CONTROLLER PROGRAMS

FORTH was designed originally as a language or tool to control instruments or devices through a computer. It enables the user to send commands to devices and receive information from them interactively with immediate feedback. Once the hardware is in place and debugged, software development is usually very straightforward. Bugs can be spotted quickly and corrections or modifications can be thoroughly tested.

Using computers in laboratory and industrial environments, the most often encounted problems are that of interfacing, i. e., how to hook things to the computer and have them talk to one another. Using FORTH, we have solved half of the problem at the computer's end, because all the resources in the computer are under immediate control.

MEMORY MAPPED I/O DEVICE

Computers using memory mapped I/O architecture are the easiest to use, because the regular memory addressing words like @ and ! can be used to control the I/O devices. Otherwise one will have to build a few special I/O instructions to talk to the peripheral devices. LSI-11 is a computer with memory mapped I/O structure. I/O interface boards built for this computer use the top page of memory space to address their internal registers and buffers, and the bottom page for interrupt vectoring. Generally each device has one control/status register and one data buffer register. Writing into the control/status register (CSR) specifies the function to be performed. Reading CSR obtains the status of the device. I/O data is routed through the data buffer register.

EXAMPLES OF I/O CONTROL

Since it is very simple to use the LSI-11 to control the I/O boards built for it, it is difficult to find good examples to illustrate the programming aspects of the interfacing. Here I selected four examples dealing with the more popular types of DEC I/O interface boards. A PROM programmer which was essentially a memory device, an A/D converter used in an image scanner, a DMA board for high speed data transfer, and a 3 axis motorized transport mechanism which I call a robot. In most cases, the reponse of the I/O device is much slower than the computation speed of the computer; therefore, programs can be written entirely in high level FORTH. In a few cases where speed is important, the critical words were coded in machine code by invoking the assembler in poly-FORTH.

PROM PROGRAMMER

With a Prom Programmer Board MRV-004 from MDB for the LSI-11 computer, burning Prom's is no more than moving bytes from one memory location to the other. This program assumes that a set of master prom's are located from octal 140000 to 147777, and a set of erased prom's are located at 150000 to 157777. Prom's are of the popular 2716's.

?CLEAN	Check the erased prom's and make sure all bits are set
DELAY	A delay loop inserted after each memory write to the prom's to be programmed.
>RAM	Copy 4096 bytes from octal 140000 to RAM at 40000, where data can be modified.
?RAM	Compare the contents in the master prom's and those in RAM. Report any discrepancy.

PROGRAMMING PROM

>PROM	Move 4096 bytes of RAM data to the prom's to be programmed. Delay 75 ms after each write operation.
?PROM	Compare the contents of the newly programmed proms with those in RAM from 40000 octal. Report discrepancies.
HELP	Helping messages to the operator who has no interests in learning the language but has to use the program.

