-bit remainder and a 16-bit quotient.

`+( w w -- w )"plus"
Addition.

`+!( n a -- )"plus store"
Increment the 16-bit value at address by n.

`,( w -- )"comma"
Compile a 16-bit value into the code area.

`-( w w -- w )"minus"
Subtract the top from the second element on the data stack.

`-TRAILING( b u -- b u )"dash trailing"
Adjust the count to eliminate any trailing white-space in the string.

`(. n -- )"dot"
Display the single value, use the current base. If BASE is DECIMAL, display as a signed number.

`.( -- ; <string> )I,C"dot quote"
Used only within a definition to compile an inline packed string terminated by the " double quote character. At run-time the string is displayed on the current output device.
Display a compiled inline packed string to the current output device. This run-time primitive is compiled by : ".

Begin a comment that is displayed to the current output device. May be used inside or outside a definition.

Display the packed string at address.

Display the standard system prompt.

Display the single value right-justified in a field of width +n, use the current base.

Display the contents of the data stack.

Floored division for 16-bit numbers. Returns only the 16-bit quotient.

Floored division for 16-bit numbers. 16-bit remainder and 16-bit quotient.

Return true if n is less than 0, negative. Comparison is signed. Also used for sign extension.

Return true if w is equal to 0.

Store a 32-bit value at aligned address.

Return the 32-bit value stored at aligned address.

Pop the 32-bit number, or the top two 16-bit numbers, from the data stack.

Duplicate the 32-bit number, or the top two 16-bit numbers, on the data stack.

Begin a colon definition to be added to the current vocabulary.

Terminate a colon definition begun with :.

Return true if n1 is less than n2. Comparison is signed.

Begin a numeric conversion.

Return true if w1 is equal to w2.

Convert a value to a printable character. Replace an unprintable character with the _ underscore character.

The pointer into the input stream.

The pointer into the input stream.
If possible, convert a code address to a name address. If not possible, return a false flag.

>\text{R}( \text{w} \rightarrow ; \text{R} \rightarrow \text{w} ) \text{"to r"}

Pop the top element of the data stack and Push it on the return stack.

?\text{( a } \rightarrow \text{ )"question"}

Display the single value stored at address, use the current base. If BASE is DECIMAL , display as a signed number.

?\text{branch( f } \rightarrow \text{ )"question branch"}

Run time routine to redirect execution to the address in the next cell if flag is false.

?\text{CSP( -- )"question c s p"}

Compare the current value of the stack pointers with the saved values. ABORT with an error message if different.

?\text{DUP( w } \rightarrow \text{ w, 0 )"question dupe"}

Duplicate the number on top of the data stack only if it is non-zero.

?\text{KEY( -- t )"question key"}

Return the status of the current input device.

?\text{RX( -- c T, F )"question r x"}

Return a character from the input device and true. Return false only if no character is pending.

?\text{STACK( -- )"question stack"}

Display an error message if the stack limits have been exceeded.

?\text{UNIQUE( $ } \rightarrow \text{ $ )"question unique"}

Display a warning massage for a duplicate definition.

@( a } \rightarrow \text{ w )"fetch"}

Return the 16-bit value stored at aligned address.

@\text{EXECUTE( a } \rightarrow \text{ )"fetch execute"}

Fetch the execution token stored at address and execute it, ie indirect execution. If the value contained in address is zero, do nothing.

\text{ABORT( -- )}

Reset the data stack and perform the function of QUIT . Note, no message is displayed.

\text{ABORT( -- ; <string> \f \rightarrow I,C"abort quote"}

Used only within a definition to compile an inline packed string terminated the " double quote character. At run-time, if the flag is false, execute the sequence of words following the string. Otherwise, the string is displayed on the current output ce, execution is then passed to an error handling routine.

\text{abort( f \rightarrow C"abort quote primitive"}

The run-time primitive compiled by ABORT" .

\text{ABS( n } \rightarrow \text{ +n )}

Return the absolute value of n.

\text{accept( b u } \rightarrow \text{ b u )}

Receive a line of u characters maximum to the an input buffer at byte address. Terminate input if a carriage return is received. Return the actual count of received characters. Perform any currently defined keyboard macros. Use the current input device.

\text{AFT( a } \rightarrow \text{ a \rightarrow I,C}

Used within a loop structure to unconditional skip a portion of code the first time thru the loop. AFT compiles the machine unconditional branch instruction and leaves an address to be resolved by THEN .

\text{AGAIN( a } \rightarrow \text{ \rightarrow I,C}

Terminate an infinite loop structure. AGAIN compiles an unconditional branch instruction, and uses the address left by BEGIN to resolve this backward branch.
AHEAD( -- a \-- )I,C
Mark the beginning of a forward branching, unconditional branch structure. AHEAD compiles the machine
unconditional branch instruction and leaves an address to be resolved by THEN.

ALIGNED( b -- a )
Convert a byte address to a word aligned address.

ALLOC( n -- )
Adjust the code area pointer by n.

AND( w w -- w )
A bitwise logical AND.

BASE( -- a )U
The system variable which holds the current numeric conversion radix.

BEGIN( -- a \-- )I,C
Mark the beginning of an indefinite loop structure. Leave an address to be resolved by UNTIL, or WHILE and
REPEAT.

BL( -- c )"b l"
Push the value of a space, the blank character, on the data stack.

branch( -- )C
Run time routine to redirect execution to the address in the next cell.

BYE( -- )
Exit Forth and return to the underlying environment or DOS.

C!( v b -- )"c store"
Store an byte value at byte address.

C@( b -- v )"c fetch"
Return the byte value stored at byte address.

CALL( a -- )C"call comma"
Assemble a 4 byte subroutine call to the designated address.

CATCH( ca -- err#/0 )
Setup a local error frame and execute the word referenced by the execution token ca. Return a non-zero error
number or zero for no error.

CELL+( a1 -- a2 )"cell plus"
Add cell size in bytes to address a1.

CELL-( a1 -- a2 )"cell minus"
Subtract cell size in bytes to address a1.

CELLS( n1 -- n2 )
Multiply n1 by the cell size in byte.

CHAR( -- c ; <string> )
Return the value of the first character in <string>. If used within a definition, use the phrase [ CHAR
<string> ] LITERAL.

CHARS( +n c -- )
Display +n of character to the current output device.

CMOVE( b1 b2 u -- )"c move"
Move u byte values from byte address b1 to b2, proceeding from lower to higher memory. Overwrite occurs if
b1<b2-b1+u.

COLD( -- )
Completely re-initialize the system, but does not re-load the system from ROM.

COMPILE( -- )C
Used only within a definition. At run-time the word following COMPILE is not executed, but its code address is copied into the code area.

**CONSOLE( -- )**
Initialize the vectored input and output devices to a terminal.

**CONTEXT( -- a )**
The variable used to specify the dictionary search order.

**COUNT( $ -- b +n )**
Return the byte address and byte count of a packed string.

**CP( -- a )“c p”**
The pointer to the next available dictionary location in code space. Since code and names are separated, the traditional DP, dictionary pointer, had to be split into CP, code pointer, and NP, name pointer.

**CRC( -- )”carriage return”**
Position the cursor at the beginning of the next line of the current output device.

**CREATE( -- ; <string> \ -- a )D**
Build a named definition. At run-time the address pointing to next available code space is pushed on the data stack.

**CSP( -- a )“c s p”**
The system variable which holds the current stack pointer. Used for error checking.

**CURRENT( -- a )**
The variable used to specify the vocabulary in which new definitions are compiled.

**DECIMAL( -- )**
Set decimal as the current BASE. Base 10.

**DEPTH( -- n )**
The number of elements on the data stack, does not include n.

**DIGIT( u -- c )**
Convert a single digit number to its character value.

**DIGIT?( c base -- v t )”digit question”**
Try to convert a character to binary digit. Return true if the digit is valid for the current base.

**dm+( a u -- p a+ )”d m plus”**
Display the 16-bit values starting at the aligned address a.

**DNEGATE( d -- -d )”d negate”**
Return the two’s complement a double number. Change the sign of a double number.

**do$( -- $ )C”do string”**
Return the address of a compiled inline packed string.

**doLIST( a -- )C”do list”**
The run-time routine which executes the list in a colon definition pointed to by a.

**doLIT( -- n )C”do literal”**
Return the in-line literal compiled by LITERAL.

**doUSER( -- )C”do user variable”**
The run-time action of user variables.

**doVAR( -- a )C”do variable”**
The run-time action of variable.

**DROP( w -- )**
Remove the top element on the data stack.

**DUMP( a u -- )**
Display the HEX and character values starting at aligned address a, for count u.

DUP( w -- w w )  "dupe"
Duplicate the top element on the data stack.

ELSE( -- \ a -- a )I,C
Used within a conditional branch structure. ELSE resolves a forward conditional branch compiled by IF . ELSE then compiles an unconditional branch instruction, leaving an address to be resolve by THEN .

EMIT( w -- )
Output a character to the current output device.

EVAL( -- )
Interpret or compile the tokens from the input stream.

EXECUTE( w -- ;R -- w )
Execute the word definition indicated by the execution token w.

EXIT( -- ;R w -- )
Compile a subroutine return.

EXPECT( b u -- )
Receive a line of u characters maximum to the an input buffer at byte address. Terminate input if a carriage return is received. The count of received characters is saved in the variable SPAN. Use the current input device. This word is vectored.

EXTRACT( d base -- d' c )
Used incremen tally to convert each digit in a number to its character value.

FILE( -- )
Specify system input from a file using pace handshake. File input is not echoed, all output messages are displayed.

FILL( b u v -- )
Fill an area at byte address of length u using the byte value v.

find( $ va -- ca f, $ F )  "find primitive"
Given a string and a dictionary entry thread, search for a name match. If found, return the code address and a true flag. If not found, the string address and a false flag.

FOR( u -- )I,C
Begin a down-counting loop. Repeat the loop till NEXT u+1 times from u to 0.

FORTH( -- )
Make the default system vocabulary FORTH the context vocabulary.

HAND( -- )
Specify system input from keyboard, no handshake. All input is echoed.

HANDLER( -- a )
The current error handler frame pointer.

HERE( -- a )
Push the address of the next free cell in the code area on the data stack.

HEX( -- )
Set hexadecimal as the current BASE . Base 16.

hi( -- )
Display the sign-on message.

HLD( -- a )  "h l d"
The pointer to a formatted numeric output string.

HOLD( c -- )
I/O( -- a )"i slash o"
An array used by CONSOLE to initialize the system input and output vectors. The vector order is 'KEY? 'KEY and 'EMIT.

IF( -- a \ f -- )I,C
Mark the beginning of a forward branching, conditional branch structure. IF compiles the machine conditional branch instruction and leaves an address to be resolved by THEN or ELSE.

IMMEDIATE( -- )
Mark the most recently created dictionary entry as a word which will execute during compilation.

INVERT( w -- w )
Bitwise logical invert. Equivalent to -1 XOR. The one's complement.

KEY( -- c )
Return a character from the current input device. If no key is ready, wait until one is available.

kTAP( b b b c -- b b b )"k tap"
The 'tap routine used for file input.

LAST( -- a )
The pointer to the name of the most recently created dictionary entry.

LITERAL( w -- \ -- w )I
Compile a number as an inline value.

M*( n n -- d )"m star"
Multiply two signed numbers. Return a 32-bit signed number.

M/MOD( d n -- r q )"m slash mod"
Floored division of a 32-bit number divided by a 16-bit number. Return a 16-bit quotient and a 16-bit remainder.

MAX( n n -- n )
Leave the greater of the two signed values.

MIN( n n -- n )
Leave the smaller of the two signed values.

MOD( n n -- r )
Floored division for 16-bit numbers. Returns only the 16-bit remainder.

NAME>( na -- ca )"name to code"
Convert a name address to a code address.

NAME?( $ -- ca f, $ F )"name question"
Given a string, search for a name match. If found, return the code address and a true flag. If not found, the string address and a false flag.

NEGATE( n -- -n )
Equivalent to 0 SWAP-. The two's complement of a number. Change the sign of a number.

NEXT( a -- \ -- ;R u -- [u-1] )I,C
Terminate a down-counting loop structure. NEXT compiles the machine loop instruction, pointing to the address left by FOR. The loop count is held on the return stack. Looping continues until the count is equal to zero.

next( -- )C
Run-time routine to terminate a down-counting FOR-NEXT loop. See NEXT.

NP( -- a )"n p"
The pointer to the next available dictionary location in name space.
NUF?( -- t )"nuf question"
Continue until paused or terminated by user. Any key will pause, while paused any key except 'enter' will
restart and return
false, enter will return true.

NULL$( -- $ )"null string"
The address of a string with a zero count.

NUMBER?( $ -- d T, $ F )"number question"
Try to convert a packed string to binary number. If possible return the number and a true. Otherwise, return
the string address
and false. A leading '$' for hexadecimal, a leading '-' for negative, and/or the decimal point for a double
number.

OR( w w -- w )
A bitwise logical OR.

OVER( w1 w2 -- w1 w2 w1 )
Copy the second element to the top of the data stack.

OVERT( -- )
Used by ; to link a successfully defined word into the search order.

PACE( -- )
Send the file transfer handshake character.

PACK$( b u $ -- $ )"pack string"
Move and convert the string at byte address with byte count to a packed string at address $ .

PAD( -- a )
Short for scratch pad. Address of a temporary buffer.

PARSE( c -- b u ;<string> )
Scan the current input stream for the given character as a delimiter. Return the beginning byte address and
count of the delimited string.

parse( b u c -- b u delta ;<string> )
Scan string for the given character as a delimiter. Return the beginning byte address and count of the
delimited string. Delta is the beginning to current offset.

PICK( +n -- w )
Copy the +nth data stack value to the top of the data stack.

PRESET( -- )
Clear the data stack, the return stack and initialize system.

QUERY( -- )
Receive a line to the current input buffer from current input device.

QUIT( -- )
Clear the return stack, set interpret state, and return control to the current command line interpreter.

R>( -- w ;R w -- )"r from"
Pop the top element of the return stack and Push it on the data stack.

RR( -- w ;R w -- w )"r fetch"
Copy the top element of the return stack and Push it on the data stack.

RECURSE( -- \ -- ;R -- a )I,C
Used only within the word currently being defined to allow self reference. Recursion.

REPEAT( a a -- \ -- )I,C
Terminate an indefinite loop structure. REPEAT compiles an unconditional branch instruction, and uses the
address left by WHILE to resolve this backward branch.

ROT( w1 w2 w3 -- w2 w3 w1 )"rote"
Rotate the top three elements on the data stack. Third element to top and all other shifted down.

RP!( a -- )C"r p store"
Set the return stack pointer to address.

RP0( -- a )"r p zero"
Return the bottom address of the return stack pointer.

RP@( -- a )"r p fetch"
Return the address of the return stack pointer.

SAME?( a1 a2 u -- a1' f -0+ )"same question"
Compare the two strings, return the beginning address of the first string and a truth flag.

SIGN( n -- )
Display a minus sign if n is negative. Must be used within <# and #> .

SP!( a -- )"s p store"
Set the data stack pointer to address.

SP0( -- a )"s p zero"
Return the bottom address of the data stack pointer.

SP@( -- a )"s p fetch"
Return the address of the data stack pointer.

SPACE( -- )
Display one space, blank character, to the current output device.

SPACES( +n -- )
Display +n spaces, blank characters, to the current output device.

SPAN( -- a )
The system variable which holds the count of characters input by EXPECT.

str( d -- b u )"s t r"???
Convert number to a string in current BASE . Signed if DECIMAL .

SWAP( w1 w2 -- w2 w1 )
Exchange the top two elements on the data stack.

TAP( b b c -- b b b ' )
Echo and store the keystroke, and update the cursor position.

temp( -- a )U
Return address of a user variable for temporary storage.

THEN( a -- \ -- )I,C
Terminate a conditional branch structure. Resolves a forward branch compiled by IF, ELSE, AHEAD, or AFT.

THROW( err# -- err# )
Reset the state of the system to the current local error frame, and update the error flag.

TIB( -- a )"t i b"
Address of the terminal input buffer.

TOKEN( -- $ ; <string> )
Scan the current input stream for a blank delimited word. Move the word to the end of the names area as a packed string. Return the address of the packed string.

TX!( c -- )"t x store"
Send a character to the output device. Primitive of EMIT.

TYPE( b u -- )
Output u characters of the string at b address to the current output device.
U.( u -- )"u dot"
Display the unsigned single value, use the current base.

U.R( u +n -- )"u dot r"
Display the unsigned single value right-justified in a field of width +n, use the current base.

U<( u1 u2 -- t )"u less"
Return true if u1 is less than u2. Comparison is unsigned.

UM*( u u -- ud )"u m star"
Multiply two unsigned 16-bit numbers. Return an unsigned 32-bit number.

UM+( u u -- ud )"u m plus"
Add two unsigned numbers and return a 32-bit sum.

UM/MOD( ud u -- ur uq )"u m slash mod"
Unsigned division of a 32-bit number divided by a 16-bit number. Return an unsigned 16-bit quotient and an
unsigned 16-bit remainder.

UNTIL( a -- f -- )I,C
Terminate an indefinite loop structure. Condition testing is done after executing the code within the loop.
UNTIL compiles the machine conditional branch instruction, and uses the address left by BEGIN to resolve this
backward branch.

UP( -- a )"u p"
Return the address of the current user area.

USER( u -- ; <string> \ -- a )D
Build a named user variable with an offset from the current user base. At run-time the address of the variable
is pushed on the
data stack.

VARIABLE( -- ; <string> \ -- a )D
Build a named variable. At run-time the address of the variable is pushed on the data stack.

VER( -- n )
Return the version code. Major revision is in the high byte and minor release in the low byte.

VOCABS( -- a )
Return the address of the first vocabulary FORTH in the vocabulary area.

WHILE( a -- a a f -- )I,C
Used within an indefinite loop structure. Condition testing is done before executing the code within the loop.
WHILE compiles the machine conditional branch instruction and leaves an address to be resolved by REPEAT.

WITHIN( u lo hi -- t )
Return true if lo <= u < hi. Comparison is unsigned and circular.

WORD( c -- $ ; <string> )
Scan the current input stream for the string delimited by 'c'. Return the address of the packed string.

WORDS( -- )
Display the words in the CONTEXT vocabulary. Display continues until paused or terminated by user.

XIO( a1 a2 a3 -- )"x i o"
Revector 'prompt, 'echo and 'tap to the code addresses on the stack.

XOR( w w -- w )
Bitwise logical Exclusive OR.

[[ -- ]"left bracket"
Begin interpreting text from the input stream. Change from compiling to interpreting.

[COMPILE]( -- ; <string> \ -- )"bracket compile"
Used only within a definition to force the compilation of the following IMMEDIATE word.
Begin a comment. The comment is terminated by the system end-of-line character. May be used inside or outside a definition.

Change from interpreting to compiling.

A keyboard macro to delete characters from the current input stream. No action is taken if the beginning of the input stream is reached.

