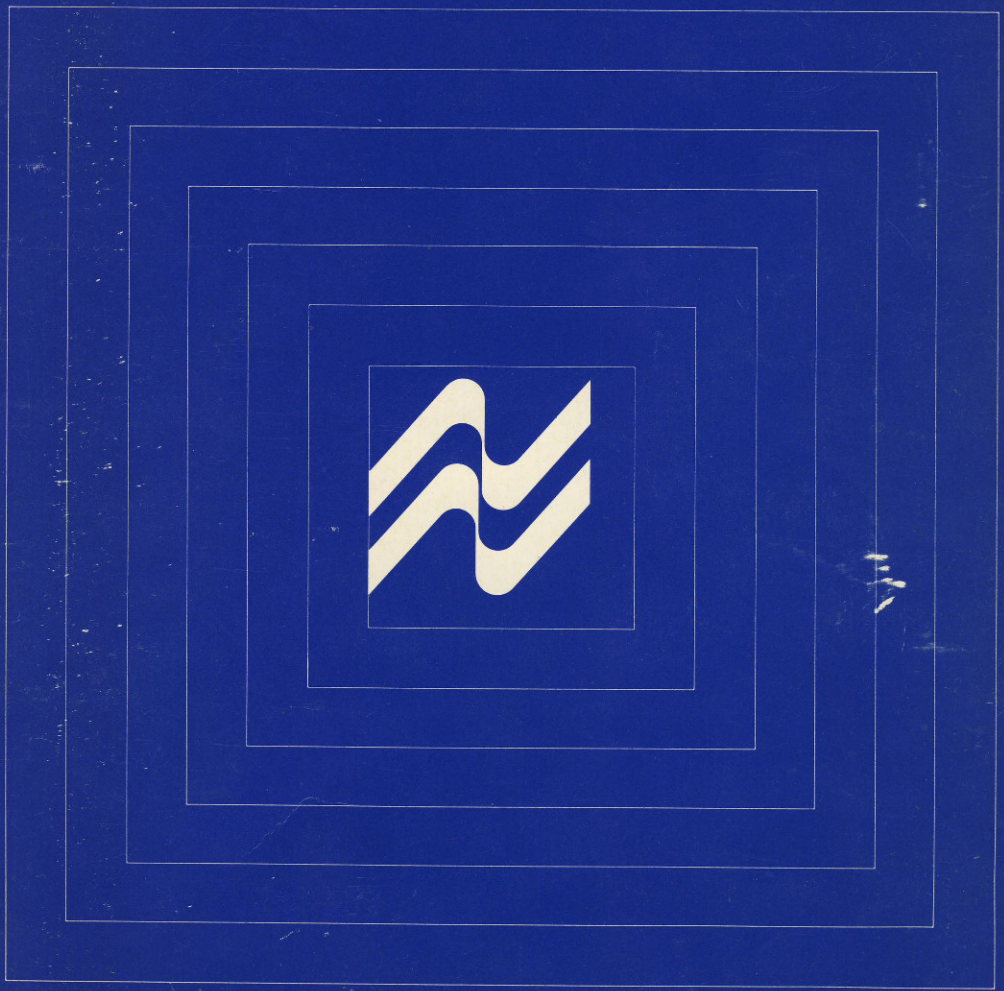
# User's Manual

National Semiconductor

# COP400 Product Development System



# COP400 Product Development System

# USER'S MANUAL

National Semiconductor

420305548-001
January 1980

2900 Semiconductor Drive, Santa Clara, California 95051
(408) 737-5000/TWX (910) 339-9240


# Table of Contents

# Table of Contents
(continued)

# Table of Contents
(continued)

# Table of Contents
(continued)

# Table Index

# Illustration Index

# 1 Introduction to the COP400 Product Development System (PDS)

## 1.1 Introduction to PDS

What is a "development system?" It is a hardware and software package designed to aid the user in developing a product incorporating a microprocessor or microcomputer. A development system provides the user with three major capabilities:

1. Creating and editing text "files" (EDITING)
2. Translating text files to microcomputer machine code (ASSEMBLING)
3. Executing and testing the machine code (DEBUGGING)

The COP400 Product Development System, hereafter called "PDS," was designed specifically to aid in the development of products using National's 400 series "Controller Oriented Processors" (COPs). This manual must be supplemented with the *COP400 Microcontroller Family Chip Users Manual*, order number 420305785-001, which gives detailed information concerning programming of COP400 devices.

PDS is a disk-oriented system. It is capable of creating and accessing files of data stored on a floppy diskette. For the user this means fast and easy access to system software, rapid access to his program files, and a convenient means of providing National with COP400 program data for the mask-making process. The user will soon learn to appreciate the ease with which the "edit-assemble-test" cycle is accomplished compared to "paper tape" systems.

Another PDS feature is the COP400 debugging capability. This includes hardware and software which enable the user to single-step through his COP400 program, breakpoint on an address, trace program execution, and dump out internal COP400 registers. This invaluable feature further speeds up the development cycle.

The user interacts with PDS via a system console such as a teletype or CRT. An optional printer can be attached to obtain program listings quickly. The PDS front panel allows the user to perform specific development functions without a system console. Connectors on the rear panel of PDS interface to an "emulator card" which emulates a particular COP400 chip in the user's system. A PROM programmer on the PDS front panel allows the emulator card to be portable, for emulation in the final environment of the user's system.

Now that we've described the basic parameters of PDS, we think it's appropriate to use a few well-chosen words on how we feel (and how we think you will) about the COP400 Microcontrollers and the COP400 Development System.

In the midst of a microcomputer industry proclaiming the development of "first" products and systems, we truly believe the COP400 Development System to be a "first" in providing the user with a compact, easily usable system for performing the above-mentioned development cycle functions with a minimal amount of physical and mental "set-up" time. The combination of portable hardware, understandable and powerful PDS system programs and commands, and the convenience of a built-in disk drive together with minimum peripheral device requirements result in an integrated system capable of performing complex software development and debug tasks with minimum effort.

One important example of this integrated-concept approach is the inclusion in the capabilities of the system of in-circuit emulation of a COP400 program in the hardware environment of the user's final system. As mentioned above, this is accomplished using the PDS emulator card which functions as a virtual COP400 device, running the user's program in its final environment. In effect, all guesswork is removed from the development cycle, with final debug being performed with the COP400 emulator card driving the user's hardware under control of the user's program.

This, however, isn't the end of the story. PDS develops and debugs programs for a versatile and powerful series of microcontrollers, the COP400 Microcontrollers. These devices are *single-chip microcomputers*, containing all device timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of small-system, dedicated applications. COP400 microcontrollers further reflect National's integrated-concept approach, with each primary COP400 device being a software-subset of the most inclusive COP400 device, the COP440. To provide the user with a choice among COP400 devices specifically tailored to the user's application at minimal cost, each part contains numerous electrical specification options, mask-programmed into the part at the same time as the user's ROM

program code. Further versatility is obtained by providing different "technology" versions of several devices; for example, the NMOS COP420 is also available as a CMOS part (COP420C) or as a low-power device (COP420L).

The final integration associated with the COP400 development cycle, and perhaps the most important, relates to the documentation describing the various COP400 devices and the COP400 Development System. Contrary to the industry practice of providing a proliferation of manuals, separately describing each device, each development system program, and hardware characteristics and "hook-ups," we've taken the unusual step of providing you with everything you may want to know but might be afraid to try and find on these subjects in *two* manuals; this manual, the *PDS Users Manual*, and the *COP400 Microcontroller Family Chip Users Manual*.

After reading these manuals and using PDS to develop a program for a COP400 device suited specifically to and debugged in the actual environment of your final system, we think you'll understand why we're particularly proud of PDS and the COP400 Microcontroller Family and why we believe that we've finally brought the economy, versatility and reliability of the microcomputer to the small-system, consumer oriented customer — or, to put it another way — that we've brought it all back home for the first time.

* * * * * * * *

**THE USER SHOULD READ THIS MANUAL THOROUGHLY BEFORE ATTEMPTING TO OPERATE THIS MACHINE.**

# 2 Description of PDS Hardware

This chapter provides a general description of PDS hardware. This hardware consists of a host CPU with support circuitry, and the following important parts:

1. Front and Rear Panels
2. COP400 Emulation and Debug Hardware

These parts will be discussed in detail in the following sections of this manual.

## 2.1 Front and Rear Panels

The PDS front panel is shown in Figure 2.1. The key switch in the lower right corner is the power switch. To its left are five other switches. The first two, labeled "Program Load," are used to load and execute two different PDS system programs. The first switch loads COP Monitor discussed in Chapter 9. The second loads a test program discussed in the *Chip Tester Manual*. The third switch causes PDS to perform a diagnostic test on its internal memory and the disk drive. The fourth switch is for PDS system initialization. The fifth switch is a spare which is not currently used by PDS. These five switches are discussed at greater length in Chapter 4.

Above the power switch is a 24-key pad, which allows PDS functions to be performed without a console. Figure 2.2 is a close-up view of the keypad. Above the keypad is a dual four-digit LED display used for displaying data requested by the keypad. These items are discussed at greater length in Chapter 9.

In the center of the PDS front panel is a quick-release socket that is used for programming PROMs. Two types of PROMs are acceptable: an MM5204 512x8 UV-erasable PROM, and an 87S296/74S474 512x8 fusible link bipolar PROM. A pin scrambler *must* be inserted when an 87S296/74S474 PROM is used. Figure 2.3 shows the pin correspondences. Either of these two PROMs can be used with a COP emulator card.

*DO NOT* ATTEMPT TO PROGRAM AN 87S296 PROM WITH PDS UNLESS THE PIN SCRAMBLER IS INSERTED.

On the left side of the PDS front panel is the floppy disk drive door. The door latches are closed and opened with the rectangular button to the left of it. In the center of the button is an indicator light which indicates that the drive is being used.

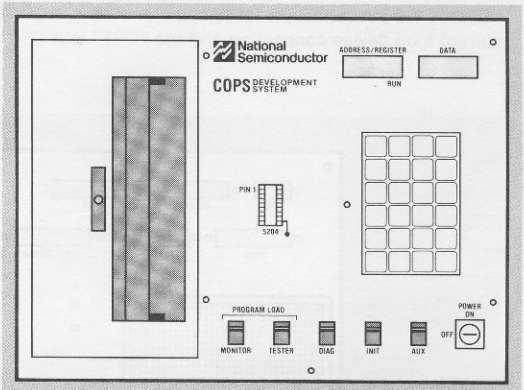



Figure 2.1 PDS Front Panel

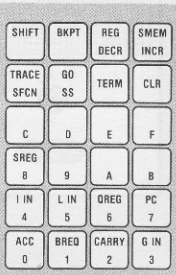



Figure 2.2 PDS Front Panel Keypad

| 74S474 Pin Number | PDS Front Panel Socket Pin Number and MM5204 PROM Pin Number |
|---|---|
| 1 | 13 |
| 2 | 11 |
| 3 | 10 |
| 4 | 9 |
| 5 | 8 |
| 6 | 7 |
| 7 | 6 |
| 8 | 5 |
| 9 | 15 |
| 10 | 16 |
| 11 | 17 |
| 12 | 24 |
| 13 | 18 |
| 14 | 19 |
| 15 | 20 |
| 16 | 21 |
| 17 | 22 |
| 18 | 4 |
| | 4 both to pin 4 |
| 20 | 2 |
| 21 | 3 |
| 22 | Not Used |
| 23 | 14 |
| 24 | 1 |

Figure 2.3. 87S296 Pin Scrambler Information

The PDS rear panel is shown in Figure 2.4. On each side there is a cooling fan filter screen. Below the left screen are a fuse holder and a plug for the power cable. Above this screen are six connectors which are used to connect peripheral devices to PDS.

Figure 2.5 is a close-up view of the rear panel connectors. The two labeled "TTY" and "CRT" are used to connect a console to PDS. The one labeled "TTY" can also be used to connect a printer. The "TTY" is a 9-pin connector for teletype or other current loop device connection and the "CRT" is a standard 25-pin RS232 connector for a CRT or other RS232 device. The center connector, labeled "PRINTER," is a standard 25-pin RS232 connector used to connect a printer to PDS.

The connectors labeled EMULATOR 1 and EMULATOR 2 interface PDS to an external Emulator Card. (For further information, see Section 2.3.)

The connector labeled TEST on the PDS rear panel is used to connect a COP400 Tester. For further information, see *COP400 Chip Tester Manual,* Order No. 420305786-001.

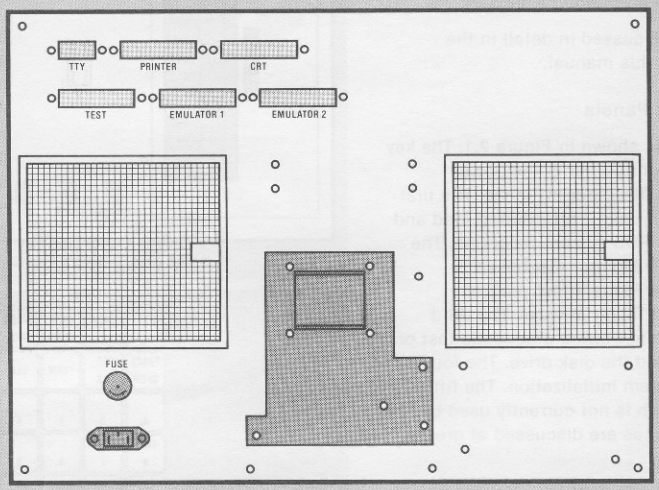



Figure 2.4 PDS Rear Panel

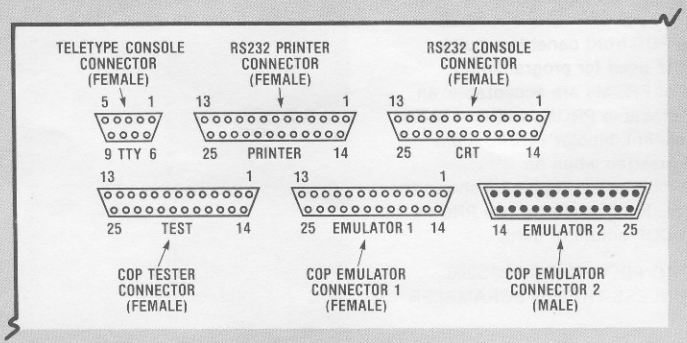



Figure 2.5 PDS Rear Panel Connectors

To use the full capabilities of PDS, a *console* for entering system commands must be connected to it. A console can be any device with a standard RS232 or current loop interface, and operating at one of the baud rates shown in Table 2.1. The user can set EVEN or NO parity, and carriage return and line feed delays from 0 to 1000 milliseconds for the console. The recommended console setups for the various allowable baud rates are as follows:

110 Baud: 8-bit data (No Parity — PDS resets bit 8 = 0), 2 Stop bits, Full Duplex operation

150–9600 Baud: 8-bit data (No Parity — PDS resets bit 8 = 0), 1 Stop bit, Full Duplex operation; *or* 7-bit data, Even Parity, 1 Stop bit, Full Duplex operation

If the console uses a current loop interface, PDS will assume a 110 Baud rate with the setup as shown above. (See Section 4.3.)

**Table 2.1. Acceptable Baud Rates for PDS Peripherals**

| |
|---|
| 110 |
| 150 |
| 300 |
| 600 |
| 1200 |
| 2400 |
| 4800 |
| 9600 |

PDS will also accept a second peripheral device, a printer, used for obtaining program listings. Any device that meets the requirements listed above for a console will also serve as a printer. The user can set EVEN or NO parity, and carriage return, line feed, form feed, and vertical tab delays from 0 to 1000 milliseconds for the printer.

When selecting peripheral devices for PDS, one capable of producing hardcopy output is essential in order to obtain program listings. Choosing a teletype for the system console has the advantage of providing hardcopy output, thus precluding the necessity for a printer. Choosing a CRT for the system console has the advantage that programs can be displayed rapidly and editing can be done more quickly. A printer is optional but is generally necessary if a non-hardcopy device is selected for a console. Table 2.2 lists some typical peripheral devices.

**Table 2.2. Recommended Peripheral Devices**

| | |
|---|---|
| CRTs: | 1. Lear Siegler model ADM-3, part number 129450<br>Lear Siegler<br>Anaheim, California |
| | 2. Hazeltine model 1500<br>Hazeltine Industrial Products Division<br>Greenlawn, NY 11740 |
| PRINTERS: | 1. Centronics model 306 with RS232 interface<br>Centronics Data Computer Corporation<br>Hudson, New Hampshire |
| | 2. FACIT model 4555 with RS232 interface<br>(Sweden) |
| | 3. G.E. TERMINET with RS232 interface |
| TTYs: | 1. Teletype model ASR3320/3JC manual read |
| | 2. Decwriter |
| | 3. Silent 700 |

Note: A Silent 700 with RS232 interface requires pins 5 and 8 be connected together.

## 2.2 PDS Peripheral Configurations

Chapter 3 includes detailed information on connection of peripheral devices to PDS. Figure 2.6 shows the minimum peripheral configuration and optional configurations of PDS. The minimum configuration consists of a teletype only, for entering system commands and obtaining program listings. The first alternative configuration uses a CRT for entering commands and a high-speed printer for program listings. The second alternate configuration uses a CRT for entering commands and a teletype for program listings. PDS can be used with *no* peripheral devices to do a limited number of tasks, e.g., COP testing and COP emulation.

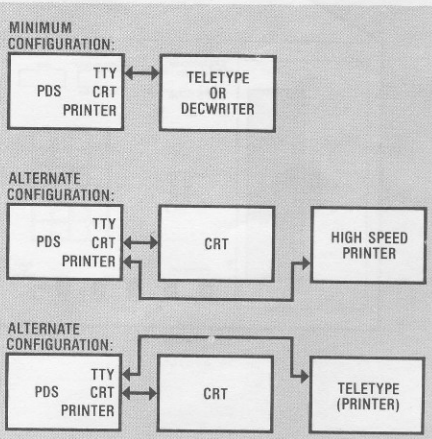



**Figure 2.6 PDS Peripheral Configurations**

## 2.3 COP400 Emulation and Debug Hardware

PDS has numerous powerful features which greatly facilitate COP400 system development. Among them are:

1. In-Circuit Emulation
2. Trace
3. Breakpoint
4. Single-Step

### In-Circuit Emulation

"In-Circuit Emulator" refers to hardware which emulates the COP400 chip and which has an EMULATION cable with a connector on it that is pin-for-pin identical to a COP400 chip and can be plugged into the user's system in place of a COP400 chip. An In-Circuit Emulator allows the user to test out his COP system in its intended environment, and to modify and re-test the program if errors are found in it. This enables the user to be absolutely certain that his system is correct before dedicating it to mask-making.

The PDS In-Circuit Emulation System is shown in Figure 2.7. The "Emulator Card" emulates the COP chip by using a special COP400 device which is identical to a masked-ROM COP, except that the ROM has been replaced by an interface to an external memory. The external memory may consist of PROMs which plug into the emulator card. Alternatively, the emulator card may be connected to a random access memory located within PDS. Since this memory is used by both the emulator card and PDS, it is called "shared memory."

Shared memory may be first loaded from a disk file, then altered and/or listed using the PDS system software (COPMON) described in Chapter 9. It is therefore simple to make COP program changes.

A drawing of the COP420 emulator card is shown in Figure 2.8. The card comes with an EMULATOR cable. The TARGET cable is supplied with the PDS. One end of the TARGET cable attaches to the 50-pin edge connector on the card. The other end splits into two connectors, one male and one female, that attach to the PDS rear panel connectors labeled "EMULATOR 1" and "EMULATOR 2." Figure 2.9 shows the pinout of these connectors. Five types of signals come across the TARGET cable:

1. Shared memory address and data lines, used by the emulator card to access shared memory.
2. +5 and −12 volt power supply lines from PDS to the emulator card. Three posts on the emulator card, labeled "$V_{CC}$," "GND," and "−12" provide user access to these supply lines. *These posts are NOT meant to allow the user to access the PDS supplies.* They are to be used for supplying power to the emulator card when it is disconnected from PDS. *The PDS power supplies should NEVER be used to power the user's system.*
3. A "RESET" signal from PDS to the COP chip "RESET" pin. This enables PDS software to reset the COP (see Chapter 9). The COP chip can

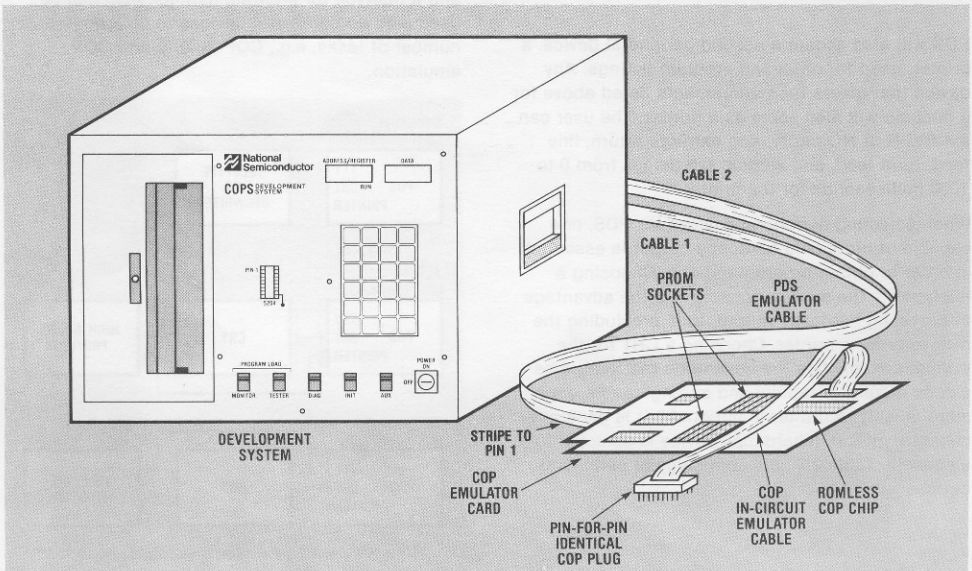



Figure 2.7 PDS In-Circuit Emulation System

also be reset by the user by pressing the button labeled "RESET" on the emulator card.

4. There are four connector posts on the right side of the emulator card labeled "1," "2," "3," and "4." The user may tie any TTL compatible signal to these posts. These signals are available to PDS via the cable. They are called "external event" signals, and are useful for testing COP systems. These signals will be discussed in more detail later in this chapter.
5. A PDS signal called "TRIGGER OUT" is available at the post labeled "T.O." on the emulator card. It is a TTL compatible signal. The purpose of this signal will be described later in this chapter.

In the center of the emulator card are two sets of holes which are used for PROM sockets. Only the 5204 sockets are provided. When PROMs are placed in these sockets and the PDS software is instructed to enter "PROM" mode (see Chapter 9) or if the PDS emulator card is disconnected from PDS, the ROM-less COP chip will access the PROMs. PROMs replace shared memory, thus providing a means of making the emulator card portable for use in the final environment of the COP chip, and releasing PDS for the next project. The sockets labeled "04" on the emulator card are for MM5204 PROMs. An MM5204 can be erased with ultra-violet light, allowing it to be used over again. However, it also requires a $-12$ volt power supply. The sockets labeled "296" on the emulator card are for 87S296 PROMs. The 87S296 is a fusible link bipolar PROM which can be programmed only once, but requires only a single $+5$ volt supply (as does a COP chip). The PROM socket labeled "PROM 0" is for COP addresses 0-X'1FF, and the socket labeled "PROM 1" is for COP addresses X'200-X'3FF.

The cable that is shown in the upper left of the emulator card in Figure 2.8 is the In-Circuit EMULATION cable. The plug on the end of this cable can be plugged directly into the COP chip socket in the user's system. The $+5$ volt power required by the emulator card is supplied by PDS via the TARGET cable. This $+5$ volt supply also appears at the EMULATION cable plug. THE USER MUST NOT CONNECT HIS SYSTEM POWER SUPPLY TO THE PDS SUPPLY VIA THE EMULATION CABLE. This could destroy one or both supplies. A jumper labeled "$V_{CC}$" on the emulator card may be cut to disconnect the PDS $+5$ volt supply from the EMULATION cable and the ROM-less COP.

The emulator card has an RC oscillator for the user's convenience. Its nominal frequency is 3.8 MHz. A jumper labeled "OSC" on the emulator card can be cut if the user wishes to use an external oscillator of his own.

Figure 2.10 summarizes the functions of the two emulator card jumpers described above.

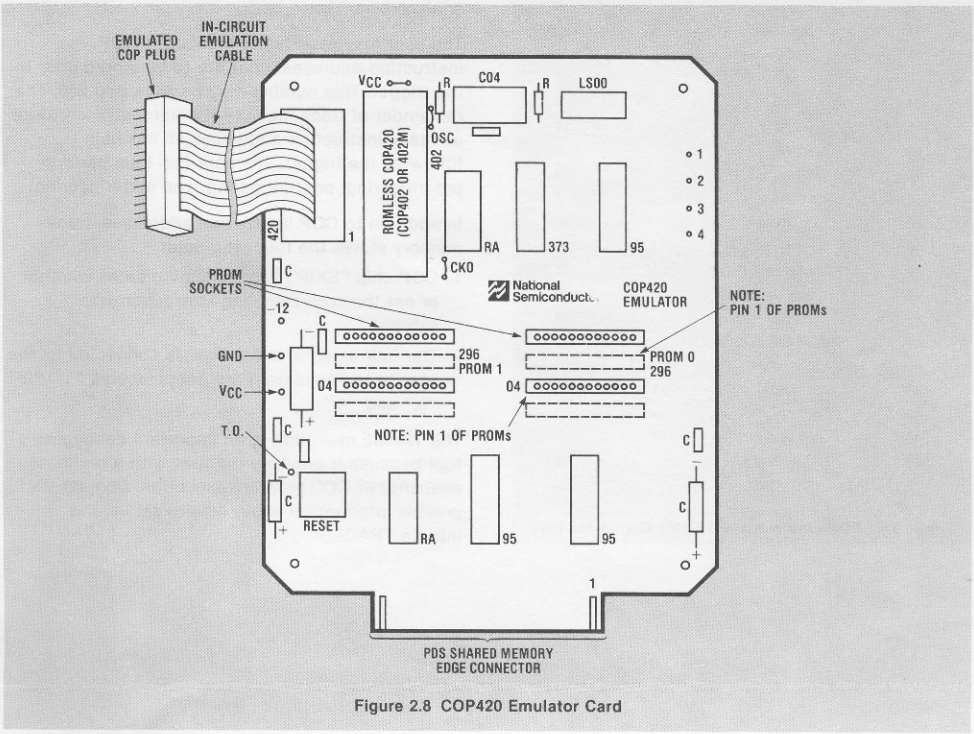



Figure 2.8 COP420 Emulator Card

| Emulator Card Connector # | Signal Name | Emulator 1 Pin # | Emulator 2 Pin # |
|---|---|---|---|
| 1 | GROUND | 13 | |
| 2 | GROUND | 25 | |
| 3 | VCC | 12 | |
| 4 | VCC | 24 | |
| 5 | External Event 2 | 11 | |
| 6 | External Event 1 | 23 | |
| 7 | External Event 4 | 10 | |
| 8 | External Event 3 | 22 | |
| 9 | CLK | 9 | |
| 10 | SKIP | 21 | |
| 11 | A8 | 8 | |
| 12 | A9 | 20 | |
| 13 | A3 | 7 | |
| 14 | A7 | 19 | |
| 15 | A1 | 6 | |
| 16 | A2 | 18 | |
| 17 | A4 | 5 | |
| 18 | A0 | 17 | |
| 19 | A6 | 4 | |
| 20 | A5 | 16 | |
| 21 | Not Used | 3 | |
| 22 | A10 | 15 | |
| 23 | Not Used | 2 | |
| 24 | Not Used | 14 | |
| 25 | Not Used | 1 | |
| 26 | Not Used | | 1 |
| 27 | Not Used | | 14 |
| 28 | Not Used | | 2 |
| 29 | Not Used | | 15 |
| 30 | Not Used | | 3 |
| 31 | Not Used | | 16 |
| 32 | Not Used | | 4 |
| 33 | B0 | | 17 |
| 34 | B7 | | 5 |
| 35 | B2 | | 18 |
| 36 | B5 | | 6 |
| 37 | B3 | | 19 |
| 38 | B4 | | 7 |
| 39 | B6 | | 20 |
| 40 | B1 | | 8 |
| 41 | TRIGGER OUT (T.O.) | | 21 |
| 42 | Not Used | | 9 |
| 43 | RST* | | 22 |
| 44 | PROM DISABLE* | | 10 |
| 45 | – 12V | 23 | |
| 46 | – 12V | 11 | |
| 47 | VCC | 24 | |
| 48 | VCC | 12 | |
| 49 | GROUND | | 25 |
| 50 | GROUND | | 13 |

**Figure 2.9. PDS Emulator Card Cable Connector Pins**

| Jumper Name | Function |
|---|---|
| "VCC" | Connects PDS +5 volt supply to ROM-less COP chip and emulation cable. Cut this jumper to disconnect PDS supply from user supply. |
| "OSC" | Connects external RC oscillator to COP oscillator input. |

**Figure 2.10. Emulator Card Jumpers**

### Trace

"Program Trace" is a *real-time* debug mode available to the user during In-Circuit Emulation.