STORE PROM DATA ON DISK

DATA	A constant specifying the starting block number where PROM data are to be stored for future usage.
SAVE	The user can store the contents of a 2716 into two disk blocks. Eight such areas are reserved for this purpose. Since consecutive bytes must be mapped to different prom's due to the LSI-11 bus structure, even bytes are extracted from a single prom and compressed into disk blocks.

```
  174 LIST

( PROM PROGRAMER ) OCTAL
: ?CLEAN   160000 150000 DO I @ DUP -1 = IF DROP ELSE
         I 10 U.R 10 U.R THEN 2 +LOOP  ;
: DELAY   ( 75 MS)   3000 0 DO LOOP ;
: >RAM    140000 40000 10000 MOVE   ;
: ?RAM    10000 0 DO
         I 40000 + @ I 140000 + @ = IF 0 DROP ELSE I 10 U.R
         I @ 10 U.R 140000 I + @ 10 U.R CR THEN 2 +LOOP ;
DECIMAL
```

```
  175 LIST

( PROM PROGRAMER ) OCTAL
: >PROM   10000 0
         DO I 40000 + @ 150000 I + ! DELAY 2 +LOOP ;
: ?PROM   10000 0 DO
         I 40000 + @ I 150000 + @ =   NOT IF I 10 U.R I 40000 +
         @   10 U.R 150000 I + @ 10 U.R CR THEN 2 +LOOP ;
DECIMAL
: HELP CR ." THE PROM PROGRAMMER COMMANDS ARE:"
     CR ." >RAM    MOVE OLD PROM DATA TO RAM MEMORY."
     CR ." ?RAM    COMPARE RAM DATA WITH OLD PROM."
     CR ." >PROM   BURN NEW PROM FROM RAM."
     CR ." ?PROM   COMPARE RAM DATA WITH NEW PROM."
     ;
HELP  CR
```

```
  176 LIST

( PRINT AND STORE PROM DATA, CHT, 2-SEP-83)
60 CONSTANT DATA
OCTAL
: SAVE ( N ---, SAVE PROM DATA TO N'TH DISK FILE)
       DUP 7 > ABORT"   NOT ENOUGHT SPACE!!!"
       CR ." ARE YOU SURE? (Y/N)"   KEY 131 - ABORT" ABORT."
       2* DATA + DUP BLOCK   140000   2000 0 DO
       OVER OVER C@ SWAP C!   2 + SWAP 1+ SWAP   LOOP
       2DROP UPDATE FLUSH
       1+ BLOCK   144000   2000 0 DO
       OVER OVER C@ SWAP C!   2 + SWAP 1+ SWAP   LOOP
       2DROP UPDATE FLUSH
       ;
DECIMAL
```

PRINT PROM DATA

TAB	Given a block number, display the byte contents of this block in tabulated format suitable for printer output.
TABLE	Display the contents of a prom previously stored in one of eight disk areas. The dump format is three bytes o prom address followed by 16 bytes in hex. Each byte is display in a 3 column field. The user wanted the data to be displayed in hex while other operations are done in octal. The HEX ... OCTAL sequence is inserted for this base switching requirement.

ANOTHER PROM PROGRAMMER

This is another prom programmer for the MDB MLSI MRV-004 EPROM module. I used this program to generate my prom based FORTH computer hosted in the LSI-11 machine. The base address of the PROM Module is set at octal 100000, and the program to be committed to proms is store in disk blocks 230 to 237.

DELAY	75 ms delay loop for 2716 programming.
PROM	Base address of the PROM Module.
?CLEAN	Check a range of memory and report any cell not having all bits set.
1BLOCK	Program one block of data into proms.
1CHECK	Compare one block of data between proms and disk.
PROGRAM	Move the entire FORTH dictionary into proms.
CHECK	Verify that the programmed proms agree with the original data on disk.

```
  177 LIST

( PRINT AND STORE PROM DATA, CHT, 2-SEP-83)
: TAB   ( BLOCK# --- )
        BLOCK   1024 0 DO    I 16 MOD
        0= IF CR 10000 0 DO LOOP  I 6 U.R THEN
        I OVER + C@ 3 U.R    LOOP    DROP    ;
: TABLE ( N --- )
        PAGE    ." BLOCK # "    DUP .
        HEX    2* DATA + DUP     TAB
        1+ TAB    OCTAL    ;
```

```
  178 LIST

(   PROM PROGRAMMER, CHT, 2-17-81)
: DELAY 4000 0 DO LOOP ;
OCTAL
100000 CONSTANT PROM
: ?CLEAN ( END-ADDR START-ADDR ---)
     DO I @ -1 = IF ELSE CR I . I ? THEN 2 +LOOP ;
: 1BLOCK   ( PROM-ADDR BLOCK# --- PROM-ADDR+1024)
     2000 0 DO OVER OVER @ SWAP ! DELAY  2+ SWAP 2+ SWAP
               2 +LOOP DROP ;
: 1CHECK 2000 0 DO OVER @ OVER @ - IF OVER DUP CR
         DELAY DELAY 10 U.R @ 10 U.R DUP @ 10 U.R
          THEN 2+ SWAP 2+ SWAP   2 +LOOP DROP ;
DECIMAL
: PROGRAM PROM 238 230 DO I BLOCK CR I . 1BLOCK LOOP DROP ;
: CHECK PROM 238 230 DO I BLOCK 1CHECK LOOP DROP ;
```

```
  179 LIST
```