Display the the string starting at the byte address b, for count u. Substitute _ the underscore character, for unprintable charact!
78C10 EFORTH
Base Machine Coded Words

Interpreter Pointer DE
Data Stack Pointer SP
Return Stack Pointer HL

Free to use: BC, EA, VA, Alternate Registers

$NEXT 48 84  EA < (DE)++
       48 28  JMP EA

doLIT 48 84  EA < (DE)++
       B4    PUSH EA
       $NEXT

doLIST 33  HL < HL -1
        33  HL < HL -1
        A6  EA < DE
        48 93  (HL) < EA
        A2    POP DE
       $NEXT

COLD 54 0000 JMP Reset Vector
BYE  54 0000 JMP Reset Vector

EXECUTE A1  POP BC
        21  JMP BC

EXIT  48 85  EA < (HL)++
       B6    DE < EA
       $NEXT

next  6A 00  B < 00
      6B 01  C < 01
      48 83  EA < (HL)
      74 B5  EA < EA - BC Skip if no borrow
      C9    JMP NEXT1
      48 93  (HL) < EA
      48 82  EA < (DE)
      B6    DE < EA
       $NEXT

NEXT1 22 22  DE < DE + 2
       32 32  HL < HL + 2
       $NEXT
?branch

6A FF  B < FF
6B FF  C < FF
A4     POP EA
74 CD  EA AND BC  Skip if no zero
C6     JMP BRAN1
22 22  DE < DE + 2
$NEXT

BRAN1 48 82  EA < (DE)
B6     DE < EA
$NEXT

branch 48 82  EA < (DE)
B6     DE < EA
$NEXT

! (w a --)

A6     EA < DE
B5     BC < EA
A2     POP DE (address)
A4     POP EA (data)
48 92  (DE) < EA
A5     EA < BC
B6     DE < EA
$NEXT

@a (a -- w)

A6     EA < DE
B5     BC < EA
A2     POP DE
48 82  EA < (DE)
B4     PUSH EA
A5     EA < BC
B6     DE < EA
$NEXT

C! (w b --)

A1     POP BC (address)
A4     POP AE (data)
09     A < EAL
39     (BC) < A
$NEXT

C@ (b -- c)

A1     POP BC
29     A < (BC)
6A 00  B < 00
1B     C < A
B1     PUSH BC
$NEXT

RP@ (--a)

B3     PUSH HL
$NEXT

RP! (a--)

A3     POP HL
$NEXT

R> (--w)

48 85  EA < (HL)++
B4     PUSH EA
$NEXT

R@ (--w)

48 83  EA < (HL)
B4  PUSH EA
$NEXT

$R (w--)  33 33  HL < HL - 2
A4  POP EA
48 93  (HL) < EA
$NEXT

SP@ (--a)  70 0E FEFF  (FFFE) < SP
70 1F FEFF  BC < (FFFE)
B1  PUSH BC
$NEXT

SP! (a--)  A1  POP BC
70 1E FEFF  (FFFE) < BC
70 0F FEFF  PC < (FFFE)
$NEXT

DROP  A4  POP EA
$NEXT

DUP  A4  POP EA
B4  PUSH EA
B4  PUSH EA
$NEXT

SWAP  A4  POP EA
A1  POP BC
B4  PUSH EA
B1  PUSH BC
$NEXT

OVER  A4  POP AE
A1  POP BC
B1  PUSH BC
B4  PUSH AE
B1  PUSH BC
$NEXT
0< (n--t)  A1  POP BC
   69 FF  A < FF
   48 06  B Shift Left,  Skip if carry
   69 00  A < 00
   1A  B < A
   1B  C < A
   B1  PUSH BC
$NEXT

AND  A1  POP BC
    A4  POP AE
    74 8D  EA < EA AND BC
    B4  PUSH EA
$NEXT

OR  A1  POP BC
    A4  POP EA
    74 9D  EA < EA OR BC
    B4  PUSH EA
$NEXT

XOR  A1  POP BC
    A4  POP EA
    74 95  EA < EA EX-OR BC
    B4  PUSH EA
$NEXT

INVERT  A1  POP BC
   69 FF  A < FF
   60 12  B < B EX-OR A
   60 13  C < C EX-OR A
   B1  PUSH BC
$NEXT

UM+  A1  POP BC
    A4  POP EA
    69 00  A < 00
    74 A5  EA < EA + BC  Skip if no carry
    69 01  A < 01
    1B  C < A
    6A 00  B < 00
    B4  PUSH EA
    B1  PUSH BC
$NEXT
+ A1 POP BC
   A4 POP EA
    74A5 EA < EA + BC Skip
   00 NOP
   B4 PUSH EA
 $NEXT

NEGATE
   A1 POP BC
  69 FF A < FF
  60 12 B < B EX-OR A
  60 13 C < C EX-OR A
   12 BC < BC + 1
   B1 PUSH BC
 $NEXT

-  A1 POP BC
  69 FF A < FF
  60 12 B < B EX-OR A
  60 13 C < C EX-OR A
   12 BC < BC + 1
   A4 POP EA
    74 A5 EA < EA + BC Skip
   00 NOP
   B4 PUSH EA
 $NEXT

0=  A4 POP EA
   A9 FF A < FF
   1A  B < A
   1B  C < A
   74 CD EA AND BC Skip if not zero
   C4 JMP BRAN2
   A9 FF A < 00
   1A  B < A
   1B  C < A
 BRAN2 B1 PUSH BC
 $NEXT

=  A4 POP EA
   A1 POP BC
  69 FF A < FF
   74 FD EA - BC Skip if zero
  69 00 A < 00
   1A  B < A
   1B  C < A
   B1 PUSH BC
 $NEXT

doUSER  48 83 EA < (HL)
  70 1F up BC < (up)
   74 A5 EA < EA +BC SKIP
   00 NOP
   B4 PUSH EA
 $NEXT
Modified
Roland PG1000

Programming
Machine Code
PUSHBUTTONS AND LED's

Each button is associated with a bit in the PA port. The number above the button is the hex value of the PA input port for that button. Note that two of the switches have no numbers associated with them.

There are 6 LED's located in the 6 non-xx buttons. These are turned on and off through the PB output port. The same port numbers as for the buttons apply to port B outputs for the LED's.

Bit zero of port B is used for serial transmission. Any serial transmission will automatically turn off all 6 LEDs. Bit one of port B is used to select between the bank of 8 numbered switches shown above and the bank of two currently unused ones in the upper right corner. PB1 must remain low to enable the 8 buttons above that are numbered.

CODE

SW@ (--- n, Read the 8 switch values in a byte. Switches are normally high.)
4C C0 A < PA
6A 0 B < 0
1B C < A
B1 PUSH BC
ENDCODE

CODE S@ (--- n, Return # of lowest switch on. n = 0,1,2,3,...,8)
4C C0 A < PA
6B 0 C < 0
74 11 FF A < A EXOR FF
74 49 FF A AND FF, Skip if NO ZERO
C4 JMP OUT
LOOP: 43 C < C+1
48 01 A Shift Right, Skip if Carry
FC JMP LOOP
OUT: 6A 0 B<0
B1 PUSH BC
ENDCODE

TST (--- ) CR BEGIN SW@ FF XOR <# # # #> TYPE
200 DELAY D EMIT NUF? UNTIL ;
ORG 0200H

; Interrupt routine for Analog to Digital Converters

DB 10H ;EXA, use alternate registers
DB 11H ;EXX

; Load ADC Address and Counter into HL. Uses FFF3.
DB 68H,0FFH ;V'<FF
DB 69H,0C6H ;A<C6
DB 1EH ;H<A, H<C6, C6xx is ADC RAM buffer area
DB 01H,0F3H ;A<(V/F3)
DB 1FH ;L<A, L<(FFF3) the slider interrupt count

; Store ADC values.
DB 4CH,0E0H ;A<CR0
DB 3DH ;(HL)+<A, store new value, increment count
DB 4CH,0E1H ;A<CR1
DB 3DH ;(HL)+<A, store new value, increment count
DB 4CH,0E2H ;A<CR2
DB 3DH ;(HL)+<A, store new value, increment count
DB 4CH,0E3H ;A<CR3
DB 3DH ;(HL)+<A, store new value, increment count

; Update Counters
DB 0FH ;A<L
DB 63H,0F3H ;(V/F3)<A, update counter.
DB 48H,25H ;A shift logical left
DB 48H,05H ;A shift left, skip if carry, if count = 40H
DB 0C7H ;JMP AHEAD if all 64 values are not converted yet.

DB 64H,1EH,01 ;MKH<MKH OR 1, disable INTAD
DB 10H,11H,0AAH,62H ;EXA,EXX,IE,RETI return from interrupt

AHEAD: DB 1AH ;B<A
DB 74H,0AH,0E0H ;B<B AND E0
DB 4CH,0C2H ;A<Pc
DB 07H,1FH ;A<A AND 1F
DB 60H,9AH ;A<A OR B
DB 4DH,0C2H ;PC<A, Load 4051s' address select bits (3 bits).
DB 64H,90H,08H ;Invert ANM bit and start next ADC convert cycle.

; Return from Interrupt.
DB 10H ;EXA
DB 11H ;EXX
DB 0AAH ;EI
DB 62H ;RETI

SERIAL INTERFACES

A 9600 Baud, software driven serial I/O is provided. PortB/bit0 is used for serial output and PortC/bit3 (INT2) is used for the serial input. The serial input is interrupt driven with a vectored interrupt routine located at 0A0H. The code words ?RX, TX!, and !!O make up the rest of the serial I/O code. Three USER variables have been set up for use by these serial I/O routines: SERIN, which holds the received character and a flag; HAFBIT, which adjusts the software timing of the receiver to read in the middle of each bit frame (set it for 1/2 the BITIME minus
5); and BITIME, which adjusts the software for a specific baud rate (17H for 9600 baud assuming a 12Mhz processor clock).

;     SERIN     ( -- a )
;     Point to host serial input. Flag in high, char in low byte.
; $USER   5,'SERIN',SERIN

;     HAFBIT     ( -- a )
;     Point to half bit time used by serial i/o routines. (Equals 06H).
; $USER   6,'HAFBIT',HAFBIT

;     BITIME     ( -- a )
;     Point to bit time used to set serial i/o baud rate. (Equals 16H).
; $USER   6,'BITIME',BITIME

ORG 00A0H
; Vectored INT2 routine for Serial Input from Host Computer.
; Uses address FFF0 as a counter location – do not use elsewhere!

DB 0B1H                    ;PUSH BC
DB 0B2H                    ;PUSH DE
DB 0B0H                    ;PUSH VA
DB 68H,0FFH                ;MVI, V<FF
DB 71H,0FOH,07H            ;MVIW, (V/F0)<07, number of bits to receive.
DB 70H,1FH,04CH,0FFH       ;LBCD, BC<(FF4C), wait for a half bit.
LOOP1: DB 53H            ;DCR, C<C-1 skip if borrow
   DB 0FEH                  ;JR, Jump to loop1
   DB 52H                    ;DCR, B<B-1 skip if borrow
   DB 0FCH                  ;JR, Jump to loop1
LOOP2: DB 70H,1FH,4EH,0FFH ;LBCD, BC<(FF4E), wait 1 bit time
   DB 53H                    ;DCR, C<C-1 skip if borrow
   DB 0FEH                  ;JR, Jump to loop2
   DB 52H                    ;DCR, B<B-1 skip if borrow
   DB 0FCH                  ;JR, Jump to loop2
   DB 04CH,0C2H             ;MOV, A<PC, read serial input on pc3
   DB 48H,31H,48H,31H        ;Rotate PC3 bit into Cy
   DB 48H,31H,48H,31H        ;RLR, A rotate right 4xs
   DB 0CH                    ;MOV, A<D, D collects the bits
   DB 48H,31H                ;RLR, shift in next bit, CY to top of D
   DB 1CH                    ;MOV, D<A
   DB 30H,0FOH              ;DCRW, (V/F0)<(V/F0)-1 skip if borrow
   DB 0E7H                 ;JR, Jump to loop2 for next bit.
   DB 70H,1FH,4EH,0FFH       ;LBCD, BC<(FF4E)
   DB 53H,0FEH,52H,0FCH      ;DCR JR DCR JR, stop bit loop time.
   DB 71H,04BH,0FFH          ;MVIW, (V/4B)<FF, load flag
   DB 0CH,63H,04AH          ;MOV STAW, A<D (V/4A)<A, load data
   DB 0A0H,0A2H,0A1H        ;POP, restore AV DE and BC
   DB 48H,44H,0             ;SKIT,NOP
   DB 0AAH,062H           ;EI RETI, enable interrupts and return

;     '?KEY     ( -- a )
;     Execution vector of ?KEY.
; $USER   5,"'?KEY',TQKEY

;     'EMIT     ( -- a )
;     Execution vector of EMIT. Points to TX! Code Word.
; $USER   5,"'EMIT',TEMIT
; ?RX
  ( -- c T | F )
; Return input character and true, or a false if no input.
Points to ?RX Code word.

$CODE  3,'?RX',QRX
DB  68H,0FFH ;MVI, V<FF
DB  01H,4BH   ;LDAW, A<(V/4B) read serial-in flag
DB  47H,0FFH ;ONI, A AND FF skip if flag not zero
DB  0CAH      ;JR, jump ahead
DB  71H,04BH,0 ;MVIW, (V/4B)<0, reset flag to zero
DB  70H,1FH,4AH,0FFH ;LBCD, BC<(FF4A), read serin data
DB  081H ;PUSH BC, push serial input data to stack
DB  69H,0FFH ;A<FF
AHEAD: DB  1BH ;C<A,
          DB  6AH,0 ;B<0
          DB  081H ;PUSH BC, push serial input flag to stack
$NEXT

; TX!
  ( c -- )
; Send character c to the output device.

$CODE  3,'TX!',TXSTO
DB  0BAH ;Disable Interrupts
DB  0A1H ;POP BC, pop char into C
DB  0B2H ;PUSH DE, store interpreter pointer
DB  0BH,1CH ;A<C, D>A, char in A and D
DB  68H,0FFH ;V<FF
DB  71H,0F0H,07H ;(V/F0)<7
DB  60H,91H ;A<A EXOR A
DB  6DH,01H ;E<01
LOOP1: DB  70H,1FH,04EH,0FFH ;BC<(FF4E) set baud
        DB  53H,0FEH,52H,0FCH ;C<C-1, JR, B<B-1, JR, jr to loop1
        DB  0CH ;A<D
        DB  07H,01H ;A<A AND 01
        DB  4DH,0C1H ;PB<A, send a bit
        DB  0CH ;A<D
        DB  48H,31H ;A rotate logical right
        DB  1CH ;D<A
        DB  0,0,0,0 ;NOPs to make rec loop = transmit loop.
        DB  30H,0F0H ;(V/F0)<(V/F0)-1 skip if borrow
        DB  0E8H ;JR, jump to loop1
        DB  0DH ;A<E
        DB  51H ;A<A-1 skip if borrow
        DB  0C6H ;JR, jump to loop2
        DB  0A2H ;POP DE, restore interpreter pointer
        DB  0AAH ;Enable Interrupts
        $NEXT ;End of routine

LOOP2: DB  6CH,03H ;D<03
        DB  1DH ;E<A
        DB  71H,0F0H,01 ;(V/F0)<01
        DB  4FH,0D7H ;JRE, jump to loop1

; !IO
  ( -- )
; Initialize the serial I/O devices.
MIDI INTERFACE

PC0 is the MIDI serial output. PC1 is the MIDI serial input. Both use the 78C10 microprocessor's internal serial I/O device. The following is set in the Cold Start Routine:

69 0B 4D D1 MCC < 0B PC0,PC1,PC3 set for I/O (4-8)
69 0A 4D D4 MC < 0A PC1 and PC3 set as inputs (4-9)
64 81 06 SMH < 6 Receive disabled, Transmit enb (7-7)
69 4E 4D CA SML < 4E Clk x16,8 bits,no parity,1 stop (7-9)

To transmit midi data xx, code it as follows: 69 xx 4D D8 (A < xx, TXB < A). When transmission is done, an INTST interrupt is generated. This software does not use an interrupt. Instead, it checks the INTFST flag using SKIT FST to wait until the flag goes high before transmitting the data.

Because of this wait period, when coding, it will save processor time to put operations between TM commands - i.e. the following code

90 TM 40 TM 40 TM OP1 OP2 OP3

is optimized by placing it in following order:

OP1 90 TM OP2 40 TM OP3 40 TM

; TM
; Wait for last transmit, then send midi byte n.
$CODE 2,'TM',TM
DB 0A1H ;POP BC
DB 0BH ;A<C
DB 48H,4AH ;SKIT FST, skip if interrupt
DB 0FDH ;JMP TO SKIT
DB 4DH,0D8H ;MOV TXB,A
$NEXT

: ENBREC ( ---, Enable Midi Receive)
$CODE
DB 64H,81H,0EH ;SMH<E
$NEXT

: RM ( --- Midi Receive: If received leave - char, FF; If nothing there leave - 00 .)
$CODE
6A,0 ;B<0
6B,0 ;C<0
48,49 ;SKIT FSR, Skip if Interrupt flag
C6 ;JMP AHEAD
4C,D9 ;A<RXB
1B ; C<A
B1 ; PUSH BC, push received data
6B FF ; C<FF
AHEAD:
B1 ; PUSH BC, push flag
$NEXT

: TST ENBREC BEGIN RM IF . THEN NUF? UNTIL ;

MEMORY BUFFERS AND REGISTERS:

C600 - C800 SLIDER RAM for the ADC/EDIT/MIDI routines.

C600-C63F ADC data, 8-bits.
C640-C67F SLAST. Last ADC value read. Used only in some operations.
C680-C6B7 Midi Byte 2 (Key#, Controller#, Program#, etc.)
C6C0-C6FF Midi Byte 1 (Status and Channel number). Disabled if bit7=0.
C700-C73F FLAG to keep track of current state of Midi operation.

Each of the 64 Sliders has a 5-byte buffer. For example, Slider #1 uses C600, C640, C680, C6C0 and C700. This buffer is serviced by the ADC interrupt routine and the main program's EDIT routines. The MIDI routineS read from it to transmit Midi data.

Serial Input Registers

FF00 Timing counter.
FF4A USER value SERIN
FF4C USER value HAFBIT
FF4E USER value BITIME

ADC Interrupt routine registers:

FF3 ADC interrupt slider counter

EDIT Routine registers:

FF00 eSLD# Slider number being edited.
FF01 eFLD LCD field being edited.
FF02 eBYTE1 Midi Byte with Status/Channel data. ENB flag.
FF03 eBYTE2 Midi Byte with Key#, Controller#, or Program#.
FF04 eBYTE3 8-bit Slider value.
FF05 eSET MODE: 0 = Slider Edit, FF = Setup mode.
FF06 eSET# Holds Setup number. Used to address Rom setups.

: eSLD# FF00 ; Edit Buffer Slider Number.
: eFLD FF01 ; Edit Buffer LCD field.
: eBYTE1 FF02 ; Edit Buffer Midi status byte.
: eBYTE2 FF03 ; Edit Buffer Midi data byte.
: eBYTE3 FF04 ; Edit Buffer Slider value.
: eSET FF05 ;
: eSET# FF06 ;

MIDI Routine registers:

FFE0 LCNT Midi Loop count.
FFE1 LST Last Midi Status sent.

Microprocessor Ram:
Tables:

The sliders are not converted in the same order in which they are physically laid out in the box. A table of 64 has been created in ROM at address 00200H to translate the slider numbers to physical position numbers:

```
48 56 40 37 53 61 45 35
51 59 43 33 49 57 41 38
54 62 46 30 32 16 08 00
24 21 13 05 29 19 11 03
44 27 17 09 01 25 22 14
06 34 50 60 52 36 20 12
04 28 18 10 02 26 42 58
07 15 23 31 39 47 55 63
```

(numbers in decimal)

The following chart shows the physical layout of the sliders on the Roland PG1000 box:

```
07 15 23 31 39 47 55 63 (External)
56 57 58 59 60 61 62 63 (External)
  60 52 36 20 12 04|28 18 10 02 26 42 58
  43 44 45 46 47 48|49 50 51 52 53 54 55
  32 16 08 00 24 21 13 05
  20 21 22 23 24 25 26 27
  28 29 30 31 32 33 34 35
  36 37 38 39 40 41 42
  48 56 40 37 53 61 45 35
  00 01 02 03 04 05 06 07
```
Utility Words
UTILITY WORDS

: CASE ( n --- , Use 'n' as a pointer to following list of words, execute  
the pointed word, then leave the current word )
R> ( Remove return stack pointer to next token, put on data stack )
SWAP ( pointer 'n' to the top )
2* + ( pointer + 2n )
@EXECUTE ( execute token at pointer + 2n, leave current word )

The CASE word is used as follows:

: XXX ( n --- ) ANYWORDS CASE PG0 PG1 PG2 ;

If n=0, PG0 is executed after CASE. If n=1, PG1 is executed after CASE. If n=2, PG2 is executed after  
CASE. Only the one PG is executed after which XXX terminates.

Warning!! If n is greater than the number of words between CASE and ';' minus one, the program will  
bomb.

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

: INCR ( n, nmax --- n+1, Increment n but set to zero if greater than nmax )
OVER 1+ < IF DROP 0 ELSE 1+ THEN ;

: DECR ( n, nmax --- n-1, Decrement n but set to nmax if less than zero )
OVER 1- 0< IF SWAP DROP ELSE DROP 1- THEN ;

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

CODE  DELAY ( n --- )
A1  POP BC
53  C < C-1, Skip if borrow
FE  JMP back 1
52  B < B-1, Skip if borrow
FC  JMP back 3
ENDCODE

************************************************
REMOTE FILE LOADING:

Build a TEXT-ONLY file in Word (or other) of the words you want to load. When using White Knight, go under Custom / Otions / Text Transfer and set the "Wait after each line sent for the ^K character (the Pacing character used in Eforth). Type in the Eforth word FILE. Character echoes are now turned off. Go under File to Send a Text File. Make sure there is the HAND command at the end of your text file to return control back to the console. Type WORDS to see if your text was loaded.
LCD Screen
The HD44780 LCD Controller

Display is 2 Lines of 16 Characters. Position addresses are 0 to F for the top line, and 40 to 4F for the bottom line. Cursor automatically increments to the next position after a character write, except for the jump between the end of the top line and beginning of the bottom line.

Characters can be taken from internal ROM which holds all the ASCII characters plus other special characters (see the chart) or from an internal RAM which must be loaded at power on by the user. This project uses only the ROM.

There are two Write modes controlled by the RS input line:
RS = 0, LCD setup commands such as Cursor on/off/ position, Clear, etc.
RS = 1, write character to cursor position.

Core LCD Words:

```
CODE  LLI  ( --- , Sets RS=0 for LCD setup commands )
  64 0A EF  Pc < Pc AND EF
ENDCODE

CODE  LLC  ( ---  , Sets RS=1 for character displaying )
  64 1A 10  Pc < Pc OR 10
ENDCODE

CODE  LCD  (n---  , Loads LCD command 'n')
  A1  POP BC
  0B  A < C
  14 00 A0  BC < A000
  39  (BC) < A
ENDCODE

: LI  (n --, Load LCD setup command 'n'.
     Exit with RS=1 for char writes.)
    LLI LCD LLC 1FF DELAY ;
```

Example LCD Display Commands:

```
1 LI    Clear LCD Screen.  C LI    Turn off the Cursor
F LI    Turn on the Cursor  80 LI    Position cursor at start

C0 LI    Position cursor at start of 2nd line.
41 LCD   Display ASCII 'A' at current cursor position,
          increment cursor.
```
Power Up Reset Routine

: LCDINIT ( Initialize LCD display )
  D7A DELAY 38 LI ( Function Set-8 bit input, 2 lines, 5x7 char )
  47E DELAY 38 LI
  017 DELAY 38 LI
  017 DELAY 38 LI
  017 DELAY 08 LI ( Display off, cursor off, blink off )
  017 DELAY 01 LI ( Clear display, home the cursor )
  1CC DELAY 02 LI ( Cursor at home - top left position )
  1CC DELAY 06 LI ( Entry Mode Set - cursor auto
    increment,shift off )
  017 DELAY 0F LI ( Display on, cursor on, blink on )
  017 DELAY ;

Display Routines

: #DISP ( n,p --- , Display 'n' as a 3-digit number in current base at LCD
    position 'p' and reset cursor to start of number )
  DUP LI SWAP ( Move cursor to 'p', and save )
  <# # # # #> ( Convert 'n' to 3 ascii digits.
    Leave address and 3 on stack )
  DROP DUP C@ LCD 1+ DUP ( Locations of the 3 digits in TIB)
  C@ LCD 1+ C@ LCD LI ( Display 3 digits and reset cursor )
 ;
  : #2DISP DUP LI SWAP <# # # > DROP DUP C@ LCD 1+ C@ LCD LI ;
  : DISP (a,p ---, Display string from address 'a' in LCD position 'p')
   LI DUP C@ 1- >R (Set cursor. First byte at string address is a count.)
   FOR 1+ DUP C@ LCD NEXT DROP (Display 'count' string characters )
 ;

MASM Macro to Compile a Stored String

SD$ MACRO STRNG
  DW DOLIT
  _LEN = $ + 4 ;;save address of count byte
  DW _LEN,EXIT ;;save cnt address on stack
String Data for DISP

Can be built with the SD$ Macro in MASM, or from FORTH in the following manner:

```
: STRING1  "$ Slider " ;
```

The word above is loaded into code space as follows:

```
DoList$|  Count  S  l  i  d  e  r  Space  EXIT
80  84  A  7  53  6C  69  64  65  72  20  94  3
```

This needs some explanation: The word "$" is an immediate word which acts in compilation mode to set up the word's code area as shown. When STRING1 is executed the word "$|" found in the code area is first to execute. This word puts the address of 'Count' on the data stack and rearranges the return stack so that the next word to execute is EXIT.