A "Trace Memory" within PDS will store 254 consecutive COP instruction addresses. This storage operation is called a "TRACE." A TRACE can be initiated immediately on user command, or it can be set up by the user to initiate automatically when one of these conditions occurs:

1. The COP chip PROGRAM COUNTER attains a specific address.
2. External Events 1 and 2 attain specific values.

The user can specify that a given number of occurrences (from 1 to 256) of the above conditions must occur before TRACE is initiated.

At the time that TRACE is initiated, a positive edge (⌐) occurs on the "Trigger Out" (T.O.) signal post on the emulator card. This signal is sometimes useful for triggering oscilloscopes or logic analyzers.

The user may specify the number of COP instruction addresses that are to be stored prior to the trigger. This number may be from 0 to 253. The remainder of trace memory will automatically store as many instruction addresses as possible following the trigger. The user can thus perform pre-triggering, post-triggering, and mid-triggering.

In addition to COP instruction addresses, trace memory stores the following data:

1. COP chip "SKIP" flag, which indicates whether or not the corresponding instruction was skipped.
2. The four external event signals connected by the user to the emulator card posts labeled "1," "2," "3," and "4."

The TRACE operation is an important debugging tool because it provides the user with a *real-time snapshot* of COP program execution. Chapter 9 will provide information about how to set up and initiate TRACES.

### Breakpoint

"Breakpoint" is a debug mode available to the user during In-Circuit Emulation. It provides a means for examining internal COP registers at specific points within program execution. A "BREAK" can be initiated immediately on user command, or it can be set up by the user to initiate automatically when one of the following conditions occurs:

1. The COP chip PROGRAM COUNTER attains a specific address.
2. External Events 1 and 2 attain specific values.

The user can specify that a given number of occurrences (from 1 to 256) of the above conditions must occur before a BREAK is initiated.

At the time that BREAK is initiated, a positive edge (⌐) occurs on the "Trigger Out" (T.O.) signal post on the emulator card. This signal is sometimes useful for triggering oscilloscopes or logic analyzers.

When BREAK is initiated, the COP chip instruction lines are switched from shared memory or PROMs over to a special transparent memory containing a *dump program*. This program causes the ROM-less COP chip to dump all internal registers and memory to PDS, where it is available for inspection by the user. The program then restores all registers and maintains the COP chip in a *waiting state* until commanded by the user to continue normal program execution. When the COP is in this wait state, it is referred to as being "breakpointed."

The BREAK operation is an important debugging tool because it provides the user with the capability of examining internal COP registers as program execution continues. BREAKPOINT is not a real-time operation. Chapter 9 will provide information about how to set up and initiate BREAKPOINTS.

### Single-Step

"Single-step" is a debug mode available to the user during In-Circuit Emulation. It provides a means for single-stepping the COP chip by "breakpointing" on each consecutive instruction. Internal COP registers are available to the user after each "STEP."

When the COP chip is being single-stepped, it is maintained in the previously discussed "breakpointed" state between "STEPS."

The STEP operation is an important debugging tool because it provides the user with a means of following program execution step-by-step. Chapter 9 will provide information about how to single-step.

# 3 PDS Installation and Verification

Figure 3.1 shows the items shipped with PDS from the factory. Chapter 2 discusses the peripheral devices that are needed to make a complete system. Installation of PDS involves connecting the peripheral devices to PDS. After installation, a verification operation must be used to test for proper operation.

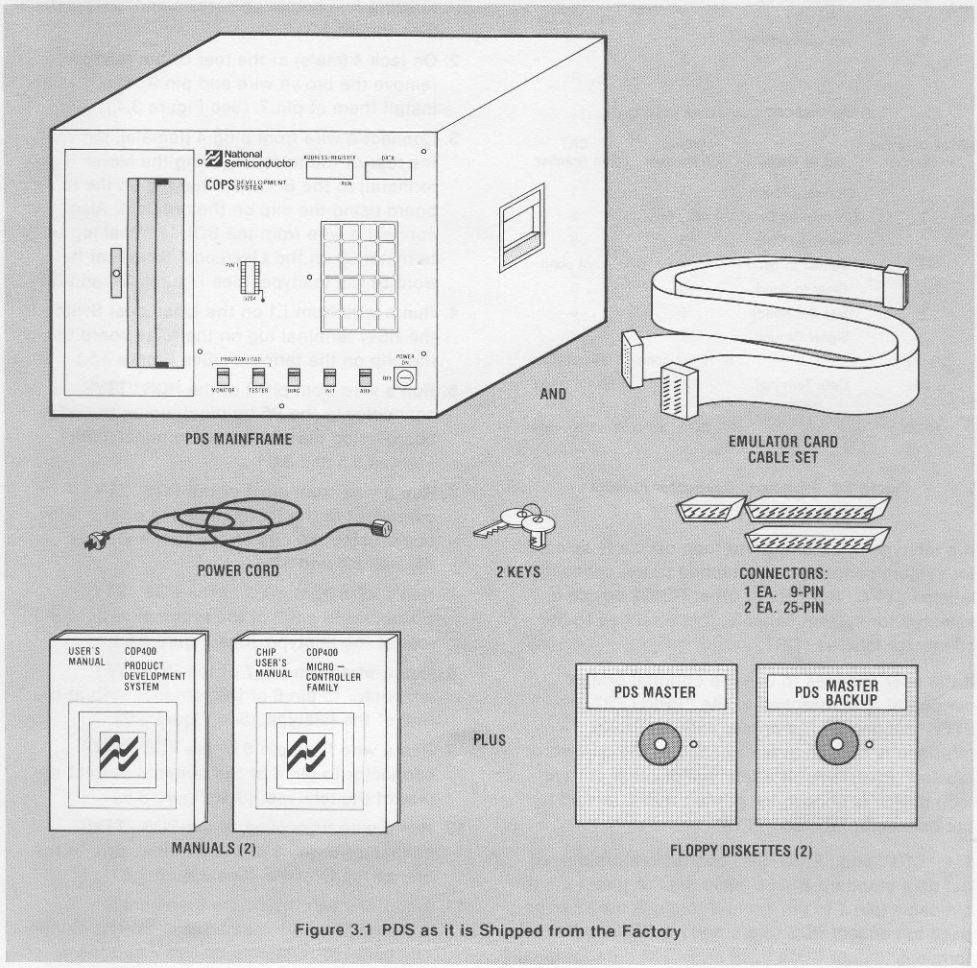



Figure 3.1 PDS as it is Shipped from the Factory

## 3.1 PDS Installation

Figure 3.2 gives the pinout for the three PDS rear-panel peripheral connectors. PDS is shipped with a matching connector for each one. The first step in installation is to make cables to connect PDS to the peripherals. The user must provide the connectors for the peripheral sides of the cables.

**TTY Connector (Current Loop)**

| Pin Number | Signal Name |
|---|---|
| 1 | TTY Xmitter (+) |
| 2 | TTY Printer (+) |
| 3 | Reader Relay (+) |
| 4 | not connected |
| 5 | not connected |
| 6 | TTY Xmitter Return (−) |
| 7 | TTY Printer Return (−) |
| 8 | Reader Relay Return (−) |
| 9 | not connected |

**Printer and CRT Connectors (RS232)**

| RE232 Data Set Pin Number | Signal Name | Printer Pin Number | CRT Pin Number |
|---|---|---|---|
| 1 | Chassis Ground | 1 | 1 |
| 2 | Transmitted Data | 2 not conn. | 2 |
| 3 | Received Data | 3 | 3 |
| 4 | Request to Send | 4 | 4 not conn. |
| 5 | Clear to Send | 5 | 5 |
| 6 | Data Set Ready | 6 | 6 |
| 7 | Signal Ground | 7 | 7 |
| 8-19 | | 8-19 not conn. | 8-19 not conn. |
| 20 | Data Terminal Ready | 20 | 20 not conn. |
| 21-25 | | 21-25 not conn. | 21-25 not conn. |

**Figure 3.2. Peripheral Connector Pinouts**

If a teletype or other current loop device is selected for system console, it is attached to the connector labeled "TTY." If a CRT or other RS232 device is selected for system console, it is attached to the connector labeled "CRT."

If the user chooses to use the optional printer peripheral, it is attached to the "PRINTER" or "TTY" connector. If a printer with an RS232 interface is chosen, it is attached to the connector labeled "PRINTER." If a teletype or other current loop device is chosen for printer, it is attached to the connector labeled "TTY."

The "CRT" and "PRINTER" connectors are pinned out as a standard RS232 "data set." A direct pin-to-pin cable (pin 1 to pin 1, pin 2 to pin 2, etc.) can be used to connect PDS to a standard RS232 "data terminal." (Most CRTs have an RS232 data terminal connector and are often supplied with a cable.)

The "TTY" connector is designed for use with a model ASR3320/3JC manual read teletype or other current loop device. If the user intends to use paper tape input to PDS, a "reader relay" must be installed on the teletype. The reader relay is available from National, part number IPC-16P/810R.

NOTE: THE READER RELAY WILL NOT NORMALLY BE NEEDED.

In order to connect teletype to PDS, the following steps must be followed. (The steps below are indicated by circled numbers in Figures 3.3 through 3.8.):

1. Install the relay board Paper Tape Reader drive circuit. A mounting tab with holes for mounting the board is located in the lower corner next to the keyboard. Mount the board with the components facing away from the keyboard, utilizing two number 6 screws and lockwashers. (See Figure 3.3.)
2. On jack 4 (male) at the rear of the teletype, remove the brown wire and pin from pin 11 and install them at pin 7. (See Figure 3.4.)
3. Connect a wire from plug 4 (female), pin 7, at the rear of the teletype (using the Molex terminal) to the SOL terminal lug on the relay board using the clip on the terminal. Also connect a wire from the SOL terminal lug to terminal L2 on the Line/Local Switch at the front of the teletype. (See Figures 3.4 and 3.5.)
4. Run a wire from L1 on the Line/Local Switch to the RSW terminal lug on the relay board using the clip on the terminal. (See Figure 3.5.)
5. Run a wire from pin 3 of the PDS "TTY" connector to the P5 terminal lug on the relay board using the clip on the terminal. (See Figures 3.5 and 3.6.)
6. Run a wire from pin 8 of the PDS "TTY" connector to the P6 terminal lug on the relay board using the clip on the terminal. (See Figures 3.5 and 3.6.)
7. Run a wire from pin 2 of the PDS "TTY" connector to pin 7 of the terminal strip at the rear of the teletype. (See Figure 3.6.)
8. Run a wire from pin 7 of the PDS "TTY" connector to pin 6 of the terminal strip at the rear of the teletype. (See Figure 3.6.)
9. Run a wire from pin 6 of the PDS "TTY" connector to pin 4 of the terminal strip at the rear of the teletype. (See Figure 3.6.)
10. Run a wire from pin 1 of the PDS "TTY" connector to pin 3 of the terminal strip at the rear of the teletype. (See Figure 3.6.)
11. Install the two thyrectors (transient suppressors) on the Line/Local Switch. Connect one between terminals L2 and 2 and the other between terminals L1 and 2. (See Figure 3.5.)

12. To set the TTY current source to 20mA, move the blue wire from terminal 3 to terminal 4 of the power resistor R1. (See Figure 3.7.)
13. To set the receive current to 20mA, move the purple wire from pin 8 to pin 9 on the terminal strip at the rear of the teletype. (See Figure 3.8.)
14. To configure the TTY for full duplex, move the white-blue wire from pin 4 to pin 6 of the terminal strip and move the brown-yellow wire from pin 3 to pin 5. (See Figure 3.8.)
15. To disable the automatic answerback option, locate the cavity behind the keyboard. Note that there are nine codebars located directly beneath the carriage, and at the front and rear of the codebars there are two additional bars, called codebar basket tie bars. These, plus the bars across the ends of the codebars, comprise the codebar basket tie bars. At the right front of the codebar basket is located a copper colored clip. Refer to *Teletype Bulletin 310B*, Volume 1, for an illustration of the codebar basket tie bar. Note that this is an illustration showing two types of clips and not actual location of the clip. On models where the clip is not provided, install a clip on the third slot from the right. A clip can be purchased from any local Teletype distributor. On models equipped with the clip, it will be found placed over the second slot. Move it to the third slot.
16. Before connecting the TTY cable to PDS, connect 110V to the TTY and put it in the line mode. Check to insure that 110V is not present on the connector that attaches to PDS. If 110V is present, check all connections to the Line/Local Switch. When there is no 110V present, the TTY may be connected to PDS.
17. Plug the PDS "TTY" connector into the connector at the rear of PDS labeled "TTY."

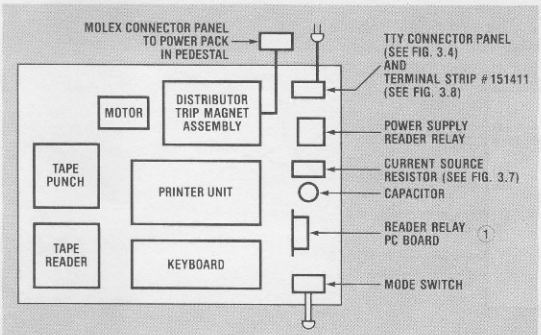



Figure 3.3 TTY Outline

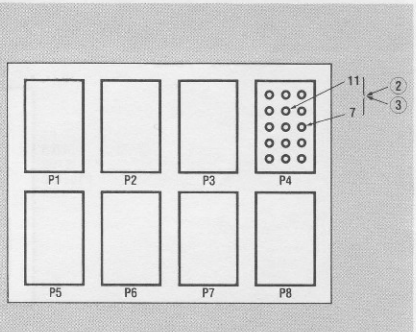



Figure 3.4 TTY Connector Panel

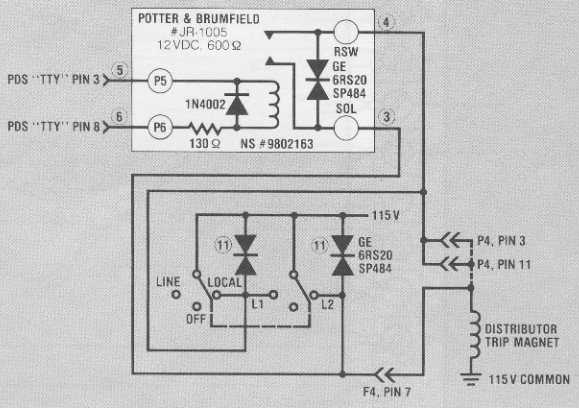



Figure 3.5 TTY Reader Relay Schematic

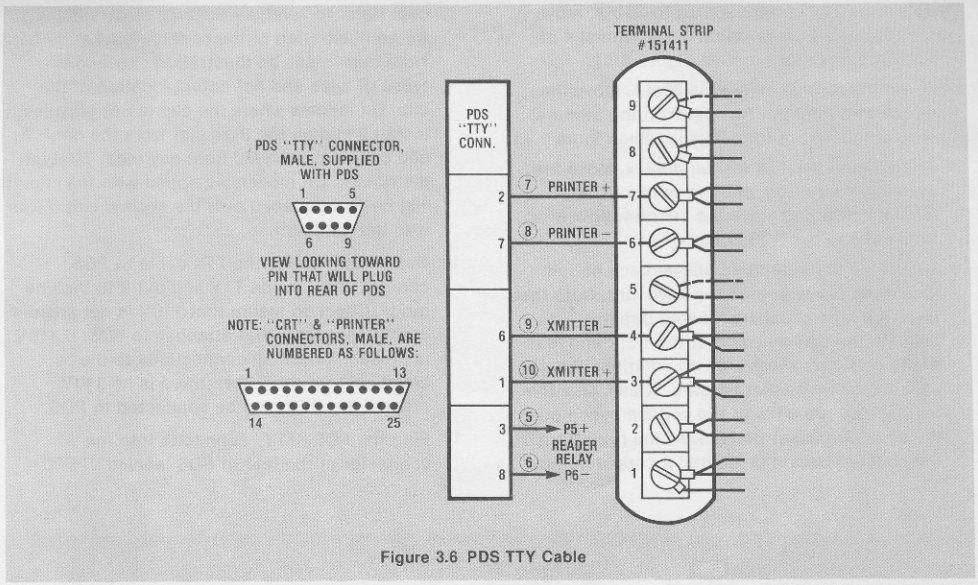



Figure 3.6 PDS TTY Cable

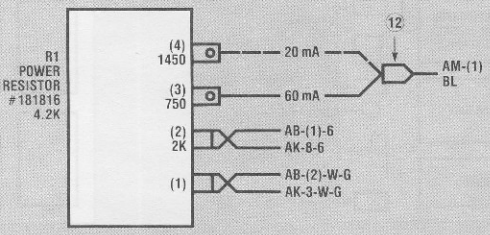



Figure 3.7 TTY Power Resistor

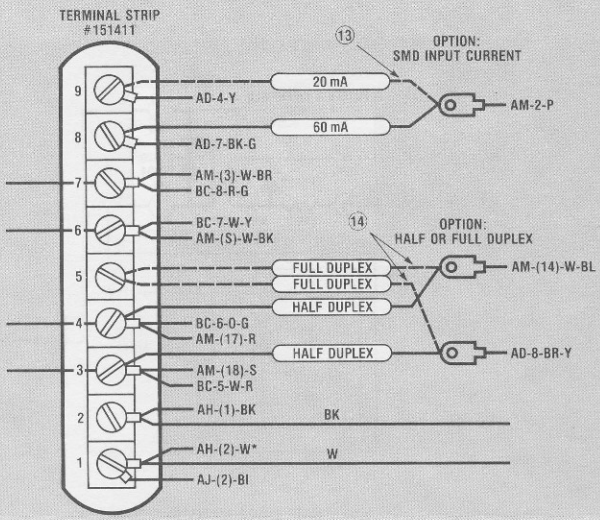



Figure 3.8 TTY Terminal Strip

The next step in PDS installation is to remove the PDS top cover by taking out the two screws located towards the rear of the cover, and slide the cover off. Then check to make sure that all six PC cards are seated firmly in their slots and all connectors are firmly fastened. Replace the cover.

The final step in installation is to connect the peripheral devices to PDS, plug the power cable into the rear of PDS, and supply power to PDS and all peripherals. At this point, PDS is ready for operation. The following verification procedure should be followed to verify proper operation. This procedure is designed not only to test PDS, but also to acquaint the user with the PDS system. The user should read this entire manual to familiarize himself with the system, then perform the verification procedure.

### 3.2 Verification of PDS Operation

The following verification procedure is intended to provide the user with both an introduction to system operation and a verification of system software and hardware. All user input is underlined and terminated by a carriage return. User responses to PDS interrogations are indicated (e.g., (CR)). The following five systems programs will be used:

1. DSKIT — to initialize a disk
2. FM — to verify the new disk and new files
3. EDIT — to create and edit a COP program
4. ASM — to assemble the COP program
5. COPMON — to debug the COP program

Before a disk can be used, it must be initialized, giving it a volume name and a directory. After the COP development system is turned on, a carriage return establishes the baud rate and calls EXEC.

Example:

```
Turn on Power and type (CR)
EXEC,REV:A
X>@DSKIT
DSKIT,REV:B
D>
```

Place metallized tape over Write Protect slot on new disk. Then insert disk. (Refer to Figure 4.1.) The initialize command allows the user to specify the volume name and header and gives the disk a directory.

```
D>I "PDSUSER","PDS USER"
READY TO RUN DESTRUCTIVE OPERATION ON DISK
  (Y/N,CR = YES)? (CR)
*** WRITING SECTOR MARKS ***
*** PERFORMING PATTERN TEST ***
*** BUILDING DIRECTORY ***
*** INITIALIZATION COMPLETE ***
```

After initialization is complete, the user may verify the operation by using FM to display the directory. A disk with FM on it must be inserted.

```
D>@FM
FM,REV:B
F>
```

Now the newly initialized disk is inserted.

```
F>D
DIRECTORY FOR: PDSUSER "PDS USER"
DIRECTORY EMPTY

SECTORS BAD:     0
SECTORS USED:    8
SECTORS FREE:  608
```

The user may obtain all the PDS master programs by duplicating that disk to his or her new disk. This is done with the duplicate command in FM. Each time the volume name is displayed, that disk must be inserted.

```
F>DU PDS TO PDSUSER

DUPLICATING VOLUME "PDS" TO VOLUME "PDSUSER"
  (Y/N,CR = YES)? (CR)

LOAD INDICATED VOLUME, PRESS RETURN
PDSUSER(CR)
PDS(CR)
PDSUSER(CR)

CREATING FILE PDSUSER:EDIT.MP

CREATING FILE PDSUSER:ASM.MP

CREATING FILE PDSUSER:COPMON.MP
PDS(CR)
PDSUSER(CR)

CREATING FILE PDSUSER:FM.MP

CREATING FILE PDSUSER:DSKIT.MP

END DUPLICATION
F>
```

### 3.3 Example Program

The user can enter a COP program using EDIT. The program to be created here will read a number from the COP420 "I" lines and add 5. The carry will be ignored. The result will be output on the "D" outputs, and the decoded 7-segment equivalent will appear on the "L" outputs. A 50% duty cycle square wave will appear on the SK output. The pulse width will increase with the magnitude of the above addition. As the user changes the data on the "I" inputs, there should be corresponding changes on the other outputs. These outputs may be examined and verified on an oscilloscope. The probes may be attached directly to the proper pins on the "COP" output cable from the emulator card. The program will be called COPEX.

```
F>@EDIT COPEX
EDIT,REV:B
CREATE NEW FILE (Y/N,CR = YES)? (CR)
AVAILABLE SECTORS: 496
E>I
    1?          .TITLE  COPEX, 'COP EXAMPLE'
    2?          CLRA
    3?          LEI  5          ; Q TO L, C TO SK ON XAS
    4? START:
    5?          ININ            ; READ I0-I3 TO A
```

```
   6?          AISC  5       ; ADD 5
   7?          OBD           ; OUTPUT A TO D0-D3
   8?          LB#           (ABORTED LINE TO INSERT A
                             NOP AFTER THE AISC)
   8? CR
E>I TO 7
   7?          NOP
   8? CR
E>I
   9?          LBI  0        ; SAVE ENTERED VALUE + 5 IN
   10?         X             ; M0
   11?         CLRA          ; SET UP A FOR
   12?         AISC  4       ; LQID ON PAGE 1
   13?         LQUID         ; PERFORM SEGMENT LOOKUP
   14?         RC
   15?         XAS           ; OUTPUT 0 TO SK
   16?         NOP
   17?         NOP           ; DELAY FOR 50% DUTY CYCLE
   18?         NOP
   19?         NOP
   20?         NOP
   21?         NOP
   22?         NOP
   23?         NOP
   24?         NOP
   25?         NOP
   26?         NOP
   27?         NOP
   28?         NOP
   29?         NOP
   30?         COMP          ; MAKE DELAY PROPORTIONAL
   31?         AISC  1       ; TO VALUE + 5
   32?         JP  .-1
   33?         SC
   34?         XAS           ; OUTPUT 1 TO SK
   35?         LD            ; GET ENTERED VALUE + 5
   36?         COMP          ; DELAY PROPORTIONAL TO
   37?         AISC  1       ; ENTERED VALUE + 5
   38?         JP  .-1
   39?         JP  START
   40?         .PAGE  1
   41?         .WORD  03F,006,05B,04F,066,06D,07D,
   42?         .WORD  007,07F,067 ; 0-9
   43?         .WORD  077,07C,039,05E,079,071 ; A-F
   44?         .END
   45?(CR)       (EXIT INPUT MODE)
E>FI
FINISH CURRENT EDIT  (Y/N,CR=YES)?(CR)
```

The user may verify the new program on the new disk by displaying the directory with FM.

```
E>@FM

FM,REV:B

F>D

DIRECTORY FOR:  PDSUSER   "PDS USER"

 FN  D NAME          TYPE          SIZE   PL   VN
  1    EDIT    .MP   MAIN PROGRAM    20    2    3
  2    ASM     .MP   MAIN PROGRAM    32    2    3
  3    COPMON  .MP   MAIN PROGRAM    32    2    3
  4    FM      .MP   MAIN PROGRAM    16    2    3
  5    DSKIT   .MP   MAIN PROGRAM    12    2    3
  6    COPEX   .SRC  SYMBOLIC         4    2    3

SECTORS BAD:    0
SECTORS USED:  124
SECTORS FREE:  492
```

The user may now assemble the COP program, displaying the assembly errors on the console.

```
F>@ASM

ASM,REV:B

A>I=COPEX,O=COPEX,L=*CN,EL

CREATING FILE PDSUSER:COPEX.LM

END PASS 1

COP CROSS ASSEMBLER  PAGE  1
COPEX  COP EXAMPLE

  13 00D 00          LQUID     ; PERFORM SEGMENT LOOKUP
ERROR UNDEFINED  @

  1 ERROR LINES

  56 ROM WORDS USED

END PASS 4

SOURCE CHECKSUM = E88F

OBJECT CHECKSUM = 0276

INPUT FILE      PDSUSER:COPEX.SRC

OBJECT FILE     PDSUSER:COPEX.LM
A>
```

The above assembly error ("LQUID" should be "LQID") can be edited with EDIT.

```
A>@EDIT COPEX

EDIT,REV:B

AVAILABLE SECTORS:   488
INPUT FILE SECTORS:    4
E>RE

EOF AT   44

E>L 10/L

  10                X
  11                CLRA        ; SET UP A FOR LQID ON
  12                AISC   4    ; PAGE 1
  13                LQUID       ; PERFORM SEGMENT LOOKUP
  14                RC
  15                XAS         ; OUTPUT O TO SK
  16                NOP
  17                ***

E>
```

