# ChannelACT™

# Operator's Manual

TECHNOLOGY INC.

Technology 80, Inc.

658 Mendelssohn Ave. North • Minneapolis, MN 55427 • (612) 542-9545

# ChannelACT

## OPERATOR'S MANUAL

Document Number: 2370-5

Released: June 1989

Revised: July, August, October, December 1989

## PREFACE

This manual provides instructions and reference information for use with the ChannelACT from Technology 80 Inc. The purpose of this manual is to answer questions pertaining to the basic applications of this module.

It is assumed that the user of the module possesses some fundamental knowledge of computer mainframe systems and operation. Such knowledge is required to successfully interface the ChannelACT with the mainframe computer. Technology 80 Inc. will be eager to assist the user with application problems pertaining to the use of this module not addressed in this manual. We recommend consulting documents provided by the manufacturers for questions relating to the computer portion of your system.

Technology 80 Inc. manufacturers and markets mainframe testing devices. These devices offer inexpensive solutions to mainframe maintenance problems. Application assistance in implementing our products and systems is available directly from Technology 80 Inc.

## PREFACE

This manual provides instructions and reference information for use with the ChannelACT from Technology 80 Inc. The purpose of this manual is to answer questions pertaining to the basic applications of this module.

It is assumed that the user of the module possesses some fundamental knowledge of computer mainframe systems and operation. Such knowledge is required to successfully interface the ChannelACT with the mainframe computer. Technology 80 Inc. will be eager to assist the user with application problems pertaining to the use of this module not addressed in this manual. We recommend consulting documents provided by the manufacturers for questions relating to the computer portion of your system.

Technology 80 Inc. manufacturers and markets mainframe testing devices. These devices offer inexpensive solutions to mainframe maintenance problems. Application assistance in implementing our products and systems is available directly from Technology 80 Inc.

## HOW TO USE THIS MANUAL

### Purpose and Organization of this Manual

This manual is meant as an instruction and reference guide for use with the ChannelACT, a mainframe I/O channel simulator developed by Technology 80, Inc.

For ease of reference, the manual is organized by task: the major sections correspond, sequentially, with the steps typically involved in using the ChannelACT. For the most part, the subsection topics are organized in the same order as they appear as options or queries on the ChannelACT's screens.

The introductory material, including the "Getting Started" section, is designed primarily for the first-time user. These sections contain information which is necessary for proper installation of the ChannelACT, and for understanding the basic principles involved in its operation.

Detailed, supplementary information concerning operation of the ChannelACT may be found in appendix form. References to this supplementary material may be found at appropriate points throughout the text.

### Stylistic Conventions Followed in the Manual

> All text strings which appear on the ChannelACT's screens are enclosed in quotes (example = "Main Menu").

> All screen prompts which require a direct response are enclosed in quotes and printed in boldface type.

> Keyboard keys are signified by characters enclosed in brackets (example = [a] or [SHIFT]).

> Step-by-step instructions which must be performed in sequence are numbered.

> Elements in lists that present non-sequential information are highlighted with a ">" symbol.

Questions concerning the use of this manual or the use of the ChannelACT should be directed to Technology 80, Inc., 658 Mendelssohn Ave. N., Minneapolis, MN 55427, (612) 542-9545.

## QUICK START

### The 15 Minute Get Acquainted Start

### I. Introduction

The ChannelACT can be useful to the channel technician after taking a few minutes to read this Quick Start Section. There are several operator controls and many useful features that can be used to perform precise tests and analysis. And while it takes a large manual to explain these features in detail, the operation of the ChannelACT is not complex. The unit is user friendly. By thoroughly reading this section, the user will gain the information necessary to effectively use the ChannelACT.

The operation of the ChannelACT is menu driven. All required operator actions are asked for on the computer screen. Just a few controls, covered below, are needed as a start to get from menu to menu. The user does need a basic knowledge of the mainframe channel in order to effectively use the ChannelACT.

### II. Starting the ChannelACT

The ChannelACT is connected to a control unit in the same way the mainframe was connected. The starting sequence of steps are:

1. Turn the corresponding channel off in the mainframe computer.
2. Turn the control unit off.
3. Disconnect the channel cable at the mainframe computer, that runs between the mainframe and the control unit.
4. Connect the channel cable into the ChannelACT Bus and Tag Cable Connectors.
5. Turn on the control unit.
6. Turn on the ChannelACT.
7. Install the ChannelACT software in diskette drive A. Or if the unit comes with a hard drive then the message "1.4 M Driver Card Installed" will appear on the screen.
8. Type in the word "ACT" when given "C:\>_" and the main menu appears. This menu is accessed when main operations need to be done such as executing the program, running the diagnostics, or exiting the program.
9. From the main menu the program development menu can be entered by pressing "2". This menu should be accessed when the user wants to write programs for the ChannelACT.

## III. Operating the ChannelACT

Programs may be written for the ChannelACT to test and maintain peripherals. To write programs enter the program development menu. Access to this menu is described in steps 8 and 9 in the previous section on starting the ChannelACT. To get a print-out of any screen type [SHIFT][PrtSc *].

## IV. A Sample Program

To write a sample program enter the program development menu. Type "2" to enter the program editor. Then type in the following sample program:

```
#
# This is a sample program for a tape drive whose address is 80 in
# hex.  The program does a system reset, writes a block of data to
# the tape, rewinds the tape, and then reads back the block and
# stores it on the PC drive.  The program demonstrates how one way
# error recovery might be implemented.
#

channel BLOCKMUX               # channel type chosen
system_reset
ccw0 \80 \07 /C                # rewind, chaining option
loadfile "test.ram" /T
data "append this sentence on",CR,LF
a: ccw0 \80 \01 /C             # write
# error recovery
if (failed)
  ccw0 \80 \27 /C              # backspace block
  restore                      # restore data
  goto a
endif
ccw0 \80 \07 /C                # rewind
ccw0 \80 \02 /C                # read
storefile "test2.ram" /T
ccw0 \80 \03                   # last command, chaining off
```

## Quick Start

---

A sample program in "C" Language has been listed below. (Note: This program would have to be compiled with a Microsoft C or a Turbo C compiler before it could be executed on the ChannelACT).

```
/********
* This is a sample program for a tape drive whose address is 0x80.
* The program does a system reset, writes a block of data to the
* tape, rewinds the tape, and then reads back the block and stores
* it on the PC drive. The program demonstrates one way error *
recovery might be implemented.
********/

#include <act.h>                /* always included */

main()
{
  initialize();                 /* always first function called */
  channel(BLOCKMUX);            /* channel type chosen */
  system_reset();
  ccw0(0x80,0x07,_C);           /* rewind, chaining option */
  ldfile("test.ram",TRANSLATED);
  ldstr("append this sentence on\r\n");
  a: ccw0(0x80,0x01,_C);        /* write */
/* error recovery */
  if (_failed_){
    ccw0(0x80,0x27,_C);         /* backspace block */
    restore();                  /* restore data */
    goto a;
  }
  ccw0(0x80,0x07,_C);           /* rewind */
  ccw0(0x80,0x02,_C);           /* read */
  stfile("test2.ram",TRANSLATED);
  ccw0(0x80,0x03,0)             /* last command, no chaining */
}
```

## V. Executing the Sample Program

Before executing the program, it should be saved. It is saved by pressing [F2] while remaining in the program editor. To execute the program type [F3].

## VI. Disconnecting the ChannelACT

The following list is the procedure for disconnecting the ChannelACT:

1. Turn the corresponding channel off in the mainframe computer (it should already be off).
2. Turn the control unit off.
3. Disconnect the channel cable at the ChannelACT, that runs between the ChannelACT and the control unit.
4. Connect the channel cable into the mainframe computer's Bus and Tag Cable Connectors.
5. Turn on the control unit.
6. Turn on the channel in the mainframe computer.

## TABLE OF CONTENTS

## LIST OF ILLUSTRATIONS

# SECTION 1: INTRODUCTION

## 1.1 GENERAL DESCRIPTION

ChannelACT is a completely self-contained FIPS channel emulator, which allows users to test, design and service compatible peripherals without tying up expensive mainframe time. Rugged construction and a compact design makes the ChannelACT portable enough to take to the field where tests can be conducted on sight. It connects to peripherals using standard Bus and Tag cables. All channel protocols including Selector, Byte Multiplexer, Block Multiplexer and 4.5 M byte data streaming rates are supported.

Channel testing can be performed by using the predefined channel sequences within the menu-driven Supervisor. The ChannelACT also allows users to program channel sequences with either the easy-to-use Extended Basic Interpreter or by using the supplied Microsoft C or Turbo C language library.

## 1.2 FEATURES

* Emulates Channel Transmissions in Real-Time
* Portable and Self-Contained: requires no external supporting device
* Connects to Standard Bus and Tag Cables
* 4.5 MB/sec Data Streaming Rates
* Programmable CCWs in Real-Time
* Supports all three channel modes: Selector, Byte Multiplexer and Block Multiplexer
* 64K Storage Buffer
* Programmable Channel Sequences using supplied Microsoft or Turbo C Language Library
* Channel Sequences programmable using the simple Extended Basic Interpreter
* Interactively steps through channel sequences
* Fully supports the following channel protocols:
    > Initial Selection
    > Data Transfer
    > Ending Procedures
    > Stack Status
    > Interface Disconnect
    > Selective Reset
    > System Reset
    > Command Retry
    > Request Sequence

## 1.3 PHYSICAL DESCRIPTION

**Dimensions:**

| | |
|---|---|
| Width | 18.25" |
| Height | 7.5" |
| Length | 21.0" |

**Weight:** Approximately 40 pounds.

**Composition:** Aluminum skin, stainless steel handle, Urethane front bezel and panel.

**External Features:**

**Front Panel Features:** (See Figure 1 - 1 on Page 1 - 3)

1. Brightness Adjustment Knob - Controls the light intensity of the Cathode Ray Tube.

2. Reset Button - Re-starts the ChannelACT and clears the memory.

3. Cathode Ray Tube - Used for viewing menus and displaying sample data.

4. Power Indicator - An LED that is illuminated when the power switch is in the "ON" position and the unit is connected to a working power supply.

5. Diskette Drive A - Used for loading and storing information from the ChannelACT.

6. Diskette Drive B or 20 Mega Byte Hard Drive - A hard drive is optional. Both are used for loading and storing information from the ChannelACT.

7. Keyboard Jack - The jack used to plug the keyboard into the ChannelACT.

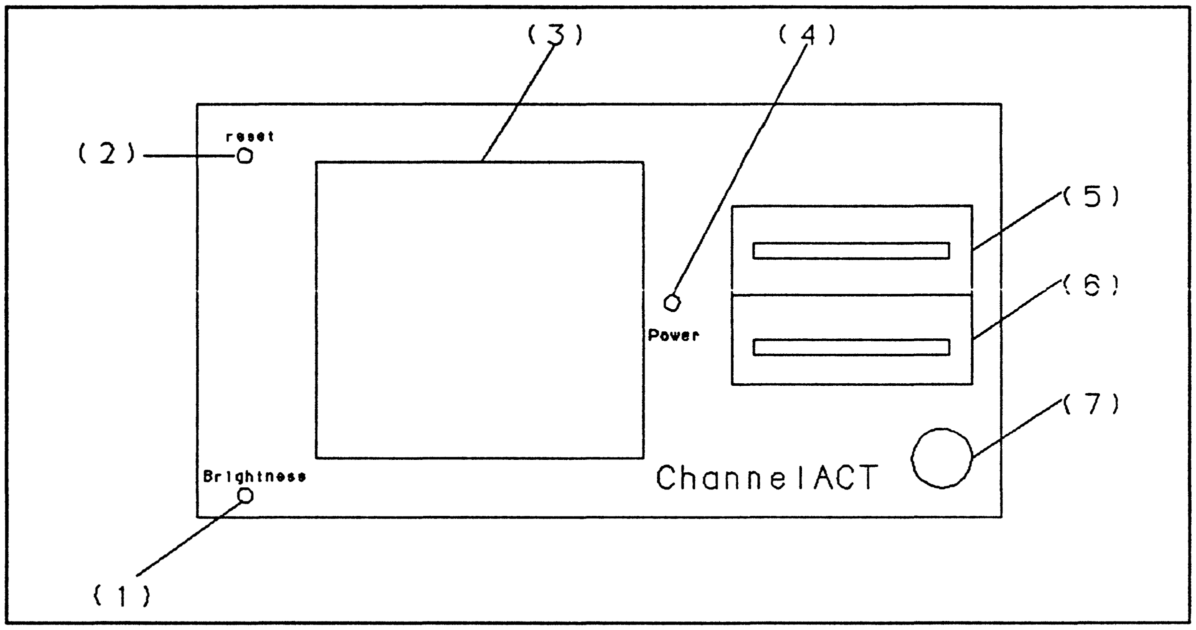


**Figure 1 - 1: ChannelACT Front Panel**

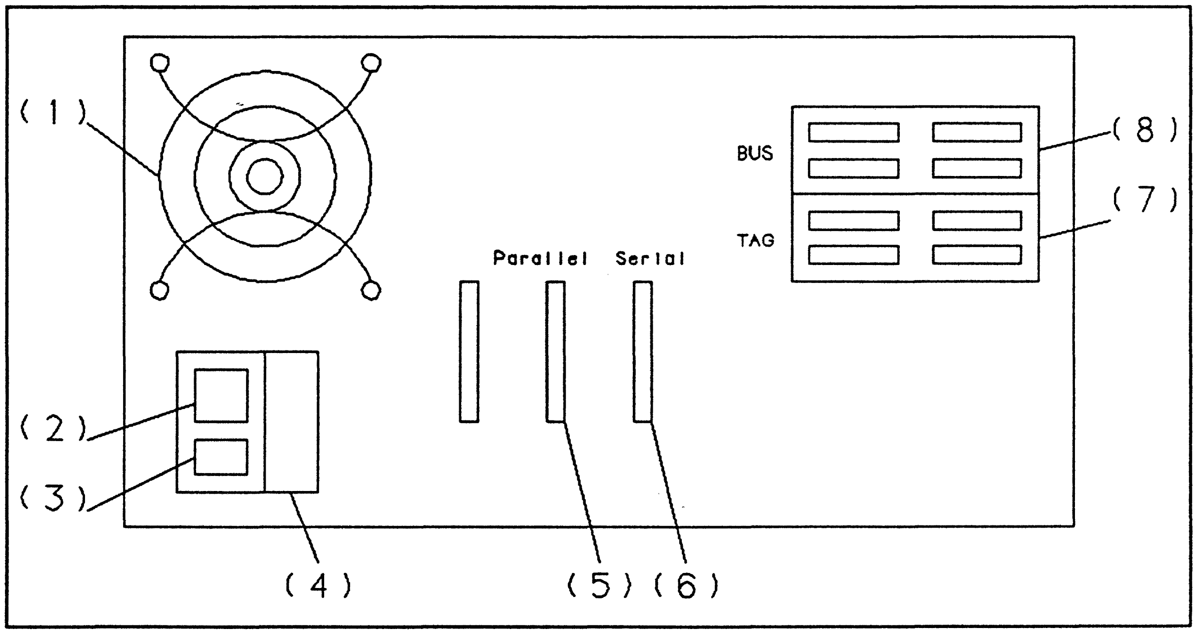


**Figure 1 - 2: ChannelACT Rear Panel**

**Rear Panel Features:** (See Figure 1 - 2 on Page 1 - 3)

1. Internal Fan - Regulates the internal temperature.

2. Power Cord - Extending from the lower left side of the rear panel, this cord should be plugged into a nominal 120 Volt, 60 Hertz power source (220 Volt, 50 Hertz models are optional).

3. Power Switch - Located on the left side when looking at the back panel, this switch is used to turn the unit on and off.

4. Fuse Connector - For proper operation, a 7 Amp fuse should be used.

5. Parallel Port - A parallel port used to connect a printer to the ChannelACT.

6. Serial Port - A RS232 port used to make external connections to the ChannelACT.

7. Tag Connectors - Two serpentine type connectors (labeled "Tag") that allow attachment of the channel Tag cables to the ChannelACT.

8. Bus Connectors - Two serpentine type connectors (labeled "Bus") that allow attachment of the channel Bus cables to the ChannelACT.

## 1.4 DEFINITIONS

Default: The pre-existing value before the user changes that value.

File Listing Screen: This screen shows the programs that are accessible to the user.

Help Message: A message that explains what a program does without having to execute the program. The message is written by the programmer. And it appears when the cursor is placed on a program when the user is in a File Listing Screen.

## 1.5 APPLICATIONS

The ChannelACT will prove useful in any of a variety of applications involving the mainframe I/O interface channel. These might include:

> End-user (on-site) maintenance of a mainframe system.
> Third-party or off-sight maintenance of a mainframe system.
> The development or expansion of a mainframe system (by inclusion of additional peripherals or hosts).
> Beta-site testing of peripherals or networking equipment.
> Laboratory testing in the development of hardware and software.