A/D CONVERTER

A/D converter is an important device which allows the compute to monitor the physical signals from the real world. The LSI-11 computer is supported by off-the-shelf A/D converter plug-in boards make by DEC and a host of other manufacturers. The one I used to read analog signals from a mechanical image scanner was made by Data Translation. As far as the control processes are concerned, it is compatible with the ADV-11A board made by DEC.

In my experiments, I used 3 input channels. The image data was on Channel 8. The starting limit switch was monitored by Channel 12, and the end limit switch was on Channel 11. After the scanner released the starting switch, 512 image data were collected, strobed by an external clock. The scanner reverse its travel at the end switch, moved back hitting the starting switch and reversed the travel direction again.

A/D INPUT

ADCSR	Control Status Register of the A/D converter.
DATA	Input data buffer of the A?D converter.
IMAGE	Image data register of the image processor.
CBCR	Control and Base Coordinate Register of the IP.
SET-CBCR	Initialize the IP so that the input data will be written directly into the image memory.
TA/D	Wait until A/D completes a conversion, read data, right shift it by 4 bits and write into the 8 bit image memory.
CA/D	Similar to TA/D except that the 12 bit input data is pushed on the data stack. This is used to check the system functions.

SCANNING THE IMAGE

ARMA/D	Set the GO bit in the ADCSR register to initiate an A/D conversion cycle.
F-START	Select the starting switch as input. Wait for the closure of this switch and exit when the switch is released again. This ensures that the scanner has left the starting position on the forward travel.
F-END	Monitor the end switch. Exit when this switch is closed and again released.
LINES	Select the image data input channel and enable the external clock to trigger the A/D conversion cycle. After the scanner leaves the starting position, read 512 data points and write them into the IP image memory.
INPUT	Acquire 485 lines of image data and fill a frame of IP image memory.

```
  180 LIST

( IMAGE SCANNER )
OCTAL
177000 CONSTANT ADCSR  177002 CONSTANT DATA
120000 CONSTANT IMAGE   117600 CONSTANT CBCR

: SET-CBCR 1000 117426 !
     100000 CBCR ! 2000 CBCR 2+ ! 177400 CBCR 4 + !
     64377 CBCR 6 + ! 4020 177000 ! ;

CODE  TA/D  BEGIN ADCSR TST B  0< END 0 DATA MOV
     0 ASL 0 ASL 0 ASL 0 ASL IMAGE 0 MOV NEXT

CODE CA/D BEGIN ADCSR TST B 0< END S -) DATA MOV  NEXT
DECIMAL
```

```
  181 LIST

( SCANNING MECHANISM ) OCTAL
: ARMA/D 1 ADCSR C! ;
: F-START 6000 ADCSR ! BEGIN ARMA/D CA/D 7777 AND 2000 > END
          BEGIN ARMA/D CA/D 7777 AND 100 < END ;
: F-END 5400 ADCSR ! BEGIN ARMA/D CA/D 7777 AND 2000 > END
        BEGIN ARMA/D CA/D 7777 AND 100 < END ;

: LINES F-START 4020 ADCSR ! 1000 0 DO TA/D LOOP
         ADCSR ? CR ;
: INPUT SET-CBCR 746 0 DO LINES LOOP ;
DECIMAL
```

```
  182 LIST
```

DMA INTERFACE

The DRV-11B board is a general purpose direct-memory-access (DMA) interface made by DEC. It is capable of transferring data directly from LSI-11 memory to a user device at the rate of 500 KW/sec.