The listing was interrupted by the user pressing a key when the error was located. The "LQUID" is replaced by a "LQID."

```
E>E 13

  13                LQUID       ; PERFORM SEGMENT LOOKUP
EDITS?              LQID        ; PERFORM SEGMENT LOOKUP
  13                LQID        ; PERFORM SEGMENT LOOKUP
EDITS?(CR)

E>FI

FINISH CURRENT EDIT  (Y/N,CR=YES)?(CR)

OK TO DELETE FILE PDSUSER:COPEX.SRC  (Y/N,CR=YES)?(CR)

E>
```

The user may now re-assemble the corrected program, obtaining an assembly load module file (COPEX.LM) and a full assembly output listing.

```
E>@ASM I=COPEX,O=COPEX,L=*PR

ASM,REV:B

OK TO DELETE FILE PDSUSER:COPEX.LM  (Y/N,CR=YES)?(CR)

CREATING FILE PDSUSER:COPEX.LM

END PASS 1

END PASS 4
A>
```

Notice that the listing was assigned to the printer. The printer listing is shown below. No assembly errors occurred.

```
COP CROSS ASSEMBLER   PAGE  1
COPEX   COP EXAMPLE

   1                .TITLE  COPEX, 'COP EXAMPLE'
 2 000  00          CLRA
 3 001  3365        LEI     5   ; Q TO L, C TO SK ON XAS
 4             START:
 5 003  3328        ININ        ; READ I0-I3 TO A
 6 005  55          AISC    5   ; ADD 5
 7 006  44          NOP
 8 007  333E        OBD         ; OUTPUT A TO D0-D3
 9 009  0F          LBI     0   ; SAVE ENTERED VALUE + 5
10 00A  06          X           ; IN M0
11 00B  00          CLRA        ; SET UP A FOR LQID ON
12 00C  54          AISC    4   ; PAGE 1
13 00D  BF          LQID        ; PERFORM SEGMENT LOOKUP
14 00E  32          RC
15 00F  4F          XAS         ; OUTPUT 0 TO SK
16 010  44          NOP
17 011  44          NOP         ; DELAY FOR 50% DUTY CYCLE
18 012  44          NOP
19 013  44          NOP
20 014  44          NOP
21 015  44          NOP
22 016  44          NOP
23 017  44          NOP
24 018  44          NOP
25 019  44          NOP
26 01A  44          NOP
27 01B  44          NOP
28 01C  44          NOP
29 01D  44          NOP
30 01E  40          COMP        ; MAKE DELAY PROPORTIONAL
31 01F  51          AISC    1   ; TO ENTERED VALUE + 5
32 020  DF          JP      .-1
33 021  22          SC
34 022  4F          XAS         ; OUTPUT 1 TO SK
35 023  05          LD          ; DELAY PROPORTIONAL TO
36 024  40          COMP        ; ENTERED VALUE + 5
37 025  51          AISC    1
38 026  E5          JP      .-1
39 027  C3          JP      START
40      0040        .PAGE   1
41 040  3F          .WORD   03F,006,05B,04F,066,06D,07D
   041  06
   042  5B
   043  4F
   044  66
   045  6D
   046  7D
42 047  07          .WORD   007,07F,067; 0-9
   048  7F
   049  67

COP CROSS ASSEMBLER   PAGE  2
COPEX   COP EXAMPLE

43 04A  77          .WORD   077,07C,039,05E,079,071 ; A-F
   04B  7C
   04C  39
   04D  5E
   04E  79
   04F  71
44                  .END

COP CROSS ASSEMBLER   PAGE  3
COPEX   COP EXAMPLE

START  0003

NO ERROR LINES

 56 ROM WORDS USED

SOURCE CHECKSUM = E85A

OBJECT CHECKSUM = 027C

INPUT FILE PDSUSER:COPEX.SRC

OBJECT FILE PDSUSER:COPEX.LM
```

The new program may be tested now using COPMON. 420 is the correct chip number. To make it easier to see the program, zeros are deposited in shared memory before loading the new program COPEX.

```
A>@COPMON

COPMON,REV:B

CHIP NUMBER (DEFAULT = 420)? (CR)

C>DE 0,0/L

C>LO COPEX

FINISHED LOADING
```

To begin execution of the program, first reset the COP, then start by giving the 'GO' command.

```
C>RE

CHIP IS RESET

C>G
```

On examining the outputs, it is discovered that the "L" outputs have the proper values, but the "D" lines do not. Also, the "square wave" on the "SK" line is incorrect. Only one half of the cycle varies with the input. A good beginning for debugging is to obtain a trace and examine the "path" of the COP.

```
C>TR

TRACE ENABLED:
 A:001 OCCUR:  1 PRIOR:  O GO:N

C>RE

CHIP IS RESET

C>G
TRACED ON  A:001 AT A:001
C>T

  0   0 A:001        E:1111
  1   1 A:002  SKIP  E:1111
  2   2 A:003        E:1111
  3   3 A:004  SKIP  E:1111
  4   4 A:005        E:1111
  5   5 A:006  SKIP  E:1111
  6   6 A:007        E:1111
  7   7 A:008  SKIP  E:1111
  8   8 A:009        E:1111
  9   9 A:00A        E:1111
 10  10 A:00B        E:1111
 11  11 A:00C        E:1111
 12  12 A:00D        E:1111
 13  13 A:044  SKIP  E:1111
 14  14 A:00E        E:1111
 15  15 A:00F        E:1111

C>T

 16  16 A:010        E:1111
 17  17 A:011        E:1111
 18  18 A:012        E:1111
 19  19 A:013        E:1111
 20  20 A:014        E:1111
 21  21 A:015        E:1111
 22  22 A:016        E:1111
 23  23 A:017        E:1111
 24  24 A:018        E:1111
 25  25 A:019        E:1111
 26  26 A:01A        E:1111
 27  27 A:01B        E:1111
 28  28 A:01C        E:1111
 29  29 A:01D        E:1111
 30  30 A:01E        E:1111
 31  31 A:01F        E:1111

C>T

 32  32 A:020        E:1111
 33  33 A:01F        E:1111
 34  34 A:020        E:1111
 35  35 A:01F        E:1111
 36  36 A:020        E:1111
 37  37 A:01F        E:1111
 38  38 A:020        E:1111
 39  39 A:01F        E:1111
 40  40 A:020        E:1111
 41  41 A:01F        E:1111
 42  42 A:020        E:1111
 43  43 A:01F        E:1111
 44  44 A:020        E:1111
 45  45 A:01F        E:1111
 46  46 A:020        E:1111
 47  47 A:01F        E:1111

C>T

 48  48 A:020        E:1111
 49  49 A:01F        E:1111
 50  50 A:020        E:1111
 51  51 A:01F        E:1111
 52  52 A:020        E:1111
 53  53 A:01F        E:1111
 54  54 A:020        E:1111
 55  55 A:01F        E:1111
 56  56 A:020        E:1111
 57  57 A:01F        E:1111
 58  58 A:020        E:1111
 59  59 A:01F        E:1111
 60  60 A:020        E:1111
 61  61 A:01F        E:1111
 62  62 A:020  SKIP  E:1111
 63  63 A:021        E:1111
```

The word "SKIP" indicates that the instruction was skipped. It also appears on the second half of two-word instructions. Notice that at trace location 13 the address is 44. This is actually the second half of the "LQID" instruction, and is the address of the data to be loaded into the "Q" register. The second instruction, "LEI 5," assigns the "Q" register to the "L" outputs. By looking at the trace, one sees that program execution has proceeded as expected, except that the loop at locations 1F and 20 was done 15 times. By examining the listing at those locations one sees that the accumulator wasn't loaded with the entered value before the first loop. The "LD" instruction before the "COMP" instruction was omitted. Single-stepping through the first several locations allows the user to inspect the COP registers, particularly the accumulator and the "B" register.

```
C>R

CHIP IS RESET

C>AU ALL

C>S

STEP A:0 B:00 C:0 G:0 I:F L:FF Q:66 S:F P:001
M0:FFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFF M3:3377777777777777

C>(CR)

STEP A:0 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:003
M0:FFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFF M3:3377777777777777

C>(CR)

STEP A:F B:00 C:0 G:0 I:F L:66 Q:66 S:F P:005
M0:FFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFF M3:3377777777777777

C>(CR)

A:006 SKIPPED

STEP A:4 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:007
M0:FFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFF M3:3377777777777777

C>(CR)

STEP A:4 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:009
M0:FFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFF

C>
```

From looking at the assembly listing, one sees that location 7 has the "OBD" instruction which puts the "B" register out to the "D" lines. After executing this instruction "B" still has zero but "A" has the correct value. A "CAB" instruction is necessary before the "OBD." Both of the mistakes in this program require instructions to be inserted when it is edited. But the "NOP" at location 10 may be easily replaced with a LD instruction, giving a much better square wave. After starting the chip, the square may be displayed again.

```
C>PU 1D,LD

C>CL

BRKPT AND TRACE CLEARED

C>G

C>
```

The program may now be re-edited.

```
C>@EDIT COPEX

EDIT,REV:B

AVAILABLE SECTORS:  480
INPUT FILE SECTORS    4
E>RE

EOF AT  44
E>L

  1              .TITLE COPEX, 'COP EXAMPLE'
  2              CLRA
  3              LEI     5    ; Q TO L, C TO SK ON XAS
  4      START:
  5              ININ         ; READ I0-I3 TO A
  6              AISC    5    ; ADD 5
  7              NOP
  8              OBD          ; OUTPUT A TO D0-D3
  9              LBI     0    ; SAVE ENTERED VALUE + 5
 10              X            ; IN M0
 11              CLRA         ; SET UP A FOR
 12# ***

E>
```

The missing CAB instruction should be inserted to line 8.

```
E>IN TO 8

  8?            CAB
  9?  (CR)

E>L

  1            .TITLE  COPEX, 'COP EXAMPLE'
  2              CLRA
  3              LEI     5    ; Q TO L, C TO SK ON
  4 START:                    ; XAS
  5              ININ         ; READ I0-I3 TO A
  6              AISC    5    ; ADD 5
  7              NOP
  8              CAB
  9              OBD          ; OUTPUT A TO D0-D3
 10              LBI     0    ; SAVE ENTERED VALUE + 5
 11              X            ; IN M0
 12              CLRA         ; SET UP A FOR
 13              AISC    4    ; LQID#***

E>L 25

 25             NOP

E>N 21

 26             NOP
 27             NOP
 28             NOP
 29             NOP
 30             NOP
 31             COMP         ; MAKE DELAY PROPORTIONAL
 32             AISC    1    ; ENTERED VALUE + 5
 33             JP      .-1
 34             SC
 35             XAS          ; OUTPUT 1 TO SK
 36             LD           ; G#***

E>
```

The missing LD instruction should be inserted to line 31.

```
E>IN to 31

 31?              LD          ; GET ENTERED VALUE + 5
 32?  (CR)

E>L 25

 25               NOP

E>N 21

 26               NOP
 27               NOP
 28               NOP
 29               NOP
 30               NOP
 31               LD          ; GET ENTERED VALUE + 5
 32               COMP        ; MAKE DELAY PROPORTIONAL
 33               AISC  1     ; TO ENTERED VALUE + 5
 34               JP    .-1
 35               SC
 36               XAS         #***

E>
```

The edit mode may be finished now, replacing the old program with the new one.

```
E>FI

FINISH CURRENT EDIT  (Y/N,CR=YES)?(CR)

OK TO DELETE FILE PDSUSER:COPEX.SRC  (Y/N,CR=YES)?(CR)

E>
```

The new program may be verified by re-assembling and testing with COPMON.

```
E>@ASM I=COPEX,O=COPEX,L=*PR

ASM,REV:B

OK TO DELETE FILE PDSUSER:COPEX.LM  (Y/N,CR=YES)?(CR)

CREATING FILE PDSUSER:COPEX.LM

END PASS 1
END PASS 4

A>
```

The new assembled program may be tested with COPMON and an oscilloscope as before to verify proper performance.

```
A>@COPMON

COPMON,REV:B

CHIP NUMBER  (DEFAULT=420) ?(CR)

C>LO COPEX

FINISHED LOADING

C>RE

CHIP IS RESET

C>G
C>
```

Now that both the source and load module files are correct, the deleted versions may be packed with the commands in file manager, giving more room on the disk for new programs.

```
C>@FM

FM,REV:B

F>
```

Now the new disk is examined and packed.

```
F>D

DIRECTORY FOR:  PDSUSER    "PDS USER"

 FN  D NAME          TYPE          SIZE   PL   VN
  1    EDIT    .MP   MAIN PROGRAM    20    2    3
  2    ASM     .MP   MAIN PROGRAM    32    2    3
  3    COPMON  .MP   MAIN PROGRAM    32    2    3
  4    FM      .MP   MAIN PROGRAM    16    2    3
  5    DSKIT   .MP   MAIN PROGRAM    12    2    3
     * COPEX   .SRC  SYMBOLIC         4    2    1
     * COPEX   .LM   LOAD MODULE      4    2    1
     * COPEX   .SRC  SYMBOLIC         4    2    2
     * COPEX   .LM   LOAD MODULE      4    2    2
  6    COPEX   .SRC  SYMBOLIC         4    2    3
  7    COPEX   .LM   LOAD MODULE      4    2    3

SECTORS BAD      0
SECTORS USED:  144
SECTORS FREE:  472

F>P

PACKING DISK  (Y/N,CR=YES)?(CR)

F>D

DIRECTORY FOR  PDSUSER    "PDS USER"

 FN  D NAME          TYPE          SIZE   PL   VN
  1    EDIT    .MP   MAIN PROGRAM    20    2    3
  2    ASM     .MP   MAIN PROGRAM    32    2    3
  3    COPMON  .MP   MAIN PROGRAM    32    2    3
  4    FM      .MP   MAIN PROGRAM    16    2    3
  5    DSKIT   .MP   MAIN PROGRAM    12    2    3
  6    COPEX   .SRC  SYMBOLIC         4    2    3
  7    COPEX   .LM   LOAD MODULE      4    2    3

SECTORS BAD:     0
SECTORS USED:  128
SECTORS FREE:  488
```

# 4 Introduction to PDS Software

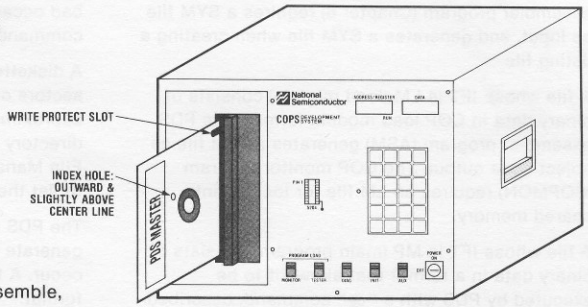



**Figure 4.1 Inserting a Diskette into the Drive**

PDS software enables the user to edit, assemble, and debug COP programs. This chapter will provide the user with a thorough introduction to these programs.

PDS software is divided into two parts:

1. Firmware in ROM.
2. Software on disk which can be loaded into RAM and executed.

The firmware includes general routines for console, printer, disk, front panel I/O system initialization, diagnostics, and disk file manipulation. The software includes a file manager, text editor, assembler, COP monitor, and diskette initialization and test programs.

## 4.1 Disk Files

A disk file is a collection of data stored on a disk and given a name. (The words "disk," "diskette," and "disc" are used interchangeably throughout this manual.) The PDS filename has the following syntax:

[VOLUME NAME:] NAME [.MODIFIER]

The brackets ([ ]) around a term indicate that the term is optional and may be left off. An example of a filename is PDS:SAMPLE.SRC. The volume name is PDS, the name is SAMPLE, and the modifier is SRC.

The "volume name" is a name given to a *diskette.* All files on a diskette have the same volume name. The volume name is given to the diskette when it is "initialized" (Chapter 6) and can be changed with the file manager program (Chapter 5). It may consist of one to eight alphanumeric characters — *blanks and special characters are not permitted.* The volume name is optional in a filename. If PDS encounters a filename with no volume name, it will use the volume name of the diskette that is currently in the PDS disk drive. If given, the volume name must be separated from the remainder of the file name by a colon.

The "name" part of a filename may consist of one to eight alphanumeric characters. *The first character must be alphabetic — blanks and special characters are not permitted.*

The "modifier" part of a filename may consist of up to three alphanumeric characters — *blanks and special characters are not permitted.* It is separated from the beginning part of the filename by a period. A period with no character following it specifies a modifier with zero characters. The modifier is usually used to describe the type of a file. For example, "SRC" is used for text files and "MP" is used for PDS system program files. This convention is not mandatory. The user may choose any modifier he wishes. The modifier and its preceding period are optional. If left off, PDS will provide a default modifier. Table 4.1 lists the default modifiers.

**Table 4.1. System Default Modifiers**

| Modifier | Definition |
|---|---|
| SRC | Source Text |
| LM | COP Load Module |
| MP | PDS System Program |
| LST | Listing File |
| SYT | Special System File |

Each file on a diskette has a unique NAME.MODIFIER combination. The user creates files using the PDS file manager, text editor, or assembler programs. PDS maintains a directory on each diskette, describing the name and other information for each file on it. The directory can be listed by using the PDS file manager program (Chapter 5).

Each file has a special number called an Internal File Type (IFT) maintained by PDS in the diskette directory. The IFT is not alterable by the user. It is used by PDS to indicate the type of data in each file (source text, system program data, etc.). This allows PDS to prevent the user from accidentally assembling a binary data file, or attempting to execute a source text file. The IFT is not related to the file modifier. The modifier is selected by the user; the PDS selects the correct IFT regardless of what modifier is used. Table 4.2 lists the PDS IFTs.

A file whose IFT is SYM (symbolic) consists of ASCII data written on the disk. The PDS file manager program (Chapter 5) generates the SYM file type when copying ASCII data to the disk or when copying another SYM file. The PDS text editor program (Chapter 7) requires a SYM file when reading data from the disk, and generates a SYM file when writing data to the disk. The PDS

assembler program (Chapter 8) requires a SYM file as input, and generates a SYM file when creating a listing file.

A file whose IFT is LM (load module) consists of binary data in COP load module format. The PDS assembler program (ASM) generates an LM file as object code output. The COP monitor program (COPMON) requires an LM file for loading into shared memory.

A file whose IFT is MP (main program) consists of binary data in a format that allows it to be executed by PDS with a "@" command, described later in this chapter. The PDS programs FM and EDIT are examples of this file type.

**Table 4.2. PDS Internal File Types**

| File Type | Definition |
|---|---|
| SYM | Symbolic Text |
| LM | COP Load Module |
| MP | PDS System Program |

PDS maintains another special number for each file, called a "protection level." This is used to prevent accidental destruction of files. Table 4.3 is a list of protection levels and their safeguard provisions. System programs such as FM and EDIT create files for the user as level 2 files. All system programs are initially level 3 files. The protect level of any file can be changed with the File Manager PROTECT command (Chapter 5).

If PDS is directed to write into an *existing file,* it wll delete the existing file and *recreate* a new file of the same name, type, and protection level. A file cannot be recreated if its protection level is 3, and if its protection level is 2, the user must give permission for recreation. A deleted file is not removed from the diskette. It still exists and can be undeleted with the File Manager UNDELETE command, provided that the disk has not been packed (Chapter 5).

**Table 4.3. Protection Levels and Safeguard Provisions**

| | User Notified of Creation? | User Approval Required to Delete or Modify File? |
|---|---|---|
| 0 | No | No |
| 1 | Yes | No |
| 2 | Yes | Yes |
| 3 | Yes | Delete/Modify not allowed |

A third special number for each file, called the "version number," is set to 1 the first time the file is created. Each time the file is recreated, as, for example, when a text file is edited using the editor program, the version number is incremented. This number is useful to help keep an up-to-date backup of a file. It is recommended that the user always keep a backup of every file, because diskettes go bad occasionally. The File Manager DUPLICATE command is used to back up a file (Chapter 5).

A diskette is divided into sectors. There are 616 sectors on each diskette. One sector will hold approximately 20 average lines of text. The diskette directory requires at least 8 sectors of its own. The File Manager DIRECTORY command can be used to list the size of each file (Chapter 5).

The PDS disk file manipulation routines will generate error messages when certain conditions occur. A file error message has the following format:

```
DISK ERROR, FILE filename
error message 1
[error message 2]
```

Table 4.4 is a list of the error messages and their meanings. Normally only the first nine messages given in the table will occur. In some messages there is no filename involved, in which case only ":." will be printed for the file name. Sometimes two error messages will be printed.

**Table 4.4. Disk File Error Messages**

| Message | Meaning |
|---|---|
| WRONG DISK VOLUME | User referred to a file on a diskette other than the one in the drive. |
| DRV NOT RDY | No disk in drive, drive door isn't shut, or diskette is jammed. |
| FILENAME SYNTAX | User typed an illegal filename. |
| END OF FILE | User tried to read past the end of the file while using the text editor. |
| END OF DISK | Diskette is full, no more data can be stored on it. See warning in Chapter 7 concerning this error. |
| CANT DELETE | Attempt to delete a file whose protect level is 3, or user didn't give permission to delete a file whose protect level is 2. |
| ILLEGAL DEVICE | User referred to an illegal device. |
| FILE NOT FOUND | Reference was made to a file that is not on the diskette. |
| NO SYNC/WRT PRTCT | Attempt to write on write-protected diskette, or else disk is bad. |
| WRT CRC ERR | Couldn't write on disk, disk may be bad. |
| RD CRC ERR | Couldn't read from disk, disk may be bad. |
| CANT RD NST | Drive not ready or disk is bad. |
| DISK/DIR FULL | Diskette is full, no more data can be stored on it. See warning in Chapter 7 concerning this error. |
| CANT RD DIR | Drive not ready or disk is bad. |
| CANT WRT NST | Drive not ready or disk is bad. |
| CANT WRT DIR | Disk may be bad. |
| RD ERR | Disk is bad. |
| WRT ERR | Disk is bad. |
| CANT MODIFY | Attempt to modify a file whose protect level is 3, or user didn't give permission to modify a file whose protect level is 2. |
| ADDR ERR | System hardware or software error. |
| ILLEGAL CMD | System hardware or software error. |

**Table 4.4. Disk File Error Messages (continued)**

| Message | Meaning |
|---|---|
| NO DISKIO ERRS | System hardware or software error. |
| NO ERRS | System hardware or software error. |
| NOT OPEN FOR RD | System hardware or software error. |
| NOT OPEN FOR WRT | System hardware or software error. |
| NOT OPEN FOR MOD | System hardware or software error. |
| ALREADY OPEN | System hardware or software error. |
| TOO MANY FILES | System hardware or software error. |
| NST/DIR DONT MATCH | System hardware or software error. |
| PAST END OF DIR | System hardware or software error. |
| BAD CHNL TBL | System hardware or software error. |
| NO END OR DIR | System hardware or software error. |
| TOO MANY VOLUMES | System hardware or software error. |

In a few system commands a "device name" is acceptable in place of a filename. A device name is specified by an asterisk followed by two alphabetic characters indicating a peripheral device. At present, only two device names are allowed. These are shown in Table 4.5.

**Table 4.5. PDS Device Names**

| Name | Device |
|---|---|
| *CN | System Console |
| *PR | Printer |

## 4.2 PDS Commands

All PDS system programs print a "sign-on" line when they are called by the user. When a system program is ready to accept a command from the user, it issues a "prompt." Each PDS system program has a unique prompt. Table 4.6 lists these programs and their prompts. The prompt indicates to the user that a program is ready to accept a *command.* Each system program has its own set of commands. Chapters 5 through 11 of this manual describe these programs and their commands.

**Table 4.6. PDS System Program Names and Prompts**

| System Program Name | Function | Prompt |
|---|---|---|
| ASM* | COP macro assembler | A> |
| COPMON* | COP monitor | C> |
| DSKIT* | Disk initialization and test | D> |
| EDIT* | Text file editor | E> |
| FM* | File manager program | F> |
| LIST* | Text file listing | L> |
| XREF* | COP program cross reference | R> |
| EXEC** | PDS executive program | X> |

*System Program on Master Diskette.

**System Program in Firmware.

To invoke a command, the user types the command name and operands on the console, followed by a *carriage return* (CR). Only the first two characters of the command name need be typed — characters following the second character are ignored. Many commands require only the *first* character to invoke them.

Some commands require *operands.* An operand is typed by the user on the same line as the command name but separated from it by one or more spaces. If there is more than one operand, they are separated from each other by commas or spaces, depending on the command.

Every command line must be terminated by a *carriage return* before PDS will perform the command.

In this manual each command will be described by giving its syntax, a description, and an example. Syntax refers to the type of operands that the command requires. Below is an example of a typical command description:

COMBINE Command —

Syntax:
COMBINE filename,filename [filename . . .] TO filename

Description: Combine the specified files into a new file with the given name. Files must be of type Symbolic.

Example:
F>C TEST1.SRC,TEST2.SRC,TEST3.SRC TO NEW.SRC
CREATING FILE CDS:NEW.SRC

Following "Syntax:" the command name (COMBINE) is given with the first one or two characters (C) underlined to indicate how many characters are necessary to invoke the command. Following the command name are the operands. The meaning of the operands is in most cases obvious. Here, for example, "filename" refers to a PDS filename, described earlier in this chapter. If the meaning of the operands is not obvious, they will be described in the "description." Brackets ([ ]) around operands indicate that the operand is optional and may be left off. If the reason for an optional operand is not obvious, it will be described in the description. Here, for example, it is obvious that if a third initial filename is given, it will also be included in the combined file. The ellipsis (. . .) indicates that an operand may be repeated several times. Capital letters such as "TO" in the operand field indicate that these letters are to be typed exactly as they are.

Following "Example:" are the program prompt characters (F>) and an example of the command, entered by the user.

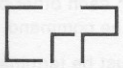
Command characters typed by the user are UNDERLINED UPPER-CASE characters. Characters typed by PDS are non-underlined UPPER-CASE characters. Lower-case characters such as range or filename represent ranges of values or names selected and typed by the user. Comments appearing in the

normal text typeface are explanatory only and are not part of the actual programs.

In some command descriptions one or more letters will be circled. This represents a control character. For example, (CR) represents a carriage return.

### 4.3 System Initialization

When PDS is powered up using the front panel key switch, or initialized using the front panel "INIT" switch, it displays



on the front panel display and waits for the user to press the "carriage return" (CR) key on the system console. When the user presses this key, PDS determines the console baud rate and type (RS232 or current loop). Note that if the console uses a *current loop* interface, PDS will assign it to a baud rate of 110 with the following setup characteristics: 8-bit data (No Parity — PDS resets bit 8 = 0), 2 Stop bits, Full Duplex operation. If the peripheral is to be set up differently, the user must follow the (CR) with an @@CONSOLE System Command to define the setup parameters of the current loop console. (See Section 4.6.) It then responds with:

```
EXEC,REV:A
X>
```

This indicates that the PDS "executive program" is ready to accept a user command. The executive program, named EXEC, is a firmware program responsible for initializing various PDS parameters when the system is powered up. The first line of the response is called the "sign-on line," and gives the program name (EXEC) and revision level (A). The second line gives the EXEC program prompt, "X>."

If a system console is not available, some PDS functions can be performed with the front panel. The "INIT" button can be used to re-initialize the system, in the same way as if power is turned off and then on again. When CrP is displayed on the PDS front panel during initialization, there are four allowable responses by the user:

1. Press carriage return (CR) on the console for normal console operation.
2. Press the "MONITOR" switch to load COPMON from disk. (See Chapter 9.)
3. Press the "TESTER" switch to load COPTST from disk. (COPTST is discussed in *COP400 Chip Tester Manual,* Order No. 420305786-001.)
4. Press the "DIAG" switch to perform a seven minute PDS diagnostic test (described in Section 4.4 below).