# SECTION 2: FUNCTIONAL DESCRIPTION

## 2.1 HOW THE ChannelACT WORKS

The ChannelACT acts like a mainframe computer. It simulates a mainframe computer through the software that is written and executed in it. Channel sequences can be written using the Extended Basic Interpreter within the Supervisor or by using the supplied Microsoft or Turbo C language library. These software programs test and service compatible peripherals.

The ChannelACT can be separated into two major systems. The first is the personal computer system and the second system is the ChannelACT card. These systems can been seen in
Figure 2 - 1. The personal computer system operates like any other IBM compatible personal computer. The ChannelACT card takes information from the personal computer and translates it to channel information. The channel information can be loaded onto the channel by the Tag and Bus Cables.

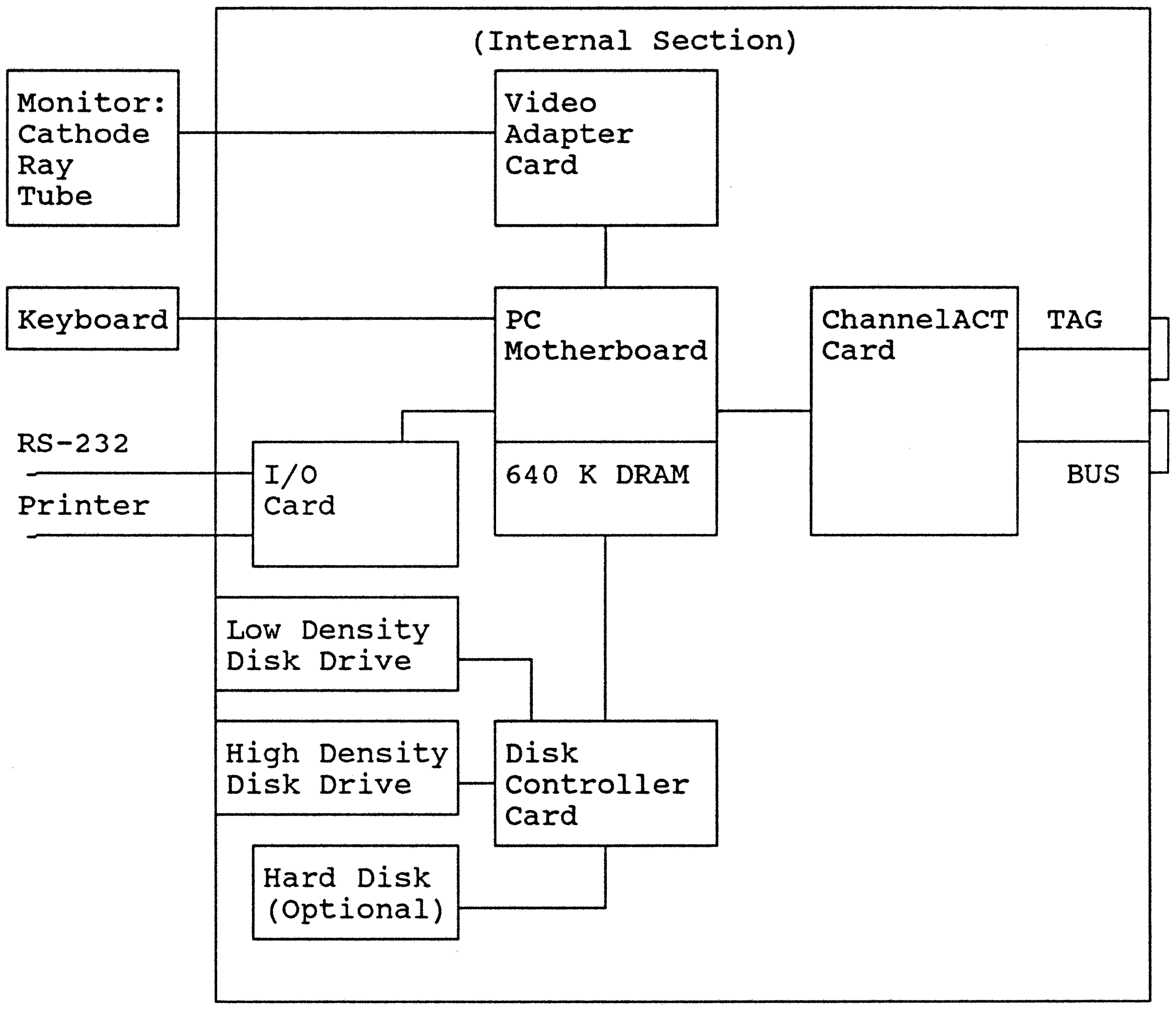


**Figure 2 - 1: ChannelACT Block Diagram**

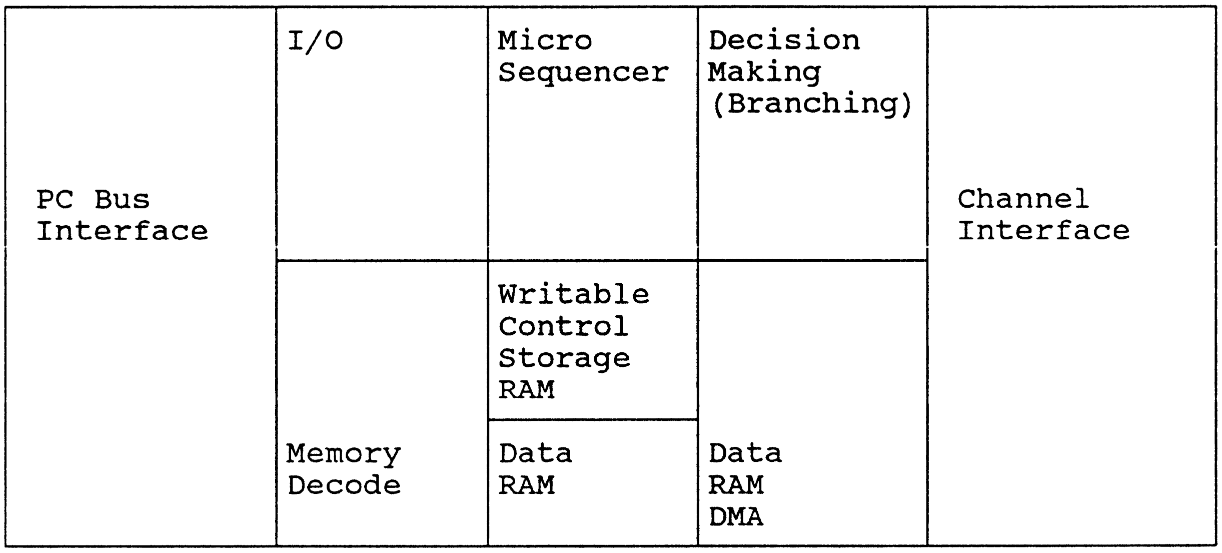


**Figure 2 - 2: ChannelACT Card** Block Diagram

## 2.2 DESCRIPTION OF THE ChannelACT CARD STAGES

The ChannelACT Card has nine stages. They are illustrated in Figure 2 - 2 and are described below.

Personal Computer (PC) Bus Interface

The PC Bus Interface contains all the buffers and the drivers needed to interface the card to the PC Bus.

Input/Output (I/O)

The Input/Output section decodes the addresses of all the PC Bus driven I/O commands. It also generates the strobes and enables needed to set up operation of the system and it reads back the status and data values.

Memory Decode

The Memory Decode decodes the addresses that allow the PC to access the Writable Control Storage and Data RAM.

Micro Sequencer

The Micro Sequencer contains the Am2910 chip and support chips that run the channel sequencer.

Writable Control Storage

RAM that contains the micro code used by the Micro Sequencer.

Data Ram

64 KiloBytes of memory to hold data that is transferred on the channel.

Decision Making

The Decision Making section contains registers and masks that allow the Am2910 to execute conditional branch instructions.

Data RAM DMA

Circuitry that transfers data between the Data RAM and the channel.

Channel Interface

The Channel Interface contains buffer, drivers and terminators that convert between TTL logic levels and channel logic levels.

# SECTION 3: GETTING STARTED

## 3.1 CONNECTING THE ChannelACT

The ChannelACT should be connected to the channel with regular Bus and Tag cables having "serpentine" connectors. Figure 3 - 1 on the following page illustrates the relationship of the ChannelACT to typical components of a mainframe system, and shows where it should be placed.

The following procedure should be followed when connecting the ChannelACT to the control unit:

1. Turn the corresponding channel off in the mainframe computer.
2. Turn the control unit off.
3. Disconnect the channel cable at the mainframe computer, that runs between the mainframe and the control unit.
4. Connect the channel cable into the ChannelACT Bus and Tag cable connectors.
5. Turn on the control unit.
6. Turn on the ChannelACT.

The ChannelACT is connected to the control unit in a similar manner as the mainframe computer is. The dark connectors are connected to the back of the ChannelACT and the light connectors are connected to the control unit. Connections should always be made between pin housings of opposite colors; lighter housings should be connected into darker housings, and darker into lighter.

Care should be taken that individual pins in the connectors are not bent. This is best accomplished by making the connection at an angle instead of straight in:

1. Rest the base of the cable pin housing to be connected on the base of the unit's pin housing.
2. Gradually bring the cable housing up and toward the connection points.
3. Gently push the two sets of connectors together. The connection will hold even though the housings do not "snap" together.
4. The connection may be secured by tightening the screws on the back of the cable pin housings.

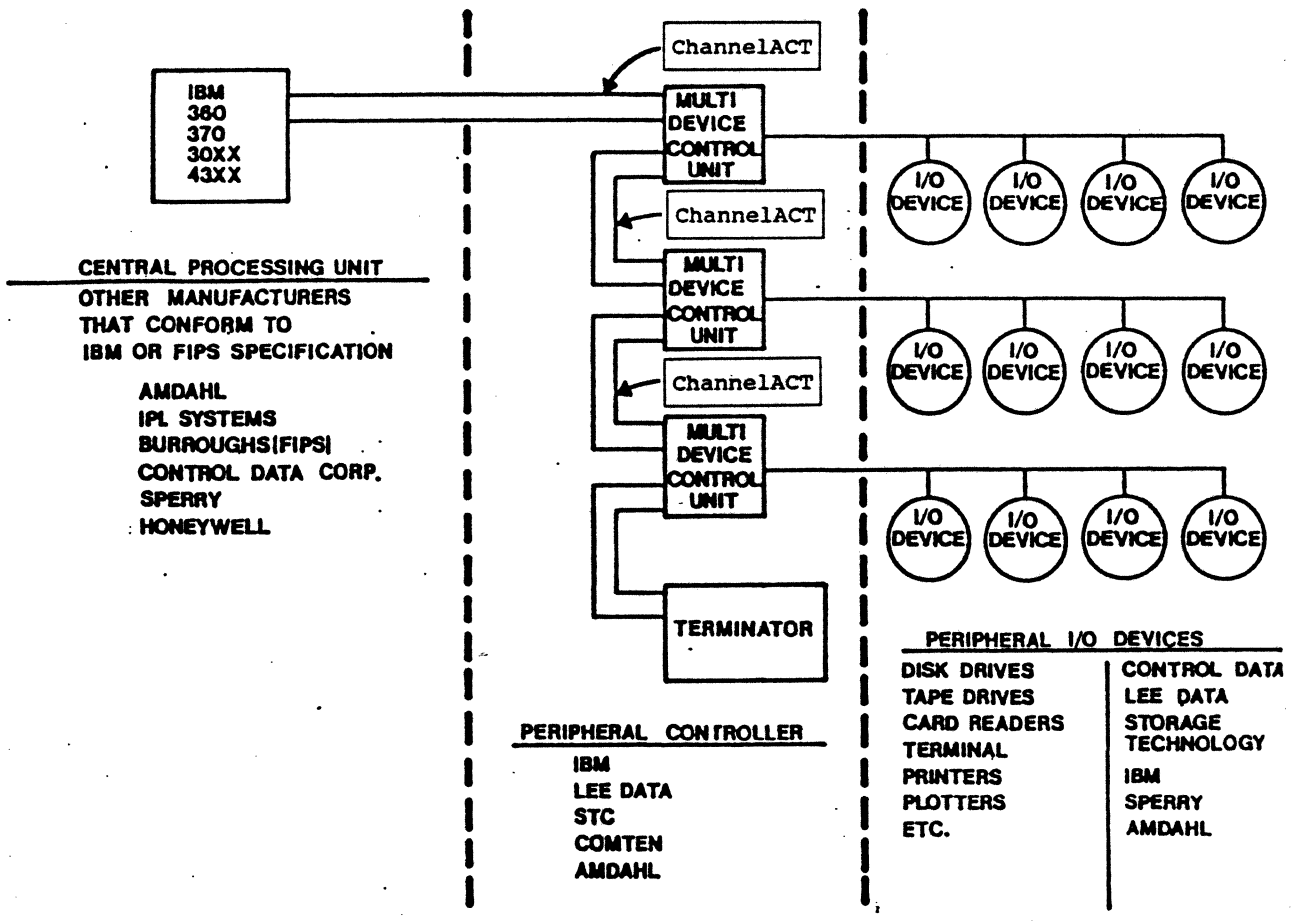


Figure 3 - 1: ChannelACT Placement

## 3.2 STARTING THE ChannelACT

**Power Source:**

The ChannelACT operates from a nominal 120 Volt, 60 Hertz power source and uses a seven amp fuse. Optionally, units which operate from 220 Volt, 50 Hertz power sources are available.

**Before "powering-up" the ChannelACT:**

1. Make sure that the power switch (located on the left side of the rear panel) is in the off ("0") position. In this position, the "0" half of the switch will be parallel with the plane of the back panel.

2. Make sure that no diskette is in the diskette drive.

**To power up the ChannelACT:**

1. Plug the power cord into an appropriate power source outlet.

2. Park the power switch in the "on" position (Press the "1" half of the switch so that it will be parallel with the plane of the back panel).

**After "power-up":**

1. A screen will appear showing the version of software being used by the unit. This screen should appear ten to fifteen seconds after power-up.

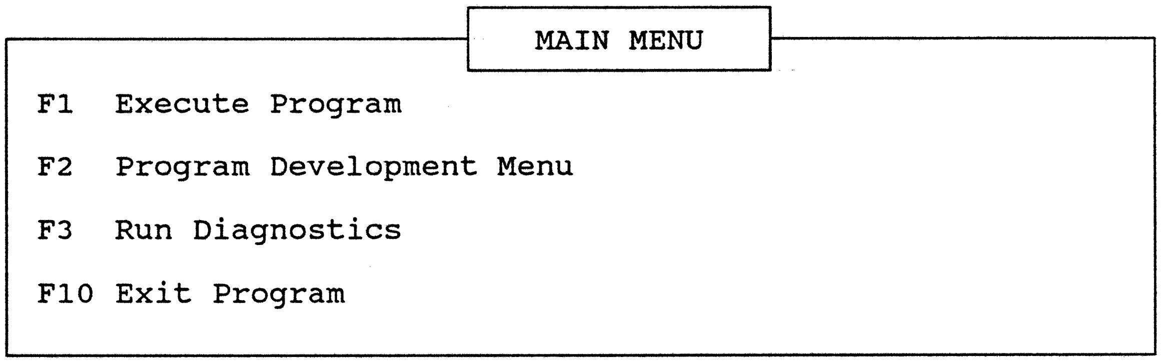
2. A series of diagnostic checks will be performed to verify that the unit's processor, internal components, and software are operational.

3. Install the ChannelACT software in diskette drive A. Or if the unit comes with a hard drive then the message "1.4 M Driver Card Installed" will appear on the screen.

4. Type the word "ACT" when given "C:\>_" and the main menu appears. This menu is illustrated in Figure 3 - 2. This menu is accessed when main operations need to be done such as executing the program, running the diagnostics, or exiting the program.

5. From the main menu the program development menu can be entered by pressing "F2". This menu is illustrated in Figure 3 - 3. This menu should be accessed when the user wants to write programs for the ChannelACT.

If any abnormalities exist, an appropriate message identifying the source of the problem will appear on the screen. In such cases, Technology 80 should be contacted for assistance in correcting the problem.