The word "$" cannot be used when forming the ROM code with MASM. The string creating macro 'SDS' is used instead. It leaves the code area looking like this:

```
doLit  addr-of-count  EXIT  Count  S  l  i  d  e  r  Space
80  xxxx  94  3  7  53  6C  69  64  65  72  20
```
LCD DISPLAY FIELDS
( **** Labels, ---- Numbers )

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 A B C D E F</th>
</tr>
</thead>
<tbody>
<tr>
<td>* * * * * * - - - - - - - - - -</td>
</tr>
<tr>
<td>FLD0   FLD1   FLD2   FLD3</td>
</tr>
<tr>
<td>* * * * * * - - - - - - - - - -</td>
</tr>
<tr>
<td>FLD4   FLD5   FLD6</td>
</tr>
</tbody>
</table>

: FLD0 80 ;  (Slider, Setup#, * MIDI Running *)
: FLD1 86 ;  (Slider number 0 to 55 )
: FLD2 88 ;  (Ch(Midi Channel), Off(Slider disabled))
: FLD3 8D ;  (Midi Channel number 0 to 15)
: FLD4 C0 ;  (Midi Status - KEY#, KEY# AT, CONTROL#, PROGRAM#, CH PRESS,
             PTCH WHL, ********)
: FLD5 C9 ;  (Midi byte value 0 to 127)
: FLD6 CD ;  (Slider value 0 to 127)

: FLDCASE  (n---f, Choose an LCD field position )
          7 AND CASE  FLD0 FLD1 FLD2 FLD3 FLD4 FLD5 FLD6 FLD0
          ;

LCD Displayed Strings

; L0             ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON  2,'L0',L0
SD$ 'Slider'

; L1             ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON  2,'L1',L1
SD$ 'Setup#'

; L2             ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON  2,'L2',L2
SD$ '* MIDI Running *'

; L20            ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON  3,'L20',L20
SD$ 'chl'

; L21            ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L21',L21
SD$ ' off'

; L40  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L40',L40
SD$ 'Key#    '  

; L41  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L41',L41
SD$ 'Key# A-T'

; L42  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L42',L42
SD$ 'Control#'

; L43  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L43',L43
SD$ 'Program#'

; L44  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L44',L44
SD$ 'Ch Press'

; L45  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L45',L45
SD$ 'Ptch Whl'

; L4X  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L4X',L4X
SD$ '*******'

; L50  ( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L50',L50
SD$ '***'

: LSTAT      (n---, Choose a Midi Status Label)
    CASE   L4X L40 L41 L42 L43 L44 L45 L4X

LCD Hardware Interface

14 pin ribbon cable:
1. GND
2. +5 v
3. Resistor to Gnd, Sets LCD back light contrast
4. RS = P4c, 78C10/21
5. R/W = Gnd, Writes only, no reads used.
6. E, Write pulse derived from address decoder – Axxx / Bxxx, ***
7. Data 0 = Pd0, 78C10/55
8. Data 1 = Pd1, 78C10/56
9. Data 2 = Pd2, 78C10/57
10. Data 3 = Pd3, 78C10/58
11. Data 4 = Pd4, 78C10/59
12. Data 5 = Pd5, 78C10/60
13. Data 6 = Pd6, 78C10/61
14. Data 7 = Pd7, 78C10/62
EDIT Words
EDIT BUFFER UTILITY ROUTINES

; eUPDAT     ( --- )
; Move data from Slider Ram to Edit Buffer.
$CODE   6,'eUPDAT',EUPDAT
DB  68H,0FFH       ;;V<FF
DB  6AH,2          ;;B<2
DB  1,0            ;;A<(V/00) Read eSLD#
DB  1BH            ;;C<A
DB  29H            ;;A<(BC) Read Translation Table
DB  6AH,0C6H       ;;B<C6
DB  1BH             ;;C<A
DB  29H             ;;A<(BC)
DB  48H,21H        ;;A SHIFT RIGHT
DB  63H,4          ;;(V/04)<A, eBYTE3 (7bits) to FF04
DB  69H,80H        ;;A<80
DB  60H,43H        ;;C<C+A
DB  29H             ;;A<(BC)
DB  63H,3H         ;;(V/03)<A, eBYTE2 to FF03
DB  69H,40H        ;;A<40
DB  60H,43H        ;;C<C+A
DB  29H             ;;A<(BC)
DB  63H,2H         ;;(V/02)<A, eBYTE1 to FF02
$NEXT

; eLOAD      ( --- )
; Load Edit Buffer data into Slider Memory.
$CODE   5,'eLOAD',ELOAD
DB  68H,0FFH       ;;V<FF
DB  6AH,2          ;;B<2
DB  1,0            ;;A<(V/00) Read eSLD#
DB  1BH            ;;C<A
DB  29H            ;;A<(BC) Read Translation Table
DB  6AH,0C6H       ;;B<C6
DB  1BH             ;;C<A
DB  69H,40H        ;;A<40
DB  60H,43H        ;;C<C+A
DB  49H,0          ;;(BC)<0, Zero to SLAST
DB  69H,40H        ;;A<40
DB  60H,43H        ;;C<C+A
DB  1,3            ;;A<(V/03)
DB  39H             ;;(BC)<A, (FF03) to Midi byte 2
DB  69H,40H        ;;A<40
DB  60H,43H        ;;C<C+A
DB  1,2            ;;A<(V/02)
DB  39H             ;;(BC)<A, (FF02) to Midi byte 1
$NEXT
; esUPDAT ( --- )
; Update only the Slider data of the Edit Buffer.
$CODE 7,'esUPDAT',esUPDAT
DB 68H,0FFH        ;;V<FF
DB 6AH,2           ;;B<2
DB 1,0             ;;A<(V/00) Read eSLD#
DB 1BH             ;;C<A
DB 29H             ;;A<(BC) Read Translation Table
DB 6AH,0C6H        ;;B<C6
DB 1BH             ;;C<A
DB 29H             ;;A<(BC)
DB 48H,21H         ;;A SHIFT RIGHT
DB 63H,4           ;;(V/04)<A, 7-bit Slider value to FF04
$NEXT

LCD DISPLAY ROUTINES

: FLDAT ( --- , Return LCD cursor to current edit field )
   eFLD C@ FLDCASE LI
   ;

: SDISP ( --- , Display the Setup operation on the LCD)
   1 LI BDEL
   L1 FLD0 DISP
   eSET# C@ FLD1 #2DISP     FLDAT
   ;

: SLDISP ( --- , Update and display Slider data )
   esUPDATE (Get latest slide value)
   eFLD C@ FLDCASE (Get current field position)
   eBYTE3 C@ 7F AND
   <# # # # #> DROP LLI FLD6 LCD LLC (Display Slide value)
   DUP C@ LCD 1+ DUP C@ LCD 1+ C@ LCD LI
   ;

: eDISP ( --- , Display Edit Buffer data on the LCD )
   L0  FLD0 DISP ( SLIDER )
   eSLD# C@ FLD1 #2DISP (Slider #)
   eBYTE3 C@ 80 AND IF L20 FLD2 DISP ( OFF )
       ELSE L21 FLD2 DISP ( CHL )
       THEN
   eBYTE1 C@ DUP F AND FLD3 #DISP ( Midi Channel # )
   2/  2/  2/  2/  7 AND LSTAT FLD4 DISP ( Midi Status )
   CF eBYTE1 C@ F0 AND <
       IF L50 FLD5 DISP ( *** in Fld 5 )
       ELSE eBYTE2 C@ FLD5 #DISP ( Number in Fld 5 )
THEN
SLDISP FLDAT               ( Slider Data # )
;
: MNDISP         ( ---, MAIN DISPLAY UPDATE used in EDIT word )
eSET C® IF SDISP ELSE eDISP THEN
;

LCD CURSOR CONTROL

: BDEL           ( ---, Long delay for poky LCD )
   8000 DELAY
;
: BL/R           ( fld---pos, Translates LCD field number to LCD position)
   DUP eFLD C! FLDCASE
;
: SL/R           ( ---, If in setup mode, toggle between fields 0 and 1 )
eFLD C® 01 AND 01 XOR DUP eFLD C! BL/R LI BDEL ;

: BLEFT          ( ---, Move cursor left & update eFLD)
   40 LED!
eSET C® IF SL/R ELSE
eFLD C® 5 DECR BL/R LI BDEL THEN
;

: BRIGHT         ( ---, Move cursor right & update eFLD)
   80 LED!
eSET C® IF SL/R ELSE
eFLD C® 5 INCR BL/R LI BDEL THEN
;

LCD FIELD INCREMENT/DECREMENT

; UDCASE          ( n---, Choose field dependent up/down routine)
eFLD C® 7 AND
   CASE U/D0 U/D1 U/D2 U/D3 U/D4 U/D5 U/D6 U/D7
;
: BUP            ( ---, Increment value at cursor field )
   10 LED!  1 UDCASE BDEL
;
: **BDOWN** ( --- , Decrement value at cursor field )
   20 LED!  0 UDCase BDEL
;

: **U/D0** (flg---, inc/dec field 0, SLIDER#/SETUP)
   DROP eSET C® IF 0 eSET C! eDISP
   ELSE FF eSET C! SDISP THEN
;

: **U/D1** (flg---, Inc/Dec fld 1 which shows the Slider OR SETUP #)
   eSET C® IF eSET# ELSE eSLD# THEN
   C® 3F ROT (slider#, max value, up/dwn flag)
   IF INCR ELSE DECR THEN ( slider# +1 or -1)
   CFLD1
;

: **CFLD1** (slider# --- , Change Slider/SETUP# in Fld1. Update Edit Buffer & LCD)
   eSET C® IF eSET# C! SDISP
   ELSE eSLD# C! eUPDAT eDISP THEN FLDAT
;

: **U/D2** (flg ---, Change Chnl or Off label in Field 2)
   IF eBYTE1 C® 80 OR eBYTE1 C! L20 FLD2 DISP ( display 'ch ' )
   ELSE eBYTE1 C® 7F AND eBYTE1 C! L21 FLD2 DISP ( display 'off' )
   THEN FLDAT
;

: **U/D3** (flg ---, Change midi channel in Field 3 )
   eBYTE1 C® F AND F ROT ( chnl#, F, u/d flg )
   IF INCR ELSE DECR THEN ( chnl# +1 or -1 )
   CFLD3
;

: **CFLD3** (channel # ---, Change midi channel in Field 3 )
   DUP eBYTE1 C® F0 AND OR (Construct eByte1 with new Chnl # )
   eBYTE1 C! FLD3 #DISP (Load into Edit buffer, Display it )
;

: **U/D4** (flg ---, Change Midi operation in the status byte1 )
   eBYTE1 C® 70 AND 2/ 2/ 2/ 2/ 7 ROT ( stat, 7, u/d flg )
   IF INCR ELSE DECR THEN ( stat +1 or -1 )
   DUP 0 = OVER 7 = OR IF DROP 1 THEN ( avoid status bytes 0x and Fx )
   CFLD4
;

: **CFLD4** ( status# ---, Change Midi operation label in field 4 )
   DUP 2* 2* 2* 80 OR eBYTE1 C® F AND OR (Construct eByte1 with new Stat#)
   eBYTE1 C! LSTAT FLD4 DISP (Load into Edit buffer, Display it )
   FLDAT
;

: **U/D5** (flg---, Change Midi data byte2 )
   eBYTE2 C® 7F ROT ( data, 7F, u/d flg )
   IF INCR ELSE DECR THEN ( data +1 or -1 )
   CFLD5
;
: CFLD5 (midi data --, Change Midi Data Byte2 in field 5 )
   CF  eBYTE1 C@  F0 AND  <  (If chnl press or ptch whl, no data here)
   IF  L50  FLD5 DISP  FLD5 LI  DROP  ( display '***' in field 5 )
   ELSE  DUP  eBYTE2 C!  FLD5  #DISP  (Load into Edit buffer, Display it )
   THEN
   ;

: U/D6 (flg---, Do nothing with Slider value field)
   DROP
   ;

: U/D7 (flg---, bogus field. Does not exist)
   DROP
   ;

: TST BEGIN  KEY  DUP  ( 1 = UP,  0 = DWN,  any other key = quit )
   31  =  IF  1  U/D3  DROP  0  ELSE
   30  =  IF  0  U/D3  0               ELSE
   1  THEN  THEN  UNTIL  ;

EDIT BUTTON ROUTINES

<table>
<thead>
<tr>
<th>8(4)</th>
<th>4(3)</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDI</td>
<td>LOAD</td>
<td>XX</td>
<td>XX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>40(7)</th>
<th>80(8)</th>
<th>10(5)</th>
<th>20(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>RIGHT</td>
<td>UP</td>
<td>DOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2(2)</th>
<th>SLD UP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(1)</td>
<td>SLIDE DWN</td>
</tr>
</tbody>
</table>

: MNCASE ( --- , Main Case statement for button routines)
   S@
   CASE  DUMMY BSDWN BSUP BLOAD BMIDI BUP BDOWN BLEFT BRIGHT
   ;

: BSDWN ( ---, Button 1 decrements the slider number)
   0 UD1 BDEL
   ;

: BSUP ( ---, Button 2 increments slider number)
   1 UD1 BDEL
   ;
: SETUP  (---, Setup Slider Ram Buffer from ROM locations 2000H)
  eSET# C@ 2000 +   C680   80 CMOVE
;

: BLOAD  (---, Button 3 loads data shown on LCD into Slider Memory)
  4 LED!
  eSET C@   IF   SETUP 0 eSET C!eUPDAT eDISP BDEL
  ELSE eLOAD eDISP BDEL   THEN
;

: BMIDI  (---, Button 4 starts the Midi program)
  1 LI BDEL
  L2 FLD0 DISP
  MIDI
;

: EDIT    (MAIN EDIT PROGRAM)

  CR DECIMAL 1 LI BDEL
  eUPDATE eDISP

  BEGIN
    ADCINIT SLDISP
    0 LED!  MNCASE
    NUF?   UNTIL HEX

;
MIDI Words
PG1000 MIDI

The main MIDI loop rotates through the 64 ADC Slider values until an enabled one is found. The loop count is stored in register FFE0. Then the Midi status is checked for the following operations:

1  KEYN      Key On or Key Off
2  KEYAT     Key On or Key Off with After Touch
3  CNTRL     Control Change
4  PRG       Program Change
5  CHAT      Channel After Touch
6  PWHL      Pitch Wheel

MIDI UTILITY ROUTINES

;  MLOOP      ( --- stat)
;  Loop thru ADC values until an enabled one is found.
$CODE   5,'MLOOP',MLOOP
DB 68H,0FFH           ;V<FF
DB 6AH,0C6H           ;B<C6
LOOPBACK:       DB 1,0E0H           ;A<(V/E0)
DB 41H              ;A<A+1 skip if carry (never carries)
DB 07H,3FH          ;A<A AND 3F
DB 63H,0E0H         ;(V/E0)<A, incremented count to FFE0
DB 46H,0C0H         ;A<A+C0
DB 1BH          ;C<A
DB 29H             ;A<(BC), get midi bytel
DB 47H,80H         ;A AND 80, skip if no zero
DB 0F2H           ;JMP LOOPBACK, if disabled
DB 7H,7FH          ;A<A AND 7F
DB 48H,21H         ;A shift right
DB 48H,21H         ;A shift right
DB 48H,21H         ;A shift right
DB 48H,21H         ;A shift right
DB 1BH              ;C<A, Midi status number
DB 6AH,0          ;B<0
DB 0B1H           ;PUSH BC, push status# on stack
$NEXT

;  ADCV       ( --- adc value)
;  Push stack with current adc value for MIDI operation.
$CODE   4,'ADCV',ADCV
DB 1,0E0H           ;A<(V/E0), Midi loop count.
DB 1BH              ;C<A
DB 6AH,0C6H         ;B<C6
DB 29H              ;A<(BC)
DB 1BH              ;C<A
DB 6AH,0          ;B<0
DB 0B1H           ;PUSH BC
$NEXT
; SLAST
( --- slast value)
; Push stack with last adc value for MIDI operation.
$CODE 4,'DIFF',DIFF
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 46H,40H ;A<A+40
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; BYT2
( --- byt2 value)
; Push stack with current BYTE2 value for MIDI operation.
$CODE 4,'BYT2',BYT2
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 46H,80H ;A<A+80
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; BYT1
( --- byt1 value)
; Push stack with current BYTE1 value for MIDI operation.
$CODE 4,'BYT1',BYT1
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 46H,0C0H ;A<A+C0
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; FLAG
( --- flag value)
; Push stack with current FLAG value for MIDI operation.
$CODE 4,'FLAG',FLAG
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 1BH ;C<A
DB 6AH,0C7H ;B<C7
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT
; FLGON
( --- )
; Store FF in FLAG of current slider.
$CODE 5,'FLGON',FLGON
DB 1,0E0H        ;A<(V/E0)
DB 1BH           ;C<A
DB 6AH,0C7H      ;B<C7
DB 69H,0FFH      ;A<FF
DB 39H           ;(BC)<A
$NEXT

; FLGOFF
( --- )
; Store 0 in FLAG of current slider.
$CODE 6,'FLGOFF',FLGOFF
DB 1,0E0H        ;A<(V/E0)
DB 1BH           ;C<A
DB 6AH,0C7H      ;B<C7
DB 69H,0         ;A<0
DB 39H           ;(BC)<A
$NEXT

; ?DIFF
(old,new --- |shifted new,FF| or |00|)
; If |old-new| > 1, then push 7-bit new and a flag of FFH.
; Else push flag of zero.
$CODE 5,'?DIFF',QDIFF
DB 0A4H          ;POP EA, new
DB 0A1H          ;POP BC, old
DB 09H           ;A<EAL
DB 60H,0E3H      ;A<0
DB 6BH,0FFH      ;C<FF
DB 60H,0EBH      ;A-C, skip if no zero, check diff=0.
DB 69H,0         ;A<0
DB 6BH,0         ;C<0, flag = 00
DB 47H,0FEH      ;A AND FE, skip if no zero
DB 0C7H          ;JMP AHEAD if |old-new| < 1.
DB 9H            ;A<EAL
DB 48H,21H       ;A SHIFT RIGHT, change 8bit to 7bit.
DB 19H           ;EAL<A
DB 0B4H          ;PUSH EA, Push 7-bit "new" value
DB 6BH,0FFH      ;C<FF, flag = FF

AHEAD:
DB 6AH,0         ;B<0
DB 0B1H          ;PUSH BC, Push flag.
$NEXT

; LDLAST
( --- )
; Moves ADC value to SLAST value
of the slider specified in MLOOP counter FFE0.
$CODE 6,'LDLAST',LDLAST
DB 1,0E0H        ;A<(V/E0)
DB 1BH           ;C<A
DB 6AH,0C6H      ;B<C6
DB 29H           ;A<(BC)
DB 19H           ;EAL<A
DB 0BH           ;A<C
: LDSTAT ( --- )
(Load current Byte1 to FFE1, last Midi status sent)
BYT1 FFE1 C!
;

: TSTAT ( --- )
(Check last Midi status sent & send new one if not =)
BYT1 DUP FFE1 C8 =
IF DROP
ELSE TM LDSTAT
THEN
;

MIDI OPERATIONS

: MCASE ( --- , Midi routines Case statement)
CASE DUMMY KEYN KEYAT CNTRL PRG CHAT PWHL DUMMY
;

: KEYN (Midi routine for Key On and Key Off.
   If the slider goes below the top of its travel and the
   note is off (as indicated by FLAG=0) then send a MIDI Note
   On message.
   If ADC=0 and the note is on (indicated by FLAG=FF)
   then send a Midi Note Off.)

   ADCV 0 =
   IF LDLAST FLAG
   IF BYT1 TM LDSTAT BYT2 TM FLGOFF 0 TM (Midi Note Off)
   ELSE EXIT
   THEN

   ELSE FLAG
   IF EXIT
   ELSE ADCV SLAST - 0<

   IF BYT1 TM LDSTAT BYT2 TM FLGON (Midi Note on)
   ADCV 2/ TM
   THEN LDLAST
   THEN

;
: **KEYAT**  (Midi routine for Key On and Key Off with Polyphonic After Touch.
   Turn on Note when the slider drops below the top of its rise.
   Continue sending polyphonic After Touch.
   Turn the Note off when the slider is brought back down to zero.)