### 4.4 Diagnostics

The PDS executive program EXEC has only a single command. This command causes a PDS diagnostic test to be performed:

Syntax: DIAGNOSE

Description: A seven minute memory diagnostic test is performed by PDS, followed by a brief disk drive test. If the memory test passes, the message:

MEMORY TEST PASSED

is displayed on the console. If it doesn't pass, the incorrect memory location will be displayed on the console. (Servicing by National will be necessary.) An initialized disk must be inserted in the disk drive for the disk test to succeed. If it does, the message:

DIAGNOSTICS PASS

will be displayed on the console. If not, the message:

DISK TEST FAILED

will be displayed.

Example:
```
X>D
DIAGNOSTICS PASS
```

As discussed previously, the diagnostic test may be performed, if a console is not available, by pressing the front panel "DIAG" switch. If the memory test (which takes about seven minutes) fails, the fail address will be given in the left side of the front panel display, and the test type (address, word, or bit) will be given in the right side. If this occurs, servicing by National is necessary. If the memory test passes, the disk test will be performed. As with the console diagnostic operation, it takes only a few seconds and requires that an initialized diskette be in the disk drive. If this test fails, diSC Err5 will be displayed on the front panel. If both tests pass, diAG PASS will be displayed on the front panel.

After diagnostics have been performed, the user can return to console operation by pressing SFUNC C TERM on the front panel keypad. This will cause re-prompting for a console carriage return (CrP).

### 4.5 Console Input

PDS uses a console input routine which has several features that allow the user to correct typing mistakes. Among the features are the ability to backspace and to abort a line using control characters. Table 4.7 describes the various control characters and their function. These control characters can be used at *any time* when the user is typing on the PDS console. If a hardcopy console is being used, most of the control characters will not be useful because of the inability to back up and change characters that have already been typed. However, the shift/O, control/Q, control/I, and carriage return characters will be useful.

**Table 4.7. PDS Console Input Control Characters**

| Character | Function |
|---|---|
| Control/H | Backspace 1 character but do not delete the character that is backspaced over. |
| Shift/0 ("←" on some TTYs) | Delete one character back. |
| Control/Q | Abort line and try again. |
| Carriage return | Line is completed. Must be entered at end of each line. |
| Control/T or Control/I | Tab. (See @@TAB Command for setting tabs.) |
| Control/X | Delete character at current CRT cursor position. |
| Control/L | Forward space one character. |
| Control/A | Insert characters before current cursor character. |
| Control/B | Backspace one word. |
| Control/F | Forward space one word. |
| Control/C | Forward space to third tab position (for comments). |
| Control/D | Same as carriage return except line is truncated at current cursor position. |
| Control/E | Forward space to end of line. |
| Control/S | Backspace to start of line. |
| Control/O or Control/Z | Forward space to next occurrence of next character typed. |
| Control/P or Control/W | Forward space one character beyond next occurrence of next character typed. |

Note: If no characters have been typed yet on a line, forward spacing will space over the last line typed, a useful means of repeating the last line. If the last line ended in "*PR," it will not appear on the repeated line.

## 4.6 PDS "System" Commands

Certain PDS system parameters can be changed by the user with "system" commands. A system command is invoked by typing "@@" followed by a command name and operands and may be entered at any time (while in any system program).

### @@CONSOLE Command

Syntax: @@CONSOLE baud[,type[,parity[,crdly[,lfdly]]]]

Description: Set console parameters. *Baud* must be one of the baud rates given in Table 2.1. *Type* must be an "R" for RS232 or a "C" for current loop console. *Parity* must be "E" for even parity or "N" for no parity. *Crdly* must be a number from 0 to 1000 representing carriage return delay in milliseconds. *Lfdly* is for line feed delay. Default parameters are RS232, no parity, zero delays. Console parameters are automatically set up when (CR) is typed at PDS initialization.

Example:
```
X>@@C 1200,R,N,10,5
```

### @@PRINTER Command

Syntax:

@@PRINTER baud [,type[,parity[,crdly[,lfdly [,ffdly[,vtdly]]]]]]

Description: Set printer parameters. Parameter description and defaults are same as for @@CONSOLE command, except that form feed and vertical tab delays are added. At system initialization time, these parameters are set to 1200 baud, RS232, no parity, zero delays.

Example:
```
X>@@P 110,C,E,20,20,500,100
```

### @@TAB Command

Syntax: @@TAB [$t_1$,[$t_2$[,$t_3$]]]

Description: Set tab columns for control/T or control/I input line control characters. Three tab columns can be set. Initial and default tabs are columns 9, 17, and 33, standard for COP400 programs.

Example:
```
X>@@T 10,20,30
```

### @@WIDTH Command

Syntax: @@WIDTH number of columns

Description: Set Printer and Console column width. At system initialization this parameter is set to 72. Minimum setting is 10, maximum setting is 80.

Example:
```
X>@@WI 80
```

## 4.7 Printer Output

IF a PDS command line has "*PR" at the end of it, PDS will direct output generated by that command to the printer. This can be done with any PDS system program command. *If a printer is not connected to the system, PDS will wait until one is connected. The system must be re-initialized to terminate this "wait" state.*

Example:

```
F>C TEST1.SRC,TEST2.SRC TO TEST3.SRC *PR
CREATING FILE CDS:TEST3.SRC
```
(This line is printed on the printer.)

## 4.8 PDS System Software

PDS is shipped from the factory with two diskettes, labeled "PDS MASTER." The PDS MASTERS contain all PDS system software. They are identical, to be used to create a new master if the old one is inadvertently destroyed. The MASTER disks contain the PDS system program files, of which there are currently seven. Figure 4.6 lists the names of these programs (marked with an asterisk, "*"). They are used to perform the following functions:

1. Disk file management
2. Text file editing
3. Text file assembly
4. Object code execution
5. Miscellaneous functions

Disk file management involves copying files, duplicating diskettes, deleting files, listing diskette directories, etc. This is accomplished with the file manager program (FM), discussed in Chapter 5.

Another disk file function is the "initialization" of new diskettes, accomplished with the disk initialization and test program (DSKIT), discussed in Chapter 6.

Text file editing involves creation of text programs and altering, inserting, and deleting lines within such files. This is accomplished with the text editor program (EDIT), discussed in Chapter 7.

Text file assembly involves the the translation of a program written in COP400 assembly language into a COP400 machine code file. This is accomplished with the COP CROSS assembler program (ASM), discussed in Chapter 8.

Object code execution involves loading PDS shared memory with a COP400 machine code file, starting COP execution, tracing program flow, and examining COP registers. This is accomplished with the COP monitor program (COPMON), discussed in Chapter 9.

Miscellaneous functions include listing text files, cross referencing COP programs, and programming PROMs.

To call a PDS system program, the user must first insert the diskette containing the program into the disk drive. This is done by pushing the rectangular "door-open" button to open the drive door (if not already open) and sliding the diskette into the drive, making sure that the index hole is facing outward and slightly upward. (See Figure 4.1.) As the disk is pushed in, a slight pressure will be felt. Keep pushing until a click is heard and the diskette hits the rear of the guidance track, then shut the drive door. *When handling diskettes, be careful not to touch the exposed magnetic surfaces.*

A small slot on one end of the diskette is used as a write protect. To write data onto one of these disks, a metallic sticker provided with the diskettes must be placed over the slot.

While the disk drive is accessing the disk, the red light in the center of the rectangular door open button will be on. *Do not* open the drive door while the light is on. Doing so may ruin the diskette.

Having inserted the disk into the drive, any PDS system program can be called by typing:

```
X>@name
```

on the system console. "Name" stands for one of the system program names given in Table 4.6 (marked with an asterisk). When the program is finished loading, it will print the sign-on line and prompt the user for a command. At that time, any valid command for that program can be entered. Also, any other system program can be called by again typing the "@name" command. (To call EXEC, the user may type only an "@," not "@EXEC").

After becoming familiar with the system program commands by reading the rest of this manual, the user is ready to begin working with PDS. The system verification procedure provided in Section 3.2 is a test to verify that PDS is working properly, and is also a handy way for the user to get a feel for using PDS. The verification procedure includes initializing a diskette with DSKIT, copying files to it with FM, creating a COP400 program file with EDIT, assembling the COP program with ASM, and finally, executing the COP program with the help of COPMON.

# 5 File Manager Program (FM)

The File Manager (FM) is a PDS system program that provides the user with an interface to disk files. FM enables the user to copy files, delete and undelete files, list the disk directory, duplicate disks, list file size and type, list space available on a disk, list and change the disk name, and perform various other functions. This chapter will describe the File Manager commands and give examples of their use.

To call FM, the user types in the @ command:

```
X>@FM
FM.REV:B
F>
```

After FM prompts for a command (F>), the user types in the necessary FM commands. These commands are summarized in Table 5.1. In commands that require a filename, if the file modifier is not specified on a filename that is to the left of "TO," FM will use "SRC" for the default modifier. If a file modifier is not specified on a filename that is to the right of "TO," FM will use the same modifier as it used for the filename to the left of "TO."

## 5.1 COMBINE FILES Command

Syntax:

COMBINE filename,filename[,filename . . .] TO filename

Description: Combine the specified disk files and save the new disk file with the specified name. All of the disk files must be of the symbolic (SYM) file type.

Example:

```
F>C FILE1.SRC,FILE2.SRC,FILE3.SRC TO TEST.SRC
CREATING FILE PDS:TEST.SRC
```

## 5.2 COPY FILE Command

Syntax: COPY filename TO filename

Description: Copy the specified disk file to a new file on the same diskette, thus creating duplicate files with two different names.

Example:

```
F>C FILE1.SRC TO SAMPLE
CREATING FILE PDS:SAMPLE.SRC
```

Table 5.1. File Manager Command Summary

| Command | Syntax | Description | Page |
|---|---|---|---|
| COMBINE FILES | C filename,filename[,filename . . .] TO filename | Combine symbolic files into a new file. | 5-1 |
| COPY FILE | C filename TO filename | Copy file with first name to a new file with the second name. | 5-1 |
| DELETE | DE filename[,filename] | Delete files on diskette. | 5-2 |
| DIRECTORY | D [option[,option . . .]] | List the diskette directory. | 5-2 |
| DUPLICATE FILE | DU volume:filename TO volume:filename | Copy file from one diskette to a second diskette. | 5-2 |
| DUPLICATE VOLUME | DU volume TO volume | Copy all files on one diskette to a second diskette. | 5-2 |
| HEADER | H ["header string"] | List or change diskette header. | 5-3 |
| LOCATE | L filename | List file type, number of sectors, protection level, and version number. | 5-3 |
| PACK FILE | P filename | Remove deleted files of the given name from the diskette directory. | 5-3 |
| PACK VOLUME | P | Remove all deleted files from the diskette directory. | 5-3 |
| PROTECT | PR filename [,plevel] | List or change file protect level. | 5-3 |
| RENAME | R filename TO filename | Rename file. | 5-3 |
| SPACE | S | List number of bad, used, and available sectors on the diskette. | 5-3 |
| UNDELETE | U filename | Undelete file. | 5-3 |
| VOLUME | V ["volume"] | List or change diskette volume name. | 5-3 |

## 5.3 DELETE Command

Syntax: DELETE filename[,filename . . .]

Description: The specified files will be marked as deleted. After a file is deleted, it remains on the diskette and its name appears in the diskette directory with an asterisk beside it. It can be undeleted using the UNDELETE command, *until the diskette is "packed" using the PACK command.* If the user tries to delete a file whose protect level is 2, he will be queried as to whether or not he really wants to delete it. The user will not be allowed to delete a file whose protect level is 3.

Example:
```
F>DE TEST.SRC, SAMPLE.MP
CANNOT DELETE FILE PDS:TEST.SRC
(protect level 3)
OK TO DELETE FILE PDS:SAMPLE.MP
(Y/N,CR = YES)? (CR)
```

## 5.4 DIRECTORY Command

Syntax: DIRECTORY [option[,option . . .]]

Description: List the diskette directory. One or both of the following options may be specified, separated by commas.

Option A — List files in alphabetical order. Otherwise, the list is done chronologically.

Option S — A "short" listing is to be made, excluding deleted files, file IFT, version number, file #, and the number of bad, used, and available sectors on the diskette.

Example: F>D A

```
DIRECTORY FOR:MASTER "PDS MASTER DISKETTE"

FN  D NAME      TYPE          SIZE  PL  VN
 2    EDIT .MP  MAIN PROGRAM    20   3   1
 1    LIST .MP  MAIN PROGRAM     8   3   1

SECTORS BAD:     0
SECTORS USED:   36
SECTORS FREE:  580
```

The first line in the above printout shows the diskette volume name (MASTER) and header (PDS MASTER DISKETTE).

The FN column is a chronological numbering of the first 99 undeleted files. In the unlikely case that there are more than 99 files on the diskette, the FN field will be blank for these files. The FN number, or F#, can be used to load a COP400 load module file into shared memory using the front panel. (See Chapter 9.)

The D (Delete) column denotes a deleted disk file with an asterisk preceding the file name.

The NAME column is an alphabetical list of the file names and modifiers.

The TYPE column indicates the file's Internal File Type (IFT).

The SIZE column indicates the number of sectors occupied by the file.

The PL column indicates the protection level.

The VN column indicates the file version number.

The sum of the bad, used, and free sectors account for the total number of sectors on a diskette (6.6). Sectors used indicates the number of sectors occupied by the files, plus a minimum of 8 sectors required by PDS.

## 5.5 DUPLICATE FILE Command

Syntax:
DUPLICATE volume:filename TO volume:filename

Description: Copy the file on the first volume to a new file on the second volume. The volume name *must* be specified as part of the filename, and the two volume names must be different. FM prompts the user to exchange diskettes in the disk drive as required to complete the transfer. The user must enter (CR) after each prompt. (Control/Q) instead of (CR) will abort the duplication.

Example:
```
F>DU VOL1:TEST.SRC TO VOL2:TEST.SRC
LOAD INDICATED VOLUME, PRESS CR
VOL1 (CR)
VOL2 (CR)
CREATING FILE VOL2:TEST.SRC
VOL1 (CR)
VOL2 (CR)
DUPLICATION COMPLETE
```

## 5.6 DUPLICATE VOLUME Command

Syntax: DUPLICATE volume TO volume

Description: Copy each undeleted file on the first volume to the second volume. The two volume names must be different. FM prompts the user to exchange diskettes in the disk drive as required to complete the transfer. The user must enter (CR) after each prompt. (Control/Q) instead of (CR) will abort the duplication. This command provides a means for making backup copies of diskettes, a recommended procedure. As many as 20 or more swaps may be

needed to duplicate diskettes that have many files or large files on them. The first volume that FM will request to be loaded is the second, or destination volume. This allows a "cleaning up" operation to be performed on it prior to the duplication in order to improve the diskette's access time.

Example:

```
F>DU VOL1 TO VOL2
LOAD INDICATED VOLUME, PRESS (CR)
VOL2 (CR)
VOL1 (CR)
VOL2 CR
CREATING FILE VOL2:FM.MP
VOL1 (CR)
VOL2 (CR)
CREATING FILE VOL2:EDIT.MP
DUPLICATION COMPLETED
```

## 5.7 HEADER Command

Syntax: HEADER ["header string"]

Description: If the header string is left off this command, the header of the diskette will be listed. If the header string is included, the current diskette header will be changed to this new one.

Example:

```
F>H "MY COP PROGRAMS"
```

## 5.8 LOCATE Command

Syntax: LOCATE filename

Description: List the file type, total sectors occupied, protection level, and version number of the specified file.

Example:

```
F>L TEST.SRC
FILE TYPE          SOURCE
TOTAL SECTORS      16
PROTECTION LEVEL   3
VERSION NUMBER     10
```

## 5.9 PACK FILE Command

Syntax: PACK filename

Description: All deleted files of the given name will be removed from the directory. The file can no longer be undeleted. Disk space that was occupied by the file is freed for use by other files.

Example:

```
F>P TEST.SRC
PACKING FILE PDS:TEST.SRC (Y/N,CR = YES)? (CR)
```

## 5.10 PACK VOLUME Command

Syntax: PACK

Description: *All* deleted files on the diskette will be removed and can not be undeleted. Disk space that was occupied by the files is freed for use by other files.

Example:

```
F>P
PACKING DISK (Y/N,CR = YES)? (CR)
```

## 5.11 PROTECT Command

Syntax: PROTECT filename [,plevel]

Description: If the protection level is not given, the protection level of the file will be listed. If the protection level is given, the file protection level will be changed to the new one.

Example:

```
F>PR TEST.SRC,3
```

## 5.12 RENAME Command

Syntax: RENAME filename TO filename

Description: Change the name of a file.

Example:

```
F>R TEST.SRC TO TEST.OLD
```

## 5.13 SPACE Command

Syntax: SPACE

Description: List the number of bad, used, and available sectors on the diskette.

Example:

```
F>S
VOLUME:MASTER
SECTORS BAD:          0
SECTORS USED:       140
SECTORS FREE:       476
```

## 5.14 UNDELETE Command

Syntax: UNDELETE filename

Description: Restore the most recently deleted version of the specified file and delete the existing one (if any). If no deleted version exists, the following message is displayed:

NO BACKUP EXISTS

If there are more than one deleted files of the same name, they can be successively undeleted and renamed, one at a time.

Example:

```
F>U TEST.SRC
```

## 5.15 VOLUME Command

Syntax: VOLUME ["volume name"]

Description: If the volume name is left off this command, the volume name of the diskette will be listed. If the volume name is included, the current diskette volume name will be changed to the new one. The volume may consist of one to eight alphanumeric characters.

Example:

```
F>V "PDS"
```

# 6 Disk Initialization and Test (DSKIT)

DSKIT is a PDS system program which allows the user to "initialize" new diskettes. Initialization consists of the following three operations:

1. Write "sector sync marks" on each of the disk's 616 sectors. This operation requires approximately 1 minute.
2. Write and verify a test pattern in each of the sectors, in order to detect bad sectors. This operation requires approximately 20 minutes.
3. Write the diskette "volume name" and "header" onto the disk, and create an empty directory. This operation requires approximately 10 seconds.

These three operations can be performed with the INITIALIZE command. Although the user will probably not have use for any of the other DSKIT commands, they are described here for completeness.

To call DSKIT, type:

```
X>@DSKIT
DSKIT.REV:B
D>
```

DSKIT is then ready to accept one of the commands listed in Table 6.1 and described in detail below.

## 6.1 INITIALIZE Command

Syntax: I[NITIALIZE "volume","header"

Description: Initialize the diskette that is currently in the disk drive, giving it the specialized volume name and header string. The volume name consists of one to eight alphanumeric characters. The header string consists of one to

**Table 6.1. DSKIT Command Summary**

| Command | Syntax | Description | Page |
|---|---|---|---|
| ADDRESS TEST | A sctrange [aopt . . .] | Test capability to access sectors in given range. | 6-2 |
| BAD SECTORS | B | Print sector numbers of bad sectors. | 6-2 |
| CLEAR | C | Clear results of previous tests. | 6-2 |
| DIRECTORY | D "volume","header" | Build an empty directory. | 6-2 |
| DUMP SECTOR | DU sctrange | Print contents of given range. | 6-2 |
| INITIALIZE | I "volume","header" | Initialize diskette. | 6-1 |
| PATTERN TEST | P sctrange [popt . . .] | Test sectors in given range. | 6-2 |
| SECTOR MARKS | S [trkrange] | Write sector marks on given track range. | 6-3 |
| STATUS | ST | Print drive status. | 6-3 |
| TEST SECTOR | T sector [topt . . .] | Test individual sector. | 6-3 |

| | | |
|---|---|---|
| sector | ::= | hex number from 0 to X'267 |
| track | ::= | hex number from 0 to X'4C |
| sctrange | ::= | sector [/sector] |
| trkrange | ::= | track [/track] |
| volume | ::= | one to eight alphanumeric characters |
| header | ::= | one to forty characters |
| aopt* | ::= | CO, NE |
| popt* | ::= | CO, ND, NE, PA, RO, RW, WO |
| topt* | ::= | CO, PA, RO, RW, WO |

*See Table 6.2 for a definition of these options.

forty characters of any type. The system will query the user regarding initialization of the diskette before beginning the operation.

Example:

```
D>I "COPS ","COP PROGRAMS"
OK TO DESTROY VOLUME "MASTER" (Y/N,CR = YES)? N
 (user forgot to put correct disk in)
D>I "COPS","COP PROGRAMS"
OK TO RUN DESTRUCTIVE OPERATION ON DISK (Y/N,CR = YES)? CR
***SECTOR MARKS COMPLETE***
***PATTERN TEST COMPLETE***
***DIRECTORY COMPLETE***
***INITIALIZATION COMPLETE***
```

## 6.2 ADDRESS TEST Command

Syntax: ADDRESS sctrange [aopt . . .]

Description: Test the addressing ability of the disk head. All sectors in the specified range (sctrange) are written in descending sequence with their sector addresses during Pass 1, then verified during Pass 2.

Valid Options: CO, NE

Example:

```
D>A 0/267
READY TO RUN DESTRUCTIVE OPERATION ON DISK (Y/N,CR = YES)? CR
***ADDRESS TEST COMPLETE***
```

**Table 6.2. DSKIT Command Option Description**

| Option | Meaning |
|---|---|
| CO — Continuous Test | Execute specified tests (RO, WO, or RW) continuously until a console interrupt is deected. |
| ND — Non-Destructive Test | Save original data before the test is begun, and restore data after the test has ended. |
| NE — No Error Messages | Suppress error messages. |
| PA — Pattern Value | Write a specified pattern (up to four hexadecimal digits) on one or more sectors. More than one pattern may be specified. |
| RO — Read-Only Test | Read previously written pattern to verify the data (primarily used to test compatibility between two drives). |
| RW — Read/Write Test | Write specified pattern on each sector, and read to verify (default mode). |
| WO — Write-Only Test | Write specified pattern on each sector but do not read. |

Note: RO, RW, and WO are mutually exclusive, i.e., only one can be used within a given option declaration.

## 6.3 BAD SECTOR Command

Syntax: BAD

Description: Print the sector numbers of all sectors that were found to be bad by the tests that were run after the last CLEAR, DIRECTORY, or INITIALIZE commands.

Example:

```
D>B
NO BAD SECTORS
```

## 6.4 CLEAR Command

Syntax: CLEAR

Description: Clear the results of all tests that have been executed up to this point. This command is performed automatically upon completion of the INITIALIZE DIRECTORY command.

Example: D>C

## 6.5 DIRECTORY COMMAND

Syntax: DIRECTORY "volume","header"

Description: Build an empty directory based on all information gathered in any preceding test. This operation should be performed after any sector tests.

Example:

```
D>DI "MASTER","PDS MASTER DISKETTE"
***DIRECTORY COMPLETE***
```

## 6.6 DUMP SECTOR Command

Syntax: DUMP sctrange

Description: Print the contents of the specified sector range in hexadecimal with the equivalent ASCII values.

Example: D>D 212/267

## 6.7 PATTERN TEST Command

Syntax: PATTERN sctrange [popt . . .]

Description: Test all sectors in the specified range. In the normal default RW (Read/Write) Mode, each sector is written with the specified pattern, then read to verify the data. A total of five patterns may be specified with the PA option, though only one pattern may be specified during RO (Read-Only) or WO (Write-Only) tests. If the PA option is not supplied, the pattern E5E5 is assumed.

Valid Options: CO, ND, NE, PA, RO, RW, WO

Example:

```
D>P 0/267 ND PA = AAAA PA = 5555
***PATTERN TEST COMPLETE***
```

### 6.8 SECTOR MARKS Command

Syntax: SECTOR [trkrange]

Description: Write the sector address marks for a new diskette. This must be followed by a PATTERN TEST command over the specified range of the diskette. The final command in this initialization sequence is DIRECTORY.

Example:
```
D>S 0/4C
***SECTOR MARKS COMPLETE***
```

### 6.9 STATUS Command

Syntax: STATUS

Description: Read sector 0 of the disk and print the resulting disk status. The status is given as four hex digits. The left byte is the number of errors encountered and the right byte indicates the type of error, as follows:

| Right Byte | Description |
|---|---|
| X'1 | No error detected |
| X'2 | Drive not ready |
| X'4 | Addressing error |
| X'8 | Missing sync/write protect |
| X'10 | Write error, CRC doesn't verify |
| X'20 | Read error, CRC doesn't verify |
| X'40 | Illegal disk command |

Example:
```
D>ST
DISK STATUS:0001
```

### 6.10 TEST SECTOR Command

Syntax: TEST sector [topt . . .]

Description: Test a sector as in the PATTERN command. This command is normally used to test the disk drive itself rather than the actual diskette. If the PA option is not supplied, the pattern E5E5 is assumed.

Valid Options: CO, PA, RO, RW, WO

Example:
```
D>T 23A WO PA=3333
```

# 7 Text File Editor (EDIT)

The text file editor (EDIT) is a PDS system program which allows the user to create and change text files that may be subsequently used as source code for assembling programs or as documentation which may be updated. A variety of commands allow the user to insert, delete, alter and list the text, and to write text to a file on floppy disk. EDIT can accept source from disk files or keyboard input. Text entered goes into the edit buffer. The edit buffer is part of the RAM reserved for system programs in the PDS system, and will hold approximately 800 lines of text. All commands with the exception of a few miscellaneous commands perform their operations on the contents of the edit buffer. The easiest way of editing text is using the disk edit mode. Disk edit mode allows the user to specify a disk file name at the beginning of an edit and have each subsequent READ or WRITE command default refer to the specified file.

## 7.1 Disk Edit Mode

Disk edit mode is entered by using the EDIT command and specifying an "edit input file" that contains the source to be edited, and optionally an "edit output file" that will contain the source after it is edited. If an edit output file is not named, the editor will replace the edit input file with the edit output file when the disk edit mode is exited. If the edit output file is named, the edit input file will not be replaced.

Operationally, when the disk edit mode is entered, the user reads a range of lines from the edit input file to the edit buffer using an ADVANCE, READ, or POSITION command. The user would perform his edits on the lines in the edit buffer, then use another ADVANCE or POSITION command to automatically write the contents of the edit buffer to the edit output file, clear the buffer, and read the next range of lines from the edit input file. The size of the edit buffer written back to the disk need not be the same size as the block read into the buffer. When the user has completed his edits, he would close the edit input file and the edit output file automatically with a FINISH or a TERMINATE command. If the user wishes to abort the disk edit mode he would enter an ABORT command.
In the disk edit mode, disk write errors will refer to a file called "EDIT.SYT," a temporary file for the disk edit mode.

Figure 7.1 shows the operational sequence.

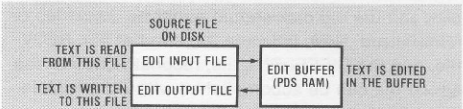



**Figure 7.1 Operational Sequences of Disk Edit Mode**

The edit buffer normally is large enough to edit the average user's source files without difficulty. However, using the "disk edit mode," the user is able to edit files larger than the edit buffer. In disk edit mode the edit buffer is treated as an "edit window." (See Figure 7.2.) The edit window (in memory) may advance through the text of the source disk file. By using the disk edit mode, the user is able to reposition large sections of text, and

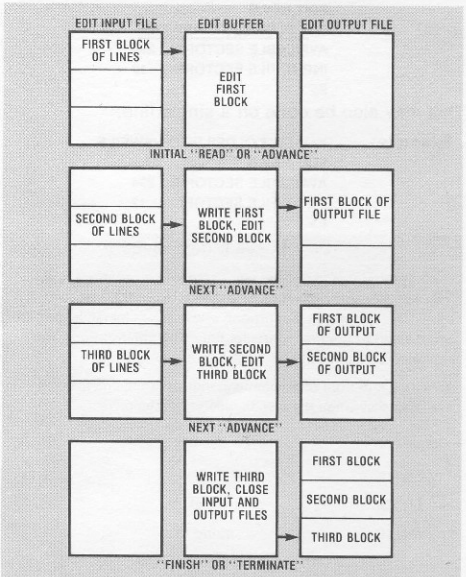



**Figure 7.2 Disk Edit Mode Edit Window Operator**

easily edit source files much larger than the edit buffer.