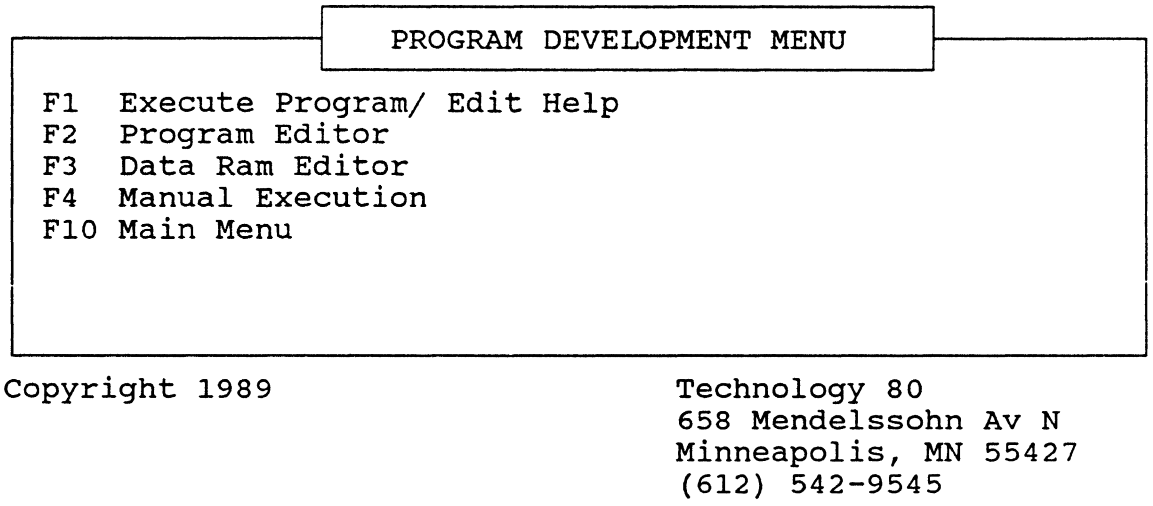


```
MAIN MENU

F1  Execute Program

F2  Program Development Menu

F3  Run Diagnostics

F10 Exit Program
```

Copyright 1989

TECHNOLOGY 80
658 Mendelssohn Av N
Minneapolis, MN 55427
(612) 542-9545

**Figure 3 - 2: The Main Menu**



PROGRAM DEVELOPMENT MENU

```
F1  Execute Program/ Edit Help
F2  Program Editor
F3  Data Ram Editor
F4  Manual Execution
F10 Main Menu
```

```
Copyright 1989                     Technology 80
                                   658 Mendelssohn Av N
                                   Minneapolis, MN 55427
                                   (612) 542-9545
```

**Figure 3 - 3: The Program Development Menu**

## 3.3 PRINTING WITH THE ChannelACT

To get a printout of any screen press [SHIFT][PrtSc *].

## 3.4 INSTALLATION CONSIDERATIONS

* Always back up disks, keep originals in a safe place
* ACT.EXE or a stand alone 'C' program will look for the microcode file SIM3.BIN in the current directory therefore keep SIM3.BIN in the same subdirectory as ACT.EXE and run all programs from that subdirectory
* Alternatively, under DOS 3.2 or later versions, locate SIM3.BIN in an "APPEND" statement in your AUTOEXEC.BAT file

  Example: `APPEND C:\SIM`

  This tells DOS to search for all auxiliary files in the C:\SIM subdirectory
* If the ACT.EXE program is to be used in demonstration mode on a different computer than the ChannelACT, include the following in that computer's CONFIG.SYS file:

  ```
  STACKS=0,0
  ```

## 3.5 TURNING OFF THE ChannelACT

The ChannelACT may be turned off at any time while it is being operated. However, no diskette should be in the unit at the time it is turned off.

To turn off the ChannelACT:

> Press the "0" half of the power switch so that it will be parallel with the plane of the back panel.

When the ChannelACT is turned off, all data currently stored in its internal memory will be lost. Turning off the ChannelACT has no effect whatsoever on current channel operations.

## SECTION 4: SOFTWARE

### 4.1- SOFTWARE OVERVIEW

The simulator software is designed to be extremely flexible. Most channel operations can be accomplished in one of three different ways:

(1) By writing a 'C' program utilizing either the Microsoft or Turbo C library.

(2) By writing a program utilizing the integrated Basic-like interpreter.

(3) By manually stepping through a channel sequence utilizing the supervisor and directly driving the channel lines.

Both types of programs, 'C' and interpreted, can be executed from within the the ChannelACT's software. A one line "help message" can be created for each program to describe its purpose. A technician will then be able to quickly pick out and run a sequence of test programs.

Also, the ChannelACT software can be run on any computer in a demonstration mode. Type "ACT/D" to start the demonstration. See page 3 - 5 for the "Installation Considerations" to help in this process.

The software that comes with the ChannelACT is listed below in the Software Packing List.

Software Packing List:

| Disk 1 | Disk 2 |
|---|---|
| ACT.EXE | ACT_MS_S |
| SIM3.BIN | ACT_MS_M |
| (SAMPLE.C) | ACT_MS_C |
| (TEST.RAM) | ACT_MS_L |
| | ACT_MS_H |
| | ACT_TC_S |
| | ACT_TC_M |
| | ACT_TC_C |
| | ACT_TC_L |
| | ACT_TC_H |
| | ACT.H |

## 4.2- ChannelACT SOFTWARE (ACT.exe)

### 4.2.1 Main Menu

The main menu offers the following selections:

**F1 Execute Program**
**F2 Program Development Menu**
**F3 Diagnostics**
**F10 Exit Program**

Selection F1 accesses the file listing screen which displays all files with extensions of "sim" (an interpretive file) or "exe" (presumably written in 'C'). A help message, if it exists, is displayed with each file. A further description of this selection is found under the description of the file listing screen, which is accessed from many different places in the supervisor.

Selection F2 displays a submenu which is described in section 4.2.2.

Selection F3 runs the internal diagnostics. If a "microcode checksum error" occurs try reinstalling the original software that came with the ChannelAct. If this doesn't work or if other errors occur, contact Technology 80.

Selection F10 exits the ChannelACT software. It exits the program and places you in DOS.

A DOS command can be executed from anywhere within the ChannelACT software. Hitting Alt-D will execute a DOS shell. Execute the DOS command and then type "exit" to return to the ChannelACT software.

A list of optional responses to errors can be found in Appedix D.

### 4.2.2 Program Development Menu

The program development menu offers the following selections:

**F1 Run Program/ Edit Help**
**F2 Program Editor**
**F3 Data Ram Editor**
**F4 Manual Execution**
**F10 Main Menu**

Selection F1 is similar to the selection F1 on the main menu, the only difference being that the help messages can be edited from here. A further description of this selection is found under the file listing screen (section 4.2.4).

A channel sequence can be executed in two different ways from the program development menu. Using selection F2, a program can be written and executed without using any of the other menu selections. Alternatively, selections F3 and F4 can be used to interactively step through the sequence. Selection F3 is used to set up the data that will be sent over the channel or to examine the data that is input over the channel. Selection F4 is used to manually execute the various simulator commands.

Selection F10 returns to the main menu. It is necessary to return to the main menu to exit the program.

When storing and retrieving files, the following default extensions are used:
"sim" for program files
"ram" for data files
It is strongly recommended that these default extensions are always used.

### 4.2.3 Editing Keys

The simulator editors use the same cursor control keys which are used by Wordstar™ and other popular editors:

Left Arrow- moves the cursor one space to the left.
Right Arrow- moves the cursor one space to the right.
Up Arrow- moves the cursor one line up.
Down Arrow- moves the cursor one line down.
Home- moves the cursor to the beginning of the current line.
End- moves the cursor to the end of the current line.
Pg Up- moves the cursor one page up.
Pg Dn- moves the cursor one page down.
(Ctrl)Pg Up- moves the cursor to the beginning of the file.
(Ctrl)Pg Dn- moves the cursor to the end of the file.
Backspace- destructively moves the cursor one space to the left.
Del- deletes the character the cursor is on.
Ins- toggles the insert mode. When the insert mode is on, the cursor appears as a block. When the insert mode is off, the cursor appears normally and characters are overwritten.

In addition, the program editor (not the data ram editor) contains the following cut-and-paste functions:

ALT M- Mark/Unmark
ALT C- Copy
ALT X- Cut
ALT P- Paste

ALT M marks the current line in reverse video. The cursor control keys will then mark additional lines. Hitting ALT M again will unmark any marked lines. Marked lines can be cut (ALT X) or copied (ALT C) to a scratch file. ALT P will paste the scratch file into the current file, inserting the lines before the line the cursor is on. Text can be copied or moved from one file to another.

### 4.2.4 File Listing Screen

```
                                       ↓
A:\*.ram                               .        <DIR>   88-07-14  16:05:46
4 files                                ..       <DIR>   88-07-14  16:05:46
283124 bytes free                      data     <DIR>   88-07-14  16:10:50
                                       test.ram   1028  88-08-12  15:29:15



Enter-Change Dir F6-Change Search F7-Change Sort F10-Cancel
```

The file listing screen is accessed by all both of the editor screens and the execute program option on the menus. The only differences are that a different default extension is used for the search pattern in each case and that different function keys are active in each case.

To load a file into the editor, use the up and down arrows to choose a file and then press enter.

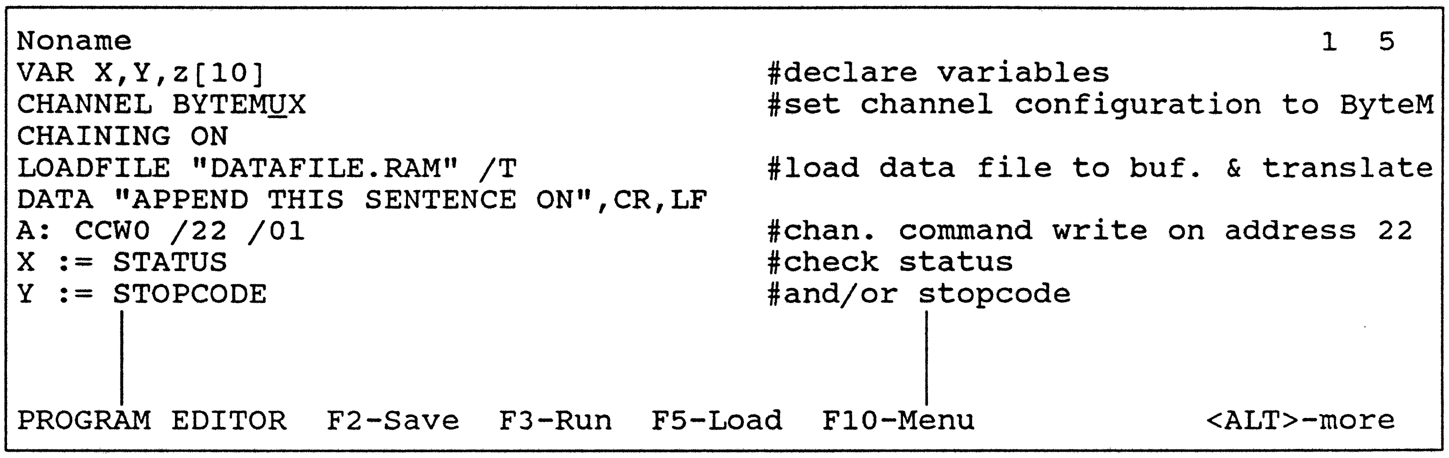
To go to a different directory, use the up and down arrows to choose that directory and press enter. To search for a different extension, or to go to another drive, use F6 and enter a different search pattern specification.

By default, the files are sorted by file name. To sort by filename extension, size, etc., press F7, use the left and right arrow to move the sort key indicator(the down arrow appearing at the top of the screen), and then press return. The date and time are put in military format, with the most significant information on the left, to make sorting by column meaningful.

To edit a help message, use the up and down arrows to choose the file whose help you want to edit and press [shift][F1]. You will then be able to type in a new help message. A help message can only be written by pressing "F1 Execute Program/Edit Help" from the Program Development Menu.

To execute a program, use the up and down arrows to choose the program you wish to execute and press F3. The program will be executed and the message <press any key to return to menu> displayed. Pressing any key will then return you to this screen.

## 4.2.5 Program Editor



```
Noname                                                                  1  5
VAR X,Y,z[10]                                   #declare variables
CHANNEL BYTEMUX                                 #set channel configuration to ByteM
CHAINING ON
LOADFILE "DATAFILE.RAM" /T                      #load data file to buf. & translate
DATA "APPEND THIS SENTENCE ON",CR,LF
A: CCW0 /22 /01                                 #chan. command write on address 22
X := STATUS                                     #check status
Y := STOPCODE                                   #and/or stopcode



PROGRAM EDITOR  F2-Save  F3-Run  F5-Load  F10-Menu            <ALT>-more
```

The program editor uses a language interpreter vary much like BASIC. What can be done on any of the other screens can be done in the program editor. After the program has been written or loaded, pressing F3 will run the program. Execution will occur on a separate screen. As displayed on that screen, pressing F7 will break execution of the program.

The standard editing keys are used- Left, Right, Up, Down, Home, End, Pg Up, Ctrl-Pg Up, Pg Dn, Ctrl-Pg Dn, Delete, Insert, and Backspace. The current cursor position is shown in the upper right hand portion of the screen. When in insert mode, the cursor will appear as a block. Holding down the alternate key will display the available cut-and-paste functions.

F2 will save the program onto a disk file. You will be prompted for the file name. If no extension is given, a default extension of "sim" is given.

F5 will load a disk file. See the information about the file listing screen in section 4.2.5.

A description of this interpretive language is given in section 4.3.

### 4.2.6 Data Ram Editor

```
 New File                                                     A  CR  LF
0000  TEST DATA..   E3  C5  E2  E3  40  C4  C1  E3  C1  0D  25


(s)F2-Save (s)F5-Load F6-Switch F7-Mne F9-Clear sF9-Restore F10-Menu
```

The data ram editor screen is split into two parts. The left part of the screen shows the EBCDIC character that corresponds to the hexadecimal number in the same relative position on the right part of the screen. A control character will appear as a dot on the left. Data can be entered on either side of the screen.

After entering the character or hexadecimal number, the cursor will automatically advance to the next position. Press F6 to change from one side of the screen to the other. The standard editing keys are used- Left, Right, Up, Down, Home, End, Pg Up, Ctrl-Pg Up, Pg Dn, Ctrl-Pg Dn, Delete, Insert, and Backspace. When in the insert mode, the cursor will appear as a block.

If the data RAM editor is entered after data is received over the channel, it will be in a "read only" mode. To be able to edit data, either the F5(load) or F9(clear) options must first be used.

After data has been written out to the channel, the data in the buffer can restored to be written out again with the (shift) F9 option. This option will only work if there has been no intervening channel read command.

The upper right hand portion of the screen shows the current character that the curser is positioned on, and the one before and after. Mnemonics will be appear here for control characters. For a list of control character mnemonics, see appendix C. To enter a mnemonic for a control character, press F7 from either side of the screen. After being prompted, enter the two or three letter mnemonic and press enter.

F2 and (shift)F2 will save the data onto a disk file. You will be prompted for the file name. If no extension is given, a default extension of "ram" is given. (shift)F2 will do an EBCDIC-to-ASCII conversion before storing the file.

F5 and (shift)F5 will load data from a disk file. See the previous page for information on the file listing screen. (shift)F5 will do an ASCII-to-EBCDIC conversion before loading the data.

A unique EBCDIC character has been assigned for every non-extended ASCII character. For a non-extended ASCII character, a ASCII-to - EBCDIC-to-ASCII conversion sequence will yield the original character. See Appendix B for the conversion matrices.

### 4.2.7 Manual Execution

```
(1)Channel:[BLOCKMUX]  (2)Buffer Size: FFFF   (3)Current Datacount: 0010

(4)SLI: 1  RQI: 1  OPI: 1  DSI: 1  SVI: 1  STI: 1  DTI: 1  ADI: 1  MKO: 1
(5)SPO: 1  SLO: 1  OPO: 1  HLO: 1  SVO: 1  DTO: 1  CDO: 1  ADO: 1

(6)Busout: D3                      (7)Busin:  FF

(8)Chaining:  Off     (9)Stacking:  Off

(10)Address: E3 [ODD]  (11)Command: 00 [ODD]  (12)Data: 03 [ODD]

(13)Sequence:  CCW0

(14)Last Address In:  E3

(15)Last Status:  0C  Channel End + Device End

(16)Last Sense:  00 00 00 00 00 00 00 00


F3-Execute F9-Clear Data sF9-Restore Data F10-Menu ALT D-Dos
```

All of the fields on the manual execution screen are displayed and/or entered as hexadecimal numbers. Use the left and right arrow keys to move around the screen. Square brackets will appear around the current input field. A value may be typed in when a cursor is given or arrow keys may be struck to change options. To select an option simply press [Enter]. If the cursor does not appear, the only input accepted will be one of the function keys or the spacebar, which will toggle the field to another value. Upon execution of a command the option to continue or abort will be given. If an error has occurred the option to retry will be given.