   ADCV 0 =
   IF LDLAST FLAG
     IF BYT1 0F AND 90 OR TM (Midi Note Off)
       90 FFE1 C! (Load LST, last status)
       BYT2 TM FLGOFF 0 TM EXIT
     ELSE EXIT
   THEN

   ELSE FLAG
     IF SLAST ADCV ?DIFF
       IF TSTAT BYTE2 TM LDLAST TM EXIT (Midi After Touch)
       ELSE EXIT
     THEN

     ELSE ADCV SLAST - 0<
       IF BYT1 F0 AND 90 OR TM (Midi Note On)
         90 FFE1 C! (Load LST, last status)
         BYT2 TM FLGON ADCV 2/ TM
       THEN LDLAST
     THEN

   

: **CNTRL**  (Midi routine for Controller Data.  If the slider value has changed, then send the new value)

   SLAST ADCV ?DIFF  (Do nothing if no change)
   IF TSTAT BYTE2 TM LDLAST TM (Send Controller data)
   THEN

   

: **PRG**  (Midi routine for Program Changes.  When the slider goes above a set threshold then send a Midi Program Change once.)

   40 ADCV <
   IF FLAG
     IF EXIT
       ELSE BYT1 TM LDSTAT FLGON BYT2 TM (Send Midi Program)
       THEN
     ELSE FLGOFF  (If ADCV goes below threshold turn off FLAG)
       THEN

   ;

80
: **CHAT**  (Midi routine for Channel After Touch. If the slider value has changed, send the new value.)

SLAST ADCV ?DIFF  (Do nothing if no change)
IF TSTAT LDLAST TM  (Send Midi Channel Pressure)
THEN
;

: **PWHL**  (Midi routine for Pitch Wheel data. If the slider value has changed, send the new value.)

SLAST ADCV ?DIFF  (Do nothing if no change)
IF TSTAT 0 TM LDLAST TM  (Send Midi Pitch Wheel)
THEN
;

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

: **MIDI**  (---, Main Midi loop)

C700 40 0 FILL  (Reset FLAGs)
C640 40 0 FILL  (Reset SLASTs)
BEGIN   ADCINIT MLOOP MCASE
    SW@ NOT 7 AND  (Leave loop by pushing buttons 1,2 or 3)
UNTIL      (Returns to originating EDIT program)
;

Main Program

**HEX**

: BDOWN 20 LED! 0 eFLD C@ 7 AND CASE U/D0 U/D1 U/D2 U/D3
U/D3 U/D4 U/D5 U/D6 U/D7 ;

: DUMMY ;

: MCASE CASE DUMMY DUMMY DUMMY BLOAD BMIDI BUP BDOWN
BLEFT BRIGHT ;

: MAIN CR 1 LI DECIMAL 1 LI eUPDAT eDISP
BEGIN   0 LED! S@ MCASE    FFFF DELAY
NUF? UNTIL ." END MAIN " CR ;

HAND
Modified
Roland PG1000

Final Assembly Code
Register Use:   Interpreter Pointer = DE
                      Data Stack Pointer  = SP
                      Return Stack Pointer  = HL

                      Free to use: BC, EA, VA, Alternate Registers.

'doList' is accessed as a subroutine through a CALT instruction
           (Call to Jump Table). This shows up as a 'DB 80H' line
           in the $COLON and $USER Macros. When executed the
           processor jumps to an address vector located at 80H. The
           vectored 'doList' code is then located at 0F0H. The word
           'call,' was changed to load 80H into the code area for a
           doLST assembly.

A 9600 Baud serial I/O is provided. PortB/bit0 is used for serial
output and PortC/bit3 (INT2) is used for the serial input. The
serial input is interrupt driven with a vectored interrupt routine
located at 0A0H. The code words ?RX, TX!, and !IO make up the
rest of the serial I/O code. Three USER variables have been set
up for use by these serial I/O routines: SERIN, which holds the
received character and a flag; HAFBIT, which adjusts the software
timing of the receiver to read in the middle of each bit frame
(set it for 1/2 the BITIME minus 5); and BITIME, which adjusts the
software for a specific baud rate (17H for 9600 baud assuming a 12Mhz
processor clock).

The 78C10 is an 8-bit micro, therefore cell aligning to even addresses
is unnecessary. The $ALIGN Macro was taken out along with the NOP's
used for cell alignment in the other Macros. All occurrences of the
word ALGNDE were erased also. The word SEE no longer works because it
relies on cell alignment.

All of the system FORTH code is to be stored in ROM (up to 32K) starting
at address 0000H. Then there is 2K of RAM starting at address COOOH.
This memory setup required the following changes:
1) Return and Data stacks and TIB moved to RAM.
   (See the Memory allocation EQU assignments.)
2) The USER variables were moved to the micro's
   internal RAM at FF00H to FFFFH.
3) PAD word was changed to move the temporary buffer
   area to RAM space.
4) The vocabulary pointers found in the word FORTH were
   moved to RAM space by creating two new USER variables,
   FHEAD and FLINK and changing DOVOC to read:
   DW FHEAD,CNTXT,STORE,EXIT.
5) NTOP and CTOP were moved to RAM space to allow dictionary
   expansion into RAM space.

Several words were added to the ROM Dictionary. The simple operators
1+,1-,2+,2-,2*,2/, were defined in machine code. The words C,,
The NEC78C10 offers the following advantages:

1) Ten 16-bit internal registers and a 16-bit ALU.
   Many 16-bit instructions for those FORTH stack operations.
2) Three 8-bit I/O ports.
3) Eight 8-bit Analog to Digital Converters.
4) Internal counters and programmable clock generators.
5) Internal hardware serial I/O. (can be used for MIDI I/O).
6) 64K address space including 256 bytes of internal RAM.

---

```
; Version control
VER             EQU     01H                     ;major release version
EXT             EQU     01H                     ;minor extension

;; Constants
COMPO           EQU     040H                    ;lexicon compile only bit
IMEDD           EQU     080H                    ;lexicon immediate bit
MASKK           EQU     07F1FH                  ;lexicon bit mask
CELLL           EQU     2                       ;size of a cell
BASEE           EQU     10                      ;default radix
VOCSS           EQU     6                       ;depth of vocabulary stack
BKSPP           EQU     8                       ;backspace
LF              EQU     10                      ;line feed
CRR             EQU     13                      ;carriage return
ERR             EQU     27                      ;error escape
TIC             EQU     39                      ;tick

CALLL           EQU     80H                     ;CALT opcodes

;; Memory allocation
0//code>--//--<name//up>--<sp//tib>--rp//em

COLDD           EQU     00100H                  ;cold start
RPP             EQU     0C2F0H                  ;start of return stack (RP0)
TIBB            EQU     0C200H                  ;terminal input buffer (TIB)
SPP             EQU     0C1F0H                  ;start of data stack (SP0)
UPP             EQU     0FF00H                  ;start of user area (UP0)
NAMEE           EQU     03FFDH                  ;name dictionary
CODEE           EQU     00300H                  ;code dictionary
CTOP            EQU     0C390H                  ;RAM code dict. expansion
NTOP            EQU     0C7FFH                  ;RAM name dict. expansion
PADD            EQU     0C300H                  ;PAD area
SLDTR           EQU     00200H                  ;Table for Slider# translate
ADCINT          EQU     00280H                  ;ADC Interrupt routine

;; Initialize assembly variables
_LINK   = 0                                     ;force a null link
_NAME   = NAMEE                                 ;initialize name pointer
_CODE   = CODEE                                 ;initialize code pointer
_USER   = 4*CELLL                               ;first user variable offset

;; Define assembly macros
```
; Compile a code definition header.

$CODE MACRO LEX,NAME,LABEL
LABEL:
_CODE = $
_LEN = (LEX AND 01FH)/CELLL
_NAME = _NAME-((_LEN+3)*CELLL)
ORG _NAME
ORC CODE NAME
_LINK = $
DB LEX,NAME
ORG _CODE
ENDM

; Compile a colon definition header.

$COLON MACRO LEX,NAME,LABEL
$CODE LEX,NAME,LABEL
DB 80H
ENDM

; Compile a user variable header.

$USER MACRO LEX,NAME,LABEL
$CODE LEX,NAME,LABEL
DB 80H
DW DOUSE,_USER
_USER = _USER+CELLL
ENDM

; Compile an inline string.

D$ MACRO FUNCT,STRNG
DW FUNCT
_LEN = $
_DB = 0,STRNG
_CODE = $
ORG _LEN
DB _CODE-_LEN-1
ORG _CODE
ENDM

; Compile a stored string.

SD$ MACRO STRNG
DW DOLIT
_LEN = $+4
_DB = LEN,EXIT
_DB = 0,STRNG
_CODE = $
ORG _LEN
DB _CODE-_LEN-1
ORG _CODE
ENDM

; Assemble inline direct threaded code ending.
$NEXT MACRO
DB 48H,84H
; EA<(DE)++, next code address
into AX
DB 48H,28H
; JMP EA, jump directly to code address
ENDM

;; Main entry points and COLD start data

MAIN SEGMENT
ASSUME CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN

ORG 0000H

ORIG:  DB 54H,00,01,00                         ; RESET vector, JMP 0100H
DB 0AAH,62H,0,0                              ; NMI vector, EI RETI
DB 8 DUP(0)                                  ; INT T0/T1 vector
DB 54H,0A0H,00H, 5 DUP(0)                   ; INT1/2 vector, JMP 00A0H
DB 8 DUP(0)                                  ; INT E1/E0 vector
DB 54H,80H,02, 5 DUP(0)                     ; INT EIN/AD vector, JMP 0280H
DB 8 DUP(0)                                  ; INT SR/ST vector
DB 48 DUP(0)                                 ; FREE
DB 32 DUP(0)                                 ; SOFTI vector at 0060H

ORG 00A0H

; Vectored INT2 routine for Serial Input from Host Computer.
; Uses address FFF0 as a counter location - do not use elsewhere!
DB 0B1H ; PUSH BC
DB 0B2H ; PUSH DE
DB 0B0H ; PUSH VA
DB 68H,0FFH ; MVI, V<FF
DB 71H,0F0H,07H ; MVIW, (V/F0)<07, number of bits to receive.
DB 70H,1FH,04CH,0FFH ; LBCD, BC<(FF4C), wait for a half bit.
DB 53H ; DCR, C<C-1 skip, LOOP1
DB 0FEH ; JR, Jump to loop1
DB 52H ; DCR, B<B-1 skip
DB 0FCH ; JR, Jump to loop1
DB 70H,1FH,4EH,0FFH ; LBCD, BC<(FF4E), wait 1 bit time, LOOP2
DB 53H ; DCR, C<C-1 skip
DB 0FEH ; JR, Jump to loop2
DB 52H ; DCR, B<B-1 skip
DB 0FCH ; JR, Jump to loop2
DB 04CH,0C2H ; MOV, A<PC, read serial input on pc3
DB 48H,31H,48H,31H ; Rotate PC3 bit into CY
DB 48H,31H,48H,31H ; RLR, A rotate right 4xs
DB 0CH ; MOV, A<D, D collects the bits
DB 48H,31H ; RLR, shift in next bit, CY to top of D
DB 1CH ; MOV, D<A
DB 30H,0F0H ; DCRW, (V/F0)<(V/F0)-1 skip
DB 0B7H ; JR, Jump to loop2 for next bit.
DB 70H,1FH,4EH,0FFH ; LBCD, BC<(FF4E)
DB 53H,0FEH,52H,0FCH ; DCR JR DCR JR, stop bit loop time.
DB 71H,04BH,0FFH ; MVIW, (V/4B)<FF, load flag
DB 0CH,63H,04AH ; MOV STAW, A<D (V/4A)<A, load data
DB 0A0H,0A2H,0A1H ; POP, restore AV DE and BC
DB 48H,44H,0 ; SKIT, NOP
DB 0AAH,062H ; EI RETI, enable interrupts and return
;; Kernel doLST routine. Always accessed by the CALT instruction: 80H
;; which is a Call Subroutine to jump to address vector located at 0080H.

ORG 00F0H
DB 33H,33H ;HL<HL-2
DB 0A6H ;EA<DE
DB 48H,93H ;(HL)<EA
DB 0A2H ;POP DE previously pushed by CALT
DB 48H,84H ;EA<(DE)+, $NEXT
DB 48H,28H ;JMP EA

ORG 0080H
DB 0F0H,0 ; set up vector to doLST

;; Table for translating Slider numbers into an orderly sequence as
;; physically set up on the Roland PG1000 board.

ORG SLDTR
DB 48,56,40,37,53,61,45,35
DB 51,59,43,33,49,57,41,38
DB 54,62,46,30,32,16,08,00
DB 24,21,13,05,29,19,11,03
DB 44,27,17,09,01,25,22,14
DB 06,34,50,60,52,36,20,12
DB 04,28,18,10,02,26,42,58
DB 07,15,23,31,39,47,55,63

ORG COLDD ;Beginning of Cold Boot
DB 69H,0FH,4DH,0D0H ;MM<0F, memory map (11-8)
DB 69H,OFFH,4DH,0D2H ;MA<FF, pa inputs (4-2)
DB 69H,00H,4DH,0D3H ;MB<00, pb outputs (4-6)
DB 64H,01H,05H ;PB<5
DB 4DH,0D7H ;MF<00, pf outputs (4-15)
DB 69H,0AH,4DH,0D4H ;MC<0A, pc1/3 inputs (4-9)
DB 69H,0BH,4DH,0D1H ;MCC<0B, pc mode (4-8)
DB 64H,02H,04H ;PC<04
DB 64H,81H,06H ;SMH<06, serial mode (7-7)
DB 69H,4EH,4DH,0CAH ;SML<4E, serial mode (7-9)
DB 04H ;SP<SPP, stack pointer=data stack
DB LOW SPP
DB HIGH SPP
DB 34H ;HL<RPP, HL=return stack pointer
DB LOW RPP
DB HIGH RPP
DB 69H,00H,4DH,0E8H ;ZCM<0, zero cross disabled (3-26)
DB 68H,OFFH ;V<FF
DB 10H,68H,0FFH,69H,0 ;V'<FF, A"<0, V<FF, A<0

;; timer setups for Midi and LCD use
DB 69H,64H,4DH,0DAH ;TM0<64, timer0 (5-1)
DB 69H,0FFH,4DH,0DBH ;TM1<FF, timer1 (5-1)
DB 64H,85H,0B3H ;TMM<B3, timer mode (5-6)
DB 44H,60H,0EAH,48H,0D3H ;ETM1<EA = EA60 (6-2)
DB 64H,83H,0CCH ;EOM<CC, timer event mode (6-14)
DB 69H,5CH,4DH,0CCH ;ETMM<5C, timer event mode (6-11)
DB 54H,00,03H ;JMP to 0300, high level cold start
; COLD start moves the following to USER variables.
; MUST BE IN SAME ORDER AS USER VARIABLES.

UZERO:          DW      4 DUP (0)               ;reserved
                DW      SPP                     ;SP0
                DW      RPP                     ;RP0
                DW      QRX                     ;'?KEY
                DW      TXSTO                   ;'EMIT
                DW      ACCEP                   ;'EXPECT
                DW      KTAP                    ;'TAP
                DW      TXSTO                   ;'ECHO
                DW      DOTOK                   ;'PROMPT
                DW      BASEE                   ;BASE
                DW      0                       ;tmp
                DW      0                       ;SPAN
                DW      0                       ;>IN
                DW      0                       ;#TIB
                DW      TIBB                    ;TIB
                DW      0                       ;CSP
                DW      INTER                   ;'EVAL
                DW      NUMBQ                   ;'NUMBER
                DW      0                       ;HLD
                DW      0                       ;HANDLER
                DW      0                       ;CONTEXT pointer
                DW      VOCSS DUP (0)           ;vocabulary stack
                DW      0                       ;CURRENT pointer
                DW      0                       ;vocabulary link pointer
                DW      0                       ;FORTH HEAD
                DW      0                       ;FORTH LINK
                DW      CTOP                    ;CP
                DW      NTOP                    ;NP
                DW      LASTN                   ;LAST
                DW      0                       ;SERIN host receive char & flag
                DW      06H                     ;HAFBIT time for serial host,
                DW      16H                     ;BITIME baud for serial host

ULAST:

ORG ADCINT

; Interrupt routine for Analog to Digital Converters
DB 10H                     ;EXA
DB 11H                     ;EXX
; Load ADC Address and Counter into HL.  Uses FFF2 and FFF3.
DB 68H,0FFH                ;V'<FF
DB 69H,0C6H                ;A<C6
DB 1EH                     ;H<A
DB 01H,0F3H                ;A<(V/F3)
DB 1FH                     ;L<A
; Store ADC 0.
DB 4CH,0E0H                ;A<CR0
DB 3DH                     ;(HL)+<A, store new value
; Store ADC 1.
DB 4CH,0E1H ;A<CR1
DB 3DH ;(HL)+<A, store new value
; Store ADC 2.
DB 4CH,0E2H ;A<CR2
DB 3DH ;(HL)+<A, store new value
; Store ADC 3.
DB 4CH,0E3H ;A<CR3
DB 3DH ;(HL)+<A, store new value
; Update Counters
DB 0FH ;A<L
DB 63H,0F3H ;(V/F3)<A, Load counter.
DB 48H,25H ;A shift logical left
DB 48H,05H ;A shift left, skip if carry (if end count)
DB 0C7H ;JMP AHEAD
DB 64H,1EH,01 ;MKH<MKH OR 1, disable interrupts
DB 10H,11H,0AAH,62H ;Return from interrupts

DB 1AH ;B<A, "AHEAD"
DB 74H,0AH,0E0H ;B<B AND E0
DB 4CH,0C2H ;A<Pc
DB 07H,1FH ;A<A AND 1F
DB 60H,9AH ;A<A OR B
DB 4DH,0C2H ;Pc<A, Load high 3 bits of slider select.
DB 64H,90H,08H ;Invert ANM bit and restart conversion.

; Return from Interrupt.
DB 10H ;EXA
DB 11H ;EXX
DB 0AAH ;EI
DB 62H ;RETI

ORG CODEE ;start code dictionary

; COLD
; The hilevel cold start sequence.
CCOLD = $ §
$COLON 4,'COLD',COLD

COLD1: DW DOLIT, UZERO, DOLIT, UPP
        DW DOLIT, ULAST - UZERO, CMOVE ;initialize user area
        DW PRESE ;initialize stack and TIB
        DW TBOOT, ATEXE ;application boot
        DW FORTH, CNTXT, AT, DUPP ;initialize search order
        DW CRRNT, DSTOR, OVERT
        DW LCDIN ;initialize LCD
        DW EDIT ;Autostart EDIT
        DW QUIT ;start interpretation
        DW BRAN, COLD1 ;just in case

;; Device dependent I/O

; BYE
; Exit eForth.
$CODE 3,'BYE', BYE
DB 54H, 0, 0 ;JMP Reset Vector

; ?RX
; Return input character and true, or a false if no input.
; TX! ( c -- )
; Send character c to the output device.

$CODE 3,'TX!',TXSTO
DB 0BAH ;Disable Interrupts
DB 0A1H ;POP BC, pop char into C
DB 0B2H ;PUSH DE, store interpreter pointer
DB 0BH,1CH ;A<C, D<A, char in A and D
DB 68H,0FFH ;V<FF
DB 71H,0F0H,07H ;(V/F0)<7
DB 60H,91H ;A<A EXOR A
DB 6DH,01H ;E<01
DB 68H,0FFH ;V<FF
DB 69H,0FFH,4DH,0C7H ;MKL<EF, enable int2 interrupt and
DB 69H,0FFH,4DH,0C6H ;MKH<FF, disable all others with mask

; !IO ( -- )
; Initialize the serial I/O devices.

$CODE 3,'!IO',STOIO
DB 69H,0EFH,4DH,0C7H ;MKL<EF, enable int2 interrupt and
DB 69H,0FFH,4DH,0C6H ;MKH<FF, disable all others with mask
DB 0AAH ;EI, enable interrupt
$NEXT

;; The kernel

; doLIT ( -- w )
; Push an inline literal.

$CODE  COMPO+5,'doLIT',DOLIT
DB 48H,84H ;EA<(DE)++
DB 0B4H ;PUSH EA
$NEXT

; EXIT ( -- )
; Terminate a colon definition.

$CODE  4,'EXIT',EXIT
DB 48H,85H ;EA<(HL)++
DB 0B6H ;DE<EA
$NEXT

; EXECUTE ( ca -- )
; Execute the word at ca.

$CODE  7,'EXECUTE',EXECU
DB 0A1H ;POP BC
DB 21H ;JMP BC

; next ( -- )
; Run time code for the single index loop.
; : next ( -- ) \ hilevel model
; r> r> dup if 1 - >r @ >r exit then drop cell+ >r ;

$CODE  COMPO+4,'next',DONXT
DB 6AH,0 ;B<00
DB 6BH,1 ;C<01
DB 48H,83H ;EA<(HL)
DB 74H,0B5H ;EA<EA-BC Skip if no borrow
DB 0C9H ;JMP NEXT1
DB 48H,93H ;(HL)<EA
DB 48H,82H ;EA<(DE)
DB 0B6H ;DE<EA
$NEXT

NEXT1: DB 22H,22H ;DE<DE+2
DB 32H,32H ;HL<HL+2
$NEXT

; ?branch ( f -- )
; Branch if flag is zero.

$CODE  COMPO+7,'?branch',QBRAN
DB 6AH,0FFH ;B<FF
DB 6BH,0FFH ;C<FF
DB 0A4H ;POP EA
DB 74H,0CDH ;EA AND BC Skip if not zero
DB 0C6H ;JMP BRAN1
DB 22H,22H ;DE<DE+2
BRAN1: DB 48H, 82H ; EA<(DE)
DB 0B6H ; DE<EA

; branch  ( -- )
; Branch to an inline address.

$CODE COMPO+6,'branch',BRAN
DB 48H, 82H ; EA<(DE)
DB 0B6H ; DE<EA

; ! ( w a -- )
; Pop the data stack to memory.