A warning has been inserted when the disk obviously has no room left for the edit. Nevertheless, care should be taken when editing to assure that there is enough room for the new edit before continuing or a disk error may occur resulting in possible loss of a substantial portion or all of the edit.

When a write error occurs on the use of the ADVANCE, FINISH, TERMINATE, or WRITE commands, the input file is closed and the output file is closed (if possible) and renamed RECV (recovery). If this happens the user will be made aware of it via a console message. The user should immediately write the buffer to a DIFFERENT disk. The bad disk should then be duplicated to a GOOD disk and the old disk should either be discarded or reinitialized. Now, between the original file, RECV file, and the DIFFERENT file, reconstruction of the edited file can be accomplished with the possible loss of only approximately 20 lines.

## 7.2 Invoking EDIT

EDIT is a line editor, which means that the edit pointer is positioned by line number. Line numbers are assigned by EDIT, and are automatically adjusted when lines are inserted or deleted. EDIT is called from the disk with the "@" command.

Example:
```
X>@EDIT
EDIT,REV:B
E>
```

A common sequence of operations is to call EDIT and then enter the disk edit mode.

Example:
```
X>@EDIT
EDIT,REV:B
E>E OLDFILE TO NEWFILE
AVAILABLE SECTORS:  294
INPUT FILE SECTORS:  12
E>
```

This may also be done on a single line.

Example:
```
X>@EDIT OLDFILE TO NEWFILE
EDIT,REV:B
AVAILABLE SECTORS:  294
INPUT FILE SECTORS:  12
E>
```

**Table 7.1. Editor Commands**

The following is a list of the edit command mnemonics and formats. *All* commands may be abbreviated to the first two characters of the command word, and some commands may be abbreviated to the first character only. The abbreviations are indicated by an underline.

An asterisk in front of a command indicates the command is available only when the disk is inserted in the drive.

| Command | Parameters | Page |
|---|---|---|
| *ABORT | | 7-13 |
| *ADVANCE | [range] | 7-10 |
| *ADVANCE | string | 7-11 |
| ALIGN | [range] [IN indent] [CO crange] | 7-9 |
| CHANGE | string TO string [IN range [,range . . .]] [N] | 7-9 |
| CHANGE | crange TO string [IN range [,range . . .]] [N] | 7-9 |
| CLEAR | | 7-6 |
| COPY | range [TO line] | 7-5 |
| DELETE | range [,range . . .] [L] | 7-6 |

**Table 7.1. Editor Command (continued)**

| Command | Parameters | Page |
|---|---|---|
| DELETE | string [IN range [,range . . .]] [L] | 7-6 |
| EDIT | [range [,range . . .]] [S] | 7-8 |
| EDIT | string [IN range [,range . . .]] [S] | 7-8 |
| *EDIT | filename [TO filename] | 7-12 |
| *FINISH | | 7-13 |
| INSERT | [TO line] | 7-4 |
| LIST | [range [,range . . .]] [S] | 7-4 |
| LIST | string [IN range[,range . . .]] [S] | 7-5 |
| MOVE | range [TO line] | 7-6 |
| NEXT | [lines] | 7-5 |
| *POSITION | range | 7-11 |
| *POSITION | string [FROM line] | 7-12 |
| READ | [lines] | 7-7 |
| *READ | [range] FROM filename [TO line] | 7-6 |
| SCALE | | 7-10 |
| *TERMINATE | | 7-13 |
| *WRITE | [range [,range . . .]] [TO filename] | 7-7 |
| *WRITE | string [IN range [,range . . .]] [TO filename] | 7-7 |

**Table 7.2. Command Format Definitions**

| Symbol/ Notation | Definition |
|---|---|
| column | is used as a single column number in the range of 1 to 80. |
| crange | (column range) is defined as:<br>column (/column)<br>where the first column specified indicates the beginning of a column range and the second column specified indicates the end of a column range. The default for the second column is the last column of the line.<br>Note: In the CHANGE command, if only the first column is specified, it indicates an insert starting at that column.<br>Note: The second column number must be equal to or greater than the first column number. |
| device | indicates an input/output device other than the disk. The legal device mnemonics are as follows:<br>Mnemonic — Device<br>*CN — Console<br>*PR — Printer |
| filename | indicates a legal disk file name. See Chapter 4 for a description of what constitutes a legal disk filename. |
| indent | indicates the number of columns to indent the first line of each paragraph in a range of lines. (Used only in the ALIGN command.) |
| line | indicates the number of a line in the edit buffer. Line may be entered as an integer in the range of 1 to 32,766 or as one of the following characters:<br>Character — Buffer Line Indicated<br>F — First line<br>P — Previous line<br>. — Current line<br>N — Next line<br>L — Last line<br>Note: The above characters may not be used in the READ command as part of the range specification. |

**Table 7.2. Editor Command Format Definitions (continued)**

| Symbol/ Notation | Definition |
|---|---|
| lines | indicates the number of lines to be read or the number of lines to be listed. Lines is an integer in the range 1 to 32,766. |
| range | is defined as:<br>line (/line)<br>where the first line specified indicates the beginning line of the range, and the second line specified indicates the ending line of the range.<br>Examples: 10/50, F/L, P, N/200, /L, 342 |
| string | is a string of 0 to 15 ASCII characters enclosed in single or double quotes.<br>If the character string contains quotes (single or double), then the quotes defining the character string must be different.<br>Examples:<br>"memory's" This would work.<br>'memory's' This would not work.<br>'"to line"' This would work.<br>""to line"" This would not work. |
| / | (slash) is entered as shown between the beginning and ending lines of a range or the beginning and ending columns of a crange. |
| [ ] | (brackets) indicate the enclosed item or items are optional. |
| . . . | (ellipsis) indicates that the previous items may be repeated if desired. |

**Table 7.3. Error Messages**

ALIGN ERROR — STOP AT line number

The length of the line the editor stopped at is greater than the maximum line width set or the column range specified. Either increase the line width or the column range, or make the line shorter and re-align the range.

BUFFER EMPTY

Attempted to perform action on an empty buffer.

BUFFER FULL

Attempted to exceed the protective limit of the edit buffer (i.e., next line may exceed the maximum buffer size).

BUFFER FULL — CHANGE IGNORED

The error message was caused by one of the following operations:

1. "EDIT line" — buffer full after edit: changes are ignored.
2. CHANGE command caused a buffer full (lines expanded), current line ("L.") is next line to be changed.

BUFFER FULL — STOP AT line number

Buffer expanded during ALIGN command, next line to be aligned is shown.

CANNOT DELETE OLD COPY OF OUTPUT FILE, NEW NAME:

Edited output file has file of same name at protection level 3, non-deletable, user must enter new name.

FILE ALREADY IN USE

Attempted to read from or write to a file currently being used as either an input or an output file in "disk edit mode." ("READ FROM file" or "WRITE TO file.")

FILE DOES NOT EXIST

On an "EDIT filename TO filename" the first filename does not exist.

**Table 7.3. Error Messages (continued)**

ILLEGAL COMMAND

Non-existent command used.

ILLEGAL OPERAND

The error message was caused by one of the following operations:

1. Illegal operand.
2. Disk not available and using disk related commands ("READ FROM file, WRITE TO file," etc.).

LINE NUMBER BEYOND RANGE

In the command "ADVANCE line [/line]" the lower line of the range has already been brought into the edit buffer or written out, and therefore is not on the disk.

NO INPUT FILE SPECIFIED

Attempted to execute a "READ [lines]" when not in disk edit mode and no input file specified.

NOT IN DISK EDIT MODE

The following commands are not available when not in disk edit mode:

ABORT, ADVANCE, FINISH, POSITION, and TERMINATE

NUMBER OVERFLOW

The error message was caused by one of the following operations:

1. The input number specified is greater than 32,766.
2. The next input line will cause the text buffer to have a line number greater than 32,767.
3. The range of lines to be copied will cause the text buffer to have a line number greater than 32,767.

OUTPUT ALREADY HAS EOF

Disk error occurred when closing file.

Attempted to execute ADVANCE, POSITION, READ, WRITE after a disk error on closing. Only valid commands are ABORT, FINISH, and TERMINATE.

RANGE WILL NOT FIT

The range of lines to be copied will cause the text buffer to exceed the maximum buffer size.

UNABLE TO ACCESS FILE

The editor is unable to transfer control to the file specified because an illegal character has been detected in the filename specified.

VOID RANGE

The lines references are not within the boundaries set by the specified ranges.

For any command which lists text, the output may be interrupted by pressing any key on the console.

## 7.3 EDIT Command Mode

Edit commands are entered from the console. Text may be entered from the console or from the disk. The prompt

```
E>
```

indicates the editor is in "command mode" and is ready to accept a new command. (The "disk edit" mode is used within and is a special version of the command mode.)

The following command formats are listed alphabetically in Table 7.1. The definitions used in the command formats are listed in Table 7.2. Table 7.3 is a list of EDIT Error Messages.

## 7.4 Commands Within the Edit Window (Buffer)

### 7.4.1 INSERT Command

Syntax: INSERT [TO line]

Description: The above command format accepts text from the console keyboard for insertion into the edit buffer. The text is inserted before the line indicated by the "TO line" option. If the "TO line" option is omitted, the text is appended to the end of the buffer. The prompt

```
line?
```

is given initially, and after each carriage return. The line number in the prompt is the actual number of the line about to be inserted. The insertion of lines causes all following line numbers to be increased by a number corresponding to the number of lines inserted.

If the line number of the insert is greater than the last line of the buffer, then the text is appended to the end of the buffer. If the line number of the insert is less than the first line of the buffer, then the text is inserted in front of the first line of the buffer.

If a (control/Q) is entered in column 1 in response to the EDIT command prompt, EDIT will enter the "insert mode" at the end of the buffer. If a (CR) is entered in column 1 in response to the EDIT prompt, EDIT will enter the "insert mode" at the current line number as if the command

```
I TO (current line)
```

had been entered. If a (control/Q) or (CR) is entered as the first character of an inserted line, the insert mode is exited. If a (control/Q) is entered in any other position, EDIT aborts that line and prompts for it again. If a (CR) is entered in any other position, it signifies the end of the current input line.

Examples:

1. Insert text before line 125.

```
E>I TO 125
125?    $BCDADD: RC
126?             JP    ASTART
127?    $INC:    SC
128?(CTRL/Q) #
E>
```

(CTRL/Q) is the first character of the line, echoed on the console as a "#," so insert mode is terminated

2. Insert text before the current line.

```
E>I TO.
128?             LD
129?             JSR   $BCDADD
130?             JSR   $CSP   (CTRL/Q) made an
130?             JMP   $DSPLY error.
131?             JSR   $MPS
132? (CTRL/Q) #
E>
```

3. Insert text before the current line (enter input mode).

```
E>(CR)
132?   (ADD NEW TEXT)
133?   (CR)(Exit input mode)
E>
```

4. Add text to the end of the buffer.

```
E>I
349?   (Text also may look like this. The text inserted is just
350?   standard ASCII characters.)
351?   (CTRL/Q) #
E>
```

5. Add more text to the end of the buffer.

```
E>(CTRL/Q) #              Enter input mode.
351?   The above command can be used to
352?   insert more text.
353?(CTRL/Q) #            Exit input mode.
```

### 7.4.2 LIST Command

This command lists text from the edit buffer. The lines are listed with their current line numbers. If the "S" (squash) option is included, the lines are left-justified, and extra blanks (more than one) are removed from between the words. The S option affects only the listing, not the text in memory.

Syntax: LIST [range[,range . . .]] [S]

Description: The above command format lists a range or ranges of lines. If the range option is omitted, the entire buffer is listed beginning at the first line of the buffer. (In disk edit mode the first line of the buffer is not necessarily line 1 of the text.) "*PR" at the end of the command line will cause the listing to be sent to the printer.

Examples:

1. List the first line of the buffer.

```
E>L F
1                .TITLE  DEMO, 'SOFTWARE EXAMPLE'
E>
```

2. List lines 430/435 on the printer.

```
E>L 430/435 *PR
430   AISC    13       These lines are printed on
431   JP      $DPINC   the printer.
432   JMP     K #      Any key (K) terminates the
E>                     listing.
```

3. List the current position of the edit buffer.

```
E>L .
432                JMP     $DSPLY
E>
```

4. List the first line, the previous through the next line, and the last line.

```
E>L F, P/N, L
1                  .TITLE  DEMO, 'SOFTWARE EXAMPLE'
431                JP      $DPINC
432                JMP     $RST
433     ;                          Comment Line.
468                .END    Start
```

5. List with and without the S option.

```
E>L 266/271 S
266     COMP; BY SETTING ALL TO ONES
267     XAS; GET CONTENTS OF S
268     COMP; S COUNTS DOWN, SO INVERT
269     SKGBZ 0; IF KEY DOWN
270     JP $NOHOLD
271     CLRA; HOLD COUNTER

E>L 266/271
266     COMP                ; BY SETTING ALL TO ONES
267     XAS                 ; GET CONTENTS OF S
268     COMP                ; S COUNTS DOWN, SO INVERT
269     SKGBZ   0;          ; IF KEY DOWN
270     JP      $NOHOLD
271     CLRA                ; HOLD COUNTER
E>
```

Syntax: LIST string [IN range[,range . . .]] [S]

Description: The above command format lists every line within the given range or ranges in which the specified string occurs. If no such lines exist, the message

VOID RANGE

is printed on the console. If the range is omitted, every occurrence of the specified string will be listed. EDIT will accept both upper- and lower-case letters. However, the user will normally use only upper-case. Within the string, upper-case characters and lower-case characters are treated as the same character. For instance:

ABC = ABc = Abc = abc = aBc = aBC = AbC

This feature is true for the LIST, ADVANCE and POSITION commands. In all other commands that have the string option (DELETE, EDIT, CHANGE, and WRITE), the string must match exactly.

Examples:

1. List all occurrences of .WORD in lines 100 through 400.

```
E>L '.WORD' IN 100/400
283                 .WORD   0FF
294         MEMORY: .WORD   03F,06,05B,04F,066,06D,07D,07
358         CRDRDR: .WORD   07F,067,077,07C,039,07E,079,
E>
```

2. List all occurrences of the string RDBUF.

```
E>L "RDBUF"
VOID RANGE
E>
```

### 7.4.3 NEXT Command

Syntax: NEXT [lines]

Description: The above command format lists lines from the edit buffer. If the number of lines option is given, the listing starts at the next line and continues until the given number of lines is listed or until the end of the buffer is reached, whichever occurs first. If the number of lines option is omitted, only the next line is listed.

Examples:

1. List the next line.

```
E>N
103                 CLRA
E>
```

This command is equivalent to: L N

2. List the next five lines.

```
E>N 5
104     $NHOLD:
105         LBI     O,CNTR
106         JSR     $BCDADD
107     K,#
E>
```

Terminate the listing by pressing any key.

### 7.4.4 COPY Command

Syntax: COPY range [TO line]

Description: The above command format copies the specified range of lines and inserts them before the line indicated by the "TO line" option. If the "TO line" option is omitted, the copied lines are appended to the end of the buffer. The copied lines are not deleted from their original location. The buffer is renumbered after the copy.

Examples:

1. Copy lines 6 through 18 and insert them before line 23.

```
E>CO 6/18 TO 23
E>
```

2. Copy lines 100 through 120 and append them to the end of the buffer.

```
E>CO 100/120
E>
```

NOTE: IF the editor is in disk edit mode and the buffer begins, for example, at line 110, only lines 110 through 120 are copied.

### 7.4.5 DELETE Command

This command deletes lines of text from the edit buffer and then renumbers the buffer. If the "L" option is specified, the lines are listed on the console as they are being deleted.

NOTE: If the "L" option is specified, striking any key will abort the deletion of the current line and any other lines that have not been deleted already.

The specific options for the delete command are described below:

Syntax: DELETE range [,range . . .] [L]

Description: The above command format deletes the specified range or ranges of lines from the edit buffer.

Examples:

1. Delete lines 94 through 98, 101, and 103 through 105.

```
E>D 94/98,101,103/105
E>
```

2. Delete lines 203 through 206 and list the deleted lines.

```
E>D 203/206 L
203     $DOT:   XAS
204             COMP
205             SKGBZ   0
206             JP      $NOHOLD
E>
```

Syntax: DELETE string [IN range[,range . . .]] [L]

Description: The above command format deletes only the lines in which the specified character string occurs. If no such lines exist, the message

VOID RANGE

is printed on the console.

NOTE: Any character string found in the text must match *exactly* the specified character string.

Examples:

1. Delete all lines that contain the character string RAMCLR. List all the lines.

```
E>D 'RAMCLR' L
158             JSR     $RAMCLR
170             JSR     $RAMCLR
234             JSR     $RAMCLR
282             JSR     $RAMCLR
E>
```

2. Delete all lines that contain the character string ABC from the range of line 100 through line 200.

```
E>D 'ABC' IN 100/200
VOID RANGE
E>
```

### 7.4.6 CLEAR Command

Syntax: CLEAR

Description: The above command format deletes all lines from the edit buffer.

Example:

Clear the edit buffer and check to see if it is cleared.

```
E>CL
CLEAR CURRENT BUFFER (Y/N,CR = YES)? (CR)

E>L                          List the contents of
BUFFER EMPTY                 the buffer.

E>
```

### 7.4.7 MOVE Command

Syntax: MOVE range [TO line]

Description: The above command format moves a range of lines and inserts them before the line specified by the "TO line" option. The lines are deleted from their original location after the move, and the text is renumbered. If the "TO line" option is not specified, the lines are appended to the end of the buffer.

Examples:

1. Move line 6 to the end of the edit buffer.

```
E>M 6
E>
```

2. Move lines 31 through 40 and insert them before line 68.

```
E>M 31/40 TO 68
E>
```

### 7.4.8 READ Command

This command reads text from a disk file into the edit buffer. The text read is merged with any existing text in the edit buffer and the buffer is renumbered. If the buffer is filled during the course of the read, the message

BUFFER FULL

is printed on the console and the command is terminated. The buffer will contain text through the last complete line read.

The specific options for the read command are described below:

Syntax: READ [range] FROM filename [TO line]

Description: The above command format reads the specified range of lines from the disk file named to the edit buffer and inserts them before the line specified by the "TO line" option. If a range of lines is not specified, the active disk file will be read until an end-of-file is detected, or until the buffer is full. If the "TO line" option is omitted, the text read will be appended to the end of the buffer. *The characters "F," "P," ".", "N," and "L" may not be used in the range option for this command.*

Examples:

1. Read the disk file named "UTILITY."

```
E>R FROM UTILITY
EOF AT 246
E>
```

This read cannot be terminated by console input.

2. Read lines 206 through 350 from the disk file named "LIST" and insert the text before line 128.

```
E>R 206/350 FROM LIST TO 128
E>
```

3. Read from the disk file named "TEST." (The editor is in disk edit mode and using "TEST.")

```
E>R 100/200 FROM TEST
FILE ALREADY IN USE
E>
```

Syntax: READ [lines]

Description: The above command format reads the specified number of lines from the input file and appends them to the edit buffer. If the number of lines is not specified, lines will be transferred until the edit buffer is full or until an end-of-file is reached.

NOTE: *The editor must be in disk edit mode when using this command format.*

Examples:

1. Read the next 12 lines from the current disk edit input file.

```
E>R 12
E>
```

2. Read the entire file.

```
E>R
BUFFER FULL
E>
```

The buffer filled before the entire file was read.

### 7.4.9 WRITE Command

This command writes text from the edit buffer to a disk file. The specific options for the write command are described below.

Syntax: WRITE [range . . .] [TO filename]

Description: The above command format writes a range or ranges of lines to the disk file named by the "TO filename" option. If the range is omitted, the entire edit buffer is written to the disk file. If the "TO filename" option is omitted and the editor is in disk edit mode, the lines are appended to the current edit output file.

Examples:

1. Write the entire contents of the buffer to the disk file named "RESUME."

```
E>W TO RESUME
OK TO DELETE PDS:RESUME.SRC (Y/N,CR = YES)? CR
```

There was an existing copy of the file.

```
CREATING FILE PDS:RESUME.SRC
E>
```

2. Write the contents of lines 1 through 200 to the disk file named "TEST 1."

```
E>W 1/200 TO TEST1
CREATING FILE PDS:TEST1.SRC
E>
```

3. Write lines 152/393 to the current disk edit output file. (The editor is in the disk edit mode.)

```
E>W 152/393
E>
```

4. Write lines 420/582 to the current disk edit output file. (The editor is not in disk edit mode.)

```
E>W 420/582
NO OUTPUT FILE SPECIFIED
E>
```

5. Write the contents of the buffer to the disk file named "TEST1." (Editor is in disk edit mode and using "TEST1.")

```
E>W TO TEST1
FILE ALREADY IN USE
E>
```

Syntax: WRITE string [IN range[,range . . .]] [TO filename]

Description: The above command format writes all the lines within the given range or ranges of lines that contain the specified character string to the disk file named by the "TO filename" option. If the range option is omitted, all lines that contain the string are written to the disk file. If the "TO filename" option is omitted and the editor is in the disk edit mode, the lines are appended to the current edit output file.

NOTE: All the character strings found in the text must match the specified character string exactly.

Examples:

1. Write all occurrences of the string XYZ to the disk file "TEST2."

```
E>W'XYZ' TO TEST2
VOID RANGE                None found.
E>
```

2. Write to the current disk edit output file all the lines, from line 168 to line 250, that contain the string "DEV02." (The editor is in disk edit mode.)

```
E>W 'DEV02' IN 168/250
E>
```

3. Write to the disk file named "TEST" all the lines that contain the string "ABCD." (The editor is not in disk edit mode.)

```
E>W 'ABCD'
NO OUTPUT FILE SPECIFIED
E>
```

**Table 7.4. EDIT Command Control Characters**

| Control Character | Description |
|---|---|
| CTRL/A | Followed by a character string and a carriage return inserts the string after the CTRL/A. A "<" is echoed on the console for the CTRL/A. |
| CTRL/B | Backspace one word. |
| CTRL/C | Advances the carriage to the third tab setting without changing any intervening characters in the line. |
| CTRL/D | Truncates the rest of the line from the current carriage position. |
| CTRL/E | Advances the carriage to one column past the last character of the current line, provided the position of the last character is less than the width. For example, if the last character is in column 65, and the width is 72, then CTRL/E will move the carriage to column 66. |
| CTRL/F | Forward space one word. |
| CTRL/H | Backspace one character. |
| CTRL/I or CTRL/T | Advances the carriage to the next tab setting, changing any intervening characters in the line to spaces. Space one if past third tab. |
| CTRL/L | Forward space one character. |
| CTRL/Q | Aborts the current line modifications if entered in any column position other than column 1. If entered in column 1, CTRL/Q aborts the EDIT command and any modifications to the current line. |
| CTRL/W or CTRL/P | Followed by any character, advances the carriage one column beyond the next occurrence of the specified character. If there are no occurrences of the character before the carriage return, the carriage does not move. |
| CTRL/X | Deletes the current character and echoes a "^" in its place. |
| CTRL/Z or CTRL/O | Followed by any character, advances the carriage to the column containing the next occurrence of the specified character. If there are no occurrences of the character before the carriage return, the carriage does not move. |
| CR | (Carriage return) in column 1 terminates modifications on the current line. |
| K | (Any key) aborts the listing of the current line. |
| SHIFT/0 | Backspace one character. |
| Underline | Backspace one character. |

### 7.4.10 EDIT Command

This command allows the user to edit a range or ranges of lines. Within a line, characters may be inserted, changed, or deleted; or the line may be extended or truncated. If the range option is omitted, the entire buffer is edited beginning at the first line. If the "S" (single) option is selected, there will be no prompt for a second edit of the same line. The control characters that may be used with this command are shown in Table 7.4. They are similar to the control characters described in Chapter 4 for the general line input routine. The line edit mode described here, however, is one of the few times when the general line input characters are not used.

Syntax: EDIT [range[,range . . .]] [S]

Description: The above command format edits a range or ranges of lines.

Examples:

1. Edit line 179.

```
E>E 179
179          JRS   IFBYP     ; IF BYPASS
EDITS?       JSR(CR)
179          JSR   IFBYP     ; IF BYPASS
EDITS? (CR)      (in column 1 terminates the edit)
E>                It is necessary to
                  correct only as far as
                  the error.
```

2. Edit the buffer starting at the current line.

```
E>E ./L
451                LEI 1
EDITS? (CR)                  No edits to this line.
452                AISC 9    9 should be changed.
EDITS? (CTRL/Z)    9         Search for the 9.
EDITS?             AISC 9    Carriage stops at 9.
EDITS?             AISC 5(CR) Correct the line.
452                AISC 5
EDITS? (CR)                  No more edits this line.
453                LBI K #   Abort the listing.
EDITS? (CTRL/Q)    #         Abort the EDIT command.
```

3. Edit lines 120 and 121 using the S option.

```
E>E 120/121 S
120                LBI 0,4   With the S option each
EDITS?             LBI 1(CR) line is presented for
121                AISC 4    editing only once.
EDITS?             AISC 5(CR)
E>
```

Syntax: EDIT string [IN range[,range . . .]] [S]

Description: The above command format edits all occurrences of the specified character string within a range or ranges of lines. The character string searched for must match exactly the character string specified. For instance, to match "ABC" the editor must find "ABC." "ABc" would not match.

Example: Edit all lines which contain the string CARRY.

```
E>E 'CARRY'
104     LBI     0, CARRY
                          Change CARRY to CRY.
EDITS? (CTRL/Z)A          Search for an "A."
EDITS?  LBI     0, CA     Carriage stops at A.
EDITS?  LBI     0, CA (CTRL/X)(CTRL/X)(CR)
EDITS?  LBI     0, C      Delete "AR" ("^^" are
104     LBI     0, CRY    echoed back).
EDITS?(CR)                No more changes.
```

### 7.4.11 CHANGE Command

This command changes a character string or a range of columns to a specified character string throughout a range or ranges of lines. The altered lines will be displayed on the console unless the "N" (no list) option is specified. Pressing any key will abort the change for the current line and the remaining lines of the given range or ranges. The specific options for this command are described below.

Syntax: CHANGE string TO string
[IN range[,range . . .]] [N]

Description: The above command format substitutes the second character string for the first character string throughout the specified range or ranges of lines. If no substitutions can be made, the message

```
VOID RANGE
```

is printed on the console.

For a character string in the text to be changed from the first character string specified in the command to the second character string specified in the command, it must match exactly the first character string (i.e., "ABC" does not match "abc").