The following information is displayed:
(1) The type of channel- "Selector","Bytemux",or "Blockmux".
(2) The current data buffer size. The default is 64K-1, the largest size possible.
(3) The current data count. This will either be equal to the number of bytes that were received over the channel, or it will be the number of bytes yet waiting to be sent over the channel.

(4) The current value of the tagin lines.
(5) The current value of the tagout lines. These lines can be manually toggled up and down.
(6) The current value of busout. To manually change the value of busout, press spacebar. A pop-up menu will appear and you will be able to put one of the values specified in #9, #10 ,or #11 on the bus, or be able to disable the bus.
(7) The last value of busin received.
(8) Whether the chaining option is "On" or "Off". While on, chaining will be signalled to the peripheral until this option is again explicitly turned off.
(9) Whether the stacking option is "On" or "Off". While on, stacking will be signalled to the peripheral until this option is again explicitly turned off.
(10) The peripheral address.
(10.5) The address parity, which can be set "Odd" (normal) or "Even".
(11) The channel command.
(11.5) The command parity.
(12) A data byte. None of the sequences use this value. This is only used if you want to manually put a value on busout with option #6 and classify it as data.
(12.5) Data Parity. This includes not only the parity of the byte specified in (12) but all data sent on busout.
(13) The sequence type. To change the sequence type, press return and a pop-up menu of choices will appear.
(14) The last address received on the channel.
(15) The last status received on the channel.
(16) The last sense bytes received when a sense sequence was executed.

To manually execute a sequence:
(1) If data is to be output over the channel, first set up the data using the data ram editor.
(2) Input the appropriate information on the screen.
(3) Press F3 to execute the microcode sequence. The bottom of the screen will now read "F7-Break". If a error message or successful execution message is not returned within an appropriate amount of time, return can be forced with the F7 key.
(4) If data was input over the channel, it can be read by going to the data RAM editor.
(5) Data can be "restored" for successive writes with the shift(F9) key.

Alternatively, the tagout and busout lines can be driven directly from this screen.

The tagin and tagout abbreviations are given below:

| | Tagin lines | | Tagout lines |
|---|---|---|---|
| SLI | Select In | SPO | Suppress Out |
| RQI | Request In | SLO | Select Out |
| OPI | Operational In | OPO | Operational Out |
| DSI | Disconnect In | HLO | Hold Out |
| SVI | Service In | SVO | Service Out |
| STI | Status In | DTO | Data Out |
| DTI | Data In | CDO | Command Out |
| ADI | Address In | ADO | Address Out |
| MKO | Markzero | | |

## 4.3 ERRORS

Programming - both modes of programming (interpretive and 'C') have certain features in common which are described in this section. Particular features of each language are then described in later sections.

Program execution:
The channel sequences will displayed on the screen as they are executed. Errors messages are also posted on the screen. Possible error messages, along with their corresponding "error code number" are listed in Appendix C. If an error occurs in the sequence, you will be prompted for a response. Type in the first letter of the desired option. Options that might appear are:

(1) Abort- program execution will be aborted.

(2) Continue- program execution passes to the next line of the program. This option should be chosen if error recovery is handled by the program.

(3) Fail- like continue, only the system variable failed (_failed_ in 'C') is set.

(4) Override- This option appears if a negative error code (usually signifying a parity error) was returned by the last channel sequence and works as the function of the same name. The error will be ignored and the sequence will continue where it left off. If the sequence was complete, the return code will be translated to what it would have been if no error had occurred. The result of a override might be another error.

(5) Retry- This option appears if an exceptional status is returned for a CCW and works as the function of the same name. The data buffer is restored (as the function "restore") and the last CCW sent is re-executed.

(6) Sense- This option appears if Unit Check occurred in the status. A sense will be performed and the result displayed. You will then be re-prompted for another choice.

Example:
```
  SYSTEM_RESET
  CCW0 \80 \07
  CCW0 \80 \01
    Exceptional Non-initial status presented
    Retry, Sense, Continue, Fail, Abort? S
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    (for command before sense) Retry, Continue, Fail, Abort? A
    Program complete
    <press any key>
```

Alternatively, automatic error handling can be programmed. See "Channel sequence options", below.

Channel sequence options:
Options can be added onto certain channel sequences that will effect the way they operate. In the interpreter, options are preceded with forward slash. In 'C', options are preceeded with an underscore and they are 'OR-ed' together.

An example in the interpretive language might be

```
  ccw0 \80 \03 /C /ER
```

where the options C and ER have been chosen.
The same example in 'C' would be

```
  ccw0(0x80,0x03,_C |_ER);
```

If no options are desired in 'C', the last argument is zero.

Permissible options are:

1. C Chaining- Chaining is indicated for any status containing Channel End or Device End
2. S Stacking- Non-zero status will be stacked. The stacking indication is then reset, so the next status presented will be accepted.
3. PA Parity Address- Incorrect(even) parity will be generated for the address. The parity error indication is then reset, so the next address generated, say on error recovery, will have the correct parity.
4. PC Parity Command- Incorrect parity will be generated for the command and then the indicator is reset.
5. PD Parity Data- Incorrect parity will be generated for all data in that sequence and then the indicator is reset.
6. EO Error Override- The override option, if permissible (a negative error code returned), is automatically chosen.
7. ER Error Retry- If Channel Command Retry (CCR) was requested, the retry will automatically be performed.
8. ES Error Sense- A sense will automatically be performed and displayed if Unit Check was contained in the last status.
9. EX Error Repeat- The retry option, if permissible and independent of whether Channel Command Retry was requested, is automatically chosen. This option generally is not recommended, as the probable result is an infinite loop.
10. EF Error Fail- The fail option is automatically chosen.

The error recovery options have the same priority as they appear in the list on page 4 -13. For example, if the options /EO /ER /ES are given, the program will first check if an override is possible. If an override is not possible, or the the override results in another error, the program then checks if a command retry was requested. If command retry was not requested, the program then checks to see if Unit Check was contained in the status. If Unit Check was contained in the status, a sense is performed. If the error has not been corrected by an override or retry, the user will be prompted for the correct action. Always specifying the option /EF will guarantee that that no operator intervention is required, but then the program must be set up to handle all possible errors.

## 4.4: PROGRAMMING IN THE PROGRAM EDIT MODE

### 4.4.1 Program Interpreter

Every operation that can be performed in the supervisor can be performed in the program interpreter. And every operation that can be performed in the program interpreter can be performed in the supervisor. It is useful to read the preceding section on the supervisor to understand the use of certain program interpreter commands.

#### 4.4.1a Special characters

Colon(:)- Indicates that the preceding text on that line is a line tag. The tag can be used as the address in a goto or gosub statement.
Pound Sign(#)- Indicates that the text between it and the next carriage return is a comment.
Backslash(\)- Indicates the following constant is hexadecimal, or in certain commands (print and input) signals a format specification.
Single quote(')- Single quotes around a character indicates that the EBCDIC value for that character is to substituted for that character, e.g. 'A' is the same as 193.
Double quotes(")- Double quotes indicates a literal string, e.g. array := "Hello, World!" assigns a literal string to the variable called array, including a terminating null character.

#### 4.4.1b Operators, level of precedence, and associativity

| Level of precedence | Type of Operation | Associativity |
|---|---|---|
| 1. () | Association | Left to right |
| 2. ˜ ! + - | Complement, Logical not, Unary plus/minus | Right to left |
| 3. * / | Multiplication, Division | Left to right |
| 4. + - | Addition, Subtraction | Left to right |
| 5. << >> | Shift left/right | Left to right |
| 6. < > <= > | Relational | Left to right |
| 7. = <> | Equality, Inequality | Left to right |
| 8. & | Bitwise and | Left to right |
| 9. ^ | Bitwise exclusive or | Left to right |
| 10. \| | Bitwise inclusive or | Left to right |
| 11. && | Logical(Boolean) and | Left to right |
| 12. \|\| | Logical(Boolean) or | Left to right |
| 13. := | Assignment | Right to left |

---

## 4.4.1c Variables and assignment

Variables may be declared anywhere in the program but must be declared before they are used. Variables must start with a letter and may include letters, numbers, and be underlined. One-dimensional arrays can be declared by putting the array size in brackets after the variable name. Subscripts are checked during the program execution to see if they are in the bounds of the array. The name of an array is equivalent to the zeroth member of the array. The effects of assigning a literal string to a member of an array is demonstrated by example below.

Example:

```
# Command           # Result

Arr    := "Eh"      # Arr[0] = 'E', Arr[1] = 'h', Arr[2] = 0
Arr[0] := "Eh"      # Arr[0] = 'E', Arr[1] = 'h', Arr[2] = 0
Arr[1] := "Eh"      # Arr[1] = 'E', Arr[2] = 'h', Arr[3] = 0
```

## 4.4.1d System variables

DATA- If data has been read over the channel into the data buffer, the data can then be read out by repeated use of this variable, thereby simulating a FIFO buffer. The variable contains the next value in the FIFO buffer. Any use of this variable advances the FIFO pointer to the next position. DATA equals -1 if the FIFO buffer is empty.
COMPARE- specifies the result of the last COMPAREFILE or COMPARERANDOM command. If compare = 1, then the comparison was successful. If compare = -1, then the comparison couldn't be done. This is because there wasn't any data read or a file error occurred. If compare = 0, then the comparison was unsuccessful.
EOF- If the pointer to the currently open input file is at end-of-file, EOF is true(non-zero). See the file manipulation commands found in section 4.3.5.
ADDRESSIN- Contains the last address received over the channel.
BUSIN- Contains the value of the busin lines.
BUSOUT- Contains the value of the busout lines.
DATACOUNT- Contains the current data count. This will either be equal to the number of bytes that were received over the channel, or it will be the number of bytes yet waiting to be sent over the channel.
MK0- Contains the value of Mark0 In.
STATUSIN- Contains the last status received over the channel.
STOPCODE- Contains the error code for the last sequence. A list of possible stop codes is given in Appendix C.
TAGIN- Contains the value of the tagin lines.
TAGOUT- Contains the value of the tagout lines.
FAILED- Failed is equal to one if the "FAIL" option is chosen in response to an error.
SENSEBYTES- Equal to the number of sense bytes received during the last sense sequence.
SENSEBUF[32]- An array containing the sense bytes received during the last sense sequence.
COMPARE- Specifies the result of the last compare operation. See the comparefile and comparerandom functions.

The order of bits in the tagin, tagout, and status variables is as follows:

| | Tagin lines | | Tagout lines | | Status | |
|---|---|---|---|---|---|---|
| SLI | Select In | SPO | Suppress Out | ATN | Attention | (MSB) |
| RQI | Request In | SLO | Select Out | MOD | Status Modifier | |
| OPI | Operational In | OPO | Operational Out | CUE | Control Unit End | |
| DSI | Disconnect In | HLO | Hold Out | BSY | Busy | |
| SVI | Service In | SVO | Service Out | CHE | Channel End | |
| STI | Status In | DTO | Data Out | DVE | Device End | |
| DTI | Data In | CDO | Command Out | CHK | Unit Check | |
| ADI | Address In | ADO | Address Out | EXC | Unit Exception | (LSB) |

The mnemonics in the table above can also be used in expressions, e.g. SLI equals one if the Select In line is high and zero if the Select In line is low.

### 4.4.2 Commands

#### 4.4.2a Variable declaration

Command: VAR (variable name) [, ...]
Description: Declares variables used in the program. These variables will be four-byte signed quantities. These statements may appear anywhere in the program. Variables must start with a letter and may include letters, numbers, and the underline character. One-dimensional arrays can be declared by putting the array size in brackets after the variable name.
Example: VAR a,A3,B_39r,Array[10]

Command: STRING (variable name) [, ...]
Description: Like VAR, only the variables will be unsigned, single byte quantities. To avoid problems with arithmetic conversions, generally only literal strings should be declared this way. If a variable declared as STRING is set equal to variable declared as VAR, the value is truncated and sign information will be lost.

---

4.4.2b Data RAM storage control

Command: LOADFILE (file name) [/t]
Description: Loads a data file into the RAM buffer. If no extension is given, a default extension of "ram" is assumed. An error will be generated if the file cannot be opened. If the "/t" option is used, an ASCII-to-EBCDIC conversion will be performed. Any data already in the buffer is overwritten.
Example: LOADFILE "a:\data\testdata.ram" /t

Command: STOREFILE (file name) [/t]
Description: Stores the data in the RAM buffer onto a disk file. If no extension is given, a default extension of "ram" is assumed. An error will be generated if the file cannot be opened. If the "/t" option is used, an EBCDIC-to-ASCII conversion will be performed.
Example: STOREFILE "a:\data\testdata.ram" /t

Command: DATA (string) | (mnemonic) | (value) [, ...]
Description: Loads data into the data ram buffer. This data will be appended to any data loaded with a loadfile command or with a prior data command. If the data in the buffer is due to input over the channel, it will be overwritten. An ASCII string will be converted to EBCDIC. As in the example, mnemonics can be used for control characters. For a list of control character mnemonics, see Appendix A.
Example: DATA "Hello",CR,\0A

Command: CLEARBUFFER
Description: Clears the data ram buffer.

Command: RESTORE
Description: After data has been written out to the channel, the data in the buffer can be restored with this command, to be written out again. This command will only work if there has been no intervening channel read command executed.

Command: LOADRANDOM (#bytes) (seed)
Description: Loads the specified number of psuedo-random bytes into the data RAM buffer. The seed can be any number to 64K and will always replicate the same series.

---

Command: COMPAREFILE (filename) [/t]
Description: After a read operation, compares the current data in the buffer to the contents of the specified file. The result of the comparison can be determined by testing the system variable compare.

```
  compare = 1 if successful comparison
            0 not successful
           -1 if no data had previously been read into the buffer
              or if the file couldn't be found
```

Command: COMPARERANDOM (#bytes) (seed)
Description: After a read operation, compares the current data in the buffer to the specified psuedo-random sequence. The result of the comparison can be determined by testing the system variable compare.

```
  compare = 1 if successful comparison
            0 not successful
           -1 if no data had previously been read into the buffer
```

## 4.4.2c Configuration and simulator control

Command: CHANNEL BYTEMUX | BLOCKMUX | SELECTOR
Description: Sets the channel configuration to either bytemux, blockmux, or selector.

Command: TIMEOUT (timeout value in seconds)
Description: Defines the timeout value for any sequence. The default value is 30 seconds.

Command: BUFSIZE (size of buffer up to 64K-1)
Description: Defines the size of the data buffer. The default size is 64K, which is the maximum size possible.

### 4.4.2d Channel sequences- group I

These sequences will not return until the peripheral disconnects, unless there is an error. The corresponding stopcodes are given for each sequence. An explanation of each stopcode number is given in Appendix C.

Command: CCW0 (address) (command) (options)
Description: Executes a Channel Command Word (CCW). The function will return when error status or device end from the selected address is presented. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,16,-2,-3,-4,-5,-6,-9,-10,-11.

Command: CCW1 (address) (command) (options)
Description: Executes a CCW. The difference between this function and CCW0 is that this function will return when channel end only is presented if not command chaining. If chaining commands, CCW1 will wait for device end. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,16,-2,-3,-4,-5,-6,-9,-10,-11.

Command: Initial (address) (command) (options)
Description: Executes a initial selection sequence. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,-1,-2,-3,-5,-6,-9,-10,-11.

Command: Request (options)
Description: Executes a request sequence. If Request In is not high, the function will wait. The possible stopcodes are 0,2,12,13,14,15,-4,-9,-10,-11.

Command: System_Reset
Description: Executes a system reset. A stopcode isn't applicable.

Command: Selective_Reset
Description: Executes a selective reset. Generally, it is only useful if Operational In is high. A stopcode isn't applicable.

Command: Interface_Disconnect (address) (options)
Description: Attempts to connect to the specified peripheral in order to do a interface disconnect sequence. The possible stopcodes are 0,1,2,4,-6.

Command: Sense (address)
Description: Executes a sense command, storing the result in the system array sensebuf. The number of sense bytes received is stored in the system variable sensebytes. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,-2,-3,-4,-5,-6,-9,-10,-11.

Command: Override
Description: If a negative return code has been received for a channel sequence, doing an override will ignore the error and continue from where the sequence left off. If the sequence was complete, the return code will be translated to what it would have been if no error had happened.

Command: Retry
Description: If the last sequence was a CCW, a retry will cause it to be executed over. It is especially useful if a channel command retry was signalled from the peripheral. Before doing the retry, the data will be restored. The possible stopcodes are 0,2,3,4,5,6,7,8,9,12,13,14,15,-2,-3,-4,-5,-6,-9,-10,-11.

---

4.4.2e Channel sequences- group II

These sequences directly drive the bus and tag lines.

Command: TAGRESET
Description: Lowers all the out tags.