WCR — Word count register. 2's complement of the word count is stored here. It is incremented after a transfer. When it is zero, an interrupt will be generated.

BAR — Base address of the memory block to be transferred.

CSR — Control-status register to control and monitor the functions of DMA board.

IDBR, ODBR — Same register address. When written, data is sent to the output port. When read, data from the input port is returned here.

STATUS — A variable to indicate whether a DMA transfer is completed or not.

INTERRUPT — Set the interrupt vector at 124 pointing to BEGIN . The interrupt routine simply increment STATUS after a DMA trnasfer is completed. This routine must be written in codes and can be very elaborate according to applications.

INPUT — Set up the DMA registers for an input block transfer. 512 words will be read into the IDATA disk block buffer.

OUTPUT — Set up registers for output block transfer. 512 words will be transferred out from the ODATA disk block buffer.

?DONE — Examine the contents of STATUS and print appropriate message.

INPUT AND OUTPUT DATA

RAMP — Initialize the output ODATA block to a ramp function.

FLAT — Initialize the output ODATA block to a constant value. This value is taken from the top of the stack.

?DATA — Dump the data in the IDATA block buffer. This is used to examine the results after a DMA input transfer. Data are displaced in hexadecimal form.

The programming part in using DMA is very simple. The tricky part is to set up the hardwares so that the right signals appear at the right pins at the right time, which is not trivial nor obvious. Armed with FORTH, I usually can point my finger to the poor hardware engineer when things don't work as they are supposed to. It is another matter when I have to do the interface myself.

```
( DRV-11B DMA INTERFACE, CHT, 14-AUG-83)
OCTAL
172410 CONSTANT WCR    ( WORD COUNT REGISTER)
172412 CONSTANT BAR    ( BASE ADDRESS REGISTER)
172414 CONSTANT CSR    ( CONTROL-STATUS REGISTER)
172416 CONSTANT IDBR   ( INPUT DATA BUFFER)
172416 CONSTANT ODBR   ( OUPUT DATA BUFFER)

VARIABLE STATUS        ( USER DEVICE STATUS)
ASSEMBLER
BEGIN   STATUS INC   124 INTERRUPT   200 126 !

DECIMAL
400 CONSTANT IDATA     ( INPUT DATA BLOCK)
401 CONSTANT ODATA     ( OUTPUT DATA BLOCK)



( DMA TRANSFER, CHT, 14-AUG-83)
OCTAL
: INPUT    0 STATUS !    IDATA BLOCK BAR !
           -1000 WCR !    121 CSR !    ;

: OUTPUT   0 STATUS !    ODATA BLOCK BAR !
           -1000 WCR !    141 CSR !    ;

: ?DONE    CR    STATUS @    IF ." Done."
           ELSE ." Not Yet."    THEN    ;

DECIMAL




( DATA INITIATION AND VERIFYING, CHT, 14-AUG-83)
: RAMP    ODATA BLOCK 1024 0 DO    I OVER !
          2 +    2 +LOOP    DROP    ;

: FLAT    ( N --- )    ODATA BLOCK 1024 0 DO
          OVER OVER !    2 +    2 +LOOP    2DROP    ;

: ?DATA   BASE @    IDATA BLOCK HEX 1024 DUMP    BASE !    ;
```

ROBOT CONTROL

In our laboratory, there was a very primitive robot, which consisted of three stepper motor driven stages: a horizontal stage as the base, a vertical stage mounted on the horizontal stage, and a rotory stage mounted on the vertical stage. All the stages can be driven in either forward or backward directions. The stepper motor controllers accept TTL pulses to drive the motor in steps.

The LSI-11 computer had a DRV-11 Parallel Interface board which put out 16 TTL output lines. We used 6 of these lines to give pulses to the stepper motor controllers. Thus we were able to move a piece of sample mounted on the rotary stage with three degrees of freedom. This was quite satisfactory for our experiments.