Examples:

1. Change the character string ABCD to 1234 throughout the entire buffer.

```
E>C 'ABCD' TO '1234'      The editor did not find
VOID RANGE                any occurrences of the
E>                        string ABCD.
```

2. Change the character string $3 to $N10 in lines 100 through 200.

```
E>L '$3'                          List all occur-
101             JP        $3      rences of the
135     $3:     LBI       $BPI    string $3.
172             JP        $3
E>C '$3' TO '$N10' IN 100/200
101             JP        $N10
135     $N10:   LBI       $BPI
172             JP        $N10
E>
```

Syntax: CHANGE crange TO string
[IN range[,range . . .]] [N]

Description: The above command format changes one or more columns to the specified character string in a range or ranges of lines. If "crange" specifies a range of columns, then the existing columns in that range are modified. If "crange" specifies a single column, then the specified character string is inserted starting at that column.

Examples:

1. List lines 30 through 35, then insert "**" in column 2 in lines 30 through 35.

```
E>L 30/35
30 = = = = = = = = = = = = = = = = = = =
31
32 READ INSTRUCTIONS BEFORE
33 TURNING ON PROCESSOR.
34
35 = = = = = = = = = = = = = = = = = = =
E>C 2 TO '**' IN 30/35
30 ** = = = = = = = = = = = = = = = = = = =
31 **
32 **READ INSTRUCTIONS BEFORE
33 **TURNING ON PROCESSOR.
34 **
35 ** = = = = = = = = = = = = = = = = = = =
```

2. Change columns 2 through 3 to ; in lines 30 through 35.

```
E>C 2/3 TO ';' IN 30/35
30 ; = = = = = = = = = = = = = = = = = = =
31 ;
32 ; READ INSTRUCTIONS BEFORE
33 ; TURNING ON PROCESSOR.
34 ;
35 ; = = = = = = = = = = = = = = = = = = =
E>;
```

This command replaces the contents of column 2 and deletes the contents of column 3. The remainder of the affected lines are moved one column to the left.

### 7.4.12 ALIGN Command

Syntax: ALIGN [range] [In indent] [CO crange]

Description: The above command format aligns a range of lines on the columns specified by the "CO crange" option. If the second column number of the crange is not specified, it defaults to the width of the line. If the "IN indent" option is specified, the first line of the paragraph (i.e., the next line after a blank line) within the range of lines is indented the number of columns specified by "indent." The first line of the range is assumed to be the start of the first paragraph.

Lines are added or deleted whenever necessary, and the text is renumbered when the align is completed. One or more blank lines defines a paragraph.

The ALIGN command removes excess spaces within each paragraph, even from within any character string contained in the paragraph. If there are one or more spaces after the following characters before alignment, two spaces will follow each character after alignment: ".", ":", "!", "?". All other characters will be followed by a single space after alignment, provided of course that they were followed by at least one space before alignment.

The listing of the range of lines that were aligned may be aborted by pressing any key.

This command is used primarily for realigning documentation after text has been added or deleted. The user should be extremely cautious when using this command since *all of the text within the range is aligned before any lines are listed.* If incorrect numbers are given, the user could align areas he had no intention of aligning. It would be advisable to practice using this command before trying it on a large source file.

Example: Align lines 1 through 5 of the following text. Indent 5 spaces in columns 20 through 60.

```
E>L

    1    THE FOLLOWING VERIFICATION PROCEDURE
    2  IS INTENDED TO PROVIDE THE USER WITH BOTH
    3  AN INTRODUCTION TO SYSTEM OPERATION AND A
    4  VERIFICATION OF SYSTEM SOFTWARE AND
    5  HARDWARE.
    6  THE FOLLOWING FIVE SYSTEMS WILL BE USED:

E>AL 1/5 IN 5 C0 20/60

    1                    THE FOLLOWING VERIFICATION
    2                PROCEDURE IS INTENDED TO PROVIDE
    3                THE USER WITH BOTH AN INTRODUCTION
    4                TO SYSTEM OPERATION AND A
    5                VERIFICATION OF SYSTEM SOFTWARE AND
    6                HARDWARE.
    7  THE FOLLOWING FIVE SYSTEMS WILL BE USED:
```

### 7.4.13 SCALE Command

Syntax: SCALE

Description: The above command format will print out a repeating string of digits starting from column 1 of the text field and continuing to column 72. This line of digits may then be compared with printed or displayed text line to determine actual column numbers.

Example:

```
E>S
        123456789-123456789-123456789-123456789-12345678
E>
```

## 7.5 Commands that Move the Edit Window

The ADVANCE and POSITION "disk edit mode" commands maintain the same line numbers as the edit input file on disk. For instance, an advance to line 100 would read lines 100 through 149 (if the size default is used). Of course, any insertions or deletions change the line numbers and the text written to the edit output file which will not, therefore, necessarily have the same line numbers as the text in the edit input file.

The "ADVANCE string" and the "POSITION string" command both have the automatic case conversion feature. That is, ABC = ABc = Abc, etc.

### 7.5.1 ADVANCE Command

This command advances the edit window forward only (in the direction of increasing line numbers). ADVANCE (rather than the POSITION command) normally is used to advance through a disk file. When advancing, prior to finding the first line or string, pressing any key will stop the advance and list the line the command was currently processing.

The specific options for the ADVANCE command are described below.

Syntax: ADVANCE [range]

Description: The above command format writes the contents of the edit buffer to the edit output file, clears the edit buffer, then copies the contents of the edit input file (starting at the next input line) to the edit output file until the lower line of the specified range is reached. Text is then read from the edit input file to the edit buffer until the upper line of the specified range is reached, or an end-of-file is reached, or the buffer is filled. If the lower line number of the specified range

already has been passed (either it was in the current buffer or it previously had been written to the edit output file), the message

```
LINE NUMBER BEYOND RANGE
```

is printed on the console, and the command is aborted.

*If only the lower line of a range is specified, the editor sets the upper line of the range to the lower line plus 49.* For example,

```
A LINE
```

is equivalent to

```
A LINE/LINE + 49
```

Examples:

1. Advance to 200.

```
E>A 200                    Equivalent to AD 200/249.
E>
```

2. Advance to 300 through 600.

```
E>A 300/600
E>
```

3. Advance to 200.

```
E>A 200
LINE NUMBER BEYOND RANGE
E>
```

Syntax: ADVANCE string

Description: The above command format writes the contents of the edit buffer to the edit output file, then copies the contents of the edit input file (starting at the next input line) to the edit output file until the specified character string is found. If the character string is found, it will be the only line written from the edit input file to the edit buffer. If the character string is not found, the contents of the edit input file are copied to the edit output file until an "EOF" (end-of-file) is found.

Examples:

1. Advance to the first occurrence of the character string $DEFAULT:.

```
E>A '$DEFAULT:'
1048     $DEFAULT:
E>
```

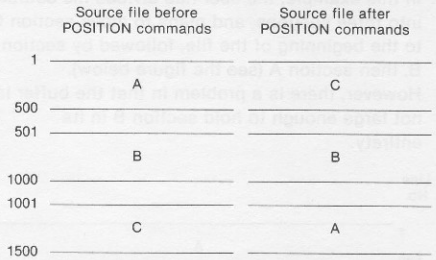
2. Advance to the first occurrence of the character string ABC.

```
E>A 'ABC'
EOF AT  276          ABC was not found.
E>
```

### 7.5.2 POSITION Command

This command moves the edit window to a new position in the edit input file. The contents of the edit buffer are written to the edit output file, the edit buffer is cleared, and then the specified lines are read into the buffer from the edit input file.

POSITION allows a user to recognize large blocks of text in his file. For instance, in the example below, suppose a user wanted to move the sections of text designated "A," "B," and "C" so that "C" was the first section of text in the source file, "B" was next, and "A" was last.




First, the user would enter disk edit mode, then position at the range of lines designated "C," then at the range of lines designated "B," then at the range of lines designated "A." Finally, he would terminate the edit. If the source file was named "TEST," then the operation would be as follows:

```
E>E TEST                    Enter disk edit mode.
E>P 1001/1500               Position at section "C."
E>P 501/1000                Position at section "B."
E>P 1/500                   Position at section "A."
E>T                         Terminate disk edit mode.
TERMINATE CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:TEST.SRC (Y/N,CR = YES)? (CR)
E>
```

The specific options for this command are described below.

Syntax: POSITION range

Description: The above command format positions the edit window at the specified range of lines. If the range is too large to fit into the edit buffer, the message

```
BUFFER FULL
```

is printed on the console, and the command terminates with the last line that will fit in the buffer. If this happens, the user may use the ADVANCE command to edit the remainder of the range, then continue. (See Example 4.)

If just the first line of the range is given, the default range will be "line/line + 49."

Examples:

1. Position at lines 100 through 700.

```
E>P 100/700                    The range was too large.
BUFFER FULL
E>
```

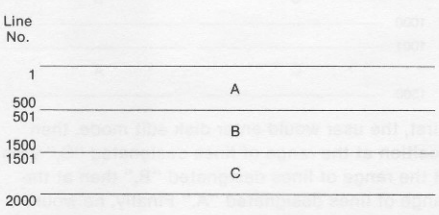
2. Position at lines 1 through 200.

```
E>P 1/200
E>
```

3. Position at line 100. (Range will be 100/149.)

```
E>P 100
E>
```

4. In this example, the user has divided his source into three sections, and plans to move section C to the beginning of the file, followed by section B, then section A (see the figure below). However, there is a problem in that the buffer is not large enough to hold section B in its entirety.




```
E>E TEST                       Enter disk edit mode.
E>P 1501/2000                  Position at section C.
E>P 501/1500                   Position at section B.
BUFFER FULL
E>L L                          List the last line in the
1000     JP        ACOOP       buffer.
E>A 1001/1500                  Advance to the end of
                               section B.
E>P 1/500                      Position at section A.
E>T                            Terminate the edit.
TERMINATE CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:TEST.SRC (Y/N,CR = YES)? (CR)
E>
```

Syntax: POSITION string [FROM line]

Description: The above command format positions the edit window at the first line in which the specified character string occurs, beginning from the line specified by the "FROM line" option. If the "FROM line" option is not specified, the search will begin from the next input file.

If a line containing the character is found, the line is listed on the console. The edit buffer will contain only that line.

Examples:

1. Position to the first occurrence of DATA beginning from line 86.

```
E>P 'DATA' FROM 86
143      LBI       0, DATA
E>
```

2. Position to the first occurrence of BLANKS.

```
E>P 'BLANKS'
683      $GR:      LBI       1, BLANKS
E>
```

## 7.6 Disk Edit Mode "Setup" and "Quit" Commands

The EDIT, FINISH, TERMINATE, and ABORT commands described in the following paragraphs allow the editor program to enter and exit disk edit mode.

Note: The user should assure that there is ample space on his disk for the edit output file before entering disk edit mode. Upon entering the disk edit mode, the size of the available disk space and the size of the input file are displayed. If the user is creating a new file, only the available disk size is displayed.

### 7.6.1 Edit Command

This command may be used to put the editor in disk edit mode. The syntax and specific options for the use of the command are described below.

Syntax: EDIT filename [TO filename]

Description: In the above command format the first named file is declared to be the edit input file and the second named file is declared to be the edit output file. If the edit output file does not exist, the editor will create one at a protection level equal to that of the input file. If the edit output file does exist, dialogue appropriate to its protection level will take place after the edit is completed. If a second file is not named, the editor will construct a provisional edit output file. If the edit is completed normally, the editor will delete the original edit input file and replace it with the edit output file. The protection level of the new edit input file will be the same as that of the old edit input file.

Examples:

1. Create a new edit output file.

```
E>E TEST1
CREATE NEW FILE (Y/N,CR = YES)? (CR)
E>
```

2. Edit disk file TEST1 to TEST2.

```
E>E TEST1 TO TEST2
E>
```

3. Edit disk file TEMPA.SRC. (Editor already in disk edit mode editing TEMPA.SRC.)

```
E>E TEMPA.SRC
FINISH CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:TEMPA.SRC (Y/N,CR = YES)? N
FILE PDS:TEMPA.SRC
CAN'T DELETE        No permission to delete file.
E>
```

4. Edit disk file B.SRC to C.SRC. (Editor already in disk edit mode editing A.SRC.)

```
E>E B.SRC TO C.SRC
FINISH CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:A.SRC (Y/N,CR = YES)? (CR)
E>                  Now ready to begin new edit.
```

5. Edit disk file B.SRC. (Editor already in disk edit mode editing A.SRC.)

```
E>E B.SRC
CONTINUE CURRENT OUTPUT FILE (Y/N,CR = YES)? (CR)
E>                  In this example, file A.SRC is
                    terminated, and file B.SRC is
                    opened.
```

### 7.6.2 FINISH Command

Syntax: FINISH

Description: The above command format appends the contents of the edit buffer and the remainder of the edit input file to the edit output file, terminates disk edit mode, and closes the edit input file and the edit output file. This is a normal completion.

If the editor is not in disk edit mode this command is ignored.

Examples:

1. Finish the current edit.

```
E>F
FINISH CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:DIVIDE.SRC (Y/N,CR = YES)? (CR)
E>
```

2. The editor was not in disk edit mode.

```
E>F
NOT IN DISK EDIT MODE
E>
```

3. Finish the current edit.

```
E>F
FINISH CURRENT EDIT (Y/N,CR = YES)? (CR)
FILE PDS:A.SRC      There was not enough space on
END OF DISK         the disk for the edit output file.
E>
```

### 7.6.3 TERMINATE Command

Syntax: TERMINATE

Description: The above command format appends *only* the contents of the edit buffer to the edit output file, terminates the edit mode, and closes the edit input file and the edit output file. This is a normal completion.

If the editor is not in disk edit mode this command is ignored.

Examples:

1. Terminate the current edit.

```
E>T
TERMINATE CURRENT EDIT (Y/N,CR = YES)? (CR)
OK TO DELETE FILE PDS:SAMPLE.SRC (Y/N,CR = YES)? (CR)
E>
```

2. The editor was not in disk edit mode.

```
E>T
NOT IN DISK EDIT MODE
E>
```

### 7.6.4 ABORT Command

Syntax: ABORT

Description: The above command format aborts the edit mode. The edit buffer is cleared, the edit input file is closed, and the edit output file is not written. If the editor is not in the disk edit mode, this command is ignored.

Examples:

1. Abort disk edit mode, then list the contents of the edit buffer.

```
E>AB
ABORT CURRENT EDIT (Y/N,CR = YES)? (CR)
E>L
BUFFER EMPTY
E>
```

2. The editor was not in disk edit mode.

```
E>AB
NOT IN DISK EDIT MODE
E>
```

# 8 COP Cross Assembler (ASM)

## 8.1 General Introduction

The COP Cross Assembler (ASM) is a PDS system program which translates symbolic program files (created with the text editor, using Assembly Language statements) into object code files (Load Modules) which contain program instructions in binary machine language format. The Load Modules, in turn, are used for loading into PDS shared-memory for debugging (see Chapter 9), for mask programming the machine code into the appropriate COP400 device (MASKTR), or for programming "test" PROMs by the PDS user. The assembler also generates an output listing containing source statements with their corresponding machine code and memory locations, error messages, and other information useful to the programmer in debugging and verifying COP400 programs. Included in the listing are some warning messages with respect to emulating the 410L/411L/420C chips with the COP400-E02 emulator.

The warnings are:

| | |
|---|---|
| *1* | RAM REGISTERS ARE NOT THE SAME AS THE 402 |
| *2* | STACK ON 410/411 HAS ONLY 2 LEVELS |
| *3* | "IT" INSTRUCTION VALID FOR 420C ONLY (TWO-BYTE NOP ON 402) |
| *W* | IF THE LISTING HAS BEEN SUPPRESSED, AS IN MACROS WHERE THE EXPANSION IS NOT LISTED OR BY USE OF LIST OPTIONS, THE *W* WILL BE PRINTED ON THE FIRST PRINTED INSTRUCTION AFTER THE LIST IS TURNED BACK ON. THIS WARNING MEANS THAT THERE WERE INSTRUCTIONS IN THE NONLISTED CODE THAT WOULD HAVE GENERATED WARNINGS HAD THE LIST BEEN ALLOWED. |

The source version number is printed on the assembled program listing to aid in tracking listings vs source files.

This chapter will describe the assembler statements, coding conventions, and other information necessary to use ASM.

To call ASM, the user types in the @ command:

```
X>@ASM I=input[,O=output][,L=List][,options]
ASM,REV:B
```

(Assembly now begins.)

or:

```
X>@ASM
ASM,REV:B
A>I=input[,O=output][,L=list][,options]
```

(Assembly now begins.)

where the assembly parameters are as follows:

| Parameter Description | Parameter Definition |
|---|---|
| Input Device (required): | |
| Disk File | I=filename |
| Output Device (optional): | |
| Disk File | O=filename |
| List Device (optional): | |
| Console | L=*CN (default) |
| Printer | L=*PR |
| Disk File | L=filename |
| Listing Options (optional): | |
| Error Listing Only | EL |
| No Symbol Table List | NM |
| No Comment List | NC |
| No Listing | NL |

The symbolic Assembly Language input to ASM is from a disk file created by the user. The default modifier for the input filename is SRC.

The machine code Load Module may be output to a disk file by the assembler. The default modifier for the output filename is LM.

An assembly Listing may be output to the console, printer, or a disk file. The default modifier for the list filename is LST.

The Load Module and Listing will be produced only if the user specifies the "O=" or "L=" parameters, respectively.

The Listing contains program assembly language statements, together with line numbers and page numbers. For assembly language statement lines which generate machine code, the hexadecimal address of memory locations and their contents are also indicated. Errors associated with assembly language statements are flagged with descriptive error messages on the appropriate statement lines. The Assembler listing also produces an alphabetical listing of all symbols used in the program together with their values. Symbols which are defined but not referenced by the program are flagged with an asterisk (*). Symbols which are referenced but undefined are flagged with a "U." The listing also indicates the number, if any, of errors encountered during the assembly, the number of ROM words (bytes) used, the source and object checksum values and the input, output and list filenames.

Examples of invoking an assembly:

1. Assemble disk file ADD.SRC; output load module to disk file ADD.LM; output full listing to printer.

   A>I = ADD,O = ADD,L = *PR (CR)

2. Assemble file DSPLY.SRC; no load module; output error list only to console.

   A>I = DSPLY,L = *CN,EL (CR)

3. Assemble disk file ABC.SRC; output load module to disk file ABC.LM; output full listing to disk file ABC.LST:

   A>I = ABC,O = ABC,L = ABC (CR)

4. Assemble disk file ABC.SRC, no listing.

   A>I = ABC,NL

Upon pressing the carriage return key associated with each of the above commands, the assembly process will begin. The user may terminate the assembly or the output of an assembler output listing by pressing *any key*. The system will then interrogate the user concerning aborting the assembly as follows:

```
CONTINUE ASSEMBLY (Y/N,CR = YES)?
```

Pressing "N (CR)" will abort the assembly, terminating the printing of an assembly output listing if in progress. Pressing "Y (CR)" or "(CR)" will result in a continuation of the assembly.

The disk containing ASM must be loaded into the disk drive prior to calling the assembler via the @ASM command. This may be the CDS master diskette or another disk to which ASM has been copied using the File Manager (see Chapter 5). After calling the Assembler, the user must insert the disk containing the source code file to be assembled into the drive disk. (If the file to be assembled is contained on the same disk as the ASM disk, no change of disks is, of course, required.)

If the user program to be assembled is a disk file which resides on the *same* disk as ASM, the user may *call and invoke* an assembly after loading the disk containing both programs by combining the call of the Assembler program and Assembler parameter specifications into one command. The following is an illustration of this technique. Note that a space must be inserted between the Assembler call (@ASM) and Assembler parameter specifications.

Example: To call the Assembler and begin an assembly of file ADD.SRC (as in Example #1 above) contained on the same disk:

```
@ASM I = ADD,O = ADD,L = *PR (CR)
ASM,REV:B
```

(Assembly of ADD.SRC now begins.)

## 8.2 The Assembly Process

In order to understand how to use the Assembler, it is helpful to have a brief introduction as to the purpose of an assembler and how it works.

If an assembler were not available, programs would have to be written in machine code. The binary code for each instruction would have to be determined and manually entered into the machine. Transfer-of-control instructions, such as JMP, would require tedious manual calculation of the JMP address to allow calculation of the machine code. Instructions with operands, such as AISC, would require manual insertion of the operand value into the machine code.

An assembler simplifies the programmer's task in several ways:

1. Each instruction is represented by an "English-like" word, called a mnemonic, instead of less intelligible binary machine code. The assembler translates the mnemonic into the appropriate code. For example, the COP400 No-OPeration instruction is represented by the mnemonic "NOP." The assembler translates this into the code 01000100.

2. Instructions which are to be referenced by transfer-of-control instructions may be labeled with an "English-like" word, called a label, and the transfer instruction may use this label. The assembler will assign the appropriate address to the label, and will use it to determine the proper machine code for the transfer instruction. For example, if the instruction at address 3A7 is CLRA and it is desired to jump to this instruction from elsewhere in the program, a label is used:

   ```
   CLEAR: CLRA
   label
   ```

   The jump instruction, rather than being JMP 3A7, is simply:

   ```
   JMP CLEAR
   ```

   The assembler calculates the appropriate address (3A7).

3. Instructions with operands may be written with the operand following the mnemonic. The assembler will insert the value of the operand into the machine code. For example, AISC 7 is translated by the assembler into the code:

   ```
   0101 0111
   AISC  7
   ```

The above three functions are present in almost every assembler. The COP assembler has several other special features which further ease the programmer's task:

4. Values may be assigned to "English-like" words, called symbols, and these symbols may be used

as the operands for instructions. For example, the value 3 may be assigned to the symbol COUNT:

```
COUNT = 3
```

and this symbol may be used as an operand for instructions:

```
LBI COUNT (Equivalent to LBI 3.)
```

This feature is often used when a value may be changed during the process of program development. In this example, only the value assigned to COUNT needs to be changed. If COUNT was not used, all LBI 3 instructions throughout the entire program would have to be changed.

5. An operand may consist of an arithmetic expression. The expression will be evaluated by the assembler and its value for the operand. Examples:
   (a) `LBI COUNT + 1`
   (b) `STII -2`
   (c) `JMP .+3`
   (d) `AISC 3*SIZE-LEN/2`

   Expressions may be used to improve program clarity and to simplify alterations of the program by assigning values to symbols at the front of the program, and using them in arithmetic expressions for instruction operands. Another use of an expression is shown in (c) above, which jumps to the current instruction plus 3, thus precluding the necessity for a label on this instruction.
6. Special assembler statements called "directives" give the user further flexibility in writing programs. Directives are available to assign a title to the program, specify the COP400 chip number and options, specify program page numbers, feed pages and lines of the output listing, perform conditional assembly of instructions, etc.
7. Assembler procedures, or *MACROs,* allow the programmer to give an "English-like" name, called the MACRO NAME, to sequences of instructions that are frequently used, and to insert these instructions into the program simply by stating the MACRO NAME.

The Assembler performs its functions by reading through the Assembly Language statements sequentially from top to bottom, generating the machine code and a program listing as it proceeds. Since it reads statements sequentially, a special problem occurs which must be overcome. Specifically, suppose the Assembler encounters the statement

```
JMP CLEAR
```

but has not yet encountered the label CLEAR. It will be unable to generate machine code for the instruction. This problem is solved by making the assembler perform two "passes" through Assembly Language statements.

Pass 1 of the assembler does not generate a Load Module or a Listing. Its purpose is to assign address values to labels. It does this by using an internal counter called a "location counter." The location counter is initialized to zero at the beginning of each pass. Each time the assembler encounters a single-byte COP400 instruction, the location counter is incremented by one. Each time the assembler encounters a double-byte COP400 instruction, the location counter is incremented by two. The location counter thus keeps track of the ROM address of the next COP400 instruction. In this respect it is similar to a COP400 program counter (PC) register. As the assembler encounters program labels, the labels are assigned the current value of the location counter. In this way, the assembler builds a table of label values which can be used during pass 2 to generate machine code for transfer of control instructions.

Pass 2 of the assembler generates the Load Module and/or Listing, as specified by the user. It uses the table of label values generated during pass 1 to calculate machine code values for transfer of control instructions. It also uses the location counter to determine the address which each COP400 instruction should occupy. The Load Module contains this address information.

The user may alter the value of the location counter with special Assembly Language statements (described later). Care must be exercised when doing this so as not to try to put two different COP instructions in the same ROM location!

## 8.3 Introduction to Assembly Language Statements

The input to the assembler consists of a sequence of Assembly Language statements. There are three types of Assembly Language statements:

1. Instruction statements, which provide a COP400 instruction mnemonic to be translated by the assembler.
2. Directive statements which provide the assembler with information or request it to perform specific tasks.
3. Assignment statements, which assign values to symbols.

Each statement is written using the following characters:

```
Letters — A through Z
Numbers — 0 through 9
Special Characters — ! $ % ' ( ) * + , - . / ; : < = > b
                     Note: "b" indicates a blank.
```

These statements are entered into the assembler input file using the text editor (Chapter 7), and following certain coding conventions. Each

statement contains from one to four fields in the following order:

label field  operation field  operand field  comment field

Since the assembler accepts free-form statements, the user may disregard specific field boundaries, provided the appropriate delimiters for each field are used. However, for clarity and readability, the use of field boundaries is highly recommended. Useful boundaries can be achieved with the PDS "control I or T" tab function described in Section 4.5. PDS initially sets tabs at columns 9, 17, and 33. The @@TAB command described in Section 4.6 can be used to change these settings if desired. The comment field may extend to column 72. Following is a description of each field.

## Label Field

The label field is optional and may contain a symbol used to identify a statement referenced by other statements. When the assembler encounters a label, it assigns it to the current value of the location counter. More than one label may appear in the label field, in which case any of the labels may be used to reference the labeled location. A label may appear by itself in a statement, in which case it refers to the next instruction or data word in the source program. *A colon (:) must be used to delimit (terminate) each label.*

Labels are the most common means of referencing address locations.

Example:

```
        JMP SUB
        .
        .
        .
        .
        .
        .
SUB:    CLRA
```

A label must conform to the following rules:

1. A label may contain one or more alphanumeric characters, the first of which must be either a letter or a dollar sign ($). Although up to 32 characters may be included, only the first *six* characters are recognized by the assembler program. Therefore, the programmer must ensure that a long label is unique in the first six characters.

   Example: LONGLA, LONGLABEL1, LONGLABEL2 } are identical to the assembler

2. If the first character in the label is a dollar sign ($), the label is defined as a *local label.* The .LOCAL directive allows the programmer to specify that local labels appearing between two .LOCAL directives are accessible only within that region of the program (see Section 8.4.3). This enables the programmer to use identical labels throughout a program without causing a conflict between label names. Within a local region, a local label must be unique in the first *four* alphanumeric characters, not including the dollar sign ($).

   Example: $ABCD, $ABCDEF } are identical labels to the assembler

3. No special characters or embedded blanks may appear within a label.

4. A label represents a memory address and, hence, must have a value ranging between 0 and the maximum ROM address of the COP400 chip being used.

Several examples of labels follow:

| Legal Labels | Illegal Labels | Reason Illegal |
|---|---|---|
| $ABC<br>LONGLA | LONGLABEL1<br>LONGLABEL2 | First six characters are not unique |
| AB2<br>$2 | 2AB<br>2CDE | First character must be a letter or a dollar sign |
| XYZ | XYZ$ | Last character is not alphanumeric |
| $ABCDEF<br>$ABC2EF | $ABCDE<br>$ABCDF | First four characters of the local labels are not unique |

A label referencing an instruction need not be on the same line as the instruction — the label will be assigned the value of the address of the first instruction location following the label. This allows the programmer, when writing source code, to devote a separate line with comments to labels, providing clearer documentation of the program and allowing for easier editing of the source code. (An edit of a "label-line" instruction often involves a change of the label location.)