Command: LOWER (busout line mnemonic)
Description: Lowers the specified busout line in the PC busout latch. The permissible mnemonics(defined above) are SPO, SLO, OPO, HLO, SVO, DTO, CDO, and ADO.

Command: RAISE (busout line mnemonic)
Description: Raises the specified busout line in the PC busout latch. The permissible mnemonics(defined above) are SPO, SLO, OPO, HLO, SVO, DTO, CDO, and ADO.

Command: ADDRESSOUT (address value) (options)
Description: Puts the specified value on busout

Command: COMMANDOUT (address value) (options)
Description: Puts the specified value on busout

Command: DATAOUT (address value) (options)
Description: Puts the specifies value on busout

Command: DISABLE_BUSOUT
Description: Enables or disables busout. If disabled, all busout lines will be 0.

## 4.4.2f Input/Output

Command: PRINT (expression) | (string) [\(format)] [,...] | [;...]

Description: Displays on the output screen literal strings and variables. A comma between values causes a tab to be printed. A semicolon between values will not cause any space to be printed. If a comma or semicolon end the command statement, the next print statement will print on the same line. Otherwise, a return is generated. The permissible format specifications are:

\d decimal- output is printed as a decimal number.
\h hexadecimal- output is printed as a hexadecimal number.
\c character- output is an EBCDIC character.
\s string- array is printed as a string.

The default format is decimal. A field width may be inserted between the backslash and the format character. If the result is longer than the field width, the output is truncated. If the result is shorter, the output is left justified for the \s format, and right justified for the other types.

Example:
```
PRINT x\2h,"squared is ",x * x\4h
PRINT                           #print extra linefeed
```

Command: INPUT [prompt,] (variable name) [\(format)]

Description: Receives input from the keyboard. First the optional prompt message is displayed. A '?' will be displayed if no prompt was specified. The permissible format specifications are:

\d decimal- input is interpreted as a decimal number.
\h hexadecimal- input is interpreted as a hexadecimal number.
\c character- the EBCDIC value of the first character input is stored in the specified variable.
\s string- the input string is stored in the specified array and is truncated if necessary.

The default format is decimal. A field width may be inserted between the backslash and the format character. For the \c and \s formats, this is the maximum number of characters that will be stored in the specified array. With the \s format, a NULL (zero) will also be stored. Field width has no meaning for the other formats. For the \d and \h formats, if the input can not completely be converted correctly, the user is asked to reinput the value.

Example:
```
INPUT "Input hexadecimal value for a:  ",a\h
```

### 4.4.2g File manipulation

Command: OPEN (filename) [/t]
Description: Opens a data file for reading. Only one data file can be open at a time. Opening a new file will automatically close any prior file opened. Data is accessed from the file with the read command, and end-of-file is detected via the EOF system variable. There is a default extension of "ram". If the /t option is specified, an ASCII-to-EBCDIC conversion will be done whenever the file is read.
Example: OPEN "data.ram" /t

Command: READ (variable) [\(format)]
Description: Reads the currently open input file. If the file was opened with the /t option, an ASCII-to-EBCDIC conversion is done. The format specification works similarly to how it works in the input command. The only permissible format specifications in this case are \c or \s, again with an optional field width specification. An attempt to read a string at end-of-file will truncate the string. If end-of-file occurred before the read, a NULL string(first array member zero) will therefore be assigned. An attempt to read a character at end-of-file will result in -1 being assigned to the variable. End-of-file can be verified with the EOF system variable.

Command: REWIND
Description: Rewinds the currently open data file to the beginning.

4.4.2h Program flow
Command: GOTO (linetag)
Description: Branches unconditionally to the line defined by linetag.
Example:

```
a3: print "Here we go again"
  |
  |
GOTO a3
```

Command: GOSUB (linetag)
Command: RETURN
Description: Branches unconditionally to and from a subroutine. Gosub's can be nested ten deep.
Example:

```
GOSUB message
  |
  |
message: print "Hi"
RETURN
```

Command: IF (expression1)
Command: ELSEIF(expression2)
Command: ELSE
Command: ENDIF
Description: If expression1 is true(nonzero), the statements between if and elseif are executed. Otherwise, if expression2 is true, the statements between elseif and else and executed. Otherwise, the statements between else and endif are executed. If statements may be nested.

Example:

```
IF (status)
  print "nonzero hexadecimal status of ",status\h
  |
  |
ENDIF
```

Command: WHILE (expression)
Command: ENDWHILE
Description: The statements between while and endwhile are repeatedly executed while expression is true(nonzero). While statements may be nested.
Example:

```
A := 5          #Loop five times
WHILE (A)
  |
  |
  A := A - 1
ENDWHILE
```

Command: FOR (variable) := (startvalue) TO (endvalue)
Command: ENDFOR
Description: if startvalue <= endvalue, program lines following the for statement are executed until the endfor statement is encountered. Then the count variable is incremented by one and the process is repeated. FOR statements may be nested.
Example:

```
FOR A := 1 TO 5  #Loop five times
  print "pass # ",A
    |
    |
ENDFOR
```

Command: END
Description: Ends program execution. This command does not have to be the last one in the program. It can appear anywhere in the program or not at all.

## 4.5: THE 'C' LANGUAGE LIBRARIES

### 4.5.1 Introduction

There are ten libraries provided. There are five memory models for both Turbo and Microsoft C:

|  | Library Name |  |
|---|---|---|
| Memory Model | Microsoft | Turbo |
| COMPACT | CS_MS_C.LIB | CS_TC_C.LIB |
| SMALL | CS_MS_S.LIB | CS_TC_S.LIB |
| MEDIUM | CS_MS_M.LIB | CS_TC_M.LIB |
| LARGE | CS_MS_L.LIB | CS_TC_L.LIB |
| HUGE | CS_MS_H.LIB | CS_TC_H.LIB |

Declarations for all the functions in the library plus all definitions referenced below are contained in the header file <act.h>.

The functions in the library are described in the following sections. The most important thing to note is that the function "initialize" must be called before any other in order for the other functions to properly work.

In the function descriptions below, constants that are in all capital letters are definitions found in the header file <act.h>.

## 4.5.2 System Variables

The following variables are declared external in the header file <act.h> and reference variables in the simulator library:

```
extern char _stopcode_;
/* the last error code returned from a sequence */
(A list of possible stopcodes is given in Appendix C.)

extern unsigned char _statusin_;
/* the last status received over the channel */

extern unsigned char _addressin_;
/* the last address received over the channel */

extern unsigned char sensebuf[];
/* contains sense bytes returned from the last sense function */

extern int sensebytes;
/* number of sense bytes returned from the last sense function */

extern unsigned char atoe[];
/* conversion array to convert from ASCII-to-EBCDIC */
/* example: ebcdic_a = atoe['a']; */

extern unsigned char etoa[];
/* conversion array to convert from EBCDIC-to-ASCII */

extern char *posmsg[];
/* array of error messages corresponding to an error code >= 0 */
/* example: printf("%s",posmsg[_stopcode_]); */

extern char *negmsg[];
/* array of error messages corresponding to an error code <= 0 */
/* example: printf("%s", negmsg[-_stopcode_]); */

extern int_failed_;
/* It is equal to one if the "FAIL" option is chosen in */
/* response to an error */
```

Each status bit has an associated "mask" defined to aid in analyzing its status:

| Mask | Status bit | |
|---|---|---|
| _ATN | Attention | (MSB) |
| _MOD | Status Modifier | |
| _CUE | Control Unit End | |
| _BSY | Busy | |
| _CHE | Channel End | |
| _DVE | Device End | |
| _CHK | Unit Check | |
| _EXC | Unit Exception | (LSB) |

For example, _CHE in binary would be 0000 1000 and can be used to isolate the channel end bit from the status. Here are two possible ways to use the mask:

```
  if (_statusin_&_CHE)   /*test for channel end*/
  if (_statusin_= =_CHE) /*test for channel end only*/
```

### 4.5.3 Simulator Control Functions

initialize
  Usage: void initialize()
  Description: Initializes the simulator operation. This function should be invoked at the beginning of every program.

timeout
  Usage: void timeout(unsigned int seconds)
  Description: Defines the timeout value for any sequence. The default value is 30 seconds.

bufsize
  Usage: void bufsize(unsigned int)
  Description: Defines the size of the data buffer. The default size is 64K-1, which is the maximum size possible.

channel
  Usage: void channel(state)
  Description: Defines what type channel the simulator is on. Permissible values of state are SELECTOR, BYTEMUX, and BLOCKMUX.

## 4.5.4 Ram Control Functions

In the following functions, the permissible values of modeflg are BINARY or TRANSLATED.
modeflg = BINARY indicates an untranslated mode.
modeflg = TRANSLATED indicates a translated mode.
ASCII is converted to EBCDIC, or vice-versa, as appropriate. There is an unique EBCDIC character assigned to every non-extended ASCII character, so an ASCII-to-EBCDIC-to-ASCII conversion sequence will generate the original character for all non-extended ASCII characters.

<u>clrbuf</u>
  Usage: void clrbuf()
  Description: Clears data from the RAM buffer. This function should be invoked before the ldbyte or ldstr functions if the old data in the RAM buffer is to be overwritten.

<u>datacount</u>
  Usage: unsigned int datacount()
  Return Value: returns the number of bytes in the data buffer, which will be either the number of bytes just read in or the number of bytes waiting to be read out.

<u>lddata</u>
  Usage: unsigned int lddata(char *str,int modeflg)
  Description: Loads a character string into the data RAM buffer, not including the terminating null character. The data is appended to the data already residing in the buffer.

<u>ldbyte</u>
  Usage: unsigned int ldbyte(char c,int modeflg)
  Description: Loads a byte of data into the data RAM buffer. The byte is appended to the data already residing in the buffer.
  Return Value:
    0 if the RAM buffer is full.
    1 if the byte was successfully loaded.
  Examples: ldbyte('a',TRANSLATED);
            ldbyte(3,BINARY);

<u>ldstr</u>
  Usage: unsigned int ldstr(char *str)
  Description: Translates a string to EBCDIC and loads it into the data RAM buffer, not including the terminating null character. The string is appended to the data already residing in the buffer.
  Return Value: Returns the number of bytes loaded.
  Example: ldstr("This is a test");

<u>lddatax</u>
  Usage: unsigned int lddatax(char *str,int modeflg,unsigned int repeat)
  Description: Like lddata, only the character pattern is repeated the specified number of times.

<u>ldbytex</u>
  Usage: unsigned int ldbytex(char c,int modeflg,unsigned int bytes)
  Description: Like ldbyte, only the character is repeated the specified number of times.

<u>ldstrx</u>
  Usage: unsigned int ldstrx(char *str,unsigned int repeat)
  Description: Like ldstr, only the character pattern is repeated the specified number of times.

<u>ldfile</u>
  Usage: long ldfile(char *filename,int modeflg)
  Description: Loads a disk file into the data RAM buffer.
  Return Value: Returns the number of bytes loaded.
    -1 indicates the file couldn't be opened.
  Example: ldfile("testdata.dat",BINARY);

<u>ldrand</u>
  Usage: unsigned int ldrand(unsigned int bytes,int seed)
  Description: Loads the specified number of pseudo-random bytes into the data ram buffer. The same seed will always replicate the same series.
  Return Value: Returns the number of bytes loaded.

<u>cmpfile</u>
  Usage: int cmpfile(char *filename, int modeflg)
  Description: After a read operation, compares the current data in the buffer to the contents of the specified file.
  Returns: 1 if successful comparison
    0 not successful
    -1 if no data had previouly been read into the buffer or if the file couldn't be opened.

<u>cmprand</u>
  Usage: int cmprand(unsigned int bytes,int seed)
  Description: After a read operation, compares the current data in the buffer to the specified pseudo-random sequence.
  Returns: 1 if successful comparison
    0 not successful
    -1 if no data had previouly been read into the buffer

<u>rdbyte</u>
  Usage: int rdbyte(int modeflg)
  Description: This function simulates reading from a FIFO buffer. After data has been input over the channel, the data can be read by repeated use of this function.
  Return Value: EOF if the FIFO buffer is empty,
    else returns the byte read.

<u>stfile</u>

```
  Usage:  long stfile(char *fstr,int modeflg)
  Description:  Stores the RAM buffer data into a disk file.
  Return Value:  0 if the file couldn't be opened.
                 1 if the data successfully stored.
  Example:  stfile("chandata.dat",TEXT);
```

<u>restore</u>

```
  Usage:  void restore(void)
```

Description: After data has been written out to the channel, this function will reset the data RAM pointer such that the data can be written out again.

## 4.5.5 Input Functions

<u>rdbusin</u>
  Usage: unsigned char rdbusin()
  Return Value: Returns the current value of busin.

<u>rdbusout</u>
  Usage: unsigned char rdbusout()
  Return Value: Returns the current value of busout.

<u>rdtagin</u>
  Usage: unsigned char rdtagin()
  Return Value: Returns the current tagin byte.

<u>rdtagout</u>
  Usage: unsigned char rdtagout()
  Return Value: Returns the current tagout byte.

<u>rdmko</u>
  Usage: unsigned char rdmk0()
  Return Value: Returns the value of the Mark Zero In Line.

To aid in testing the results of rdtagin and rdtagout each bit has an associated "mask" defined:

| Mask | Tagin line | Mask | Tagout line | |
|---|---|---|---|---|
| _SLI | Select In | _SPO | Suppress Out | (MSB) |
| _RQI | Request In | _SLO | Select Out | |
| _OPI | Operational In | _OPO | Operational Out | |
| _DSI | Disconnect In | _HLO | Hold Out | |
| _SVI | Service In | _SVO | Service Out | |
| _STI | Status In | _DTO | Data Out | |
| _DTI | Data In | _CDO | Command Out | |
| _ADI | Address In | _ADO | Address Out | (LSB) |

For example, _SVO in binary would be 00001000 and can be used to isolate the service out bit from the tagout byte.
Here are two possible ways to use the mask:

```
  if (rdtagout() & _SVO)    /* test for service out */
          |
  if (rdtagout() == _SVO)   /* test for service out only */
          |
```

## 4.5.5a Channel sequences- group I

These sequences will not return until the peripheral disconnects, unless there is an error. If the function returns a value, that value is a stopcode. Alternatively the system variable (_stopcode_) also contains the stopcode for the last sequence executed. An explanation of the stopcode numbers is given in Appendix C.

ccw0

Usage: int ccw0(unsigned char address, unsigned char command, unsigned int options)

Description: Executes a CCW. The function will return when error status or device end from the selected address is presented. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,16,-2,-3,-4,-5,-6,-9,-10,-11.

ccw1

Usage: int ccw1(unsigned char address, unsigned char command, unsigned int options)

Description: Executes a CCW. The difference between this function and CCW0 is that this function will return when channel end only is presented if not command chaining. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,16,-2,-3, -4,-5,-6,-9,-10,-11.

initial

Usage: int initial(unsigned char address, unsigned char command, unsigned int options)

Description: Executes a initial selection sequence. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,-1,-2,-3, -5,-6,-9,-10,-11.

request

Usage: int request(unsigned int options)

Description: Executes a request sequence. If Request In is not high, the function will wait. The possible stopcodes are 0,2,12,13,14,15,16,-4,-9,-10,-11.

system_reset

Usage: void system_reset()

Description: Execute a system reset. A stopcode is n't applicable.

selective_reset

Usage: void selective_reset()

Description: Executes a selective reset. Generally, it is only useful if Operational In is high. A stopcode isn't applicable.

<u>interface_disconnect</u>
Usage: int interface_disconnect(unsigned char address)
Description: Attempts to connect to the specified peripheral in order to do a interface disconnect sequence. The possible stopcodes are 0,1,2,4,-6.

<u>sense</u>
Usage: int sense(unsigned char address)
Description: Executes a sense command, storing the result in the system array sensebuf. The number of sense bytes received is stored in the system variable sensebytes. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,-2,-3,-4,-5,-6,-9,-10,-11.

<u>override</u>
Usage: int override()
Description: If a negative return code has been received for a channel sequence, doing an override will ignore the error and continue from where the sequence left off. If the sequence was complete, the return code will be translated to what it would have been if no error had happened.

<u>retry</u>
Usage: int retry()
Description: If the last sequence was a CCW, a retry will cause it to be executed over. Especially useful if a channel command retry was signalled from the peripheral. Before doing the retry, the data will be restored. The possible stopcodes are 0,2,4,5,6,7,8,9,12,13,14,15,-2,-3,-4,-5,-6,-9,
-10,-11.