$CODE 1,'!',STORE
DB 0A1H ; POP BC, address
DB 0A4H ; POP EA, data
DB 09H ; A<EAL
DB 39H ; (BC)<A
DB 12H ; BC<BC+1
DB 08H ; A<EAH
DB 39H ; (BC)<A

; @ ( a -- w )
; Push data at memory location to the data stack.

$CODE 1,'@',AT
DB 0A1H ; POP BC
DB 29H ; A<(BC)
DB 19H ; EAL<A
DB 12H ; BC<BC+1
DB 29H ; A<(BC)
DB 18H ; EAH<A
DB 0B4H ; PUSH EA

; C! ( c b -- )
; Pop the data stack to byte memory.

$CODE 2,'C!','CSTOR
DB 0A1H ; POP BC address
DB 0A4H ; POP AE data
DB 09H ; A<EAL
DB 39H ; (BC)<A

; C@ ( b -- c )
; Push byte memory location to the data stack.

$CODE 2,'C@','CAT
DB 0A1H ; POP BC
DB 29H ; A<(BC)
DB 6AH, 0 ; B<00
DB 1BH ; C<A
DB 0B1H ; PUSH BC
$NEXT

; RP@ ( -- a )
; Push the current RP to the data stack.

$CODE 3,'RP@',RPAT
DB 0B3H ;PUSH HL
$NEXT

; RP! ( a -- )
; Set the return stack pointer.

$CODE COMPO+3,'RP!',RPSTO
DB 0A3H ;POP HL
$NEXT

; R> ( -- w )
; Pop the return stack to the data stack.

$CODE 2,'R>',RFROM
DB 48H,85H ;EA<(HL)++
DB 0B4H ;PUSH EA
$NEXT

; R@ ( -- w )
; Copy top of return stack to the data stack.

$CODE 2,'R@',RAT
DB 48H,83H ;EA<(HL)
DB 0B4H ;PUSH EA
$NEXT

; >R ( w -- )
; Push the data stack to the return stack.

$CODE COMPO+2,'>R',TOR
DB 33H,33H ;HL<HL-2
DB 0A4H ;POP EA
DB 48H,93H ;(HL)<EA
$NEXT

; SP@ ( -- a )
; Push the current data stack pointer.

$CODE 3,'SP@',SPAT
DB 70H,0EH,0FEH,0FFH ;(FFFE)<SP
DB 70H,1FH,0FEH,0FFH ;BC<(FFFE)
DB 0B1H ;PUSH BC
$NEXT

; SP! ( a -- )
; Set the data stack pointer.

$CODE 3,'SP!',SPSTO
DB 0A1H ;POP BC
DB 70H,1EH,0FEH,0FFH ;(FFFE)<BC
DB 70H,0FH,0FEH,0FFH ;PC<(FFFE)
$NEXT
;  DROP
(  w -- )
;  Discard top stack item.

$CODE 4,'DROP',DROP
DB 0A4H ;POP EA
$NEXT

;  DUP
(  w -- w w )
;  Duplicate the top stack item.

$CODE 3,'DUP',DUPP
DB 0A4H ;POP EA
DB 0B4H ;PUSH EA
DB 0B4H ;PUSH EA
$NEXT

;  SWAP
(  w1 w2 -- w2 w1 )
;  Exchange top two stack items.

$CODE 4,'SWAP',SWAP
DB 0A4H ;POP EA
DB 0A1H ;POP BC
DB 0B4H ;PUSH EA
DB 0B1H ;PUSH BC
$NEXT

;  OVER
(  w1 w2 -- w1 w2 w1 )
;  Copy second stack item to top.

$CODE 4,'OVER',OVER
DB 0A4H ;POP AE
DB 0A1H ;POP BC
DB 0B1H ;PUSH BC
DB 0B4H ;PUSH AE
DB 0B1H ;PUSH BC
$NEXT

;  0<
(  n -- t )
;  Return true if n is negative.

$CODE 2,'0<',ZLESS
DB 0A1H ;POP BC
DB 69H,0FFH ;A<FF
DB 48H,06H ;B Shift Left, Skip if carry
DB 69H,0 ;A<00
DB 1AH ;B<A
DB 1BH ;C<A
DB 0B1H ;PUSH BC
$NEXT

;  AND
(  w w -- w )
;  Bitwise AND.

$CODE 3,'AND',ANDD
DB 0A1H ;POP BC
DB 0A4H ;POP AE
DB 74H,8DH ;EA<EA AND BC
; OR   ( w w -- w )
;      Bitwise inclusive OR.

$CODE 2,'OR',ORR
DB 0A1H ;POP BC
DB 0A4H ;POP EA
DB 74H,9DH ;EA<EA OR BC
DB 0B4H ;PUSH EA
$NEXT

; XOR  ( w w -- w )
;      Bitwise exclusive OR.

$CODE 3,'XOR',XORR
DB 0A1H ;POP BC
DB 0A4H ;POP EA
DB 74H,95H ;EA<EA EX-OR BC
DB 0B4H ;PUSH EA
$NEXT

; UM+  ( w w -- w cy )
;      Add two numbers, return the sum and carry flag.

$CODE 3,'UM+',UPLUS
DB 0A1H ;POP BC
DB 0A4H ;POP EA
DB 69H,0 ;A<00
DB 74H,0A5H ;EA<EA+BC Skip if no carry
DB 41H ;A<A+1
DB 1BH ;C<A
DB 6AH,0 ;B<00
DB 0B4H ;PUSH EA
DB 0B1H ;PUSH BC
$NEXT

;; System and user variables

; doVAR  ( -- a )
;      Run time routine for VARIABLE and CREATE.

$COLON COMPO+5,'doVAR',DOVAR
DW RFROM,EXIT

; UP    ( -- a )
;      Pointer to the user area.

$COLON 2,'UP',UP
DW DOVAR
DW UPP

; doUSER  ( -- a )
;      Run time routine for user variables.

$COLON COMPO+6,'doUSER',DOUSE
DW RFROM,AT,UP,AT,PLUS,EXIT
; SP0 ( -- a )
; Pointer to bottom of the data stack.
$USER 3,'SP0',SZERO

; RP0 ( -- a )
; Pointer to bottom of the return stack.
$USER 3,'RP0',RZERO

; '?KEY ( -- a )
; Execution vector of ?KEY.
$USER 5,"'?KEY",TQKEY

; 'EMIT ( -- a )
; Execution vector of EMIT.
$USER 5,"'EMIT",TEMIT

; 'EXPECT ( -- a )
; Execution vector of EXPECT.
$USER 7,"'EXPECT",TEXPE

; 'TAP ( -- a )
; Execution vector of TAP.
$USER 4,"'TAP",TTAP

; 'ECHO ( -- a )
; Execution vector of ECHO.
$USER 5,"'ECHO",TECHO

; 'PROMPT ( -- a )
; Execution vector of PROMPT.
$USER 7,"'PROMPT",TPROM

; BASE ( -- a )
; Storage of the radix base for numeric I/O.
$USER 4,'BASE',BASE

; tmp ( -- a )
; A temporary storage location used in parse and find.
$USER COMPO+3,'tmp',TEMP

; SPAN ( -- a )
; Hold character count received by EXPECT.
$USER 4,'SPAN',SPAN

; >IN ( -- a )
; Hold the character pointer while parsing input stream.
$USER 3,'>IN',INN

; #TIB ( -- a )
; Hold the current count and address of the terminal input buffer.

$USER 4,'#TIB',NTIB
_USER = _USER+CELLL

; CSP ( -- a )
; Hold the stack pointer for error checking.

$USER 3,'CSP',CSP

; 'EVAL ( -- a )
; Execution vector of EVAL.

$USER 5,'EVAL',TEVAL

; 'NUMBER ( -- a )
; Execution vector of NUMBER?.

$USER 7,'NUMBER',TNUMB

; HLD ( -- a )
; Hold a pointer in building a numeric output string.

$USER 3,'HLD',HLD

; HANDLER ( -- a )
; Hold the return stack pointer for error handling.

$USER 7,'HANDLER',HANDL

; CONTEXT ( -- a )
; A area to specify vocabulary search order.

$USER 7,'CONTEXT',CNTXT
_USER = _USER+VOCSS*CELLL ;vocabulary stack

; CURRENT ( -- a )
; Point to the vocabulary to be extended.

$USER 7,'CURRENT',CRRNT
_USER = _USER+CELLL ;vocabulary link pointer

; FHEAD ( -- a )
; Point to the FORTH vocab head pointer.
$USER 5,'FHEAD',FHEAD

; FLINK ( -- a )
; Point to the FORTH vocab link pointer.
$USER 5,'FLINK',FLINK

; CP ( -- a )
; Point to the top of the code dictionary.

$USER 2,'CP',CP
NP
(Point to the bottom of the name dictionary.)

$USER 2,'NP',NP

LAST
(Point to the last name in the name dictionary.)

$USER 4,'LAST',LAST

SERIN
(Point to host serial input. Flag in high, char in low byte.)

$USER 5,'SERIN',SERIN

HAFBIT
(Point to half bit time used by serial i/o routines.)

$USER 6,'HAFBIT',HAFBIT

BITIME
(Point to bit time used to set serial i/o baud rate.)

$USER 6,'BITIME',BITIME

; Common functions

doVOC
(Run time action of VOCABULARY's.)

$COLON COMPO+5,'doVOC',DOVOC
    DW FHEAD,CNTXT,STORE,EXIT

FORTH
(Make FORTH the context vocabulary.)

$COLON 5,'FORTH',FORTH
    DW DOVOC,EXIT
    ; Head and Link pointers normally here were moved to User Ram.

?DUP
(Dup tos if its is not zero.)

$CODE 4,'?DUP',QDUP
    DB 6AH,0FFH ;B<FF
    DB 6BH,0FFH ;C<FF
    DB 0A4H ;POP EA
    DB 74H,0DDH ;EA AND BC, Skip if zero
    DB 0B4H ;PUSH EA
    DB 0B4H ;PUSH EA
    $NEXT

ROT
(Rot 3rd item to top.)

$COLON 3,'ROT',ROT
    DW TOR,SWAP,RFROM,SWAP,EXIT
; 2DROP  ( w w -- )
; Discard two items on stack.

$CODE 5,'2DROP',DDROP
DB 0A4H,0A4H ;POP EA, POP EA
$NEXT

; 2DUP  ( w1 w2 -- w1 w2 w1 w2 )
; Duplicate top two items.

$CODE 4,'2DUP',DDUP
DB 0A4H,0A1H ;POP EA, POP BC
DB 0B1H,0B4H ;PUSH BC, PUSH EA
DB 0B1H,0B4H ;PUSH BC, PUSH EA
$NEXT

; +  ( w w -- sum )
; Add top two items.

$CODE 1,'+',PLUS
DB 0A1H,0A4H ;POP BC, POP EA
DB 74H,0A5H ;EA<EA+BC, Skip
DB 0 ;NOP
DB 0B4H ;PUSH EA
$NEXT

; D+  ( d d -- d )
; Double addition, as an example using UM+.

; $COLON 2,'D+',DPLUS
; DW TOR,SWAP,TOR,UPLUS
; DW RFROM,RFROM,PLUS,PLUS,EXIT

; NOT  ( w -- w )
; One's complement of tos.

$CODE 3,'NOT',INVER
DB 0A1H ;POP BC
DB 69H,0FFH ;A<FF
DB 60H,12H ;B<B EX-OR A
DB 60H,13H ;C<C EX-OR A
DB 0B1H ;PUSH BC
$NEXT

; NEGATE  ( n -- -n )
; Two's complement of tos.

$CODE 6,'NEGATE',NEGAT
DB 0A1H ;POP BC
DB 69H,0FFH ;A<FF
DB 60H,12H ;B<B EX-OR A
DB 60H,13H ;C<C EX-OR A
DB 12H ;BC<BC+1
DB 0B1H ;PUSH BC
$NEXT
; DNEGATE ( d -- -d )
; Two's complement of top double.

$COLON 7,'DNEGATE',DNEGA
DW INVER,TOR,INVER
DW DOLIT,1,UPLUS
DW RFROM,PLUS,EXIT

; - ( n1 n2 -- n1-n2 )
; Subtraction.

$CODE 1, '-', SUBB
DB 0A1H ; POP BC
DB 069H,0FFH ; A<FF
DB 060H,12H ; B<B EX-OR A
DB 060H,13H ; C<C EX-OR A
DB 12H ; BC<BC+1
DB 0A4H ; POP EA
DB 74H,0A5H ; EA<EA+BC Skip
DB 0 ; NOP
DB 0B4H ; PUSH EA
$NEXT

; ABS ( n -- n )
; Return the absolute value of n.

$COLON 3,'ABS', ABSS
DW DUPP, ZLESS
DW QBRAN,ABS1
DW NEGAT
ABS1: DW EXIT

; = ( w w -- t )
; Return true if top two are equal.

$CODE 1, '=', EQUAL
DB 0A4H,0A1H ; POP EA, POP BC
DB 69H,0FFH ; A<FF
DB 74H,0FDH ; EA-BC, Skip if zero
DB 69H,00H ; A<00
DB 1AH,1BH ; B<A, C<A
DB 0B1H ; PUSH BC
$NEXT

; U< ( u u -- t )
; Unsigned compare of top two items.

$COLON 2,'U<', ULESS
DW DDUP, XORR, ZLESS
DW QBRAN, ULES1
DW SWAP, DROP, ZLESS, EXIT
ULES1: DW SUBB, ZLESS, EXIT

; < ( n1 n2 -- t )
; Signed compare of top two items.

$COLON 1,'<', LESS
DW DDUP, XORR, ZLESS
DW QBRAN,LESS
DW DROP,ZLESS,EXIT
LESS1:
DW SUBB,ZLESS,EXIT

; MAX
( n n -- n )
Return the greater of two top stack items.

$CODE 3,'MAX',MAX
DB 0A4H,0A1H ;POP EA, POP BC
DB 74H,0BDH ;EA-BC, Skip if borrow
DB 0C2H ;Jump to Push EA
DB 0B1H ;PUSH BC
DB 0C1H ;Jump to next
DB 0B4H ;PUSH EA
$NEXT

; MIN
( n n -- n )
Return the smaller of top two stack items.

$CODE 3,'MIN',MIN
DB 0A4H,0A1H ;POP EA, POP BC
DB 74H,0BDH ;EA-BC, Skip if borrow
DB 0C2H ;Jump to Push EA
DB 0B4H ;PUSH EA
DB 0C1H ;Jump to next
DB 0B1H ;PUSH BC
$NEXT

; WITHIN
( u ul uh -- t )
Return true if u is within the range of ul and uh.

$COLON 6,'WITHIN',WITHI
DW OVER,SUBB,TOR ;ul <= u < uh
DW SUBB,RFROM,ULESS,EXIT

;; Quick Operators

; 1+ ( n -- n+1 )
$CODE 2,'1+',ONEP
DB 0A1H ;POP BC
DB 12H ;BC<BC+1
DB 0B1H ;PUSH BC
$NEXT

; 1- ( n -- n-1 )
$CODE 2,'1-','ONEM
DB 0A1H ;POP BC
DB 013H ;BC<BC-1
DB 0B1H ;PUSH BC
$NEXT

; 2+ ( n -- n+2 )
$CODE 2,'2+','TWOP
DB 0A1H ;POP BC
DB 12H,12H ;BC<BC+2
DB 0B1H ;PUSH BC
$NEXT

; 2-
( n -- n-2 )
$CODE 2,'2-',TWOM
DB 0A1H ;POP BC
DB 13H,13H ;BC<BC-2
DB 0B1H ;PUSH BC
$NEXT

; 2*
( n -- n*2 )
$CODE 2,'2*',TWOSL
DB 0A4H ;POP EA
DB 48H,0A4H ;EA Logical Shift Left
DB 0B4H ;PUSH EA
$NEXT

; 2/
( n -- n/2 )
$CODE 2,'2/',TWOSR
DB 0A4H ;POP EA
DB 48H,0A0H ;EA Logical Shift Right
DB 0B4H ;PUSH EA
$NEXT

;; Divide

; UM/MOD
( udl udh u -- ur uq )
Unsigned divide of a double by a single. Return mod and quotient.

$COLON 6,'UM/MOD',UMMOD
DW DDUP,ULESS
DW QBRAN,UMM4
DW NEGAT,DOLIT,15,TOR
Umm1:
DW TOR,DUPP,UPLUS
DW TOR,TOR,DUPP,UPLUS
DW RFROM,PLUS,DUPP
DW RFROM,RAT,SWAP,TOR
DW UPLUS,RFROM,ORR
DW QBRAN,UMM2
DW TOR,DROP,ONEP,RFROM
DW BRAN,UMM3
Umm2:
DW DROP
Umm3:
DW RFROM
DW DNNXT,UMM1
DW DROP,SWAP,EXIT
Umm4:
DW DROP,DDROP
DW DOLIT,-1,DUPP,EXIT ;overflow, return max

; M/MOD
( d n -- r q )
Signed floored divide of double by single. Return mod and quotient.

$COLON 5,'M/MOD',MSMOD
DW DUPP,ZLESS,DUPP,TOR
DW QBRAN,MMOD1
DW NEGAT,TOR,DNEGA,RFROM
MMOD1:          DW      TOR,DUPP,ZLESS
                DW      QBRAN,MMOD2
                DW      RAT,PLUS
MMOD2:          DW      RFROM,UMMOD,RFROM
                DW      QBRAN,MMOD3
                DW      SWAP,NEGAT,SWAP
MMOD3:          DW      EXIT

;   /MOD        ( n n -- r q )
;   Signed divide. Return mod and quotient.

$COLON  4,'/MOD',SLMOD
DW      OVER,ZLESS,SWAP,MSMOD,EXIT

;   MOD         ( n n -- r )
;   Signed divide. Return mod only.

$COLON  3,'MOD',MODD
DW      SLMOD,DROP,EXIT

;   /           ( n n -- q )
;   Signed divide. Return quotient only.

$COLON  1,'/',SLASH
DW      SLMOD,SWAP,DROP,EXIT

;; Multiply

;   UM*         ( u u -- ud )
;   Unsigned multiply. Return double product.

$COLON  3,'UM*',UMSTA
DW      DOLIT,0,SWAP,DOLIT,15,TOR
UMST1:          DW      DUPP,UPLUS,TOR,TOR
                DW      DUPP,UPLUS,RFROM,PLUS,RFROM
                DW      QBRAN,UMST2
UMST2:          DW      TOR,OVER,UPLUS,RFROM,PLUS
                DW      DONXT,UMST1
                DW      ROT,DROP,EXIT

;   *           ( n n -- n )
;   Signed multiply. Return single product.

$COLON  1,'*',STAR
DW      UMSTA,DROP,EXIT

;   M*          ( n n -- d )
;   Signed multiply. Return double product.

$COLON  2,'M*',MSTAR
DW      DDUP,XORR,ZLESS,TOR
DW      ABSS,SWAP,ABSS,UMSTA
DW      RFROM
DW      QBRAN,MSTA1
DW      DNEGA
MSTA1:          DW      EXIT

;   */MOD       ( n1 n2 n3 -- r q )
Multiply n1 and n2, then divide by n3. Return mod and quotient.

\[ n1 \times n2 \div n3 \]

Miller, Lord, and Sussman, 
CONSISTENT SCHEME, 1977

ulty n1 by n2, then divide by n3. Return quotient only.

\[ \frac{n1}{n2} \div n3 \]

;; Miscellaneous

BL

( -- 32 )
Return 32, the blank character.

\[ 32 \]

>&CHAR

( c -- c )
Filter non-printing characters.

\[ c \rightarrow \ _ \]

DEPTH

( -- n )
Return the depth of the data stack.

\[ n \]

PICK

( ... +n -- ... w )
Copy the nth stack item to tos.

\[ w \rightarrow \ ... +n \]

+!

( n a -- )
Add n to the contents at address a.

\[ a + n \]

2!

( d a -- )
Store the double integer to address a.

\[ d \rightarrow a \]
;  2@  ( a -- d )
;  Fetch double integer from address a.

$COLON  2,'2@',DAT
DW       DUPP,TWOP,AT
DW       SWAP,AT,EXIT

;  COUNT  ( b -- b +n )
;  Return count byte of a string and add 1 to byte address.

$COLON  5,'COUNT',COUNT
DW       DUPP,ONEP
DW       SWAP,AT,EXIT

;  HERE  ( -- a )
;  Return the top of the code dictionary.

$COLON  4,'HERE',HERE
DW       CP,AT,EXIT

;  PAD  ( -- a )
;  Return the address of a temporary buffer.

$COLON  3,'PAD',PAD
DW       DOLIT,PADD,EXIT

;  TIB  ( -- a )
;  Return the address of the terminal input buffer.

$COLON  3,'TIB',TIB
DW       NTIB,TWOP,AT,EXIT

;  @EXECUTE  ( a -- )
;  Execute vector stored in address a.

$COLON  8,'@EXECUTE',ATEXE
DW       AT,QDUP       ;?address or zero
DW       QBRAN,EXE1
DW       EXECU        ;execute if non-zero
EXE1:    DW       EXIT       ;do nothing if zero

;  CMOVE  ( b1 b2 u -- )
;  Copy u bytes from b1 to b2.

$COLON  5,'CMOVE',CMOVE
DW       TOR
DW       BRAN,CMOV2
CMOV1:   DW       TOR,DUPP,CAT
          DW       RAT,CSTOR
          DW       ONEP
          DW       RFROM,ONEP
CMOV2:   DW       DONXT,CMOV1
          DW       DDROP,EXIT

;  FILL  ( b u c -- )
;  Fill u bytes of character c to area beginning at b.
$COLON  4,'FILL',FILL
DW      SWAP, TOR, SWAP
DW      BRAN, FILL2

FILL1:
DW      DDUP, CSTOR, ONEP

FILL2:
DW      DONXT, FILL1
DW      DDROP, EXIT

;   -TRAILING   ( u b -- b u )
; Adjust the count to eliminate trailing white space.