THE STEPPER MOTORS

The stepper motors were the Slo-Syn type, 200 steps per revolution. In the linear stages, the stepper motor was directly coupled to a 1 mm pitch screw which drives a plate-form. In the rotary stage, the screw drives the plateform through a fine-toothed cam. The drive ratio is 50 steps for 1 degree.

The stages could be driven at 500 Hz pulse rate from dead stop. To gain higher speed, the driver had to be slewed, i.e., starting at 500 Hz and gradually increasing the rate. They could be slewed to 3000 Hz rate if the load was not excessive. I will describe both the constant speed and slewing routines.

CONSTANT SPEED STEPPER CONTROL

DROUT	The address of the output buffer in DRV-11 board.
MS	A delay of 1 mili-second.
DRIVER	Variable holding the bit pattern to be given to the stepper controllers via DRV-11 board.
PULSE	Invert the contents of DRIVER and send them to the stepper controllers. Wait 1 ms, clear the signals, and wait another 1 ms. If executed in a loop, active lines specified by DRIVER will be toggled at th 500 Hz rate.
PULSES	Pulse the output lines n times, given n on stack.

CONSTANT SPEED STEPPER CONTROL

HORZ — A double integer variable holding the current horizontal position in number of steps from origin.

VERT — A double integer variable holding the current vertical position in number of steps from origin.

ROTARY — A double integer variable holding the current rotary position in number of steps from origin.

STORAGE — An array to store the current positions.

D-INIT — Initialize all the position counters to zero, thus define the home position or the origin of the stages.

DRIVE — Given the number of pulses to be sent out to the stepper controller and the active line patterns on the stack, send out these pulses.

RAMPING THE STEPPER CONTROLLERS

MS — Number of dummy loops is specified on the stack, permitting the pulse width to be varied.

FASTP — Send out n pulses at a high rate of 2000 Hz.

SLOWP — Send out n pulses at a low rate of 500 Hz.

RAMP — Send out 5 pulses at a rate specified by the number on the stack.

/RAMP — In 210 steps, increase the driving pulse rate from 500 Hz to 2000 Hz. Ramp-up the motor.

\RAMP — In 210 steps, decrease the driving pulse rate from 2000 Hz to 500 Hz. Ramp-down the motor.

PULSE — Drive the stepper motors with n pulses. If n is less than 420, drive the steppers at the 500 Hz rate. If n is greater than 420, ramp the pulse rate to 2000 Hz and drive at this rate. At the end, ramp down to 500 Hz and stop.

KEEP TRACK OF THE ROBOT MOVEMENTS

30 LOAD 31 LOAD — Load the double integer extensions.

PMAX — The maximum number of pulses that can be handled by PULSES.

DPULSES — The number of pulses are given by a double integer on the stack. Send out groups of 32767 pulses to reduce the stack number to less than 32767. Send the rest of the pulses.

DDRIVE — Pulse the stepper controllers by the pattern at top of stack d times. d is under n on the stack.

DRIGHT, DLEFT, DUPP, DDOWN, DCW, DCCW — Drive the stages right, left, up, down, cw, or ccw by the steps specified by the double integer on the stack. Update the stage positions to keep track of the robot movement.

120 LIST

```
( ROBOT CONTROL, CHT, 4/23/81 )
OCTAL 167772 CONSTANT DROUT DECIMAL
: MS 50 0 DO LOOP ;

VARIABLE DRIVER ( BIT PATTERN TO DRIVE SPECIFIED STEPPER )
: PULSE DRIVER @ -1 XOR DROUT ! MS  -1 DROUT ! MS ; ( 500 HZ )
: PULSES ( N --- )   0 DO PULSE LOOP ;
: VDR DRIVER ;
VARIABLE HORZ 2 ALLOT   VARIABLE VERT 2 ALLOT
VARIABLE ROTARY 2 ALLOT  ( POSITION COUNTERS )
VARIABLE STORAGE 12 ALLOT ( FOR INTERRUPT )

: D-INIT   0 0    2DUP HORZ 2!   2DUP VERT 2!
     ROTARY 2!    STORAGE 14 ERASE   ;
: DRIVE ( PULSE DRIVER --- PULSE ) DRIVER ! DUP PULSES ;
```