## 4.5.5b Channel sequences- group II

These sequences directly drive the bus and tag lines.
If the function returns a value, it is the errorcode for that sequence, which could also be accessed through the _stopcode_ system variable.

tagreset
  Usage:  void tagreset()
  Description:  Lowers all the out tags.

lower
  Usage:  void lower(int tagno)
  Description:  Lowers the specified busout line in the PC busout latch. The permissible values of tagno are SPO, SLO, OPO, HLO, SVO, DTO, CDO, and ADO.

raise
  Usage:  void raise(int tagno)
  Description:  Raises the specified busout line in the PC busout latch. The permissible values of tagno are SPO, SLO, OPO, HLO, SVO, DTO, CDO, and ADO.

addressout
  Usage:  void addressout(unsigned char value, unsigned int options)
  Description:  Puts the specified value on busout

commandout
  Usage:  void commandout(unsigned char value, unsigned int options)
  Description:  Puts the specified value on busout

dataout
  Usage:  void dataout(unsigned char value, unsigned int options)
  Description:  Puts the specified value on busout

disable_busout
  Usage:  void disable_busout()
  Description:  Disables busout.

## SECTION 5: MAINTENANCE AND WARRANTY

### 5.1: LIMITED WARRANTY

Technology 80 Inc. warrants the ChannelACT to be free from defects in workmanship and materials under normal, intended use and service in its original, unmodified condition (unless such modifications are made by Technology 80 Inc.) for the appropriate period, set forth below from the date of delivery to the purchaser. If any section of the ChannelACT is found defective within the terms of this warranty, the sole responsibility of Technology 80 Inc. shall be to repair, or at its option to replace, such defective section, provided further that the ChannelACT is returned to a repair depot designated by Technology 80 Inc. with transportation charges prepaid by the purchaser. Return shipping charges will be billed to the purchaser or shipment made collect at the purchaser's option. Technology 80 Inc. assumes no responsibility for damage in shipment and any insurance charges covering such possible damage must be paid by the purchaser. Any charges for customs clearance, or other related charges, are excluded from warranty coverage and are to be paid by the purchaser. All replaced parts, components or materials shall become the property of Technology 80 Inc.

If Technology 80 Inc. determines in its sole judgement that the ChannelACT is not defective within the terms of the warranty, the purchaser shall pay Technology 80 Inc. all costs of handling, transportation and testing at the prevailing Technology 80 Inc. rates. If the unit is determined to be defective from causes not covered by this warranty, the purchaser shall be so notified and instructions obtained as to its desired disposition. If the purchaser requests that repairs of such defects be made by Technology 80 Inc., such repairs shall be performed pursuant to the Technology 80 Inc. Service Policy. Damage resulting from moving portable units is not covered by warranty.

The components of the ChannelACT and their respective warranty periods are as follows:

| Section Covered | Warranty Period |
|---|---|
| Circuit boards, cabinet, card cage | one (1) year |
| Tape drive, display, keyboard, external connectors | ninety (90) days |
| Software, firmware | one (1) year |

Software or firmware warranty runs only to the magnetic media or semiconductor chip on which such software or firmware is recorded. The program itself is warranted only for the version included at delivery, and Technology 80 Inc. has no obligation under this warranty to replace such program with later versions.

All the above warranties are contingent upon proper use of the ChannelACT. These warranties will not apply: (i) if adjustment, repair or parts replacement is required because of accident; unusual physical, electrical or electro-magnetic stress; neglect; misuse; failure of electric power, air conditioning, or humidity control; transportation; failure of rotating media not furnished by Technology 80 Inc.; operation with media not meeting or not maintained in accordance with Technology 80 Inc. specification; or use or operation other than in the manner for which it was designed or intended; or (ii) if the ChannelACT has been modified by the purchaser or user; or (iii) where Technology 80 Inc. serial numbers or warranty date decals have been removed or altered. ChannelACT may contain used parts that are equivalent to new in performance, when used in the ChannelACT.

EXCEPT FOR THE EXPRESS WARRANTIES STATED HEREIN, TECHNOLOGY 80 INC. DISCLAIMS ALL WARRANTIES ON PRODUCTS SUBJECT HERETO, INCLUDING WITHOUT LIMITATION ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. THE LIABILITY OF TECHNOLOGY 80 FOR DEFECTS ON THE ChannelACT IS LIMITED TO THE EXPRESS WARRANTIES SET FORTH HEREIN, AND SUCH EXPRESS WARRANTIES ARE IN LIEU OF ANY AND ALL OTHER WARRANTIES, OBLIGATIONS OR LIABILITIES ON THE PART OF TECHNOLOGY 80 INC. FOR A DEFECTIVE PRODUCT OR ARISING OUT OF, OR IN CONNECTION WITH, THE PERFORMANCE OF THE PRODUCT SUBJECT HERETO. IN NO EVENT WILL TECHNOLOGY 80 BE LIABLE FOR: (A) SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OR (B) ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, ARISING OUT OF OR IN CONNECTION WITH THIS CONTRACT OR THE USE OR PERFORMANCE OF TECHNOLOGY 80 INC. PRODUCTS, WHETHER IN AN ACTION OF CONTRACT OR TORT, INCLUDING NEGLIGENCE. THE LIABILITY OF TECHNOLOGY INC. FOR DAMAGE TO PROPERTY SHALL BE LIMITED TO PHYSICAL DAMAGE DIRECTLY CAUSED BY THE SOLE NEGLIGENCE OF TECHNOLOGY 80 INC. AND SHALL IN NO EVENT EXCEED TEN THOUSAND DOLLARS ($10,000.00). IN ADDITION TO THE FOREGOING, TECHNOLOGY 80 INC. WILL NOT BE LIABLE FOR THE UNAUTHORIZED USE OF INFORMATION OBTAINED FROM THE ChannelACT, INCLUDING WITHOUT LIMITATION THE UNAUTHORIZED USE OF THE PURCHASER'S OR USER'S ACCESS CODE. No Technology 80 Inc. employees, agents, or representatives, nor any other persons, are authorized to assume or agree to any other warranties or liabilities binding on Technology 80 Inc.

## 5.2: SERVICE POLICY

Before returning a unit for repair, call the factory for assistance in verifying that the unit is defective. If it is determined that the unit should be returned, a Return Material Authorization (RMA) number will be assigned. Carefully package the unit and ship prepaid (insured suggested) to the point designated by the factory. Include with the unit a short statement of the malfunction along with a reference to the RMA number. Include the name, address and phone number of a technical person who may be contacted in the event additional information is needed. If the unit is out of warranty or the unit is damaged outside of warranty, also include a properly executed purchase order. If an estimate or repair charge is desired before work is done, so state on the purchase order form. Under no circumstances return any item with freight collect as it will not be accepted. Be sure the unit is packaged properly. Technology 80 Inc. will not be responsible for damage due to improper packaging of items returned for service or repair.

## APPENDIX A: EBCDIC CONTROL CHARACTER MNEMONICS

| Hex | Decimal | Mnemonic | Hex | Decimal | Mnemonic |
|---|---|---|---|---|---|
| 00 | 0 | NUL | 22 | 34 | FS |
| 01 | 1 | SOH | 23 | 35 | WUS |
| 02 | 2 | STX | 24 | 36 | BYP |
| 03 | 3 | ETX | 25 | 37 | LF |
| 04 | 4 | SEL | 26 | 38 | ETB |
| 05 | 5 | HT | 27 | 39 | ESC |
| 06 | 6 | RNL | 28 | 40 | SA |
| 07 | 7 | DEL | 29 | 41 | SFE |
| 08 | 8 | GE | 2A | 42 | SM |
| 09 | 9 | SPS | 2B | 43 | CSP |
| 0A | 10 | RPT | 2C | 44 | MFA |
| 0B | 11 | VT | 2D | 45 | ENQ |
| 0C | 12 | FF | 2E | 46 | ACK |
| 0D | 13 | CR | 2F | 47 | BEL |
| 0E | 14 | SO | 30 | 48 | |
| 0F | 15 | SI | 31 | 49 | |
| 10 | 16 | DLE | 32 | 50 | SYN |
| 11 | 17 | DC1 | 33 | 51 | IR |
| 12 | 18 | DC2 | 34 | 52 | PP |
| 13 | 19 | DC3 | 35 | 53 | TRN |
| 14 | 20 | RES | 36 | 54 | NBS |
| 15 | 21 | NL | 37 | 55 | EOT |
| 16 | 22 | BS | 38 | 56 | SBS |
| 17 | 23 | POC | 39 | 57 | IT |
| 18 | 24 | CAN | 3A | 58 | RFF |
| 19 | 25 | EM | 3B | 59 | CU3 |
| 1A | 26 | UBS | 3C | 60 | DC4 |
| 1B | 27 | CU1 | 3D | 61 | NAK |
| 1C | 28 | IFS | 3E | 62 | |
| 1D | 29 | IGS | 3F | 63 | SUB |
| 1E | 30 | IRS | 40 | 64 | SP |
| 1F | 31 | ITB | 41 | 65 | RSP |
| 20 | 32 | DS | 42 | 66 | |
| 21 | 33 | SOS | | | |

## APPENDIX B: ASCII/EBCDIC CONVERSIONS

The following rules are applied to conversions:
(1) A unique EBCDIC character is assigned to every non-extended ASCII character.
(2) An ASCII-to-EBCDIC-to-ASCII conversion done on non-extended ASCII characters will yield the original character.
(3) Every extended ASCII character converts to NUL(value 0).
(4) Every printable EBCDIC character converts to its closest ASCII equivalent, within the limits of rule #2.
(5) Control characters are converted to their equivalent if there is one, otherwise they are converted to NUL(value 0).

ASCII-to-EBCDIC conversion matrix

The ASCII value of the character is read as a hexadecimal number. The first hex digit is read from the left column. The second hex digit is read across the top. The table then gives the converted EBCDIC value in hex. If the entry is blank, the character is converted to a NUL(value 0).

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | 01 | 02 | 03 | 37 | 2D | 2E | 2F | 16 | 05 | 25 | 0B | 0C | 0D | 0E | 0F |
| 1 | 10 | 11 | 12 | 13 | 3C | 3D | 32 | 26 | 18 | 19 | 3F | 27 | 22 | 1D | 1E | 1F |
| 2 | 40 | 5A | 7F | 7B | 5B | 6C | 50 | 7D | 4D | 5D | 5C | 4E | 6B | 60 | 4B | 61 |
| 3 | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | 7A | 5E | 4C | 7E | 6E | 6F |
| 4 | 7C | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | D1 | D2 | D3 | D4 | D5 | D6 |
| 5 | D7 | D8 | D9 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | AD | E0 | BD | 4F | 6D |
| 6 | 5F | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 91 | 92 | 93 | 94 | 95 | 96 |
| 7 | 97 | 98 | 99 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | C0 | 6A | D0 | A1 | 07 |
| 8 | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | |
| A | | | | | | | | | | | | | | | | |
| B | | | | | | | | | | | | | | | | |
| C | | | | | | | | | | | | | | | | |
| D | | | | | | | | | | | | | | | | |
| E | | | | | | | | | | | | | | | | |
| F | | | | | | | | | | | | | | | | |

## EBCDIC-to-ASCII conversion matrix

The EBCDIC value of the character is read as a hexadecimal number. The first hex digit is read from the left column. The second hex digit is read across the top. The table then gives the converted ASCII result. If possible, the ASCII character is printed, otherwise its value is shown in hexadecimal. If the entry is blank, the character is converted to a NUL(value 0).

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | 01 | 02 | 03 | | 09 | | 7F | | | | 0B | 0C | 0D | 0E | 0F |
| 1 | 10 | 11 | 12 | 13 | | | 08 | | 18 | 19 | | | | 1D | 1E | 1F |
| 2 | | | 1C | | | 0A | 17 | 1B | | | | | | 05 | 06 | 07 |
| 3 | | | 16 | | | | | 04 | | | | | 14 | 15 | | 1A |
| 4 | 20 | 20 | | | | | | | | | ¢ | . | < | ( | + | ˆ |
| 5 | & | | | | | | | | | | ! | $ | * | ) | ; | ` |
| 6 | - | / | | | | | | | | | \| | , | % | _ | > | ? |
| 7 | | | | | | | | | | | : | # | @ | ' | = | " |
| 8 | | a | b | c | d | e | f | g | h | i | | { | ≤ | ( | + | + |
| 9 | | j | k | l | m | n | o | p | q | r | | } | ╬ | ) | ± | █ |
| A | - | ˜ | s | t | u | v | w | x | y | z | | └ | ┌ | [ | ≥ | • |
| B | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | ┘ | ┐ | ] | ╪ | - |
| C | { | A | B | C | D | E | F | G | H | I | - | | | | | |
| D | } | J | K | L | M | N | O | P | Q | R | | | | | | |
| E | \ | 20 | S | T | U | V | W | X | Y | Z | | | | | | |
| F | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | | | |

## APPENDIX C: STOPCODES LIST

| | |
|---|---|
| 0 | Timeout or user break |
| 1 | Successful execution |
| 2 | Selection unsuccessful |
| 3 | Data transfer complete |
| 4 | Short Busy indication |
| 5 | Initial status presented; command not accepted |
| 6 | Initial status presented; command retry (immediate) requested |
| 7 | Initial status presented; command retry (non-immediate) requested |
| 8 | Initial status presented |
| 9 | Initial status stacked |
| 10 | Data transfer request |
| 11 | Data transfer stopped |
| 12 | Exceptional non-initial status presented |
| 13 | Non-initial status presented; command retry (immediate) requested |
| 14 | Non-initial status presented; command retry (non-immediate) requested |
| 15 | Non-initial status presented |
| 16 | Non-initial status stacked |

| | |
|---|---|
| 0 | Timeout or user break |
| -1 | Pending request when went to issue command |
| -2 | Incorrect address returned |
| -3 | Parity error in initial selection address |
| -4 | Parity error in polling address |
| -5 | Parity error in initial status |
| -6 | Parity error in short busy status |
| -7 | Parity error in data byte |
| -8 | Parity error in data block; ending status has not been presented |
| -9 | Parity error in data block; ending status has been presented |
| -10 | Parity error in ending status |
| -11 | Parity error in both data block and ending status |

# APPENDIX D: SERPENTINE CONNECTOR PIN ASSIGNMENTS

## Bus 0 (Light Gray or White)

```
       B                      G
  ++++++++++++           ++++++++++++
02            13       02            13
  ++++++++++++           ++++++++++++
       D                      J
```

| | B | | G |
|---|---|---|---|
| 02 | Shield | 02 | Shield |
| 03 | Bus 0 Out P | 03 | Bus 0 In P |
| 04 | Shield | 04 | Shield |
| 05 | Bus 0 Out 1 | 05 | Bus 0 In 1 |
| 06 | * | 06 | * |
| 07 | Shield | 07 | Shield |
| 08 | Bus 0 Out 3 | 08 | Bus 0 In 3 |
| 09 | Shield | 09 | Shield |
| 10 | Bus 0 Out 5 | 10 | Bus 0 In 5 |
| 11 | Shield for B06* | 11 | Shield for G06 |
| 12 | Bus 0 Out 7 | 12 | Bus 0 In 7 |
| 13 | Shield | 13 | Shield |

| | D | J | |
|---|---|---|---|
| 02 | * | 02 | * |
| 03 | Shield* | 03 | Shield* |
| 04 | Bus 0 Out 0 | 04 | Bus 0 In 0 |
| 05 | Shield | 05 | Shield |
| 06 | Bus 0 Out 2 | 06 | Bus 0 In 2 |
| 07 | Shield | 07 | Shield |
| 08 | Shield | 08 | Shield |
| 09 | Bus 0 Out 4 | 09 | Bus 0 In 4 |
| 10 | Shield | 10 | Shield |
| 11 | Bus 0 Out 6 | 11 | Bus 0 In 6 |
| 12 | Shield | 12 | Shield |
| 13 | Mark 0 Out | 13 | Mark 0 In |

* Reserved for future use

## Bus 0 (Dark Gray or Black)

```
       D                    J
  ++++++++++++         ++++++++++++
 02          13       02          13
  ++++++++++++         ++++++++++++
       B                    G
```

|    | D              |    | J              |
|----|----------------|----|----------------|
| 02 | *              | 02 | *              |
| 03 | Shield*        | 03 | Shield*        |
| 04 | Bus 0 Out 0    | 04 | Bus 0 In 0     |
| 05 | Shield         | 05 | Shield         |
| 06 | Bus 0 Out 2    | 06 | Bus 0 In 2     |
| 07 | Shield         | 07 | Shield         |
| 08 | Shield         | 08 | Shield         |
| 09 | Bus 0 Out 4    | 09 | Bus 0 In 4     |
| 10 | Shield         | 10 | Shield         |
| 11 | Bus 0 Out 6    | 11 | Bus 0 In 6     |
| 12 | Shield         | 12 | Shield         |
| 13 | Mark 0 Out     | 13 | Mark 0 In      |

|    | B                |    | G                |
|----|------------------|----|------------------|
| 02 | Shield           | 02 | Shield           |
| 03 | Bus 0 Out P      | 03 | Bus 0 In P       |
| 04 | Shield           | 04 | Shield           |
| 05 | Bus 0 Out 1      | 05 | Bus 0 In 1       |
| 06 | *                | 06 | *                |
| 07 | Shield           | 07 | Shield           |
| 08 | Bus 0 Out 3      | 08 | Bus 0 In 3       |
| 09 | Shield           | 09 | Shield           |
| 10 | Bus 0 Out 5      | 10 | Bus 0 In 5       |
| 11 | Shield for B06*  | 11 | Shield for G06*  |
| 12 | Bus 0 Out 7      | 12 | Bus 0 In 7       |
| 13 | Shield           | 13 | Shield           |