$COLON  9,'-TRAILING',DTRAI
DW      TOR
DW      BRAN, DTRA2

DTRA1:
DW      BLANK, OVER, RAT, PLUS, CAT, LESS
DW      QBRAN, DTRA2
DW      RFROM, ONEP, EXIT ; adjusted count

DTRA2:
DW      DONXT, DTRA1
DW      DOLIT, 0, EXIT ; count=0

;   PACK$   ( b u a -- a )
; Build a counted string with u characters from b. Null fill.

$COLON  5,'PACK$',PACKS
DW      DUPP, TOR ; strings only on cell boundary
DW      OVER, DUPP, DOLIT, 0
DW      DOLIT, CELL, UMMOD, DROP ; count mod cell
DW      SUBB, OVER, PLUS
DW      DOLIT, 0, SWAP, STORE ; null fill cell
DW      DDUP, CSTOR, ONEP ; save count
DW      SWAP, CMOVE, RFROM, EXIT ; move string

;; Numeric output, single precision

;   DIGIT   ( u -- c )
; Convert digit u to a character.

$COLON  5,'DIGIT',DIGIT
DW      DOLIT, 9, OVER, LESS
DW      DOLIT, 7, ANDD, PLUS
DW      DOLIT, '0', PLUS, EXIT

;   EXTRACT   ( n base -- n c )
; Extract the least significant digit from n.

$COLON  7,'EXTRACT',EXTRC
DW      DOLIT, 0, SWAP, UMMOD
DW      SWAP, DIGIT, EXIT

;   <#   ( -- )
; Initiate the numeric output process.

$COLON  2,'<#',BDIGS
DW      PAD, HLD, STORE, EXIT

;   HOLD   ( c -- )
; Insert a character into the numeric output string.

$COLON  4,'HOLD',HOLD
DW      HLD,AT,ONEM
DW      DUPP,HLD,STORE,CSTOR,EXIT

;   #    ( u -- u )
; Extract one digit from u and append the digit to output string.

$COLON 1,'#',DIG
DW      BASE,AT,EXTRC,HOLD,EXIT

;   #S    ( u -- 0 )
; Convert u until all digits are added to the output string.

$COLON 2,'#S',DIGS
DIGS1:
   DW      DIG,DUPP
   DW      QBRAN,DIGS2
DIGS2:
   DW      EXIT

;   SIGN   ( n -- )
; Add a minus sign to the numeric output string.

$COLON 4,'SIGN',SIGN
DW      ZLESS
DW      QBRAN,SIGN1
DW      DOLIT,'-',HOLD
SIGN1:
   DW      EXIT

;   #>    ( w -- b u )
; Prepare the output string to be TYPE'd.

$COLON 2,'#>',EDIGS
DW      DROP,HLD,AT
DW      PAD,OVER,SUBB,EXIT

;   str    ( n -- b u )
; Convert a signed integer to a numeric string.

$COLON 3,'str',STR
DW      DUPP,TOR,ABSS
DW      BDIGS,DIGS,RFROM
DW      SIGN,EDIGS,EXIT

;   HEX    ( -- )
; Use radix 16 as base for numeric conversions.

$COLON 3,'HEX',HEX
DW      DOLIT,16,BASE,STORE,EXIT

;   DECIMAL   ( -- )
; Use radix 10 as base for numeric conversions.

$COLON 7,'DECIMAL',DECIM
DW      DOLIT,10,BASE,STORE,EXIT

;; Numeric input, single precision

;   DIGIT?   ( c base -- u t )
; Convert a character to its numeric value. A flag indicates success.

$COLON 6,'DIGIT?',DGTQ
DW TOR,DOLIT,'0',SUBB
DW DOLIT,9,OVER,LESS
DW QBRAN,DGTQ1
DW DOLIT,7,SUBB
DW DUPP,DOLIT,10,LESS,ORR
DGTQ1:
DW DUPP,RFROM,ULESS,EXIT

; NUMBER? ( a -- n T | a F )
; Convert a number string to integer. Push a flag on tos.

$COLON 7,'NUMBER?',NUMBQ
DW BASE,AT,TOR,DOLIT,0,OVER,COUNT
DW OVER,CAT,DOLIT,'%',EQUAL
DW QBRAN,NUMQ1
DW HEX,SWAP,ONEP
DW SWAP,ONEM
NUMQ1:
DW OVER,CAT,DOLIT,'%-',EQUAL,TOR
DW SWAP,RAT,SUBB,SWAP,RAT,PLUS,QDUP
DW QBRAN,NUMQ6
DW ONEM,TOR
NUMQ2:
DW DUPP,TOR,CAT,BASE,AT,DIGTQ
DW QBRAN,NUMQ4
DW SWAP,BASE,AT,STAR,PLUS,RFROM
DW ONEP
DW DONXT,NUMQ2
DW RAT,SWAP,DROP
DW QBRAN,NUMQ3
DW NEGAT
NUMQ3:
DW SWAP
DW BRAN,NUMQ5
NUMQ4:
DW RFROM,RFROM,DDROP,DDROP,DOLIT,0
NUMQ5:
DW DUPP
NUMQ6:
DW RFROM,DDROP
DW RFROM,BASE,STORE,EXIT

;; Basic I/O

; ?KEY ( -- c T | F )
; Return input character and true, or a false if no input.

$COLON 4,'?KEY',QKEY
DW TQKEY,ATEXE,EXIT

; KEY ( -- c )
; Wait for and return an input character.

$COLON 3,'KEY',KEY
KEY1:
DW QKEY
DW QBRAN,KEY1
DW EXIT

; EMIT ( c -- )
; Send a character to the output device.
; NUF? ( -- t )
; Return false if no input, else pause and if CR return true.

$COLON 4,'NUF?',NUFQ
DW QKEY,DUPP
DW QBRAN,NUFQ1
DW DDROP,KEY,DOLIT,CRR,EQUAL
NUFQ1:
DW EXIT

; PACE ( -- )
; Send a pace character for the file downloading process.

$COLON 4,'PACE',PACE
DW DOLIT,11,EMIT,EXIT

; SPACE ( -- )
; Send the blank character to the output device.

$COLON 5,'SPACE',SPACE
DW BLANK,EMIT,EXIT

; SPACES ( +n -- )
; Send n spaces to the output device.

$COLON 6,'SPACES',SPACS
DW DOLIT,0,MAX,TOR
DW BRAN,CHAR2
CHAR1:
DW SPACE
CHAR2:
DW DONXT,CHAR1
DW EXIT

; TYPE ( b u -- )
; Output u characters from b.

$COLON 4,'TYPE',TYPEE
DW TOR
DW BRAN,TYPE2
TYPE1:
DW DUPP,CAT,EMIT
DW ONEP
TYPE2:
DW DONXT,TYPE1
DW DROP,EXIT

; CR ( -- )
; Output a carriage return and a line feed.

$COLON 2,'CR',CR
DW DOLIT,CRR,EMIT
DW DOLIT,LF,EMIT,EXIT

; do$ ( -- a )
; Return the address of a compiled string.

$COLON COMPO+3,'do$',DOSTR
DW RFROM,RAT,RFROM,COUNT,PLUS
DW TOR,SWAP,TOR,EXIT
; $"| ( -- a )
; Run time routine compiled by ". Return address of a compiled string.

$COLON  COMPO+3,'$"|',STRQP
DW   DOSTR,EXIT ;force a call to do$

; ."| ( -- )
; Run time routine of ". Output a compiled string.

$COLON  COMPO+3,'."|',DOTQP
DW   DOSTR,COUNT,TYPEE,EXIT

; .R ( n +n -- )
; Display an integer in a field of n columns, right justified.

$COLON  2,'.R',DOTR
DW   TOR,STR,RFROM,OVER,SUBB
DW   SPACS,TYPEE,EXIT

; U.R ( u +n -- )
; Display an unsigned integer in n column, right justified.

$COLON  3,'U.R',UDOTR
DW   TOR,BDIGS,DIGS,EDIGS
DW   RFROM,OVER,SUBB
DW   SPACS,TYPEE,EXIT

; U. ( u -- )
; Display an unsigned integer in free format.

$COLON  2,'U.',UDOT
DW   BDIGS,DIGS,EDIGS
DW   SPACE,TYPEE,EXIT

; . ( w -- )
; Display an integer in free format, preceded by a space.

$COLON  1,'.',DOT
DW   BASE,AT,DOLIT,10,XORR  ;?decimal
DW   QBRAN,DOT1
DW   UDOT,EXIT         ;no, display unsigned

DOT1:
DW   STR,SPACE,TYPEE,EXIT ;yes, display signed

; ? ( a -- )
; Display the contents in a memory cell.

$COLON  1,'?',QUEST
DW   AT,DOT,EXIT

;; Parsing

; parse ( b u c -- b u delta ; <string> )
; Scan string delimited by c. Return found string and its offset.

$COLON  5,'parse',PARS
DW   TEMP,STORE,OVER,TOR,DUPP

110
H:

; PARSE ( c -- b u ; <string> )
; Scan input stream and return counted string delimited by c.

$COLON 5,'PARSE',PARSE
DW TOR,TIB,INN,AT,PLUS ;current input buffer pointer
DW NTIB,AT,INN,AT,SUBB ;remaining count
DW RFROM,PARS,INN,PSTOR,EXIT

; .( ( -- )
; Output following string up to next ).

$COLON IMEDD+2,'./(,DOTPR
DW DOLIT,''),PARSE,TYPEE,EXIT

; ( ( -- )
; Ignore following string up to next ). A comment.

$COLON IMEDD+1,./(,PAREN
DW DOLIT,''),PARSE,DDROP,EXIT

; \ ( ( -- )
; Ignore following text till the end of line.

$COLON IMEDD+1,\',BKSLA
DW NTIB,AT,INN,STORE,EXIT

; CHAR ( -- c )
; Parse next word and return its first character.

$COLON 4,'CHAR',CHAR
DW BLANK,PARSE,DROP,CAT,EXIT

; TOKEN ( -- a ; <string> )
; Parse a word from input stream and copy it to name dictionary.

$COLON 5,'TOKEN',TOKEN
DW BLANK,PARSE,DOLIT,31,MIN
DW NP,AT,OVER,SUBB,TWOM
DW PACKS,EXIT

; WORD ( c -- a ; <string> )
; Parse a word from input stream and copy it to code dictionary.

$COLON 4,'WORD',WORDDD
DW PARSE,HERE,PACKS,EXIT

;; Dictionary search

; NAME> ( na -- ca )
; Return a code address given a name address.

$COLON 5,'NAME>',NAMET
DW TWOM,TWOM,AT,EXIT

; SAME? ( a a u -- a a f \ -0+ )
; Compare u cells in two strings. Return 0 if identical.

$COLON 5,'SAME?',SAMEQ
DW TOR
DW BRAN,SAME2
SAME1: DW OVER,RAT,TWOSL,PLUS,AT
DW OVER,RAT,TWOSL,PLUS,AT
DW SUBB,QDUP
DW QBRAN,SAME2
DW RFROM,DROP,EXIT ;strings not equal
SAME2: DW DONXT,SAME1
DW DOLIT,0,EXIT ;strings equal

; find ( a va -- ca na | a F )
; Search a vocabulary for a string. Return ca and na if succeeded.

$COLON 4,'find',FIND
DW SWAP,DUPP,CAT
DW DOLIT,CELLL,SLASH,TEMP,STORE
DW DUPP,AT,TOR,TWOP,SWAP
FIND1: DW AT,DUPP
DW QBRAN,FIND6
DW DUPP,AT,DOLIT,MASKK,ANDD,RAT,XORR
DW QBRAN,FIND2
DW TWOP,DOLIT,-1 ;true flag
DW BRAN,FIND3
FIND2: DW TWOP,TEMP,AT,SAMEQ
FIND3: DW BRAN,FIND4
FIND6: DW RFROM,DROP
DW SWAP,TWOM,SWAP,EXIT
FIND4: DW QBRAN,FIND5
DW TWOM,TWOM
DW BRAN,FIND1
FIND5:          DW      RFROM,DROP,SWAP,DROP
          DW      TWOM
          DW      DUPP,NAMET,SWAP,EXIT

;   NAME?       ( a -- ca na | a F )
;         Search all context vocabularies for a string.
$COLON  5,'NAME?',NAMEQ
          DW      CNTXT,DUPP,DAT,XORR       ;?context=also
          DW      QBRAN,NAMQ1
          DW      TWOM                     ;no, start with context
NAMQ1:          DW      TOR
NAMQ2:          DW      RFROM,TWOP,DUPP,TOR   ;next in search order
          DW      AT,QDUP
          DW      QBRAN,NAMQ3
          DW      FIND,QDUP                 ;search vocabulary
          DW      QBRAN,NAMQ2
          DW      RFROM,DROP,EXIT           ;found name
NAMQ3:          DW      RFROM,DROP
          DW      DOLIT,0,EXIT             ;false flag

;;  Terminal response

;   ^H          ( bot eot cur -- bot eot cur )
;         Backup the cursor by one character.
$COLON  2,'^H',BKSP
          DW      TOR,OVER,RFROM,SWAP,OVER,XORR
          DW      QBRAN,BACK1
          DW      DOLIT,BKSPPP,TECHO,ATEXE,ONEM
          DW      BLANK,TECHO,ATEXE
          DW      DOLIT,BKSPPP,TECHO,ATEXE
BACK1:          DW      EXIT

;   TAP         ( bot eot cur c -- bot eot cur )
;         Accept and echo the key stroke and bump the cursor.
$COLON  3,'TAP',TAP
          DW      DUPP,TECHO,ATEXE
          DW      OVER,CSTOR,ONEP,EXIT

;   kTAP        ( bot eot cur c -- bot eot cur )
;         Process a key stroke, CR or backspace.
$COLON  4,'kTAP',KTAP
          DW      DUPP,DOLIT,CRR,XORR
          DW      QBRAN,KTAP2
          DW      DOLIT,BKSPPP,XORR
          DW      QBRAN,KTAP1
          DW      BLANK,TAP,EXIT
KTAP1:          DW      BKSP,EXIT
KTAP2:          DW      DROP,SWAP,DROP,DUPP,EXIT

;   accept      ( b u -- b u )
;         Accept characters to input buffer. Return with actual count.
$COLON  6,'accept',ACCEP
          DW      OVER,PLUS,OVER
ACCP1:          DW      DDUP,XORR  
               DW      QBRAN,ACCP4  
               DW      KEY,DUPP  
               DW      BLANK,SUBB,DOLIT,95,ULESS  
               DW      BLANK,DOLIT,127,WITHI  
               DW      QBRAN,ACCP2  
               DW      TAP  
               DW      BRAN,ACCP3  
ACCP2:          DW      TTAP,ATEXE  
ACCP3:          DW      BRAN,ACCP1  
ACCP4:          DW      DROP,OVER,SUBB,EXIT  

;   EXPECT    ( b u -- )  
;   Accept input stream and store count in SPAN.  
$COLON  6,'EXPECT',EXPEC  
               DW      TEXPE,ATEXE,SPAN,STORE,DROP,EXIT  

;   QUERY    ( -- )  
;   Accept input stream to terminal input buffer.  
$COLON  5,'QUERY',QUERY  
               DW      TIB,DOLIT,80,TEXPE,ATEXE,NTIB,STORE  
               DW      DROP,DOLIT,0,INN,STORE,EXIT  

;; Error handling  

;; CATCH      ( ca -- 0 | err# )  
;   Execute word at ca and set up an error frame for it.  
$COLON  5,'CATCH',CATCH  
               DW      SPAT,TOR,HANDL,AT,TOR ;save error frame  
               DW      RPAT,HANDL,STORE,EXECU ;execute  
               DW      RFROM,HANDL,STORE ;restore error frame  
               DW      RFROM,DROP,DOLIT,0,EXIT ;no error  

;   THROW      ( err# -- err# )  
;   Reset system to current local error frame and update error flag.  
$COLON  5,'THROW',THROW  
               DW      HANDL,AT,RPSTO ;restore return stack  
               DW      RFROM,HANDL,STORE ;restore handler frame  
               DW      RFROM,SWAP,TOR,SPSTO ;restore data stack  
               DW      DROP,RFROM,EXIT  

;   NULL$      ( -- a )  
;   Return address of a null string with zero count.  
$COLON  5,'NULL$',NULLS  
               DW      DOVAR ;emulate CREATE  
               DW      0  
               DB      99,111,121,111,116,101  

;   ABORT     ( -- )  
;   Reset data stack and jump to QUIT.  
$COLON  5,'ABORT',ABORT  
               DW      NULLS,THROW  

114
; abort"  ( f -- )
; Run time routine of ABORT" . Abort with a message.

$COLON  COMPO+6,'abort"',ABORQ
DW  QBRAN,ABOR1 ;text flag
DW  DOSTR,THROW ;pass error string
ABOR1:    DW  DOSTR,DROP,EXIT ;drop error

;; The text interpreter

;  $INTERPRET  ( a -- )
; Interpret a word. If failed, try to convert it to an integer.

$COLON  10,'$INTERPRET',INTER
DW  NAMEQ,QDUP ;?defined
DW  QBRAN,INTE1
DW  AT,DOLIT,COMPO,ANDD ;?compile only lexicon bits
D$  ABORQ,' compile only'
DW  EXECU,EXIT ;execute defined word
INTE1:    DW  TNUMB,ATEXE ;convert a number
DW  QBRAN,INTE2
DW  EXIT
INTE2:    DW  THROW ;error

;  [           ( -- )
; Start the text interpreter.

$COLON  IMEDD+1,'
',LBRAC
DW  DOLIT,INTER,TEVAL,STORE,EXIT

;  .OK         ( -- )
; Display 'ok' only while interpreting.

$COLON  3,'.OK',DOTOK
DW  DOLIT,INTER,TEVAL,AT,EQUAL
DW  QBRAN,DOTO1
D$  DOTQP,' ok'
DOTO1:    DW  CR,EXIT

;  ?STACK      ( -- )
; Abort if the data stack underflows.

$COLON  6,'?STACK',QSTAC
DW  DEPTH,2LESS ;check only for underflow
D$  ABORQ,' underflow'
DW  EXIT

;  EVAL        ( -- )
; Interpret the input stream.

$COLON  4,'EVAL',EVAL
EVAL1:    DW  TOKEN,DUPP,CAT ;?input stream empty
DW  QBRAN,EVAL2
DW  TEVAL,ATEXE,QSTAC ;evaluate input, check stack
DW  BRAN,EVAL1
EVAL2:    DW  DROP,TPROM,ATEXE,EXIT ;prompt
;; Shell

;; PRESET
;; ( -- )
;; Reset data stack pointer and the terminal input buffer.

$COLON 6,'PRESET',PRESE
DW SZERO,AT,SPSTO
DW DOLIT,TIBB,NTIB,TWOP,STORE,EXIT

;; xio
;; ( a a a -- )
;; Reset the I/O vectors 'EXPECT', 'TAP', 'ECHO and 'PROMPT.

$COLON COMPO+3,'xio',XIO
DW DOLIT,ACCEP,TEXPE,DSTOR
DW TECHO,DSTOR,EXIT

;; FILE
;; ( -- )
;; Select I/O vectors for file download.

$COLON 4,'FILE',FILE
DW DOLIT,PACE,DOLIT,DROP
DW DOLIT,KTAP,XIO,EXIT

;; HAND
;; ( -- )
;; Select I/O vectors for terminal interface.

$COLON 4,'HAND',HAND
DW DOLIT,DOTOK,DOLIT,EMIT
DW DOLIT,KTAP,XIO,EXIT

;; I/O
;; ( -- a )
;; Array to store default I/O vectors.

$COLON 3,'I/O',ISLO
DW DOVAR                   ;emulate CREATE
DW QRX,TXSTO               ;default I/O vectors

;; CONSOLE
;; ( -- )
;; Initiate terminal interface.

$COLON 7,'CONSOLE',CONSO
DW ISLO,DAT,TQKEY,DSTOR    ;restore default I/O device
DW HAND,EXIT               ;keyboard input

;; QUIT
;; ( -- )
;; Reset return stack pointer and start text interpreter.

$COLON 4,'QUIT',QUIT
QUIT1: DW RZERO,AT,RPSTO          ;reset return stack pointer
QUIT2: DW LBRAC                   ;start interpretation
       DW QUERY                   ;get input
       DW DOLIT,EVAL,CATCH,QDUP   ;evaluate input
       DW QBRAN,QUIT2             ;continue till error
       DW TPROM,AT,SWAP           ;save input device
       DW CONSO,NULLS,OVER,XORR   ;display error message
       DW QBRAN,QUIT3
       DW SPACE,COUNT,TYPEE       ;error message
       D$ DOTQP,' ? '             ;error prompt
QUIT3:          DW      DOLIT,DOTOK,XORR        ;?file input
                DW      QBRAN,QUIT4
                DW      DOLIT,ERR,EMIT          ;file error, tell host
QUIT4:          DW      PRESE                   ;some cleanup
                DW      BRAN,QUIT1

;; The compiler

;   ' ( -- ca )
; Search context vocabularies for the next word in input stream.

$COLON  1,"",TICK
        DW      TOKEN,NAMEQ             ;?defined
        DW      QBRAN,TICK1
        DW      EXIT                    ;yes, push code address
TICK1:          DW      THROW                   ;no, error

;   ALLOT ( n -- )
; Allocate n bytes to the code dictionary.

$COLON  5,'ALLOT',ALLOT
        DW      CP,PSTOR,EXIT           ;adjust code pointer

;   , ( w -- )
; Compile an integer into the code dictionary.