121 LIST

```
( ROBOT CONTROL, CHT, 4/28/81, WITH RAMPING )
OCTAL 167772 CONSTANT DROUT DECIMAL
: MS ( N --- ) 0 DO LOOP ; ( VARIABLE PULSE WIDTH )
VARIABLE DRIVER ( BIT PATTERN TO DRIVE SPECIFIED STEPPER )
: PULSE DRIVER @ -1 XOR DROUT ! DUP MS -1 DROUT ! MS ;
: FASTP ( N---) 0 DO 8 PULSE LOOP ; ( 2000 HZ )
: SLOWP ( N---) 0 DO 50 PULSE LOOP ; (  400 HZ )
: RAMP 5 0 DO DUP PULSE LOOP ;
: /RAMP 50 42 0 DO RAMP 1- LOOP DROP ;
: \RAMP 20 42 0 DO RAMP 1+ LOOP DROP ;
: PULSES ( N---) ?DUP IF   DUP 420 > IF 420 - /RAMP FASTP \RAMP
     ELSE SLOWP THEN    THEN ;
VARIABLE HORZ 2 ALLOT   VARIABLE VERT 2 ALLOT
VARIABLE ROTARY 2 ALLOT  ( POSITION COUNTERS )
VARIABLE STORAGE 12 ALLOT ( FOR INTERRUPT )
```

122 LIST

```
( ROBOT MOVEMENTS BY DOUBLE INTEGERS, CHT, 4/23/81 )

30 LOAD 31 LOAD ( 32 BIT )
32767 0 2CONSTANT PMAX

: DPULSES ( D --- )   BEGIN   PMAX 2OVER D<
     IF   PMAX D-   32767 PULSES   AGAIN   DROP PULSES   ;
: DDRIVE  ( D N --- D )   DRIVER !   2DUP DPULSES   ;

: DRIGHT ( D --- )   4 DDRIVE   HORZ 2@ D+ HORZ 2!   ;
: DLEFT  ( D --- )   8 DDRIVE   HORZ 2@ 2SWAP D-   HORZ 2! ;
: DUPP   ( D --- )   1 DDRIVE   VERT 2@ D+ VERT 2!   ;
: DDOWN  ( D --- )   2 DDRIVE   VERT 2@ 2SWAP D-   VERT 2! ;
: DCW    ( D --- )  16 DDRIVE   ROTARY 2@ D+ ROTARY 2!   ;
: DCCW   ( D --- )  32 DDRIVE   ROTARY 2@ 2SWAP D- ROTARY 2! ;
```

INTERRUPT THE ROBOT MOTION

SAVE — Save the current robot position in the STORAGE array. The first number in STORAGE is set to 1, indicating that the robot motion is being interrupted.

RESTORE — Move the robot back to the position previously stored in STORAGE. Clear the first number in STORAGE to zero. However, if the first number in STORAGE is zero already, abort the process because the robot motion had not been interrupted.

ROBOT POSITION MANAGEMENT

D-INIT — Initialize the position counters and STORAGE to zero. Identify the current robot position as HOME.

HOME — Return the robot to its home position.

?ESCAPE — If the console had received a Z key, save the current position counters in STORAGE and then enter the interrupt state, in which all the normal robot control commands can be executed to move the robot. Upon returning from the interrupt state, restore the robot to the position when interrupt occured.

INTERRUPT HANDLER

XRESET — Clear the data stack.