Example:

```
SUB:    ; SUBTRACT ROUTINE
        CLRA ; CLEAR ACCUMULATOR
        .
        .
        .
```

Note: The label "SUB" will be assigned the value of the address of the CLRA instruction.

The label field may also contain a symbol, without a following ":". This format is used for the assignment statement (Section 8.4.2).

## Operation Field

The operation field is mandatory and contains an identifier indicating which type of statement it is.

In an *instruction statement,* the operation field contains the mnemonic name of the desired instruction. For example:

| label | operation |
|---|---|
| SUB: | CLRA |

Valid COP400 instruction mnemonics are provided in Table 8.2. The operation field of an instruction statement is often called a *mnemonic field.*

In a *directive statement* the operation field contains a "." immediately followed by the name of the desired directive. For example:

```
.END
```

Valid directive names are provided in Table 8.6.

In an *assignment statement* the operation field contains a " = ". (See Section 8.4.2.)

*One or more blanks terminate the operation field.*

### Operand Field

The operand field contains entries that identify data to be acted upon by the operation defined in the operation field. Many statements do not require use of the operand field. For those that do, the operand field usually consists of one or two expressions, separated by a comma.

An expression is composed of *terms.* There are 7 types of terms:

1. A *decimal constant* is a decimal number that does not begin with zero. Leading zeros for decimal data are not permitted, except for the simple case of the constant 0.

   Examples: 3,234,-10.

2. A *hexadecimal constant* term is a hexadecimal number that starts with "X'" or with a leading zero.

   Examples: X'23A,07B,X'F.

3. A *string constant* term is a single character enclosed in single quote marks.

   Examples: 'Z,' '$,' '3.'

   To use a single quote mark for a string constant, write four quote marks: ''''

4. A *label* term is described above under the label field description.

5. A *symbol* term is constructed in the same way as a label term, but is used differently. (See Section 8.4.2.)

6. The *location counter* term is a single dot (".").
   The dot represents the location counter, and, if it appears within an expression, it is replaced by the current value of the location counter.

   Example: JMP . + 2

7. A *lower-half* term is represented by L(term). An *upper-half* term is represented by H(term). When the assembler encounters one of these in an expression, it replaces it with either the lower or the upper 8 bits of the value of the symbol, respectively.

   Examples: H(X'172F) is replaced by X'17
   L(X'172F) is replaced by X'2F
   H(X'00FF) is replaced by X'00
   L(X'00FF) is replaced by X'FF

Terms are represented internally in the assembler in 16-bit binary notation. Negative numbers are represented by two's complement notation. In this notation, the negative of a number is formed by complementing each bit in the data word and adding one to the complemented number. The sign of the number is indicated by the most significant bit. When the most significant bit is "0," the number is positive or zero; when the most significant bit is "1," the number is negative. The maximum range for a 16-bit number in this format is $7FFF_{16}$ ($+32767_{10}$) to $8000_{16}$ ($-32768_{10}$).

String constants are represented internally by the appropriate 8-bit ASCII code.

An *expression* may consist of a single term.

Examples:
```
5
X'3C
'Q'
SUB
.
H(H'3CF)
L(SUB)
```

Alternatively, an expression may consist of two or more terms combined using the operators shown in Table 8.1.

Examples:
```
36 + SUB
X'3F0-10
X'7F&'Q'
3*5!XYZ
%SUB/2
```

The multiterm expression is evaluated by the assembler program in a left-to-right order

**Table 8.1. ASM Arithmetic and Logical Operators**

| Operator | Function | Type |
|---|---|---|
| + | Addition | Binary |
| - | Subtraction | Unary or Binary |
| * | Multiplication | Binary |
| / | Division | Binary |
| % | Logical NOT | Unary |
| & | Logical AND | Binary |
| ! | Logical OR | Binary |
| < | "Less Than" | Binary |
| = | "Equal To" | Binary |
| > | "Greater Than" | Binary |

regardless of the operators used between the terms. However, parentheses are permitted for the purpose of grouping the terms of a multiterm expression. They may be nested up to nine-deep within a multiterm expression, with the innermost parenthetical operation being resolved first.

The constructs "A<B," "A = B," and "A>B" cause the specified comparison to be made. The result is

"1" if the comparison is true and "0" if the condition is false.

Examples:

```
I=(2+3*(4+5))/6
L(TABLE)+X'10
100-1
ENTRY1+ENTRY 2-4
A>B * DISKAD
(100-1*12)+H(X'300)
```

multiterm expressions

The magnitude of the expression must be compatible with the memory storage available for the expression. For example, if the expression is to be stored in an 8-bit memory word, then the value of the expression must not exceed X'FF.

Example:

```
JMP X'40+CHAR
```

Expression value must not exceed X'3FF for COP420 (1024 bytes of program memory).

If the expression is used in conjunction with the JP instruction to transfer control to a new ROM word on the *same* page, then the value must not exceed $N*40_{16}+3E_{16}$ or precede $N*40_{16}$ where N = the number of the current page.

Example:

```
.PAGE     0
  .       .
  .       .
  .       .
  .       .
JP        TABLE+4
```

Expression value must not exceed $3E_{16}$ or precede 0.

On some statements, there are a mandatory first expression and an optional second expression. When such a statement is encountered by the assembler and the operand field contains two expressions, the assembler will left shift the value of the left expression by four bits, and will then add to it the value of the right expression, which must evaluate to less than $16_{10}$ (4 bits). This feature is useful on the LBI instruction.

Example:

```
LBI 3,15
```

In this example, the assembler evaluates the left expression (3), shifts it left four bits to obtain the value X'30, then evaluates the right expression (15), and finally adds it to X'30, obtaining a result of X'3F. This value is then used to determine the correct machine code for the LBI instruction. The above example is thus equivalent to:

```
LBI X'3F
```

### Comment Field

Comments are *optional* descriptive notes which are printed on the assembler output listing for programmer reference and documentation. *Comments should be included throughout the program to explain subroutine linkages, data formats, algorithms used, formats of inputs processed, and so forth.* A comment may follow a statement on the same line, or the comment may be entered on one or more separate statement lines. The comment has no effect on the assembled Load Module (.LM) file.

The following conventions apply to comments:

1. A comment *must* be preceded by a semicolon (;).
2. All ASCII characters, including blanks, may be used in comments.
3. Comments should not extend beyond column 72, but a comment may be carried over on the following line (preceded by a semicolon).

   Note: When listing a COP400 program on the system printer, comments are listed to column 63 only.

   Example:

```
Label      Operand    Comment
GETVAL: JSR   SAVREG  ; LOAD MEMORY DATA INTO A
```

The label, GETVAL, is a label name for the address of this instruction. Thus, GETVAL can be used in other statements (preceding or following) to reference this statement. The instruction mnemonic JSR specifies the COP400 instruction. The operand field for the JSR instruction is the symbol SAVREG. The comment field is separated from the operand field by a semicolon (";"). Spaces on each side of the semicolon are optional. The comment allows the programmer to quickly identify the operation performed by the instruction.

## 8.4 Assembler Statements

The following sections describe the COP Assembly Language statements in detail. Some statements have *optional fields.* These optional fields will be enclosed in *brackets* ([ ]) to indicate that they are optional.

### 8.4.1 Instruction Statements

There are approximately 60 COP400 instructions. All are applicable to the COP440. A few of these are inapplicable to the COP420. (The COP420 instruction set is a subset of the COP440's.) The COP410's instruction set, in turn, is a subset of the 420's — it lacks some of the 420's instructions (410 < 420 < 440). Also, the COP421 and COP411, because of their lack of specific inputs, do not contain some instructions that are present in their related COP devices, the COP420 and COP410, respectively. The programmer, therefore, should refer to the *COP400 Microcontroller Family Chip User's Manual* for specific information on the instruction sets of the particular COP devices for which the assembly code is being written.

COP400 series Assembler Instruction statements fall within one of the following six classes:

- Arithmetic Instructions
- Transfer of Control Instructions

- Memory Reference Instructions
- Register Reference Instructions
- Test Instructions
- Input/Output Instructions

Table 8.2 contains a summary of the COP400 series instruction set, grouped according to one of the above set classes. Additional instructions which will be included in the COP400 instruction set are not indicated. (Refer to COP440 data sheet.) This table provides the assembly mnemonic and operand, hex code, machine code (binary), data flow, skip conditions and description for each instruction. Refer to Table 8.3 for definitions of symbols used in describing the COP400 series instruction set. The Notes to Table 8.2 provide additional information to assist the user in understanding the operations of specific instructions. For further detailed information on the nature and use of the COP400 series instruction set, *examples of assembly language routines and programming techniques,* information on the electrical specifications and architecture of each COP400 series device, see the *COP400 Microcontroller Family Chip User's Manual.* Refer to Table 8.4 for an alphabetical listing of all COP400 series instructions showing their hexadecimal opcode. Also refer to Table 8.5 for a hexadecimal opcode ordered list of the COP400 series instruction set. These latter two tables do not include references to the additional COP400 instructions.

**Table 8.2. COP400 Instruction Set**

| Mnemonic | Operand | Hex Code | Machine Language Code (Binary) | Data Flow | Skip Conditions | Description |
|---|---|---|---|---|---|---|
| ARITHMETIC INSTRUCTIONS | | | | | | |
| ASC | | 30 | 0 0 1 1 \| 0 0 0 0 | A + C + RAM(B) → A<br>Carry → C | Carry | Add with Carry, Skip on Carry |
| ADD | | 31 | 0 0 1 1 \| 0 0 0 1 | A + RAM(B) → A | None | Add A to RAM |
| ADT | | 4A | 0 1 0 0 \| 1 0 1 0 | A + $10_{10}$ → A | None | Add Ten to A |
| AISC | y | 5– | 0 1 0 1 \| y | A + y → A | Carry | Add Immediate, Skip on Carry (y ≠ 0) |
| CASC | | 10 | 0 0 0 1 \| 0 0 0 0 | $\overline{A}$ + RAM(B) + C → A<br>Carry → C | Carry | Complement and Add with Carry, Skip on Carry |
| CLRA | | 00 | 0 0 0 0 \| 0 0 0 0 | 0 → A | None | Clear A |
| COMP | | 40 | 0 1 0 0 \| 0 0 0 0 | $\overline{A}$ → A | None | Ones complement of A to A |
| NOP | | 44 | 0 1 0 0 \| 0 1 0 0 | None | None | No Operation |
| RC | | 32 | 0 0 1 1 \| 0 0 1 0 | "0" → C | None | Reset C |
| SC | | 22 | 0 0 1 0 \| 0 0 1 0 | "1" → C | None | Set C |
| XOR | | 02 | 0 0 0 0 \| 0 0 1 0 | A ⊕ RAM(B) → A | None | Exclusive-OR A with RAM |

Table 8.2. COP400 Instruction Set (continued)

| Mnemonic | Operand | Hex Code | Machine Language Code (Binary) | Data Flow | Skip Conditions | Description |
|---|---|---|---|---|---|---|
| TRANSFER OF CONTROL INSTRUCTIONS | | | | | | |
| JID | | FF | 1 1 1 1 \| 1 1 1 1 | ROM ($PC_{9:8}$,A,M) → $PC_{7:0}$ | None | Jump Indirect (Note 3) |
| JMP | a | 6– | 0 1 1 0 \| 0 0 \| $a_{9:8}$ | a → PC | None | Jump |
| | | -- | $a_{7:0}$ | | | |
| JP | a | -- | 1 \| $a_{6:0}$ (pages 2,3 only) | a → $PC_{6:0}$ | None | Jump within Page (Note 4) |
| | | | or | | | |
| | | -- | 1 1 \| $a_{5:0}$ (all other pages) | a → $PC_{5:0}$ | | |
| JSRP | a | -- | 1 0 \| $a_{5:0}$ | PC + 1 → SA → SB → SC<br>0010 → $PC_{9:6}$<br>a → $PC_{5:0}$ | None | Jump to Subroutine Page (Note 5) |
| JSR | a | 6– | 0 1 1 0 \| 1 0 \| $a_{9:8}$ | PC + 1 → SA → SB → a → PC | None | Jump to Subroutine |
| | | -- | $a_{7:0}$ | | | |
| RET | | 48 | 0 1 0 0 \| 1 0 0 0 | SC → SB → SA → PC | None | Return from Subroutine |
| RETSK | | 49 | 0 1 0 0 \| 1 0 0 1 | SC → SB → SA → PC | Always Skip on Return | Return from Subroutine then Skip |
| MEMORY REFERENCE INSTRUCTIONS | | | | | | |
| CAMQ | | 33 | 0 0 1 1 \| 0 0 1 1 | A → $Q_{7:4}$<br>RAM(B) → $Q_{3:0}$ | None | Copy A, RAM to Q |
| | | 3C | 0 0 1 1 \| 1 1 0 0 | | | |
| CQMA | | 33 | 0 0 1 1 \| 0 0 1 1 | $Q_{7:4}$ → RAM(B)<br>$Q_{3:0}$ → A | None | Copy Q to RAM, A |
| | | 2C | 0 0 1 0 \| 1 1 0 0 | | | |
| LD | r | -5 | 0 0 \| r \| 0 1 0 1 | RAM(B) → A<br>Br ⊕ r → Br | None | Load RAM into A, Exclusive-OR Br with r |
| LDD | r,d | 23 | 0 0 1 0 \| 0 0 1 1 | RAM(r,d) → A | None | Load A with RAM pointed to directly by r,d |
| | | -- | 0 0 \| r \| d | | | |
| LQID | | BF | 1 0 1 1 \| 1 1 1 1 | ROM($PC_{9:8}$,A,M) → Q<br>SB → SC | None | Load Q Indirect (Note 3) |
| RMB | 0 | 4C | 0 1 0 0 \| 1 1 0 0 | 0 → $RAM(B)_0$ | None | Reset RAM Bit |
| | 1 | 45 | 0 1 0 0 \| 0 1 0 1 | 0 → $RAM(B)_1$ | | |
| | 2 | 42 | 0 1 0 0 \| 0 0 1 0 | 0 → $RAM(B)_2$ | | |
| | 3 | 43 | 0 1 0 0 \| 0 0 1 1 | 0 → $RAM(B)_3$ | | |
| SMB | 0 | 4D | 0 1 0 0 \| 1 1 0 1 | 1 → $RAM(B)_0$ | None | Set RAM Bit |
| | 1 | 47 | 0 1 0 0 \| 0 1 1 1 | 1 → $RAM(B)_1$ | | |
| | 2 | 46 | 0 1 0 0 \| 0 1 1 0 | 1 → $RAM(B)_2$ | | |
| | 3 | 4B | 0 1 0 0 \| 1 0 1 1 | 1 → $RAM(B)_3$ | | |

Table 8.2. COP400 Instruction Set (continued)

| Mnemonic | Operand | Hex Code | Machine Language Code (Binary) | Data Flow | Skip Conditions | Description |
|---|---|---|---|---|---|---|
| MEMORY REFERENCE INSTRUCTIONS (continued) | | | | | | |
| STII | y | 7– | \|0 1 1 1\| y \| | y → RAM(B)<br>Bd + 1 → Bd | None | Store Memory Immediate and Increment Bd |
| X | r | –6 | \|0 0\| r \|0 1 1 0\| | RAM(B) ⟷ A<br>Br ⊕ r → Br | None | Exchange RAM with A, Exclusive-OR Br with r |
| XAD | r,d | 23<br>-- | \|0 0 1 0\|0 0 1 1\|<br>\|1 0\| r \| d \| | RAM(r,d) ⟷ A | None | Exchange A with RAM pointed to directly by r,d |
| XDS | r | –7 | \|0 0\| r \|0 1 1 1\| | RAM(B) ⟷ A<br>Bd – 1 → Bd<br>Br ⊕ r → Br | Bd decrements past 0 | Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r |
| XIS | r | –4 | \|0 0\| r \|0 1 0 0\| | RAM(B) ⟷ A<br>Bd + 1 → Bd<br>Br ⊕ r → Br | Bd increments past 15<br>Exclusive-OR Br with r | Exchange RAM with A and Increment Bd, |
| REGISTER REFERENCE INSTRUCTIONS | | | | | | |
| CAB | | 50 | \|0 1 0 1\|0 0 0 0\| | A → Bd | None | Copy A to Bd |
| CBA | | 4E | \|0 1 0 0\|1 1 1 0\| | Bd → A | None | Copy Bd to A |
| LBI | r,d | -- | \|0 0\| r \| (d – 1)\|<br>(d = 0, 9:15)<br>or<br>\|0 0 1 1\|0 0 1 1\|<br>\|1 0\| r \| d \|<br>(any d) | r,d → B | Skip until not a LBI | Load B Immediate with r,d (Note 6) |
| LEI | y | 33<br>6– | \|0 0 1 1\|0 0 1 1\|<br>\|0 1 1 0\| y \| | y → EN | None | Load EN Immediate (Note 7) |
| XABR | | 12 | \|0 0 0 1\|0 0 1 0\| | A ⟷ Br ($0,0 \rightarrow A_3,A_2$) | None | Exchange A with Br |
| TEST INSTRUCTIONS | | | | | | |
| SKC | | 20 | \|0 0 1 0\|0 0 0 0\| | | C = "1" | Skip if C is True |
| SKE | | 21 | \|0 0 1 0\|0 0 0 1\| | | A = RAM(B) | Skip if A Equals RAM |
| SKGZ | | 33<br>21 | \|0 0 1 1\|0 0 1 1\|<br>\|0 0 1 0\|0 0 0 1\| | | $G_{3:0} = 0$ | Skip if G is Zero (all 4 bits) |
| SKGBZ | | 33 | \|0 0 1 1\|0 0 1 1\| | 1st byte | | Skip if G Bit is Zero |
| | 0 | 01 | \|0 0 0 0\|0 0 0 1\| | 2nd byte | $G_0 = 0$ | |
| | 1 | 11 | \|0 0 0 1\|0 0 0 1\| | 2nd byte | $G_1 = 0$ | |
| | 2 | 03 | \|0 0 0 0\|0 0 1 1\| | 2nd byte | $G_2 = 0$ | |
| | 3 | 13 | \|0 0 0 1\|0 0 1 1\| | 2nd byte | $G_3 = 0$ | |
| SKMBZ | 0 | 01 | \|0 0 0 0\|0 0 0 1\| | | $RAM(B)_0 = 0$ | Skip if RAM Bit is Zero |
| | 1 | 11 | \|0 0 0 1\|0 0 0 1\| | | $RAM(B)_1 = 0$ | |
| | 2 | 03 | \|0 0 0 0\|0 0 1 1\| | | $RAM(B)_2 = 0$ | |
| | 3 | 13 | \|0 0 0 1\|0 0 1 1\| | | $RAM(B)_3 = 0$ | |
| SKT | | 41 | \|0 1 0 0\|0 0 0 1\| | | A time-base counter carry has occurred since last test | Skip on Timer (Note 3) |

Table 8.2. COP400 Instruction Set (continued)

| Mnemonic | Operand | Hex Code | Machine Language Code (Binary) | Data Flow | Skip Conditions | Description |
|---|---|---|---|---|---|---|
| INPUT/OUTPUT INSTRUCTIONS | | | | | | |
| ING | | 33 | \|0 0 1 1\|0 0 1 1\| | G → A | None | Input G Ports to A |
| | | 2A | \|0 0 1 0\|1 0 1 0\| | | | |
| ININ | | 33 | \|0 0 1 1\|0 0 1 1\| | IN → A | None | Input IN Inputs to A (Note 2) |
| | | 28 | \|0 0 1 0\|1 0 0 0\| | | | |
| INIL | | 33 | \|0 0 1 1\|0 0 1 1\| | $IL_3$,"1","0",$IL_0$ → A | None | Input IL Latches to A (Notes 2 and 3) |
| | | 29 | \|0 0 1 0\|1 0 0 1\| | | | |
| INL | | 33 | \|0 0 1 1\|0 0 1 1\| | $L_{7:4}$ → RAM(B)<br>$L_{3:0}$ → A | None | Input L Ports to RAM, A |
| | | 2E | \|0 0 1 0\|1 1 1 0\| | | | |
| OBD | | 33 | \|0 0 1 1\|0 0 1 1\| | Bd → D | None | Output Bd to D Outputs |
| | | 3E | \|0 0 1 1\|1 1 1 0\| | | | |
| OGI | y | 33 | \|0 0 1 1\|0 0 1 1\| | y → G | None | Output to G Ports Immediate |
| | | 5- | \|0 1 0 1\| y \| | | | |
| OMG | | 33 | \|0 0 1 1\|0 0 1 1\| | RAM(B) → G | None | Output RAM to G Ports |
| | | 3A | \|0 0 1 1\|1 0 1 0\| | | | |
| XAS | | 4F | \|0 1 0 0\|1 1 1 1\| | A ⟷ SIO, C → SK | None | Exchange A with SIO (Note 3) |

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, $A_3$ indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The INI and INIL instructions are not available on the 24-pin COP421 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see data sheet.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 ($1001_2$), the lower 4 bits of the LBI instruction equal 8 ($1000_2$). To load 0, the lower 4 bits of the LBI instruction should equal 15 ($1111_2$).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

**Table 8.3. COP400 Instruction Set Table Symbols**

| Symbol | Definition |
|---|---|
| INTERNAL ARCHITECTURE SYMBOLS | |
| A | 4-bit Accumulator |
| B | 6-bit RAM Address Register |
| Br | Upper 2 bits of B (register address) |
| Bd | Lower 4 bits of B (digit address) |
| C | 1-bit Carry Register |
| D | 4-bit Data Output Port |
| EN | 4-bit Enable Register |
| G | 4-bit Register to latch data for G I/O Port |
| IL | Two 1-bit Latches associated with the $IN_3$ or $IN_0$ Inputs |
| IN | 4-bit Input Port |
| L | 8-bit TRI-STATE I/O Port |
| M | 4-bit contents of RAM Memory pointed to by B Register |
| PC | 10-bit ROM Address Register (program counter) |
| Q | 8-bit Register to latch data for L I/O Port |
| SA | 10-bit Subroutine Save Register A |
| SB | 10-bit Subroutine Save Register B |
| SC | 10-bit Subroutine Save Register C |
| SIO | 4-bit Shift Register and Counter |
| SK | Logic-Controlled Clock Output |

| Symbol | Definition |
|---|---|
| INSTRUCTION OPERAND SYMBOLS | |
| d | 4-bit Operand Field, 0-15 binary (RAM Digit Select) |
| r | 2-bit Operand Field, 0-3 binary (RAM Register Select) |
| a | 10-bit Operand Field, 0-1024 binary (ROM Address) |
| y | 4-bit Operand Field, 0-15 binary (Immediate Data) |
| RAM(s) | Contents of RAM location addressed by s |
| ROM(t) | Contents of ROM location addressed by t |
| OPERATIONAL SYMBOLS | |
| + | Plus |
| − | Minus |
| → | Replaces |
| ←→ | Is exchanged with |
| = | Is equal to |
| A | The ones complement of A |
| ⊕ | Exclusive-OR |
| : | Range of values |

**Table 8.4. Alphabetical Mnemonic Index of COP400 Instructions**

| Instruction | Hexadecimal Op Code | Description |
|---|---|---|
| ADD | 31 | Add A to RAM |
| ADT | 4A | Add Ten to A |
| AISC 1-15 | 51-5F | Add Immediate, Skip on Carry |
| ASC | 30 | Add with Carry, Skip on Carry |
| CAB | 50 | Copy A to Bd |
| CAMQ* | 33/3C | Copy A, RAM to Q |
| CASC | 10 | Complement and Add with Carry, Skip on Carry |
| CBA | 4E | Copy Bd to A |
| CLRA | 0 | Clear A |
| COMP | 40 | Ones complement of A to A |
| CQMA* | 33/2C | Copy A to RAM, A |
| ING* | 33/2A | Input G ports to A |
| INIL* | 33/0 | Input IL Latches to A |
| ININ | 33/28 | Input IN Inputs to A |
| INL* | 33/2E | Input L Ports to M, A |
| JID | FF | Jump Indirect |
| JMP* | 60-63/0-FF | Jump |
| JP | 80-BE,C0-CD | Jump within Page |
| JSR* | 68-6B/0-FF | Jump to Subroutine |
| JSRP | 80-BE | Jump to Subroutine Page |
| LBI 0,9-15,0 | 8-F | Load B Immediate (Single-byte) |
| LBI 1,9-15,0 | 18-1F | |
| LBI 2,9-15,0 | 28-2F | |
| LBI 3,9-15,0 | 38-3F | |
| LBI* 0,1-8 | 33/81-88 | Load B Immediate (Double-byte) |
| LBI* 1,1-8 | 33/91-98 | |
| LBI* 2,1-8 | 33/A1-A8 | |
| LBI* 3,1-8 | 33/B1-B8 | |
| LD 0,1,2,3 | 5,15,25,35 | Load RAM into A |
| LDD* 0-3,0-15 | 23/0-F | Load A with RAM |
| LEI* 0-15 | 33/60-6F | Load EN Immediate |
| LQID | BF | Load Q Indirect |
| NOP | 44 | No Operation |
| OBD* | 33/3E | Output Bd to D Outputs |
| OGI* | 33/50-5F | Output to G Ports Immediate |
| OMG* | 33/3A | Output RAM to G Ports |
| RC | 32 | Reset C |
| RET | 48 | Return |
| RETSK | 49 | Return then Skip |
| RMB 0,1,2,3 | 4C,45,42,43 | Reset RAM Bit |
| SC | 22 | Set C |
| SMB 0,1,2,3 | 4D,47,46,48 | Set RAM Bit |
| SKC | 20 | Skip if C is True |
| SKE | 21 | Skip if A Equals |
| SKGBZ* 0,1,2,3 | 33/1,11,3,13 | Skip if G Bit is Zero |
| SKGZ* | 33/21 | Skip if G Equals Zero (all 4 bits) |
| SKMBZ 0,1,2,3 | 1,11,3,13 | Skip if RAM Bit is Zero |
| SKT | 41 | Skip on Timer |
| STII | 70-7F | Store Memory Immediate and Increment Bd |
| X 0,1,2,3 | 6,16,26,36 | Exchange RAM with A |
| XABR | 12 | Exchange A with Br |
| XAD* 0-3,0-15 | 23/80-BF | Exchange A with RAM |
| XDS 0,1,2,3 | 7,17,27,37 | Exchange RAM with A and Decrement Bd |
| XIS 0,1,2,3 | 4,14,24,34 | Exchange RAM with A and Increment Bd |
| XOR | 2 | Exclusive-OR A with RAM |

*Double-byte Instruction.