$COLON  1,',',COMMA
        DW      HERE,DUPP,TWOP         ;cell boundary
        DW      CP,STORE,STORE,EXIT     ;adjust code pointer, compile

;   C, ( b -- )
; Compile a byte into the code dictionary

$COLON  2,'C,',CCOMMA
        DW      HERE,DUPP,ONEP
        DW      CP,STORE,CSTOR,EXIT

;   [COMPILE] ( -- ; <string> )
; Compile the next immediate word into code dictionary.

$COLON  IMEDD+9,'[COMPILE]','BCOMP
        DW      TICK,COMMA,EXIT

;   COMPILE ( -- )
; Compile the next address in colon list to code dictionary.

$COLON  COMPO+7,'COMPILE','COMPI
        DW      RFROM,DUPP,AT,COMMA     ;compile address
        DW      TWOP,TOR,EXIT          ;adjust return address

;   LITERAL ( w -- )
; Compile tos to code dictionary as an integer literal.

$COLON  IMEDD+7,'LITERAL','LITER
        DW      COMPI,DOLIT,COMMA,EXIT

;   $," ( -- )
; Compile a literal string up to next ".

117
$COLON 3,",",STRCQ
DW  DOLIT,",",WORDD           ;move string to code dictionary
DW  COUNT,PLUS                   ;calculate aligned end of string
DW  CP,STORE,EXIT                ;adjust the code pointer

; RECURSE ( -- )
; Make the current word available for compilation.

$COLON IMEDD+7,'RECURSE',RECUR
DW  LAST,AT,NAMET,COMMA,EXIT

;; Structures

; FOR ( -- a )
; Start a FOR-NEXT loop structure in a colon definition.

$COLON IMEDD+3,'FOR',FOR
DW  COMPI,TOR,HERE,EXIT

; BEGIN ( -- a )
; Start an infinite or indefinite loop structure.

$COLON IMEDD+5,'BEGIN',BEGIN
DW  HERE,EXIT

; NEXT ( a -- )
; Terminate a FOR-NEXT loop structure.

$COLON IMEDD+4,'NEXT',NEXT
DW  COMPI,DONXT,COMMA,EXIT

; UNTIL ( a -- )
; Terminate a BEGIN-UNTIL indefinite loop structure.

$COLON IMEDD+5,'UNTIL',UNTIL
DW  COMPI,QBRAN,COMMA,EXIT

; AGAIN ( a -- )
; Terminate a BEGIN-AGAIN infinite loop structure.

$COLON IMEDD+5,'AGAIN',AGAIN
DW  COMPI,BRAN,COMMA,EXIT

; IF ( -- A )
; Begin a conditional branch structure.

$COLON IMEDD+2,'IF',IFF
DW  COMPI,QBRAN,HERE
DW  DOLIT,0,COMMA,EXIT

; AHEAD ( -- A )
; Compile a forward branch instruction.

$COLON IMEDD+5,'AHEAD',AHEAD
DW  COMPI,BRAN,HERE,DOLIT,0,COMMA,EXIT

; REPEAT ( A a -- )
; Terminate a BEGIN-WHILE-REPEAT indefinite loop.

$COLON IMEDD+6,'REPEAT',REPEA
DW AGAIN,HERE,SWAP,STORE,EXIT

; THEN
( A -- )
; Terminate a conditional branch structure.

$COLON IMEDD+4,'THEN',THENN
DW HERE,SWAP,STORE,EXIT

; AFT
( a -- a A )
; Jump to THEN in a FOR-AFT-THEN-NEXT loop the first time through.

$COLON IMEDD+3,'AFT',AFT
DW DROP,AHEAD,BEGIN,SWAP,EXIT

; ELSE
( A -- A )
; Start the false clause in an IF-ELSE-THEN structure.

$COLON IMEDD+4,'ELSE',ELSEE
DW AHEAD,SWAP,THENN,EXIT

; WHILE
( a -- A a )
; Conditional branch out of a BEGIN-WHILE-REPEAT loop.

$COLON IMEDD+5,'WHILE',WHILE
DW IFF,SWAP,EXIT

; ABORT"
( -- ; <string> )
; Conditional abort with an error message.

$COLON IMEDD+6,'ABORT"',ABRTQ
DW COMPI,ABORQ,STRCQ,EXIT

; "$"
( -- ; <string> )
; Compile an inline string literal.

$COLON IMEDD+2,'$"',STRQ
DW COMPI,STRQP,STRCQ,EXIT

; ."
( -- ; <string> )
; Compile an inline string literal to be typed out at run time.

$COLON IMEDD+2,'."',DOTQ
DW COMPI,DOTQP,STRCQ,EXIT

;; Name compiler

; ?UNIQUE
( a -- a )
; Display a warning message if the word already exists.

$COLON 7,'?UNIQUE',UNIQU
DW DUPP,NAMEQ ;?name exists
DW QBRAK,UNIQ1 ;redefinitions are OK
DS DOTQP,' reDef ' ;but warn the user
DW OVER,COUNT,TYPEE ;just in case its not planned

UNIQ1:
DW DROP,EXIT
; $,n  ( na -- )
; Build a new dictionary name using the string at na.

$COLON 3,'$',SNAME
DW   DUPP,CAT      ;?null input
DW   QBRAN,PNAM1   ;?redefinition
DW   UNIQU         ;?redefinition
DW   DUPP,LAST,STORE ;save na for vocabulary link
DW   HERE,SWAP     ;align code address
DW   TWOM          ;link address
DW   CRRNT,AT,AT,OVER,STORE
DW   TWOM,DUPP,NP,STORE ;adjust name pointer
DW   STORE,EXIT    ;save code pointer

PNAM1: DS $STRQP,' name' ;null input
DW  THROW

;; FORTH compiler

; $COMPILE  ( a -- )
; Compile next word to code dictionary as a token or literal.

$COLON 8,],'$COMPILE',SCOMP
DW   NAMEQ,QDUP     ;?defined
DW   QBRAN,SCOM2     ;?immediate
DW   AT,DOLIT,IMEDD,ANDD ;?immediate
DW   QBRAN,SCOM1     ;its immediate, execute
SCOM1: DW   COMMA,EXIT  ;its not immediate, compile
SCOM2: DW   TNUMB,ATEXE ;try to convert to number
DW   QBRAN,SCOM3     ;compile number as integer
SCOM3: DW   LITER,EXIT  ;error

; CCOMPILE   ( a -- )
; Compile next byte to code dictionary as machine code.

$COLON 8,],'CCOMPILE',CCOMP
DW   NAMEQ,QDUP     ;?defined
DW   QBRAN,CCOM2     ;?immediate
DW   AT,DOLIT,IMEDD,ANDD ;?immediate
DW   QBRAN,CCOM1     ;its immediate, execute
CCOM1: DW   DROP,EXIT   ;its not immediate,drop
CCOM2: DW   TNUMB,ATEXE ;try to convert to number
DW   QBRAN,CCOM3     ;compile as code byte
CCOM3: DW   CCOMMA,EXIT ;error

; OVERT     ( -- )
; Link a new word into the current vocabulary.

$COLON 5,]'OVERT',OVERT
DW   LAST,AT,CRRNT,AT,STORE,EXIT

; ;     ( -- )
; Terminate a colon definition.
( -- )
Start compiling the words in the input stream.

;   ]
; Start compiling the words in the input stream.

; call,   ( ca -- )
Assemble a call instruction to doLST.

;   ; IMMEDIATE   ( -- )
Make the last compiled word an immediate word.

;; Defining words

;   USER        ( u -- ; <string> )
Compile a new user variable.

; CREATE      ( -- ; <string> )
Compile a new array entry without allocating code space.

; VARIABLE    ( -- ; <string> )
Compile a new variable initialized to 0.

; CODE        ( -- )
Start a new code definition using next word as its name.
; ENCODE ( -- )
; Terminate a code definition

$COLON IMEDD+COMPO+7,'ENCODE',ENDCD
DW DOLIT,48H,COMMA,DOLIT,84H,COMMA ;$NEXT
DW DOLIT,48H,COMMA,DOLIT,28H,COMMA
DW LBRAC,OVERT,EXIT

;; Tools

; _TYPE ( b u -- )
; Display a string. Filter non-printing characters.

$COLON 5,'_TYPE',UTYPE
DW TOR ;start count down loop
DW BRAN,UTYP2 ;skip first pass
UTYP1: DW DUPP,CAT,TCHAR,EMIT ;display only printable
DW ONEP ;increment address
UTYP2: DW DONXT,UTYP1 ;loop till done
DW DROP,EXIT

; dm+ ( a u -- a )
; Dump u bytes from , leaving a+u on the stack.

$COLON 3,'dm+',DMP
DW OVER,DOLIT,4,UDOTR ;display address
DW SPACE,TOR ;start count down loop
DW BRAN,PDUM2 ;skip first pass
PDUM1: DW DUPP,CAT,DOLIT,3,UDOTR ;display numeric data
DW ONEP ;increment address
PDUM2: DW DONXT,PDUM1 ;loop till done
DW DROP,EXIT

; DUMP ( a u -- )
; Dump u bytes from a, in a formatted manner.

$COLON 4,'DUMP',DUMP
DW BASE,AT,TOR,HEX ;save radix, set hex
DW DOLIT,16,SLASH ;change count to lines
DW TOR ;start count down loop
DUMP1: DW CR,DOLIT,16,DDUP,DMP ;display numeric
DW ROT,ROT
DW SPACE,SPACE,UTYPE ;display printable characters
DW NUFQ,INVER ;user control
DW QBRAN,DUMP2
DW DONXT,DUMP1 ;loop till done
DW BRAN,DUMP3
DUMP2: DW RFROM,DROP ;cleanup loop stack, early exit
DUMP3: DW DROP,RFROM,BASE,STORE ;restore radix
DW EXIT

; .S ( ... -- ... )
; Display the contents of the data stack.

$COLON 2,'.S',DOTS
DW CR,DEPTH ;stack depth
DW TOR ;start count down loop
DW BRAN,DOTS2 ;skip first pass
DOTS1:    DW      RAT,PICK,DOT         ;index stack, display contents
DOTS2:    DW      DONXT,DOTS1       ;loop till done
D$      DOTQP,' <sp'
DW      EXIT

;   !CSP        ( -- )
;   Save stack pointer in CSP for error checking.
$COLON  4,'!CSP',STCSP
DW      SPAT,CSP,STORE,EXIT     ;save pointer

;   ?CSP        ( -- )
;   Abort if stack pointer differs from that saved in CSP.
$COLON  4,'?CSP',QCSP
DW      SPAT,CSP,AT,XORR        ;compare pointers
D$      ABORQ,'stacks'         ;abort if different
DW      EXIT

;   >NAME       ( ca -- na | F )
;   Convert code address to a name address.
$COLON  5,'>NAME',TNAME
DW      CRRNT                   ;vocabulary link
TNAM1:          DW      TWOP,AT,QDUP       ;check all vocabularies
DW      QBRAN,TNAM4
DW      DDUP

TNAM2:          DW      AT,DUPP            ;?last word in a vocabulary
DW      QBRAN,TNAM3
DW      DDUP,NAMET,XORR         ;compare
DW      QBRAN,TNAM3
DW      TWOM                   ;continue with next word
DW      BRAN,TNAM2

TNAM3:          DW      SWAP,DROP,QDUP
DW      QBRAN,TNAM1
DW      SWAP,DROP,SWAP,DROP,EXIT
TNAM4:          DW      DROP,DOLIT,0,EXIT       ;false flag

;   .ID         ( na -- )
;   Display the name at address.
$COLON  3,'.ID',DOTID
DW      QDUP                    ;if zero no name
DW      QBRAN, DOTI1
DW      COUNT,DOLIT,01FH,ANDD   ;mask lexicon bits
DW      UTYPE,EXIT              ;display name string

DOTI1:          D$      DOTQP,' {noName}'
DW      EXIT

;   WORDS       ( -- )
;   Display the names in the context vocabulary.
$COLON  5,'WORDS',WORDS
DW      CR,CNTXT,AT             ;only in context
WORS1:          DW      AT,QDUP       ;?at end of list
DW      QBRAN, WORS2
DW      DUPP,SPACE,DOTID        ;display a name
DW      TWOM,NUFQ               ;user control
DW QBRAN, WORS1
DW DROP

WORS2:
DW EXIT

;; Hardware reset

; VER ( -- n )
; Return the version number of this implementation.

$COLON 3, 'VER', VERSN
DW DOLIT, VER*256 + EXT, EXIT

; hi ( -- )
; Display the sign-on message of eForth.

$COLON 2, 'hi', HI
DW STOIO, CR ; initialize I/O
DS DOTQP, 'eForth v'
DW BASE, AT, HEX
DW VERSN, BDIGS, DIG, DIG
DW DOLIT, '.', HOLD
DW DIGS, EDIGS, TYPEE
DW BASE, STORE, CR, EXIT

; 'BOOT ( -- a )
; The application startup vector.

$COLON 5, 'BOOT', TBOOT
DW DOVAR

; SEE ( --word-- )
; Decompiles word.

$COLON 3, 'SEE', SEE
DW TICK
DW CR, ONEP
SEE1:
DW DUPP, DUPP, SPACE, DOT, DOLIT, 07CH, EMIT, AT, DUPP
DW QBRAN, SEE2
DW TNAME
SEE2:
DW QDUP
DW QBRAN, SEE3
DW DOTID
DW BRAN, SEE4
SEE3:
DW DUPP, AT, UDOT
SEE4:
DW TWOP, NUFQ
DW QBRAN, SEE1
DW DROP, EXIT

; ADCINIT ( -- )
; Init routine for starting ADC Interrupts

$CODE 7, 'ADCINIT', ADCIN
DB 64H, 4EH, 1 ; MKH AND 1, skip if not zero
DB 0D1H ; JUMP TO $NEXT
DB 64H, 0AH, 1FH ; PC<PC AND 1F
DB 68H, 0FFH ; V<FF
DB 69H, 0 ; A<0
DB 63H, 0F3H ; (V/F3)<A
; ADCOFF
; ( --- )
; Disable ADC Interrupts.
$CODE 6,'ADCOFF',ADCOF
DB 64H,1EH,1 ;;MKH < MKH OR 1
$NEXT

; TM
; ( n -- )
; Wait for last transmit, then send midi byte n.
$CODE 2,'TM',TM
DB 0A1H ;POP BC
DB 0BH ;A<C
DB 48H,4AH ;SKIT FST, skip if interrupt
DB 0FDH ;JMP TO SKIT
DB 4DH,0D8H ;MOV TXB,A
$NEXT

; DELAY
; ( n -- )
; Wait for n loops.
$CODE 5,'DELAY',DELAY
DB 0A1H ;POP BC
DB 53H ;C<C-1, Skip if borrow
DB 0FEH ;JMP
DB 52H ;B<B-1, Skip if borrow
DB 0FCH ;JMP
$NEXT

; LCD
; ( n -- )
; Load control n to LCD display.
$CODE 3,'LCD',LCD
DB 0A1H ;POP BC
DB 0BH ;A<C
DB 14H,0,0A0H ;BC<A000
DB 39H ;(BC)<A
$NEXT

; LLI
; ( --- )
; Sets RS=0 for LCD setup commands.
$CODE 3,'LLI',LLI
DB 64H,0AH,0EFH ;Pc<Pc AND EF
$NEXT

; LLC
; ( --- )
; Sets RS=1 for LCD character loading
$CODE 3,'LLC',LLC
DB 64H,1AH,10H ;Pc<Pc OR 10
$NEXT

; LI
; ( n --- )
; load LCD setup instruction n, exit ready for char loads
$COLON 2,'LI',LI
DW LLI,LCD,LLC,DOLIT,01FFH,DELAY,EXIT

125
; LCDINIT ( -- )
; Initialize LCD display.
$COLON 7,'LCDINIT',LCDIN
DW DOLIT,0D7AH,DELAY
DW DOLIT,038H,LI
DW DOLIT,047EH,DELAY
DW DOLIT,038H,LI
DW DOLIT,017H,DELAY
DW DOLIT,038H,LI
DW DOLIT,017H,DELAY
DW DOLIT,038H,LI
DW DOLIT,017H,DELAY
DW DOLIT,08H,LI
DW DOLIT,017H,DELAY
DW DOLIT,01H,LI
DW DOLIT,01CCH,DELAY
DW DOLIT,02H,LI
DW DOLIT,01CCH,DELAY
DW DOLIT,06H,LI
DW DOLIT,17H,DELAY
DW DOLIT,0EH,LI
DW DOLIT,17H,DELAY
DW EXIT

; #DISP  ( n,p --- )
; Display n as a 3-digit number at LCD position p.
$COLON 5,'#DISP',NDISP
DW DUPP,LI,SWAP
DW BDIGS,DIG,DIG,DIG,EDIGS
DW DROP,DUPP,CAT,LCD,ONEP
DW DUPP,CAT,LCD,ONEP,CAT,LCD,LI,EXIT

; #2DISP  ( n,p --- )
; Display n as a 3-digit number at LCD position p.
$COLON 6,'#2DISP',N2DISP
DW DUPP,LI,SWAP
DW BDIGS,DIG,DIG,DIG,EDIGS
DW DROP,DUPP,CAT,LCD,ONEP
DW CAT,LCD,LI,EXIT

; DISP  ( a,p --- )
; Display packed string at a to LCD position p.
$COLON 4,'DISP',DISP
DW LI,DUPP,CAT,ONEM,TOR

DISP1: DW ONEP
DW DUPP,CAT,LCD
DW DONXT,DISP1
DW DROP,EXIT

; CASE  ( n --- )
; Execute one of a list of words pointed to by n.
$COLON 4,'CASE',CASE
DW RFROM,SWAP,TWOSL,PLUS
DW ATEXE,EXIT
; INCR
; Increment n mod nmax.
INCR:     DW ONEP
;
;
; DECR
; Decrement n mod nmax.
DECR:     DW ONEM
;
;
; SW@
; Read Roland switches as a byte.
SWAT:     DB 4CH,0C0H  ; ;A<PA
          DB 6AH,0      ; ;B<0
          DB 1BH         ; ;C<A
          DB 0B1H        ; ;PUSH BC
$NEXT
;
;
; S@
; Return number of the lowest Roland switch on.
SAT:      DB 4CH,0C0H  ; ;A<PA
          DB 6BH,0      ; ;C<0
          DB 74H,11H,0FFH  ; ;A<EXOR FFF
          DB 74H,49H,0FFH  ; ;A AND FFF, SKIP IF NO ZERO
          DB 0C4H        ; ;JMP OUT
          DB 43H         ; ;C=C+1, LOOP1
          DB 48H,1       ; ;A SHIFT RIGHT, SKIP IF CARRY
          DB 0FCH        ; ;JMP LOOP1
          DB 6AH,0       ; ;B<0, OUT
          DB 0B1H        ; ;PUSH BC
$NEXT
;
;
; LED!
; Turn on/off Roland LED's.
LEDB:     DB 0A1H       ; ;POP BC
          DB 0BH         ; ;A<C
          DB 74H,9H,0FCH  ; ;A<AND FC
          DB 74H,19H,1    ; ;A<A OR 1
          DB 4DH,0C1H    ; ;PB<A
$NEXT
;
;
; eUPDAT
; Move data from Slider Ram to Edit Buffer.
eUPDAT:   $CODE  6,'eUPDAT',EUPDAT
; eLOAD
; Load Edit Buffer data into Slider Memory.
$CODE 5,'eLOAD',ELOAD
DB 68H,0FFH ;;V<FF
DB 6AH,2 ;;B<2
DB 1,0 ;;A<(V/00) Read eSLD#
DB 1BH ;;C<A
DB 29H ;;A<(BC) Read Translation Table
DB 6AH,0C6H ;;B<C6
DB 1BH ;;C<A
DB 29H ;;A<(BC)
DB 48H,21H ;;A SHIFT RIGHT
DB 63H,4 ;;(V/04)<A, eBYTE3
DB 69H,80H ;;A<80
DB 60H,43H ;;C<C+A
DB 29H ;;A<(BC)
DB 63H,3H ;;(V/03)<A, eBYTE2
DB 69H,40H ;;A<40
DB 60H,43H ;;C<C+A
DB 29H ;;A<(BC)
DB 63H,2H ;;(V/02)<A, eBYTE1
$NEXT

; esUPDAT
; Update only the Slider data of the Edit Buffer.
$CODE 7,'esUPDAT',ESUPDAT
DB 68H,0FFH ;;V<FF
DB 6AH,2 ;;B<2
DB 1,0 ;;A<(V/00) Read eSLD#
DB 1BH ;;C<A
DB 29H ;;A<(BC) Read Translation Table
DB 6AH,0C6H ;;B<C6
DB 1BH ;;C<A
DB 29H ;;A<(BC)
DB 49H,0 ;;(BC)<0, LAST
DB 69H,40H ;;A<40
DB 60H,43H ;;C<C+A
DB 49H,0 ;;(BC)<0, LAST
DB 69H,40H ;;A<40
DB 60H,43H ;;C<C+A
DB 1,3 ;;A<(V/03)
DB 39H ;;(BC)<A, eBYTE2
DB 69H,40H ;;A<40
DB 60H,43H ;;C<C+A
DB 39H ;;(BC)<A, eBYTE1
$NEXT
; eSLD#  ( --- FF00 )
; Edit Buffer Slider number.
$COLON 5,'eSLD#',ESLDN
DW DOLIT,OFF00H,EXIT

; eFLD  ( --- FF01 )
; Edit Buffer LCD Field.
$COLON 4,'eFLD',EFLD
DW DOLIT,OFF01H,EXIT

; eBYTE1  ( --- FF07 )
; Edit Buffer Midi Status/Chnl byte.
$COLON 6,'eBYTE1',EBYT1
DW DOLIT,OFF02H,EXIT