XNUMBER — Modified NUMBER routine. When an error occurs, only print a '?' on console to avoid aborting back to the normal state.

XENTRY — Modified text interpreter. Do not abort when an error occurs, but keep the operation within this loop.

XESCAPE — The entry point of the interrupted state. It uses XENTRY to interpret commands typed in on the console.

R — Abort the interrupt state and return to the normal FORTH system by dropping two addresses on the return stack. If the first number in STORAGE is not 1, don't do anything, because the system is not in the interrupted state.

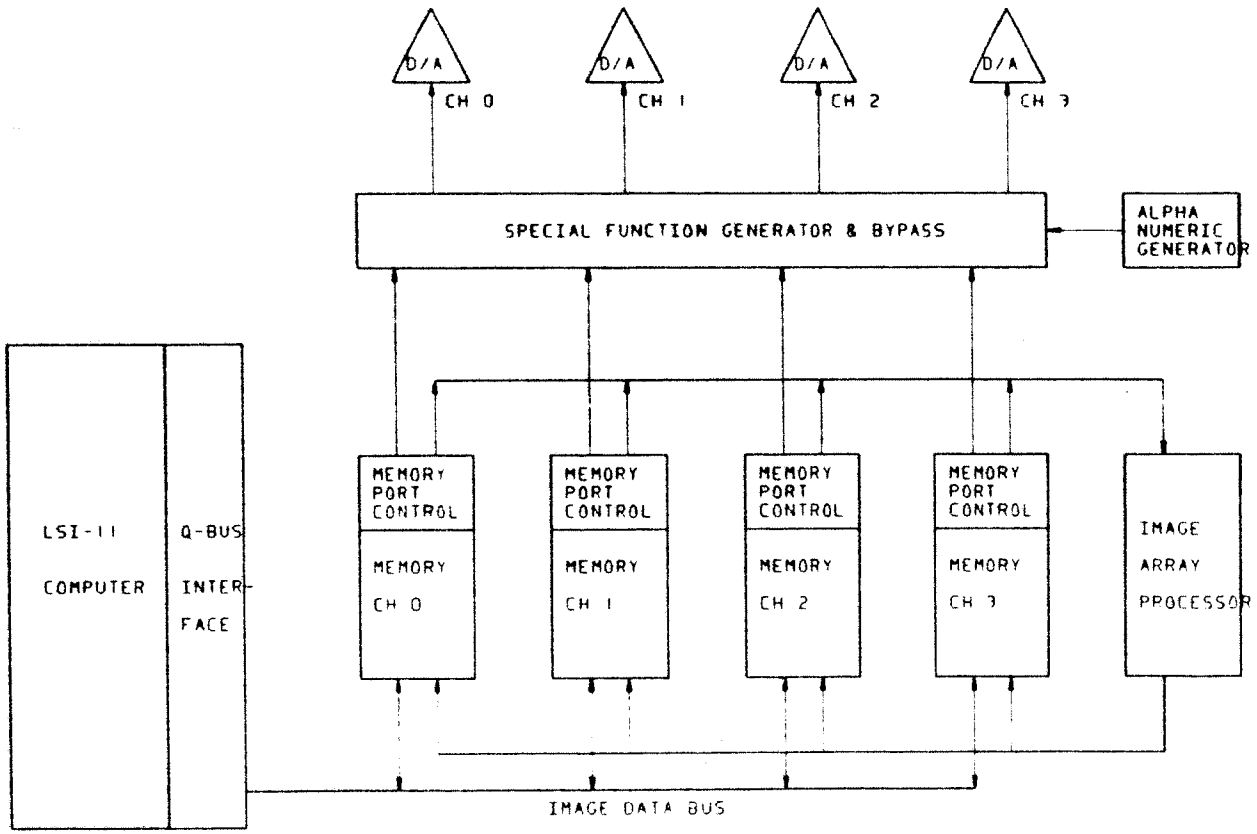
ABORT — The standard abort command without output message and also not requiring a flag on the stack.

```
  123 LIST

( INTERRUPT SAVE & RESTORE, CHT, 4/23/81 )

: SAVE  ( SAVE POSITION IN STORAGE )     STORAGE
        DUP 2+ HORZ 2@ ROT 2!    DUP 6 + VERT 2@ ROT 2!
        DUP 10 + ROTARY 2@ ROT 2!    1 SWAP !    ;

: RESTORE STORAGE DUP @ IF    DUP 2+ 2@ HORZ 2@
        2OVER 2OVER D<    IF 2SWAP D- DLEFT ELSE D- DRIGHT THEN
      DUP 6 + 2@ VERT 2@        2OVER 2OVER D<
        IF 2SWAP D- DDOWN    ELSE D- DUPP    THEN
      DUP 10 + 2@ ROTARY 2@        2OVER 2OVER D<
        IF 2SWAP D- DCCW      ELSE D- DCW     THEN 0 SWAP !
      ELSE DROP 1 ABORT" CAN'T RESTORE!"    THEN ;




  124 LIST

( HOME THE ROBOT, CHT, 4/23/81 )
: D-INIT    0 0     2DUP HORZ 2!    2DUP VERT 2!
     ROTARY 2!     STORAGE 14 ERASE    ;

: HOME ( RETURN TO STARTING POSITION )
       HORZ 2@ DUP 0<    IF DABS DRIGHT ELSE DLEFT THEN
       VERT 2@ DUP 0<    IF DABS DUPP ELSE DDOWN   THEN