* Reserved for future use

## Appendix D: Serpentine Connector Pin Assignments

**(Light Gray or White)**

```
      B                    G
 ++++++++++++         ++++++++++++
02          13       02          13
 ++++++++++++         ++++++++++++
      D                    J
```

|  | B |  | G |
|---|---|---|---|
| 02 | Shield | 02 | Shield |
| 03 | Operational In | 03 | Clock Out |
| 04 | Shield | 04 | Shield |
| 05 | Address In | 05 | Metering In |
| 06 | * | 06 | * |
| 07 | Shield | 07 | Shield |
| 08 | Select In | 08 | Data In |
| 09 | Shield | 09 | Shield |
| 10 | Address Out | 10 | Data Out |
| 11 | Shield for B06* | 11 | Shield for G06 |
| 12 | Suppress Out | 12 | Hold Out |
| 13 | Shield | 13 | Shield |

|  | D |  | J |
|---|---|---|---|
| 02 | * | 02 | * |
| 03 | Shield* | 03 | Shield |
| 04 | Status In | 04 | Metering Out |
| 05 | Shield | 05 | Shield |
| 06 | Service In | 06 | Request In |
| 07 | Shield | 07 | Shield |
| 08 | Shield | 08 | Shield |
| 09 | Select Out | 09 | (Special Use) |
| 10 | Shield | 10 | Shield |
| 11 | Command Out | 11 | Disconnect In |
| 12 | Shield | 12 | Shield |
| 13 | Service Out | 13 | Operational Out |

* Reserved for future use

## Tag (Dark Gray or Black)

```
        D                      J
  ++++++++++++           ++++++++++++
02            13       02            13
  ++++++++++++           ++++++++++++
        B                      G
```

| | D | | J |
|---|---|---|---|
| 02 | * | 02 | * |
| 03 | Shield* | 03 | Shield* |
| 04 | Status In | 04 | Metering Out |
| 05 | Shield | 05 | Shield |
| 06 | Service In | 06 | Request In |
| 07 | Shield | 07 | Shield |
| 08 | Shield | 08 | Shield |
| 09 | Select Out | 09 | (Special Use) |
| 10 | Shield | 10 | Shield |
| 11 | Command Out | 11 | Disconnect In |
| 12 | Shield | 12 | Shield |
| 13 | Service Out | 13 | Operational Out |
| | **B** | | **G** |
| 02 | Shield | 02 | Shield |
| 03 | Operational In | 03 | Clock Out |
| 04 | Shield | 04 | Shield |
| 05 | Address In | 05 | Metering In |
| 06 | * | 06 | * |
| 07 | Shield | 07 | Shield |
| 08 | Select In | 08 | Data In |
| 09 | Shield | 09 | Shield |
| 10 | Address Out | 10 | Data Out |
| 11 | Shield for B06* | 11 | Shield for G06* |
| 12 | Suppress Out | 12 | Hold Out |
| 13 | Shield | 13 | Shield |

* Reserved for future use

# Appendix D: Serpentine Connector Pin Assignments

**Bus 1 (Light Gray or White)**

```
      B                  G
 ++++++++++++       ++++++++++++
02          13     02          13
 ++++++++++++       ++++++++++++
      D                  J
```

|  | B |  | G |
|---|---|---|---|
| 02 | Shield | 02 | Shield |
| 03 | Bus 1 Out P | 03 | Bus 1 In P |
| 04 | Shield | 04 | Shield |
| 05 | Bus 1 Out 1 | 05 | Bus 1 In 1 |
| 06 | * | 06 | * |
| 07 | Shield | 07 | Shield |
| 08 | Bus 1 Out 3 | 08 | Bus 1 In 3 |
| 09 | Shield | 09 | Shield |
| 10 | Bus 1 Out 5 | 10 | Bus 1 In 5 |
| 11 | Shield for B06* | 11 | Shield for G06 |
| 12 | Bus 1 Out 7 | 12 | Bus 1 In 7 |
| 13 | Shield | 13 | Shield |

|  | B |  | J |
|---|---|---|---|
| 02 | Mark Out P | 02 | Mark In P |
| 03 | Shield | 03 | Shield |
| 04 | Bus 1 Out 0 | 04 | Bus 1 In 0 |
| 05 | Shield | 05 | Shield |
| 06 | Bus 1 Out 2 | 06 | Bus 1 In 2 |
| 07 | Shield | 07 | Shield |
| 08 | Shield | 08 | Shield |
| 09 | Bus 1 Out 4 | 09 | Bus 1 In 4 |
| 10 | Shield | 10 | Shield |
| 11 | Bus 1 Out 6 | 11 | Bus 1 In 6 |
| 12 | Shield | 12 | Shield |
| 13 | Mark 1 Out | 13 | Mark 1 In |

* Reserved for future use

## Bus 1 (Dark Gray or Black)

```
         D                   J
   ++++++++++++        ++++++++++++
 02           13     02           13
   ++++++++++++        ++++++++++++
         B                   G
```

| | D | | J |
|---|---|---|---|
| 02 | Mark Out P | 02 | Mark In P |
| 03 | Shield | 03 | Shield |
| 04 | Bus 1 Out 0 | 04 | Bus 1 In 0 |
| 05 | Shield | 05 | Shield |
| 06 | Bus 1 Out 2 | 06 | Bus 1 In 2 |
| 07 | Shield | 07 | Shield |
| 08 | Shield | 08 | Shield |
| 09 | Bus 1 Out 4 | 09 | Bus 1 In 4 |
| 10 | Shield | 10 | Shield |
| 11 | Bus 1 Out 6 | 11 | Bus 1 In 6 |
| 12 | Shield | 12 | Shield |
| 13 | Mark 1 Out | 13 | Mark 1 In |

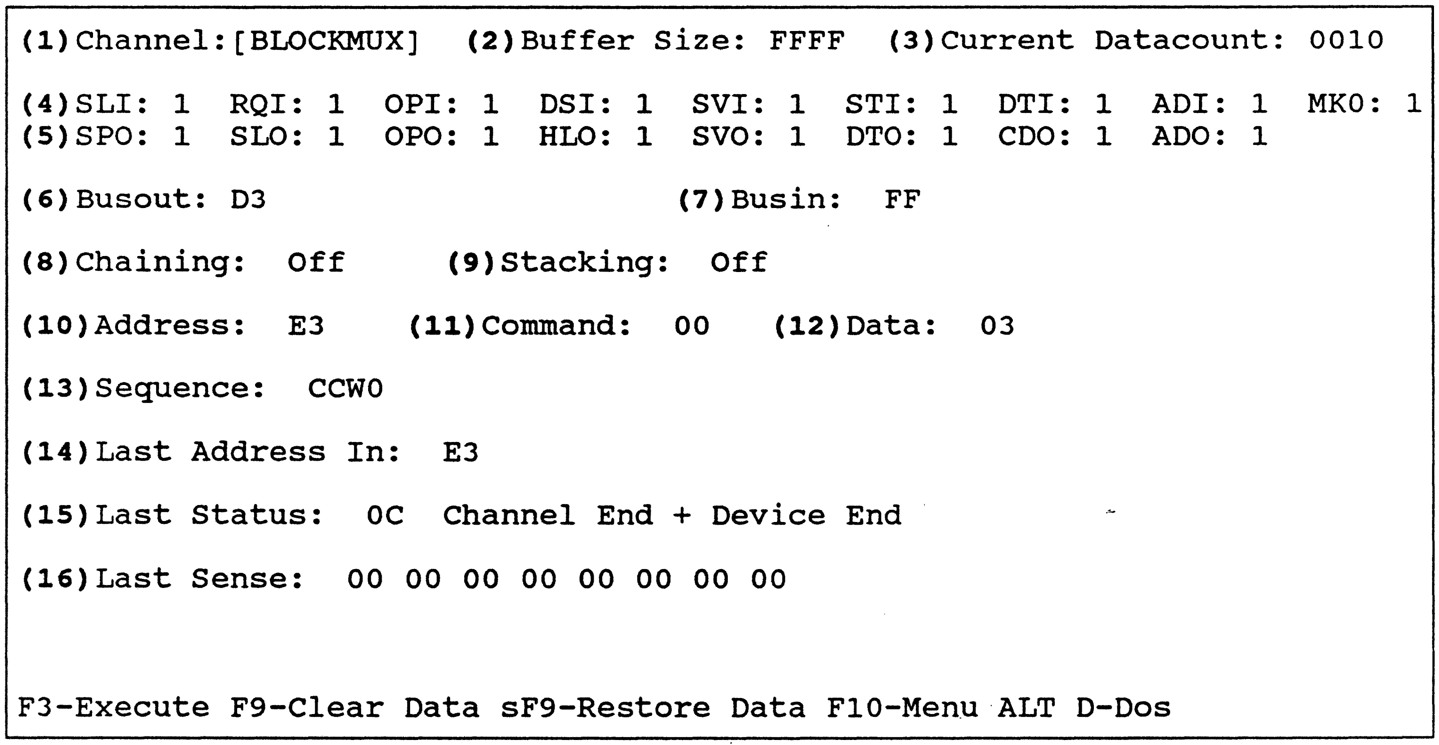
| | B | | G |
|---|---|---|---|
| 02 | Shield | 02 | Shield |
| 03 | Bus 1 Out P | 03 | Bus 1 In P |
| 04 | Shield | 04 | Shield |
| 05 | Bus 1 Out 1 | 05 | Bus 1 In 1 |
| 06 | * | 06 | * |
| 07 | Shield | 07 | Shield |
| 08 | Bus 1 Out 3 | 08 | Bus 1 In 3 |
| 09 | Shield | 09 | Shield |
| 10 | Bus 1 Out 5 | 10 | Bus 1 In 5 |
| 11 | Shield for B06* | 11 | Shield for G06* |
| 12 | Bus 1 Out 7 | 12 | Bus 1 In 7 |
| 13 | Shield | 13 | Shield |

* Reserved for future use

# APPENDIX E: UNDERSTANDING THE MANUAL EXECUTION SCREEN

## I. Introduction

The Manual Execution Screen can be confusing to someone who has little experience working with mainframe computers. This appendix should help explain what each option on this screen does. Each option has been numbered and a description of that option is given below the figure.



```
(1)Channel:[BLOCKMUX]  (2)Buffer Size: FFFF  (3)Current Datacount: 0010

(4)SLI: 1  RQI: 1  OPI: 1  DSI: 1  SVI: 1  STI: 1  DTI: 1  ADI: 1  MKO: 1
(5)SPO: 1  SLO: 1  OPO: 1  HLO: 1  SVO: 1  DTO: 1  CDO: 1  ADO: 1

(6)Busout: D3                      (7)Busin:  FF

(8)Chaining:  Off      (9)Stacking:  Off

(10)Address:  E3    (11)Command:  00    (12)Data:  03

(13)Sequence:  CCW0

(14)Last Address In:  E3

(15)Last Status:  0C  Channel End + Device End

(16)Last Sense:  00 00 00 00 00 00 00 00


F3-Execute F9-Clear Data sF9-Restore Data F10-Menu ALT D-Dos
```

**Figure E - 1: The Manual Execution Screen**

(1) Channel: Three types of channels may be selected - "Selector","Bytemux",or "Blockmux".

ByteMux: A number of devices can be operated simultaneously on the ByteMux Channel (Byte Multiplexor Channel). The I/O devices time-share the ByteMux Channel on a byte basis (bytes of data to or from the various devices are interleaved with one another). Burst mode can be forced on the ByteMux Channel by an I/O device holding up OPI.

BlockMux: A number of devices can be operated simultaneously on the BlockMux Channel (Block Multiplexor Channel). The I/O devices time-share the BlockMux Channel on a block basis (blocks of data to or from the various devices are interleaved with one another). Burst mode is forced on the BlockMux Channel by the channel holding up SLO.

Selector: Only one I/O device can be selected on this channel at any one time. Once selected, a complete record is transferred over the standard I/O interface one byte at a time. Once the record has been transferred, the channel is free to select another I/O device. A Selector Channel can only operate in "burst mode". It is suited to use with higher speed I/O devices, such as Magnetic Tape Units, Disk Units, and Drums.

(2) Buffer Size: This displays the current data buffer size. The default is 63,999, the largest size possible. The size of the buffer may be adjusted from 0 to 63,999.

(3) Current Datacount: This displays the current data count. This will either be equal to the number of bytes that were received over the channel, or it will be the number of bytes yet waiting to be sent over the channel. It can not be greater than the buffer size.

(4) Tagin Lines: These are lines that go into the mainframe computer from the control unit. They may be either high (1) or low (0). Tag lines, in most cases, have the function of identifying data (information) on the "Bus In" lines or the "Bus Out" lines. A Tag line that goes high indicates to the channel or the control unit that data (information) has been placed on the Bus, and identifies the function of this data. In some cases, Tag lines are also used as "sequence control" responses by the channel to statuses or data sent by a control unit. In these cases, the high Tag line is not directly related to the data on the Bus at the time it is raised.

Select In (SLI): Control unit I/O device selection is controlled by these signals. "Select Out" and "Select

In" form a loop from the channel through each control unit to the cable terminator block ("Select Out") and back through each control unit to the channel ("Select In"). Control unit priority is established because the rise of "Select Out" is a effective only to the first control unit. If the first control unit is not selected it will propagate the "Select Out" signal to the next control unit . A selected control unit will respond to "Select Out" with OPI. Once OPI rises the control unit must keep it up until the current signal sequence is completed. If the selected control unit is busy when "Select Out" is detected, the response is "Status In".

Request In (RQI): "Request In" is a line from the control unit to the channel and indicates that the control unit is ready to transmit information or data and is therefore requesting to be selected. "Request In" can be signaled by more than one control unit at a time.

Operational In (OPI): "Operational In" is a line from the control unit to the channel and is used to notify the channel that an I/O device has been selected. When 'Op In' is raised for a particular signal sequence, it must stay high until all required information is transmitted between the channel and the control unit.

Disconnect In (DSI): "Disconnect In" is a line from the control unit to the channel. It responds high when the "Service Out" line goes high. It is disabled when the operational line is disabled.

Service In (SVI): "Service In" is a Tag line from the control unit to the channel. It is used to signal the channel when the selected control unit wants to transmit or receive a byte of information. During a read operation, SVI rises when data is placed on "Bus In". During a write operation, SVI rises when data is required on "Bus Out". "Service In" must stay up until the rise of either "Service Out", "Command Out", or "Address Out".

Status In (STI): "Status In" is a Tag line from the control unit to the channel. It is used to signal the channel when the selected control unit has placed status information on "Bus In". The channel responds with either "Service Out" or "Command Out" depending on whether or not it accepted the Status byte. It is operated when the control unit detects a malfunction.

Data In (DTI): "Data In" is a tag line from the control unit to the channel. It is used to signal the channel that the data from "Busin" has been received.

Address In (ADI): "Address In" is a Tag line from the control unit to the channel. It is used to notify the channel to the address of the selected I/O device has been placed on "Bus In". "Address In" must stay high until the rise of "Command Out" from the channel. "Address In" must drop before 'Command Out' may drop.

Mark 0 In (MOI): "Mark 0 In" is a tag line from the control unit to the channel. It is used in early data-bus-width indication and it is enabled when OPI goes high. It is also used during a command retry.

(5) Tagout Lines: These are lines that go out of the mainframe computer and into the control unit. They may be either high (1) or low (0).

Suppress Out (SPO): "Suppress Out" is a line from the channel to the control unit. It is used both alone and with the out Tag lines to perform the following special functions: suppress data, suppress status, command chaining and selective reset. This line is used alone or in conjunction with the out-Tag lines to provide the following special functions: suppress data, suppress status, command chaining, and selective reset. The primary function of SPO is to prevent control units from beginning a re-selection (control unit initiated) sequence: A control unit cannot activate RQI as long as SPO is active from the channel.