; eBYTE2  ( --- FF06 )
; Edit Buffer Midi Key#, Controller#, or Program# byte.
$COLON 6,'eBYTE2',EBYT2
DW DOLIT,OFF03H,EXIT

; eBYTE3  ( --- FF04 )
; Edit Buffer Slider value.
$COLON 6,'eBYTE3',EBYT3
DW DOLIT,OFF04H,EXIT

; eSET  ( --- FF05 )
; Flag indicating Slider or Setup operation.
$COLON 4,'eSET',ESET
DW DOLIT,OFF05H,EXIT

; eSET#  ( --- FF06 )
; Holds Setup number.
$COLON 5,'eSET#',ESETN
DW DOLIT,OFF06H,EXIT

; FLD0  ( --- 80 )
; LCD Field start.
$COLON 4,'FLD0',FLD0
DW DOLIT,080H,EXIT

; FLD1  ( --- 86 )
; LCD Field start.
$COLON 4,'FLD1',FLD01
DW DOLIT,086H,EXIT

; FLD2  ( --- 8A )
; LCD Field start.
$COLON 4,'FLD2',FLD2
DW DOLIT,088H,EXIT

; FLD3  ( --- 8D )
; LCD Field start.
$COLON  4,'FLD3',FLD3
DW      DOLIT,08DH,EXIT

; FLD4
( --- C0 )
; LCD Field start.
$COLON  4,'FLD4',FLD4
DW      DOLIT,0C0H,EXIT

; FLD5
( --- C9 )
; LCD Field start.
$COLON  4,'FLD5',FLD5
DW      DOLIT,0C9H,EXIT

; FLD6
( --- CD )
; LCD Field start.
$COLON  4,'FLD6',FLD6
DW      DOLIT,0CDH,EXIT

; L0
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  2,'L0',L0
SD$  'Slider'

; L1
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  2,'L1',L1
SD$  'Setup#'

; L2
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  2,'L2',L2
SD$  '* MIDI Running *'

; L20
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  3,'L20',L20
SD$  ' chl '

; L21
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  3,'L21',L21
SD$  ' off '

; L40
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  3,'L40',L40
SD$  'Key#    '

; L41
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  3,'L41',L41
SD$  'Key# A-T'

; L42
( --- a )
Packed string. 'a' is addr of count byte.
$COLON  3,'L42',L42
SD$  'Control#'
; L43
( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L43',L43
SD$ 'Program#'

; L44
( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L44',L44
SD$ 'Ch Press'

; L45
( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L45',L45
SD$ 'Ptch Whl'

; L4X
( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L4X',L4X
SD$ '******'

; L50
( --- a )
; Packed string. 'a' is addr of count byte.
$COLON 3,'L50',L50
SD$ '***'

; FLDCASE
( n -- f )
; Choose an LCD field position.
$COLON 7,'FLDCASE',FLDCS
DW DOLIT,7H,ANDD,CASE
DW FLD0,FLD01,FLD2,FLD3,FLD4,FLD5,FLD6,FLD0
DW EXIT

; FLDAT
( --- )
; Return LCD cursor to current field.
$COLON 5,'FLDAT',FLDAT
DW EFLD,CAT,FLDCS,LI,EXIT

; LSTAT
( n --- )
; Choose a midi status label.
$COLON 5,'LSTAT',LSTAT
DW CASE,L4X,L40,L41,L42,L43,L44,L45,L4X,EXIT

;

; SLDISP
( --- )
; Slider data update and display.
$COLON 6,'SLDISP',SLDISP
DW ESUPDAT,EFLD,CAT,FLDCS
DW EBYT3,CAT,DOLIT,07FH,ANDD
DW BDIGS,DIG,DIG,DIG,EDIGS
DW DROP,LLI,FLD6,LCD,LLC
DW DUPP,CAT,LCD,ONEP,DUPP,CAT,LCD
DW ONEP,CAT,LCD,LI,EXIT

; eDISP
( --- )
; Display the Edit buffer on the LCD
$COLON 5,'eDISP',eDISP

131
; SDISP        ( --- )
; Display the Setup operation on the LCD.
$COLON  5,'SDISP',SDISP
DW      DOLIT,01,LI,BDEL
DW      L1,FLD0,DISP
DW      ESETN,CAT,FLD01,N2DISP,FLDAT,EXIT

; MNDISP        ( --- )
; Main display routine for updating LCD display.
$COLON  6,'MNDISP',MNDISP
DW      ESET,CAT
DW      QBRAN,MNDIS1
DW      SDISP,EXIT
MNDIS1:    DW      EDISP,EXIT

; BDEL         ( --- )
; Long delay at end of button routines.
$COLON  4,'BDEL',BDEL
DW      DOLIT,08000H,DELAY,EXIT

; UDCASE        ( n --- )
; Choose an up/down routine from list.
$COLON  6,'UDCASE',UDCS
DW      EFLD,CAT,DOLIT,01,LI,BDEL
DW      L1,FLD0,DISP
DW      ESETN,CAT,FLD01,N2DISP,FLDAT,EXIT

; BL/R         ( fld --- pos )
; Translates LCD field number to a position number.
$COLON  4,'BL/R',BLR
DW      DUPP,EFLD,CSTOR,FLDCS
DW      EXIT

; SL/R         ( --- )
; Limit cursor movement only between fields 0 and 1.
$COLON  4,'SL/R',SLR
DW      EFLD,CAT,DOLIT,01,ANDD,DOLIT,01,XORR,DUPP
DW      EFLD,CSTOR,BLR,LI,BDEL,EXIT

; BLEFT        ( --- )
; Moves the LCD cursor to next field. Loads eFLD.
$COLON 5,'BLEFT',BLEFT
DW DOLIT,40H,LEDB
DW ESET,CAT
DW QBRAN,BLEFT1
DW SLR,EXIT
BLEFT1: DW EFLD,CAT,DOLIT,5,DECR,BLR,LI,BDEL,EXIT

; BRIGHT ( --- )
; Moves the LCD cursor to next field. Loads eFLD.
$COLON 6,'BRIGHT',BRIGH
DW DOLIT,80H,LEDB
DW ESET,CAT
DW QBRAN,BRIGH1
DW SLR,EXIT
BRIGH1: DW EFLD,CAT,DOLIT,5,INCR,BLR,LI,BDEL,EXIT

; SETUP ( --- )
; Setup Slider full Ram buffer from ROM, or MIDI in.
$COLON 5,'SETUP',SETUP
DW ESETN,CAT,DOLIT,2000H,PLUS
DW DOLIT,0C680H,DOLIT,80H,CMOVE
DW EXIT

; LOAD ( --- )
; Load Buffer data shown on LCD into Slider Memory.
$COLON 5,'LOAD',BLOAD
DW DOLIT,4,LEDB
DW ESET,CAT
DW QBRAN,BLOAD1
DW SETUP,DOLIT,0,ESET,CSTOR,EUPDAT,EDISP,BDEL,EXIT
BLOAD1: DW ELOAD,EDISP,BDEL,EXIT

; BMIDI ( --- )
; Start the Midi program.
$COLON 5,'BMIDI',BMIDI
DW DOLIT,1,LI,BDEL
DW L2,FLD0,DISP
DW MIDI,EXIT

; BUP ( --- )
; Increment value in LCD cursor field.
$COLON 3,'BUP',BUP
DW DOLIT,10H,LEDB
DW DOLIT,1,UDCS,BDEL,EXIT

; BDOWN ( --- )
; Decrement value in LCD cursor field.
$COLON 5,'BDOWN',BDOWN
DW DOLIT,20H,LEDB
DW DOLIT,0,UDCS,BDEL,EXIT

; U/D0 ( i/d --- )
; Field increment/decrement routine.
$COLON 4,'U/D0',UD0
DW DROP,ESET,CAT
DW QBRAN,UD0A
UD0A:     DW      DOLIT,0, ESET, CSTOR, EDISP, EXIT

; U/D7    ( i/d --- )
; Field increment/decrement routine. (bogus field)
 $COLON  4,'U/D7', UD7
 DW      DROP, EXIT

; U/D6    ( i/d --- )
; Field increment/decrement routine.
 $COLON  4,'U/D6', UD6
 DW      DROP, EXIT

; U/D1    ( i/d --- )
; Field increment/decrement routine.
 $COLON  4,'U/D1', UD1
 DW      ESET, CAT
 DW      QBRAN, UD1C
 DW      ESETN
 DW      BRAN, UD1D
 UD1C:    DW      ESLDN
 UD1D:    DW      CAT, DOLIT, 3FH, ROT
 DW      QBRAN, UD1A
 DW      INCR
 DW      BRAN, UD1B
 UD1A:    DW      DECR
 UD1B:    DW      CFLD1, EXIT

; CFLD1   ( sld# --- )
; Change Slider# in field 1. Update Edit buffer & LCD.
 $COLON  5,'CFLD1', CFLD1
 DW      ESET, CAT
 DW      QBRAN, CFLA
 DW      ESETN, CSTOR, SDISP
 DW      BRAN, CFLB
 CFLA:    DW      ESLDN, CSTOR, EUPDAT, EDISP, FLDAT
 CFLB:    DW      FLDAT, EXIT

; U/D2    ( i/d --- )
; Ch/Off Field increment/decrement routine.
 $COLON  4,'U/D2', UD2
 DW      QBRAN, UD2A
 DW      EBYT1, CAT, DOLIT, 80H, ORR
 DW      EBYT1, CSTOR, L20, FLD2, DISP
 DW      BRAN, UD2B
 UD2A:    DW      EBYT1, CAT, DOLIT, 7FH, ANDD
 DW      EBYT1, CSTOR, L21, FLD2, DISP
 UD2B:    DW      FLDAT, EXIT

; U/D3    ( i/d --- )
; Field increment/decrement routine.
 $COLON  4,'U/D3', UD3
 DW      EBYT1, CAT, DOLIT, 0FH, ANDD, DOLIT, 0FH, ROT
 DW      QBRAN, UD3A
 DW      INCR
 DW      BRAN, UD3B
 UD3A:    DW      DECR
 UD3B:    DW      CFLD3, EXIT
; CFLD3
  ( chnl --- )
; Change midi channel in field 3.
$COLON  5,'CFLD3',CFLD3
DW  DUPP,EBYT1,CAT,DOLIT,0F0H
  ANDD,ORR,EBYT1,CSTOR,FLD3,NDISP,EXIT

; U/D4
  ( i/d --- )
; Field increment/decrement routine.
$COLON  4,'U/D4',UD4
DW  EBYT1,CAT,DOLIT,70H,ANDD
DW  TWOSR,TWOSR,TWOSR,TWOSR,DOLIT,7,ROT
DW  QBRAN,UD4A
DW  INCR
DW  BRAN,UD4B
UD4A:
  DW  DECR
UD4B:
  DW  DUPP,DOLIT,0,EQUAL,OVER,DOLIT,7H,EQUAL,ORR
  QBRAN,UD4C
  DW  DROP,DOLIT,1
UD4C:
  DW  CFLD4,EXIT

; CFLD4
  ( status --- )
; Change Midi operation label in field 4.
$COLON  5,'CFLD4',CFLD4
DW  DUPP,TWOSL,TWOSL,TWOSL,TWOSL
DW  DOLIT,80H,ORR,EBYT1,CAT
DW  DOLIT,0FH,ANDD,ORR,EBYT1,CSTOR
DW  LSTAT,FLD4,DISP,FLDAT,EXIT

; U/D5
  ( i/d --- )
; Field increment/decrement routine.
$COLON  4,'U/D5',UD5
DW  EBYT2,CAT,DOLIT,07FH,ROT
DW  QBRAN,UD5A
DW  INCR
DW  BRAN,UD5B
UD5A:
  DW  DECR
UD5B:
  DW  CFLD5,EXIT

; CFLD5
  ( data --- )
; Change Midi data byte in field 5.
$COLON  5,'CFLD5',CFLD5
DW  DOLIT,0CFH,EBYT1,CAT,DOLIT,0F0H,ANDD,LESS
DW  QBRAN,UD5AA
DW  L50,FLD5,DISP,FLD5,LI,DROP
DW  BRAN,UD5BB
UD5AA:
  DW  DUPP,EBYT2,CSTOR,FLD5,NDISP
UD5BB:
  DW  EXIT

; BSUP
  ( --- )
; Button 1.  Increments Slider number.
$COLON  4,'BSUP',BSUP
DW  DOLIT,1,UD1,BDEL,EXIT

; BSDWN
  ( --- )
; Button 2.  Decrements Slider Number.
$COLON  5,'BSDWN',BSDWN
DW  DOLIT,0,UD1,BDEL,EXIT
; MCASE
( --- )
Button case for Main.

; MLOOP
( --- stat)
Loop thru ADC values until an enabled one is found.

; ADCV
( --- adc value)
Push stack with current adc value for MIDI operation.

; SLAST
( --- diff value)
Push stack with current diff value for MIDI operation.
; BYT2
( --- byt2 value)
; Push stack with current BYTE2 value for MIDI operation.
; CODE 4,'BYT2',BYT2
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 46H,80H ;A<A+80
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; BYT1
( --- byt1 value)
; Push stack with current BYTE1 value for MIDI operation.
; CODE 4,'BYT1',BYT1
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 46H,0C0H ;A<A+C0
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; FLAG
( --- flag value)
; Push stack with current FLAG value for MIDI operation.
; CODE 4,'FLAG',FLAG
DB 1,0E0H ;A<(V/E0), Midi loop count.
DB 1BH ;C<A
DB 6AH,0C7H ;B<C7
DB 29H ;A<(BC)
DB 1BH ;C<A
DB 6AH,0 ;B<0
DB 0B1H ;PUSH BC
$NEXT

; FLGON
( --- )
; Store FF in FLAG of current slider.
; CODE 5,'FLGON',FLGON
DB 1,0E0H ;A<(V/E0)
DB 1BH ;C<A
DB 6AH,0C7H ;B<C7
DB 69H,0FFH ;A<FF
DB 39H ;(BC)<A
$NEXT

; FLGOFF
( --- )
; Store 0 in FLAG of current slider.
; CODE 6,'FLGOFF',FLGOFF
DB 1,0E0H ;A<(V/E0)
DB 1BH ;C<A
DB 6AH,0C7H ;B<C7
DB 69H,0 ;A<0
DB 39H ;(BC)<A
$NEXT

; ?DIFF
(old,new --- /shifted new,0F/ OR /00/)
; Flag=0F if /old-new/>1, else Flag=0.
$CODE 5,'?DIFF',QDIFF
DB 0A4H ;POP EA, new
DB 0A1H ;POP BC, old
DB 09H ;A<EAL
DB 60H,0E3H ;A<A-C

DB 6BH,0FFH ;C<FF
DB 60H,0EBH ;A-C, skip if no zero
DB 6BH,0 ;C<0
DB 47H,0FEH ;A AND FE, skip if no zero
DB 0C7H ;JMP AHEAD
DB 9H ;A<EAL
DB 48H,21H ;A SHIFT RIGHT
DB 19H ;EAL<A
DB 0B4H ;PUSH EA
DB 6BH,0FFH ;C<FF

DB 6AH,0 ;B<0, AHEAD
DB 0B1H ;PUSH BC
$NEXT

; LDLAST
( --- )
; Moves ADC value to SLAST value in current MLOOP slider buffer.
$CODE 6,'LDLAST',LDLAST
DB 1,0E0H ;A<(V/E0)
DB 1BH ;C<A
DB 6AH,0C6H ;B<C6
DB 29H ;A<(BC)
DB 19H ;EAL<A
DB 0BH ;A<C
DB 46H,40H ;A<A+40
DB 1BH ;C<A
DB 09H ;A<EAL
DB 39H ;(BC)<A
$NEXT

; DUMMY
( --- )
; Do nothing dummy.
$COLON 5,'DUMMY',DUMMY
DW EXIT

; LDSTAT
( --- )
; Load current Byte1 to FFE1, last Midi status sent.
$COLON 6,'LDSTAT',LDSTAT
DW BYT1,DOLIT,0FFE1H,CSTOR,EXIT

; KEYN
( --- )
; Midi routine for Key On and Key Off.
$COLON 4,'KEYN',KEYN
DW ADCV,DOLIT,0,EQUAL
DW QBRAN,KEYN1
DW LDLAST,FLAG
DW QBRAN,KEYN4

138
DW      BYTE1,TM,LDSTAT,BYTE2,TM,FLGOFF,DOLIT,0,TM,EXIT

KEYN1:          DW      FLAG
DW      QBRAN,KEYN2
DW      EXIT

KEYN2:          DW      ADCV,SLAST,SUBB,ZLESS
DW      QBRAN,KEYN3
DW      BYTE1,TM,LDSTAT,BYTE2,TM,FLGON,ADCV,TWOSR,TM

KEYN3:          DW      LDLAST

KEYN4:          DW      EXIT

;  TSTAT  ( --- )
;  Test status byte, send midi status if not last sent.
$COLON  5,'TSTAT',TSTAT
DW      BYTE1,DUPP,DOLIT,OFFE1H,CAT,EQUAL
DW      QBRAN,TSTAT1
DW      DROP,BRAN,TSTAT2

TSTAT1:         DW      TM,LDSTAT

TSTAT2:         DW      EXIT

;  KEYAT  ( --- )
;  Midi routine for Key On and Key Off with Poly After Touch.
$COLON  5,'KEYAT',KEYAT
DW      ADCV,DOLIT,0,EQUAL
DW      QBRAN,KEYAT1
DW      LDLAST,FLAG
DW      QBRAN,KEYAT4
DW      BYTE1,DOLIT,0FH,ANDD,DOLIT,90H,ORR,TM
DW      DOLIT,90H,DOLIT,OFFE1H,CSTOR
DW      BYTE2,TM,FLGOFF,DOLIT,0,TM,EXIT

KEYAT1:         DW      FLAG
DW      QBRAN,KEYAT2
DW      SLAST,ADCV
DW      QDIFF
DW      QBRAN,KEYAT4
DW      TSTAT,BYTE2,TM,LDLAST,TM,EXIT

KEYAT2:         DW      ADCV,SLAST,SUBB,ZLESS
DW      QBRAN,KEYAT3
DW      BYTE1,DOLIT,0FH,ANDD,DOLIT,90H,ORR,TM
DW      DOLIT,90H,DOLIT,OFFE1H,CSTOR
DW      BYTE2,TM,FLGON,ADCV,TWOSR,TM

KEYAT3:         DW      LDLAST

KEYAT4:         DW      EXIT

;  CNTRL  ( --- )
;  Midi Routine for controller data.
$COLON  5,'CNTRL',CNTRL
DW      SLAST,ADCV
DW      QDIFF
DW      QBRAN,CNTRL1
DW      TSTAT,BYTE2,TM,LDLAST,TM,EXIT

CNTRL1          DW      EXIT

;  PRG  ( --- )
;  Midi Routine for program changes.
$COLON  3,'PRG',PRG
DW      DOLIT,40H,ADCV,LESS
DW      QBRAN,PRG2
DW      FLAG
;  CHAT         ( --- )
;  Midi Routine for Channel Pressure.
$COLON  4,'CHAT',CHAT
DW      SLAST,ADCV
DW      QDIFF
DW      QBRAN,CHAT1
DW      TSTAT,LDLAST,TM,EXIT
CHAT1:          DW      EXIT

;  PWHL         ( --- )
;  Midi Routine for Pitch Wheel.
$COLON  4,'PWHL',PWHL
DW      SLAST,ADCV
DW      QDIFF
DW      QBRAN,PWHL1
DW      TSTAT,DOLIT,0,TM,LDLAST,TM,EXIT
PWHL1:          DW      EXIT

;  MCASE        ( --- )
;  Midi Routines Case Statement.
$COLON  5,'MCASE',MCASE
DW      CASE,DUMMY
DW      KEYN,KEYAT,CNTRL,PRG,CHAT,PWHL,DUMMY
DW      EXIT

;  MIDI         ( --- )
;  Main Midi Loop.
$COLON  4,'MIDI',MIDI
DW      DOLIT,0C700H,DOLIT,40H,DOLIT,0,FILL
DW      DOLIT,0C640H,DOLIT,40H,DOLIT,0,FILL
MIDI1:          DW      ADCIN,MLOOP,MCASE,SWAT,INVER,DOLIT,07H,ANDD
DW      QBRAN,MIDI1
DW      EXIT

;  EDIT         ( --- )
;  MAIN SLIDER EDIT PROGRAM.
$COLON  4,'EDIT',EDIT

DW      CR,DECIM,DOLIT,1,LI,BDEL
DW      EUPDAT,EDISP
EDIT1:          DW      ADCIN,SLDISP,DOLIT,0,LEDB,MNCASE,NUFQ
DW      QBRAN,EDIT1
DW      HEX,EXIT

;  ENBRM         ( --- )
;  Enable Midi Receive,
$CODE   5,'ENBRM',ENBRM
DB      64H,81H,0EH     ;SMH<E
$NEXT

;  RM           ( --- b,f)
;  Receive Midi. If midi received, returns the data plus true,
; else returns false flag.
$CODE 2,'RM',RM
DB  6AH,0 ;B<0
DB  6BH,0 ;C<0
DB  48H,49H ;SKIT FSR, skip if interrupt flg
DB  0C6H ;JMP AHEAD
DB  4CH,0D9H ;A<RXB
DB  1BH ;C<A
DB  0B1H ;PUSH BC, received byte
DB  6BH,0FFH ;C<FF
DB  0B1H ;PUSH BC, the flag, AHEAD
$NEXT

;===============================================================================

LASTN EQU _NAME+4 ;last name address
NTOPP EQU _NAME-0 ;next available memory in ROM name
dictionary
CTOPP EQU $+0 ;next available memory in ROM code
dictionary

MAIN ENDS
END ORIG

;===============================================================================

141