       ROTARY 2@ DUP 0< IF DABS DCW  ELSE DCCW    THEN
       0 DUP 2DUP HORZ 2!    2DUP VERT 2!    ROTARY 2! ;

OCTAL
: ?ESCAPE  177562 @ 177 AND 132 =
       IF STORAGE @ IF CR ." CAN'T INTERRUPT!" KEY-START
                    ELSE SAVE XESCAPE THEN    THEN ;
DECIMAL



  125 LIST

( ESCAPE AND REENTRY, CHT, 4/23/81 )
CODE XRESET    S S0 U) MOV    S -) CLR    NEXT

: XNUMBER    DUP 1+ C@    45 =    DUP >R    +
     0 BEGIN    SWAP DIGIT    IF ROT BASE @ * +    AGAIN
     C@ 32 -    IF ." ?"    XRESET THEN    R> IF MINUS THEN ;

: XENTRY     BEGIN -'    IF XNUMBER    ELSE EXECUTE
     ?STACK IF ." DATA ERROR " XRESET THEN THEN 0 END   ;

: XESCAPE   BEGIN CR ." OK. TYPE 'R' & CR TO RETURN. "
       0 BLK ! 0 >IN !    S0 @ 80 EXPECT    XENTRY    0 END ;

: R STORAGE @ IF RESTORE KEY-START    R> R> 2DROP
              ELSE CR ." CAN'T DO IT!"    THEN ;
: ABORT 1 ABORT" ABORTED!" CR ;
```

CONTROL ROBOT THROUGH KEYBOARD

SLEW Enter into a loop. 6 special function keys will cause the robot to move at slow speed. Ascii 24 code will terminate this loop. The special function keys on a TEC-70 terminal were used to implement this control command. Other regular key can be used by changing the code number in this routine.

```
  126 LIST

( SLEW THE ROBOT, CHT, 4/23/81 )
: SLEW  BEGIN KEY
     DUP 0= IF 20 0 DRIGHT THEN
     DUP 4 =   IF 20 0 DLEFT THEN
     DUP 8 =   IF 20 0 DUPP THEN
     DUP 12 =  IF 20 0 DDOWN THEN
     DUP 16 =  IF 20 0 DCW THEN
     DUP 20 =  IF 20 0 DCCW THEN
     24 = END  ;
```



1. Image Processor Architecture