Select Out (SLO): Control unit I/O device selection is controlled by these signals. "Select Out" and "Select In" form a loop from the channel through each control unit to the cable terminator block ("Select Out") and back through each control unit to the channel ("Select In"). Control unit priority is established because the rise of "Select Out" is a effective only to the first control unit. If the first control unit is not selected it will propagate the "Select Out" signal to the next control unit . A selected control unit will respond to "Select Out" with OPI. Once OPI rises the control unit must keep it up until the current signal sequence is completed. If the selected control unit is busy when "Select Out" is detected, the response is "Status In".

Operational Out (OPO): "Operational Out" is a line from the channel to the control unit. With the exception of "Suppress Out", all lines from the channel are significant only when OPO is high. Whenever OPO is low, all "In" lines from the control unit must drop and any operation currently in process over the interface must be reset. Selective reset and a system reset are both a result of dropping the OPO signal. OPO is normal as long as the channel is operating. This line is normally active and indicates that the channel is operational. As long as power is up in the channel and the channel is able to operate, this line should be active. Except for SPO, all "out Tags" are significant only when OPO is up. When OPO drops, all "in Tags" drop within 1.5 Microseconds, and any operation currently in progress over the interface is reset.

Hold Out (HLO): "Hold Out" is a line from the channel to all attached control units . It is used with "Select Out" to provide an enable for "Select Out". HLO is also used to minimize the propagation of the fall of "Select Out" (once "Hold Out" falls all the control units should drop "Select Out").

Service Out (SVO): "Service Out" is raised by the channel to indicate to the selected control unit that SVI or STI has been recognized. When raised in response to SVI, it indicates (on a read operation) that the channel has accepted the information on "Bus In" or (on a write operation) that the data requested by SVI has been placed on "Bus Out". When raised in response to STI, it indicates that the channel has accepted status information on "Bus In".

Data Out (DTO): "Data Out" is a Tag line from the channel to the control unit. It is used to signal the control unit that data has been sent out.

Command Out (CDO): "Command Out" is a Tag line from the channel to the control unit and is used to signal the selected I/O device in response to "Address In", "Status In", or "Service In". This signal is raised by the channel to signal a selected control unit in response to a signal on ADI, STI, DTI, or SVI. During an initial-selection sequence, CDO rising in response to ADI indicates to the I/O device that the channel has placed a "command byte" on the bus which indicates the I/O operation to be performed. CDO rising in response to SVI or DTI always means "stop the data transfer in progress". During a control-unit initiated sequence, CDO in response to ADI means "proceed". CDO in response to STI means "stack (retain) status".

Address Out (ADO): "Address Out" is a Tag line from the channel to all attached control units. It has two functions:

1. I/O device selection: "Address Out" is used to signal all control units to decode the device address on "Bus Out". The control unit that is addressed will respond with "Operational In" when "Select Out" comes up.
2. Disconnect operation: If "Address Out" is high, and "Hold Out" is low, the presently connected control unit must drop its "Operational In", thus disconnecting from the interface. "Address Out" will remain up until "Operational In" drops. This signal is raised by the channel to indicate to the control units that the address of the device the channel wants to select for an I/O operation has been placed on the "Bus Out".

(6) Busout: "Busout" is a set of nine lines consisting of eight information lines and one parity line. It is used to transmit addresses, commands, control orders, and data to the control units. To manually change the value of Busout, press enter. A pop-up menu will appear and you will be able to put one of the values specified in (9), (10) ,or (11) on the bus, or be able to disable the bus. The information contained on "Busout" is indicated by the Tagout lines:

1. When "Address Out" is high, "Busout" specifies the address of the I/O device the channel wants to communicate with.
2. When "Command Out" is high in response to "Address In", "Busout" specifies a command.
3. When "Service Out" is high in response to "Service In", "Bus Out" contains data.

Disable: This option is used when neither the address, nor the data, nor the sequence should be placed on the bus.
Enable Address: This option places the address on the bus.
Enable Data: This option places the data on the bus.
Enable Sequence: This option places the sequence on the bus.

(7) Busin: "Busin" is a set of nine lines consisting of eight information lines and one parity line. It is used to transmit

addresses, commands, control orders, and data to the channel. The value shown in the "Busout" display is the last value received into the ChannelACT on the "Busin" line. The type of information transmitted over "Busin" is indicated by the Tagin lines:
1. When "Address In" is high, "Bus In" contains the address of the currently selected I/O device.
2. When "Status In" is high, "Bus In" contains a byte of information describing the status of the selected I/O device or control unit.
3. When "Service In" is high, "Bus In" contains a byte of data or it contains the sense byte which describes the status of the device in detail.

(8) Chaining: The "Chaining option" can be turned "On" or "Off". While on, "Chaining" will be signalled to the peripheral until this option is again explicitly turned off. The "Chaining" sequence control is indicated, if indicated, at the time an I/O device presents ending status to the channel (at the conclusion of a data transfer). "Chaining" is indicated if SPO is up at the time SVO is raised in response to STI. "Chaining" means that another initial selection sequence (re-selection) is to occur for the I/O device in operation immediately following the presentation of "device end", provided that no unusual conditions were encountered during execution of the current operation.

(9) Stacking: The "Stacking" option can be turned "On" or "Off". While on, stacking will be signalled to the peripheral until this option is again explicitly turned off. "Stacking" is used when conditions preclude acceptance of status from the control unit. It may occur during any sequences except the "short-busy" sequence. It causes retention of status information at the control unit or I/O device until that status is accepted during a subsequent sequence. Stacking is indicated by the rise of CDO in response to STI. When it occurs, the control unit disconnects from the interface after SLO is down (OPI falls). CDO remains up until OPI falls. Any attempt to perform a control unit-initiated sequence in order to present status is under control of SPO (see "Suppress Status").

(10) Address: This option shows the peripheral address.

(11) Command: This option shows the channel command.

(12) Data: This option is only used if you want to manually put a data byte value on "Busout" with option (6) and classify it as data. None of the sequences use this value.

(13) Sequence: This option displays the sequence type. To change the sequence type, press return and a pop-up menu of choices will appear. They are as follows:

CCW0: CCW0 stands for "Channel Command Word 0". It executes commands during the time that the peripheral is connected to the system. It searches for both the channel end and the device end to indicate complete execution of the CCW.

CCW1: CCW1 is similar to CCW0 except it searches for the channel end and not the device end. When the chaining is on CCW0 and CCW1 act the same.

Initial Selection: Used when the channel wishes to establish communications with a particular I/O device.

Request: This option is used to get the final status if CCW doesn't give it.

Sense: Data is obtained from sense indicators rather than from a record source as in a "read" command.

System Reset: A "System Reset" may occur at any time, and is used to reset all control units and devices that are on-line. A "System Reset" is indicated whenever OPO and SPO are down concurrently and the I/O device is in the "online" mode. This condition causes OPI to fall and causes all control units and their attached I/O devices, along with their status, to be reset. The control units are in a busy state for the duration of their reset procedure. "System Reset" can prepare an I/O device for an initial program loading sequence.

Selective Reset:"Selective Reset" is generated by the channel, and may occur any time OPI is high. "Selective Reset" is indicated whenever SPO is high and OPO drops. This condition causes OPI to fall and causes the particular I/O device in the operation and its status to be reset. The operation in process proceeds to a normal stopping point, if applicable, with no further data transfer.

Interface Disconnect: "Interface Disconnect" is used by the channel to signal the control unit to end execution of an on-going I/O operation. If HLO is low and ADO rises or if ADO is high and HLO falls, the presently connected control unit drops OPI, thus disconnecting from the interface.

(14) Last Address In: The last address received on the channel.

(15) Last Status: The last status received on the channel.

(16) Last Sense: The last sense bytes received when a sense

sequence was executed.

# APPENDIX F:  MORE SAMPLE PROGRAMS

**Program 1:  Write, Read and Store Test**

This test writes 100 blocks of the data files "TMSG.RAM" and "1982.RAM" to the tape.  Following the writes, the tape rewinds and reads all 101 blocks.  Next, the 101st block is stored to the diskette with the name "Data.Ram".  The diskette file or data buffer can be displayed to see if the data in block 101 is correct. If so, the test was executed properly.

```
#************************TEST NAME: 1982.SIM******************
var a                     #A IS A VARIABLE
channel blockmux          #CHANNEL TYPE IS BLOCKMUX
system_reset              #RESET CHANNEL
ccw0 \80 \07              #PUT 07 ON ADDRESS 80 (REWIND)
ccw0 \80 \C3              #PUT C3 ON ADDRESS 80 (MODESET-21600BPI)
loadfile "1982"/T         #LOAD FILE 1982 FROM DISK AND TRANSLATE
for a := 1 to 100         #DEFINE VALUE RANGE FOR VARIABLE A
CCW0 \80 \01              #WRITE DATA BUFFER TO CHANNEL
restore                   #RESTORE DATA IN BUFFER FOR NEXT COMMAND
ENDFOR                    #LOOK FOR NEXT VALUE OF VAR A OR END
loadfile "TMSG"/T         #LD. FILE TMSG.RAM FROM DISK & TRANSLATE
CCW0 \80 \01              #WRITE DATA BUFFER TO CHANNEL
CCW0 \80 \07              #PUT 07 ON ADDRESS 80 (REWIND)
FOR a := 1 TO 101         #READ TO BLOCK SIX (FIND MSG)
CCW0 \80 \02              #ADDRESS 80 (READ FIRST BLOCK OF DATA)
ENDFOR                    #LOOK FOR NEXT VALUE OF A OR END
storefile "data.ram"/T    #STORES ONE BLOCK OF DATA TO SOURCE DISK
```

**Program 2: Write, Rewind, Read and Store Test**

This test writes 500 blocks of the file "32K.RAM". It writes the file "TMSG.RAM" as the 501st block then rewinds the tape. All 501 blocks are read and the 501st block is stored to the diskette with the filename "Data.RAM". The diskette file or the data buffer can be displayed to see if the data is correct. If it is, the test was executed correctly and is complete.

```
#*************************TEST NAME:  BIG.SIM******************
var a                       #A IS A VARIABLE
channel blockmux            #CHANNEL TYPE IS BLOCKMUX
system_reset                #RESET CHANNEL
ccw0 \80 \07                #PUT 07 ON ADDRESS 80 (REWIND)
loadfile "32K"/T            #LD. FILE 32K.RAM FROM DISK & TRANSLATE
for a := 1 to 500           #DEFINE VALUE RANGE FOR VARIABLE A
CCW0 \80 \01                #WRITE DATA BUFFER TO CHANNEL
restore                     #RESTORE DATA IN BUFFER FOR NEXT COMMAND
ENDFOR                      #LOOK FOR NEXT VALUE OF VAR A OR END
loadfile "TMSG"/T           #LD. FILE TMSG.RAM FROM DISK & TRANSLATE
CCW0 \80 \01                #WRITE DATA BUFFER TO CHANNEL
CCW0 \80 \07                #PUT 07 ON ADDRESS 80 (REWIND)
FOR a := 1 TO 501           #READ TO BLOCK SIX (FIND MSG)
CCW0 \80 \02                #ADDRESS 80 (READ FIRST BLOCK OF DATA)
ENDFOR                      #LOOK FOR NEXT VALUE OF A OR END
storefile "data.ram"/T      #STORES ONE BLOCK OF DATA TO SOURCE DISK
```

**Program 3: Recalibrate, Seek and Read Test**

First the actuator is recalibrated by placing the heads on cylinder track 00. Then this test does five sequential "seeks". A "seek" command positions the actuator. These commands are followed by reads of Count/Key/Data. The "seeks" begin at cylinder 01, track 03 and end at cylinder 05, track 03. The heads are then recalibrated, meaning they "seek" the cylinder 00 track.

```
#*************************TEST NAME:  DKRDLP.SIM***************
VAR A                       #A IS A VARIABLE
SYSTEM_RESET                #RESETS CHANNEL & DEVICES
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
CLEARBUFFER                 #CLEAR DATA BUFFER
FOR A := 1TO5               #DEFINE RANGE OF A
DATA 00,00,00,A,00,03       #6 BYTE CYL. & HD. ADD.
CCW0 \61 \07                #SEEK CYL. & HD.
CLEARBUFFER                 #CLEAR DATA BUFFER
CCW1 \61 \1E                #R\D. CNT,KEY,DAT
CLEARBUFFER                 #CLEAR DATA BUFFER
ENDFOR                      #END OF TEST LOOP
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
```

**Program 4: Recalibrate, Read, Seek, Write and Basic Sense Test**

First this test moves the heads to cylinder 00, track 00. Second, "home address" (record 0) is read and "sense" is read. Third, it does five sequential "seeks" (beginning at cylinder 01, track 03, record 01). Next, it searches for the record identification verifying the position then it writes the file "18K.RAM" to the track for each "seek". Following the fifth cycle, the test does a "basic sense" which completes operations.

```
#************************TEST NAME:  DKWRLP.SIM*************
VAR A
SYSTEM_RESET                #RESETS CHANNEL & DEVICES
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
CCW0 \61 \1A                #READ HOME ADDRESS
CCW0 \61 \04                #BASIC SENSE
FOR A :=1to5                #SET RANGE OF A
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,00,00,A,00,03       #6 BYTE CYL. ADR.
CCW0 \61 \07                #SEEK CYL. & HD
CLEARBUFFER
DATA 00,A,00,03,01          #CYL, HD, & REC. NUMBER
CCW0 \61 \31                #SEARCH I.D. EQUAL
while (!MOD)                #RE-
  retry                     #   TRY
endwhile                    #       LOOP
CLEARBUFFER                 #CLEAR DATA BUFFER
LOADFILE "18K"/T            #LOAD 4K.RAM FROM DISK TO BUFFER
CCW0 \61 \05                #WRT KEY, & DATA TO DISK
ENDFOR                      #END OF LOOP SEQUENCE
CLEARBUFFER                 #CLEAR DATA BUFFER
CCW0 \61 \04                #BASIC SENSE
```

**Program 5: Recalibrate, Seek and Read Test**

First this test recalibrates the heads placing them on cylinder 00, track 00. Then it "seeks" to cylinder 03, track 03. Next it looks for the identification of cylinder 03, track 03, record 01 and reads Count/Key/Data.

```
#************************TEST NAME:  DSKRD.SIM***************
SYSTEM_RESET                #RESETS CHANNEL & DEVICES
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,00,00,03,00,03      #6 BYTE CYL. & HD. ADD.
CCW0 \61 \07                #SEEK CYL. & HD.
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,03,00,03,01         #VALUE FOR SEARCH ID
CCW0 \61 \31                #SEARCH FOR SPECIFIED ID
CCW0 \61 \1E                #RD. CNT,KEY,DAT
```

**Program 6: File Mask Setting Test**

First this test places the actuator at cylinder 00, track 00. Then it "seeks" to cylinder 03, track 03 and sets the file mask. Third it searches for identification that equals cylinder 03, track 03, record 01. When found, it writes file "18K.RAM" to the disk with Count/Key/Data.

```
#*************************TEST NAME:  DSKWR.SIM****************
SYSTEM_RESET                #RESETS CHANNEL & DEVICES
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,00,00,03,00,03      #6 BYTE CYL. ADR.
CCW0 \61 \07                #SEEK CYL. & HD
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,03,00,03,01         #CYL, HD, & REC. NUMBER
CCW0 \61 \1F                #SET FILE MASK
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,03,00,03,01         #CYL, HD, & REC. NUMBER
CCW0 \61 \31                #SEARCH I.D. EQUAL
while (!MOD)
  retry
endwhile
CLEARBUFFER                 #CLEAR DATA BUFFER
LOADFILE "18K"/T            #LOAD 8K.RAM FROM DISK TO BUFFER
CCW0 \61 \05                #WRT KEY, & DATA TO DISK
CCW0 \61 \03                #NO OP
```

**Program 7: Long Seek Test**

This is a long "seek test". It recalibrates the actuator placing the heads on cylinder 00, track 00. It then "seeks" to cylinder 0247, track 03. From here it repeats the sequence. It recalibrates and "seeks" infinitely until the operator hits the function key [F7] to terminate the test.

```
#*************************TEST NAME:  LGSKLP.SIM***************
SYSTEM_RESET                #RESETS CHANNEL & DEVICES
CCW0 \61 \13                #RECALIBRATE(PUTS HDS TO CYL.00 SEC.00)
AA: CLEARBUFFER             #CLEAR DATA BUFFER
DATA 00,00,00,00,00,03      #6 BYTE CYL. & HD. ADD.
CCW0 \61 \07                #SEEK CYL. & HD.
CLEARBUFFER                 #CLEAR DATA BUFFER
DATA 00,00,02,47,00,03      #6 BYTE CYL. & HD. ADD
CCW0 \61 \07                #SEEK CYL. & HD.
GOTO AA                     #LOOP BACK TO AA
```

# Index

TECHNOLOGY 80 INC.

658 Mendelssohn Ave. No.
Minneapolis, MN 55427
(800) 545-2980
FAX (612) 542-9785