## Table 8.5. Table of COP400 Instructions Listed by Op Codes (Hexadecimal)

| | | |
|---|---|---|
| 00 | CLRA | |
| 01 | SKMBZ | 0 |
| 02 | XOR | |
| 03 | SKMBZ | 2 |
| 04 | XIS | 0 |
| 05 | LD | 0 |
| 06 | X | 0 |
| 07 | XDS | 0 |
| 08 | LBI | 0,9 |
| 09 | LBI | 0,10 |
| 0A | LBI | 0,11 |
| 0B | LBI | 0,12 |
| 0C | LBI | 0,13 |
| 0D | LBI | 0,14 |
| 0E | LBI | 0,15 |
| 0F | LBI | 0,0 |
| 10 | CASC | |
| 11 | SKMBZ | 1 |
| 12 | XABR | |
| 13 | SKMBZ | 3 |
| 14 | XIS | 0 |
| 15 | LD | 1 |
| 16 | X | 1 |
| 17 | XDS | 1 |
| 18 | LBI | 1,9 |
| 19 | LBI | 1,10 |
| 1A | LBI | 1,11 |
| 1B | LBI | 1,12 |
| 1C | LBI | 1,13 |
| 1D | LBI | 1,14 |
| 1E | LBI | 1,15 |
| 1F | LBI | 1,0 |
| 20 | SKC | |
| 21 | SKE | |
| 22 | SC | |
| 23 | LDD/XAD** | |
| 24 | XIS | 2 |
| 25 | LD | 2 |
| 26 | X | 2 |
| 27 | XDS | 2 |
| 28 | LBI | 2,9 |
| 29 | LBI | 2,10 |
| 2A | LBI | 2,11 |
| 2B | LBI | 2,12 |
| 2C | LBI | 2,13 |
| 2D | LBI | 2,14 |
| 2E | LBI | 2,15 |
| 2F | LBI | 2,0 |
| 30 | ASC | |
| 31 | ADD | |
| 32 | RC | |
| 33 | TWO WORD* (except LDD, XAD, JMP, JSR) | |
| 34 | XIS | 3 |
| 35 | LD | 3 |
| 36 | X | 3 |
| 37 | XDS | 3 |
| 38 | LBI | 3,9 |
| 39 | LBI | 3,10 |
| 3A | LBI | 3,11 |
| 3B | LBI | 3,12 |
| 3C | LBI | 3,13 |
| 3D | LBI | 3,14 |
| 3E | LBI | 3,15 |
| 3F | LBI | 3,0 |
| 40 | COMP | |
| 41 | SKT | |
| 42 | RMB | 2 |
| 43 | RMB | 3 |
| 44 | NOP | |
| 45 | RMB | 1 |
| 46 | SMB | 2 |
| 47 | SMB | 1 |
| 48 | RET | |
| 49 | RETSK | |
| 4A | ADT | |
| 4B | SMB | 3 |
| 4C | RMB | 0 |
| 4D | SMB | 0 |
| 4E | CBA | |
| 4F | XAS | |
| 50 | CAB | |
| 51 | AISC | 1 |
| 52 | AISC | 2 |
| 53 | AISC | 3 |
| 54 | AISC | 4 |
| 55 | AISC | 5 |
| 56 | AISC | 6 |
| 57 | AISC | 7 |
| 58 | AISC | 8 |
| 59 | AISC | 9 |
| 5A | AISC | 10 |
| 5B | AISC | 11 |
| 5C | AISC | 12 |
| 5D | AISC | 13 |
| 5E | AISC | 14 |
| 5F | AISC | 15 |
| 60 | JMP*** | to page 0, 1, 2, or 3 |
| 61 | JMP*** | to page 4, 5, 6, or 7 |
| 62 | JMP*** | to page 8, 9, 10, or 11 |
| 63 | JMP*** | to page 12, 13, 14, or 15 |
| 64 | invalid | |
| 65 | invalid | |
| 66 | invalid | |
| 67 | invalid | |
| 68 | JSR*** | to page 0, 1, 2, or 3 |
| 69 | JSR*** | to page 4, 5, 6, or 7 |
| 6A | JSR*** | to page 8, 9, 10, or 11 |
| 6B | JSR*** | to page 12, 13, 14, or 15 |
| 6C | invalid for COP420, COP410 | |
| 6D | invalid for COP420, COP410 | |
| 6E | invalid for COP420, COP410 | |
| 6F | invalid for COP420, COP410 | |
| 70 | STII | 0 |
| 71 | STII | 1 |
| 72 | STII | 2 |
| 73 | STII | 3 |
| 74 | STII | 4 |
| 75 | STII | 5 |
| 76 | STII | 6 |
| 77 | STII | 7 |
| 78 | STII | 8 |
| 79 | STII | 9 |
| 7A | STII | 10 |
| 7B | STII | 11 |
| 7C | STII | 12 |
| 7D | STII | 13 |
| 7E | STII | 14 |
| 7F | STII | 15 |
| 80-BE | JSRP to word XX ($0-3F_{16}$) or JP to page 2, word XX ($0-3F_{16}$): opcode = 80 + XX) | |
| BF | LQID | |
| C0-CE | JP to word XX ($0-3F_{16}$): opcode = C0 + XX | |
| FF | JID | |

***Two-Word Instructions**

First word: 33, second word: see table below.

| | | |
|---|---|---|
| 1 | SKGBZ | 0 |
| 3 | SKGBZ | 2 |
| 11 | SKGBZ | 1 |
| 13 | SKGBZ | 3 |
| 21 | SKGZ | |
| 28 | ININ | |
| 29 | INIL | |
| 2A | ING | |
| 2C | CQMA | |
| 2E | INL | |
| 3A | OMG | |
| 3C | CAMQ | |
| 3E | OBD | |
| 50 | OGI | 0 |
| 51 | OGI | 1 |
| 52 | OGI | 2 |
| 53 | OGI | 3 |
| 54 | OGI | 4 |
| 55 | OGI | 5 |
| 56 | OGI | 6 |
| 57 | OGI | 7 |
| 58 | OGI | 8 |
| 59 | OGI | 9 |
| 5A | OGI | 10 |
| 5B | OGI | 11 |
| 5C | OGI | 12 |
| 5D | OGI | 13 |
| 5E | OGI | 14 |
| 5F | OGI | 15 |
| 60 | LEI | 0 |
| 61 | LEI | 1 |
| 62 | LEI | 2 |
| 63 | LEI | 3 |
| 64 | LEI | 4 |
| 65 | LEI | 5 |
| 66 | LEI | 6 |
| 67 | LEI | 7 |
| 68 | LEI | 8 |
| 69 | LEI | 9 |
| 6A | LEI | 10 |
| 6B | LEI | 11 |

## Table 8.5. Table of COP400 Instructions Listed by Op Codes (Hexadecimal) (continued)

| | | |
|---|---|---|
| 6C | LEI | 12 |
| 6D | LEI | 13 |
| 6E | LEI | 14 |
| 6F | LEI | 15 |
| 81 | LBI | 0,1 |
| 82 | LBI | 0,2 |
| 83 | LBI | 0,3 |
| 84 | LBI | 0,4 |
| 85 | LBI | 0,5 |
| 86 | LBI | 0,6 |
| 87 | LBI | 0,7 |
| 88 | LBI | 0,8 |
| 91 | LBI | 1,1 |
| 92 | LBI | 1,2 |
| 93 | LBI | 1,3 |
| 94 | LBI | 1,4 |
| 95 | LBI | 1,5 |
| 96 | LBI | 1,6 |
| 97 | LBI | 1,7 |
| 98 | LBI | 1,8 |
| A1 | LBI | 2,1 |
| A2 | LBI | 2,2 |
| A3 | LBI | 2,3 |
| A4 | LBI | 2,4 |
| A5 | LBI | 2,5 |
| A6 | LBI | 2,6 |
| A7 | LBI | 2,7 |
| A8 | LBI | 2,8 |
| B1 | LBI | 3,1 |
| B2 | LBI | 3,2 |
| B3 | LBI | 3,3 |
| B4 | LBI | 3,4 |
| B5 | LBI | 3,5 |
| B6 | LBI | 3,6 |
| B7 | LBI | 3,7 |
| B8 | LBI | 3,8 |

****LDD/XAD Instruction**

First word: 23, second word: see table below.

| | | |
|---|---|---|
| 0 | LDD | 0,0 |
| 1 | LDD | 0,1 |
| 2 | LDD | 0,2 |
| 3 | LDD | 0,3 |
| 4 | LDD | 0,4 |
| 5 | LDD | 0,5 |
| 6 | LDD | 0,6 |
| 7 | LDD | 0,7 |
| 8 | LDD | 0,8 |
| 9 | LDD | 0,9 |
| A | LDD | 0,10 |
| B | LDD | 0,11 |
| C | LDD | 0,12 |
| D | LDD | 0,13 |
| E | LDD | 0,14 |
| F | LDD | 0,15 |
| 10 | LDD | 1,0 |
| 11 | LDD | 1,1 |
| 12 | LDD | 1,2 |
| 13 | LDD | 1,3 |
| 14 | LDD | 1,4 |
| 15 | LDD | 1,5 |
| 16 | LDD | 1,6 |
| 17 | LDD | 1,7 |
| 18 | LDD | 1,8 |
| 19 | LDD | 1,9 |
| 1A | LDD | 1,10 |
| 1B | LDD | 1,11 |
| 1C | LDD | 1,12 |
| 1D | LDD | 1,13 |
| 1E | LDD | 1,14 |
| 1F | LDD | 1,15 |
| 20 | LDD | 2,0 |
| 21 | LDD | 2,1 |
| 22 | LDD | 2,2 |
| 23 | LDD | 2,3 |
| 24 | LDD | 2,4 |
| 25 | LDD | 2,5 |
| 26 | LDD | 2,6 |
| 27 | LDD | 2,7 |
| 28 | LDD | 2,8 |
| 29 | LDD | 2,9 |
| 2A | LDD | 2,10 |
| 2B | LDD | 2,11 |
| 2C | LDD | 2,12 |
| 2D | LDD | 2,13 |
| 2E | LDD | 2,14 |
| 2F | LDD | 2,15 |
| 30 | LDD | 3,0 |
| 31 | LDD | 3,1 |
| 32 | LDD | 3,2 |
| 33 | LDD | 3,3 |
| 34 | LDD | 3,4 |
| 35 | LDD | 3,5 |
| 36 | LDD | 3,6 |
| 37 | LDD | 3,7 |
| 38 | LDD | 3,8 |
| 39 | LDD | 3,9 |
| 3A | LDD | 3,10 |
| 3B | LDD | 3,11 |
| 3C | LDD | 3,12 |
| 3D | LDD | 3,13 |
| 3E | LDD | 3,14 |
| 3F | LDD | 3,15 |
| 80 | XAD | 0,0 |
| 81 | XAD | 0,1 |
| 82 | XAD | 0,2 |
| 83 | XAD | 0,3 |
| 84 | XAD | 0,4 |
| 85 | XAD | 0,5 |
| 86 | XAD | 0,6 |
| 87 | XAD | 0,7 |
| 88 | XAD | 0,8 |
| 89 | XAD | 0,9 |
| 8A | XAD | 0,10 |
| 8B | XAD | 0,11 |
| 8C | XAD | 0,12 |
| 8D | XAD | 0,13 |
| 8E | XAD | 0,14 |
| 8F | XAD | 0,15 |
| 90 | XAD | 1,0 |
| 91 | XAD | 1,1 |
| 92 | XAD | 1,2 |
| 93 | XAD | 1,3 |
| 94 | XAD | 1,4 |
| 95 | XAD | 1,5 |
| 96 | XAD | 1,6 |
| 97 | XAD | 1,7 |
| 98 | XAD | 1,8 |
| 99 | XAD | 1,9 |
| 9A | XAD | 1,10 |
| 9B | XAD | 1,11 |
| 9C | XAD | 1,12 |
| 9D | XAD | 1,13 |
| 9E | XAD | 1,14 |
| 9F | XAD | 1,15 |
| A0 | XAD | 2,0 |
| A1 | XAD | 2,1 |
| A2 | XAD | 2,2 |
| A3 | XAD | 2,3 |
| A4 | XAD | 2,4 |
| A5 | XAD | 2,5 |
| A6 | XAD | 2,6 |
| A7 | XAD | 2,7 |
| A8 | XAD | 2,8 |
| A9 | XAD | 2,9 |
| AA | XAD | 2,10 |
| AB | XAD | 2,11 |
| AC | XAD | 2,12 |
| AD | XAD | 2,13 |
| AE | XAD | 2,14 |
| AF | XAD | 2,15 |
| B0 | XAD | 3,0 |
| B1 | XAD | 3,1 |
| B2 | XAD | 3,2 |
| B3 | XAD | 3,3 |
| B4 | XAD | 3,4 |
| B5 | XAD | 3,5 |
| B6 | XAD | 3,6 |
| B7 | XAD | 3,7 |
| B8 | XAD | 3,8 |
| B9 | XAD | 3,9 |
| BA | XAD | 3,10 |
| BB | XAD | 3,11 |
| BC | XAD | 3,12 |
| BD | XAD | 3,13 |
| BE | XAD | 3,14 |
| BF | XAD | 3,15 |

*** 0 + XX : JSR or JMP to page 0, 4, 10, or 14, word XX ($0-3F_{16}$); 0-3F
40 + XX : JSR or JMP to page 1, 5, 11, or 15, word XX ($0-3F_{16}$); 40-7F
80 + XX : JSR or JMP to page 2, 6, 12, or 16, word XX ($0-3F_{16}$); 80-BF
C0 + XX : JSR or JMP to page 3, 7, 13, or 17, word XX ($0-3F_{16}$); C0-FF

### 8.4.2 Assignment Statements

symbol = expression[,expression][;comments]

The Assignment Statement assigns the value of the expression on the right of the equals sign to the symbol on the left of the equals sign. If two expressions are given, the value of the leftmost is shifted left by four bits, and the rightmost expression, which must evaluate to less than $16_{10}$, is added to this value. The Assignment Statement does not generate machine code. It simply assigns a value to a symbol. When the symbol is *used* in a COP instruction statement operand field, the assigned value is used to generate code.

```
Examples:   X = 3,15      ; ASSIGN VALUE OF X'3F to
                          ; "X"
            LBI  X        ; GENERATE LBI 3,15
                          ; INSTRUCTION CODE
            Y = 5         ; ASSIGN VALUE OF 5 TO "Y"
            AISC Y        ; GENERATE AISC 5
                          ; INSTRUCTION CODE
```

The Assignment Statement may also refer to the current value of the location counter. The location counter symbol (".") may appear on both sides of the Assignment Statement equals sign.

If it appears on the left, it is assigned the value of the expression to the right side of the equals sign. In that case the expression on the right must be defined during the first pass so that the pass 1 label assignments may be made.

```
Examples:   . = X'20      ; SET LOCATION COUNTER TO
                          ; ADDRESS X'20
            . = . + 10    ; RESERVE 10 LOCATIONS
                          ; FOR LATER USE
            LOC =         ; SAVE CURRENT LOCATION
                          ; COUNTER VALUE IN "LOC"
```

If the symbol on the left of the equals sign is not a ".", then the expression on the right need not have a value during pass 1 but the expression must have a value during pass 2. This permits only one level of forward referencing. An example of more than one level of forward referencing is included in the following examples:

```
THD:   A = B + 2   This expression is undefined
                   during pass 2 because it appears
                   before B is defined below. It is
                   therefore invalid; B is defined
                   only after pass 2. See "SND"
                   below.

SND:   B = C - 1   This expression is undefined
                   during pass 1 because it appears
                   before C is defined below. It is
                   defined during pass 2 because C
                   was defined during pass 1.

FST:   C = 25      This expression is absolute,
                   defined during pass 1.
```

A symbol may be assigned only one value during an assembly with an Assignment Statement. (Attempting to redefine the value of the symbol will result in an error message.) The .SET directive, however, allows symbol values to be redefined during an assembly (see Section 8.4.3).

### 8.4.3 Directive Statements

Directive Statements control the assembly process and may generate data in the object program. The directive name may be preceded by one or more labels, and may be followed by a comment. It occupies the operation field and may require an operand field expression as determined by the particular directive statement.

Assembler directive statements and their functions are summarized in Table 8.6. Note that all directive statements begin with a period for easy visual differentiation from the instruction statement mnemonics in the output listing. Each directive statement is described in more detail in the following sections. For further examples of use of assembler directives in writing COP400 programs, refer to Chapter 4, *COP400 Microcontroller Family Chip User's Manual.*

**Table 8.6. Summary of Assembler Directives**

| Directive | Function | Page |
|---|---|---|
| .ADDR | Address constant generation | 8-17 |
| .CHIP | Identification of COP400 device | 8-18 |
| .DO | Begin Macro-time looping* | 8-24 |
| .ELSE | Conditional assembly directive | 8-19 |
| .END | Physical end of source program | 8-15 |
| .ENDDO | End Macro-time looping* | 8-24 |
| .ENDIF | Conditional assembly directive | 8-19 |
| .ENDM | End Macro definition* | 8-20 |
| .ERROR | Macro error message generation** | 8-24 |
| .EXIT | Exit DO loop* | 8-24 |
| .FORM | Output Listing top-of-form | 8-16 |
| .IF | Conditional assembly directive | 8-19 |
| .IFC | Macro conditional assembly** | 8-23 |
| .INCLD | Include disk file source code | 8-19 |
| .LIST | Listing output control | 8-15 |
| .LOCAL | Establish a new local symbol region | 8-17 |
| .MACRO | Begin Macro definition* | 8-20 |
| .MDEL | Macro delete** | 8-24 |
| .MLOC | Macro local symbol designation* | 8-22 |
| .OPT | Define COP400 device options | 8-18 |
| .PAGE | Set assembler location counter to page address | 8-18 |
| .SET | Assign values to varibles | 8-24 |
| .SPACE | Space n lines on Output Listing | 8-16 |
| .TITLE | Identification of program | 8-15 |
| .WORD | 8-bit data generation | 8-16 |

*Used only in Macro definitions. **Macro related directive.

### .TITLE Directive

Syntax: .TITLE symbol,['string'] [;comments]

Description: The .TITLE directive identifies the load module and output listing in which it appears with a symbolic name and an *optional* definitive title string. If a .TITLE directive does not appear in the program, the load module and output listing are given the name MAINPR. If more than one .TITLE directive is used, the last one encountered specifies the symbolic name. The symbolic name must meet the symbol construction restrictions discussed in Section 8.3. The string must be 26 or fewer characters long for it to appear fully on the output listing. Single quotes (') must appear at the beginning and end of the character string.

Example:
```
.TITLE  TBLKP, 'TABLE LOOKUP'
```

### .END Directive

Syntax: .END

Description: The .END directive signifies the physical end of the source program. All assembly source statements appearing after this directive are ignored. *All Assembler programs must terminate with the .END directive.*

Example:
```
.
.       ; SOURCE CODE
.
.END    ; END OF PROGRAM
```

### .LIST Directive

Syntax: .LIST expression [;comments]

Description: The .LIST directive controls listing of the source program. This includes listing of assembled code in general, listing of unassembled code caused by the .IF and .IFC directives, listing of MACRO expansions and listing of code generated by the .INCLD directive.

Control of the various list options depends upon the state of the six least significant bits of the evaluated expression in the operand field (bits 5 through 0). Table 8.7 shows the options available, their associated bit weights and assembler default values.

Options are usually combined to give the desired type of listing.

Examples:

1. Full Master listing:
```
.LIST 1
```
2. Full Master listing and listing of all code expanded during macro calls:
```
.LIST X'D
   or
.LIST 01!0C
```
3. Suppress listing:
```
.LIST 0
```

**Table 8.7. List Options**

| Control Function | Bit Positions | Binary Value | 6-Bit Hex Value | Description |
|---|---|---|---|---|
| Master List | 0 | 0 | 00 | Suppress all listing |
| | | 1 | 01 | *Full listing |
| .IF List | 1 | 0 | 00 | *Suppress listing of unassembled code |
| | | 1 | 02 | Full listing (of .IFs and .IFCs) |
| Macro List | 2,3 | 00 | 00 | *List only macro calls |
| | | 10 | 08 | List only code generated by macro calls |
| | | 11 | 0C | List all code expanded during macro calls |
| Binary List | 4 | 0 | 00 | List only the first two bytes of generated data |
| | | 1 | 10 | *List all the binary output by statements generating more than one word (e.g., .ASCII) |
| Include List | 5 | 0 | 00 | *List only error lines for the included file |
| | | 1 | 20 | List the included file (source statements from the included files are listed without line numbers) |

*Indicates default.

### .SPACE Directive

Syntax: [label] .SPACE expression [;comments]

Description: The .SPACE directive skips forward a number of lines on the output listing as specified by the expression in the operand field.

Example: Skip 20 lines.

```
.SPACE 20
```

### .FORM Directive

Syntax: .FORM ['string'] [;comments]

Description: The .FORM directive spaces forward to the top of the next page of the output listing (form feed). The optional string is printed as a page title on each page until a .FORM directive containing a new string is encountered. No action is taken (except for a new page title) if the .FORM directive is encountered immediately after an *assembler generated* top-of-page request which occurs when an output listing is full.

Example:

```
.FORM'BCD ARITHMETIC ROUTINES'
; FORM FEED
```

The string must be 26 or fewer characters to be fully printed on the output listing.

### .WORD Directive

Syntax: [label:] .WORD expression[,expression . . .] [;comments]

Description: The .WORD directive stores consecutively in memory one 8-bit byte of data for each given expression. If the directive has a label, it refers to the address of the first expression. The value of each expression must be in the range −128 to +127 for signed data or 0 to 255 for unsigned data.

Examples:

1. Single expression without a label.

```
.WORD    X'FF
```

2. Multiple expression with a label.

```
TBL:    .WORD MPR-10, X'FF
```

Note: "TBL" will be assigned by the address location occupied by the first byte of the multiple expression, i.e., MPR-10.

The hexadecimal value of ASCII characters may be stored in memory using the .WORD directive and an operand expression specifying character strings or their hexadecimal equivalents. (See Table 8.8, ASCII Character Set in Hexadecimal Representation.)

Example: Both of the following directive formats will store the hex value of the ASCII word "HELLO" in successive memory locations:

```
.WORD  'H','E','L','L','O'          ; CHARACTER STRING
.WORD  X'48,X'45,X'4C,X'4C,2'4F     ; HEX EQUIVALENTS
```

In the smaller system dedicated applications in which COP400 devices are commonly used, a more typical function of the .WORD directive is to place 7-segment decode data in ROM for output to the digits of a LED or VF display. Table 8.9 provides the 7-segment binary and hexadecimal values associated with the display numerals 0 through 9, with and without the Decimal Point bit on and with the contents of ROM ($I_7$-$I_0$) assigned to Sa-Sg, D.P. as well as to D.P., Sg-Sa.

**Table 8.8. ASCII Character Set in Hexadecimal Representation**

| Char. | 7-Bit Hex Number | Char. | 7-Bit Hex Number | Char. | 7-Bit Hex Number | Char. | 7-Bit Hex Number |
|---|---|---|---|---|---|---|---|
| NUM | 00 | SP | 20 | @ | 40 | \ | 60 |
| SOH | 01 | ! | 21 | A | 41 | A | 61 |
| STX | 02 | " | 22 | B | 42 | b | 62 |
| ETX | 03 | # | 23 | C | 43 | c | 63 |
| EOT | 04 | $ | 24 | D | 44 | d | 64 |
| ENQ | 05 | % | 25 | E | 45 | e | 65 |
| ACK | 06 | & | 26 | F | 46 | f | 66 |
| BEL | 07 | ' | 27 | G | 47 | g | 67 |
| BS | 08 | ( | 28 | H | 48 | h | 68 |
| HT | 09 | ) | 29 | I | 49 | i | 69 |
| LF | 0A | * | 2A | J | 4A | j | 6A |
| VT | 0B | + | 2B | K | 4B | k | 6B |
| FF | 0C | , | 2C | L | 4C | l | 6C |
| CR | 0D | – | 2D | M | 4D | m | 6D |
| SO | 0E | . | 2E | N | 4E | n | 6E |
| SI | 0F | / | 2F | O | 4F | o | 6F |
| DLE | 10 | 0 | 30 | P | 50 | p | 70 |
| DC1 | 11 | 1 | 31 | Q | 51 | q | 71 |
| DC2 | 12 | 2 | 32 | R | 52 | r | 72 |
| DC3 | 13 | 3 | 33 | S | 53 | s | 73 |
| DC4 | 14 | 4 | 34 | T | 54 | t | 74 |
| NAK | 15 | 5 | 35 | U | 55 | u | 75 |
| SYN | 16 | 6 | 36 | V | 56 | v | 76 |
| ETB | 17 | 7 | 37 | W | 57 | w | 77 |
| CAN | 18 | 8 | 38 | X | 58 | x | 78 |
| EM | 19 | 9 | 39 | Y | 59 | y | 79 |
| SUB | 1A | : | 3A | Z | 5A | z | 7A |
| ESC | 1B | ; | 3B | [ | 5B | | 7B |
| FS | 1C | < | 3C | \ | 5C | | 7C |
| GS | 1D | = | 3D | ] | 5D | ALT | 7D |
| RS | 1E | > | 3E | ↑ | 5E | ESC | 7E |
| US | 1F | ? | 3F | ← | 5F | DEL, rubout | 7F |

Table 8.9. Display Digit Segments


| Binary Values: Sa-Sg, D.P. → $I_7$-$I_0$ | | | | | | | | Hexadecimal Values: Sa-Sg, D.P. → $I_7$-$I_0$ | | Hexadecimal Values: D.P., Sg-Sa → $I_7$-$I_0$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sa | Sb | Sc | Sd | Se | Sf | Sg | D.P. Off/On | D.P. Off | D.P. On | D.P. Off | D.P. On |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0/1 | FC | FD | 3F | BF |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0/1 | 60 | 61 | 06 | 86 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0/1 | DA | DB | 5B | DB |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0/1 | F2 | F3 | 4F | CF |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0/1 | 66 | 67 | 66 | E6 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0/1 | B6 | B7 | 6D | ED |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0/1 | BE | BF | 7D | FD |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0/1 | E0 | E1 | 07 | 87 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0/1 | FE | FF | 7F | FF |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0/1 | E6 | E7 | 67 | E7 |

### .ADDR Directive

Syntax: [label:] .ADDR expression[,expression . . .] [;comments]

Description: The .ADDR directive generates 8-bit bytes as specified by one or more expressions in the operand field of this directive and places them in successive memory locations. These expressions are usually labels and are used as address pointers by the COP400 *JID* (Jump Indirect) instruction which transfers program control to the contents of the address generated by the .ADDR directive.

This directive masks out the upper 8 bits of the expression specified in the operand field, and places the lower 8 bits in successive memory locations. Next, the lower 8 bits of the symbol or expression are masked and a comparison is made of the upper 8 bits with the current location counter address to ensure that the address generated by the .ADDR directive is in the same 4-page ROM block as the assembler location counter — this test is necessary since the *JID* instruction must access a pointer and transfer program control within the current 4-page program ROM "block." If this test indicates an out-of-range expression, an error message will be generated upon assembly and listed on the assembler output listing. For further information on the operation, restrictions associated with, and use of the COP400 JID instruction, see *COP400 Microcontroller Family Chip User's Manual,* Sections 3.2 and 4.1.

Example: Create an address pointer table to be used by the COP400 JID instruction.

Assuming that program labels TBL1, TBL2 and TBL3 are located at memory locations 01D3, 01DF and 02C0, respectively, with the .ADDR directive placed in the program source code preceding memory location 01C0 using an Assignment Statement, then

```
.=X'1C0     ; SET LOCATION POINTER TO
            ; ROM LOCATION X'01C0
.ADDR TBL1,TBL2,TBL3
```

will place the following address pointer data in the following memory locations:

| Address (HEX) | Data (HEX) | |
|---|---|---|
| 01C0 | D3 | (lower 8 bits of address of TBL1 label) |
| 01C1 | DF | (lower 8 bits of address of TBL2 label) |
| 01C2 | XX | (ERROR message will be generated — TBL3 address is out of range for .ADDR directive) |

For further examples of the use of the .ADDR directive in conjunction with the JID instruction, see *COP400 Microcontroller Family Chip User's Manual,* Section 5.3.

### .PAGE Directive

Syntax: .PAGE [expression] [; comments]

Description: The .PAGE directive changes the assembler's location counter to the address of the beginning of the ROM page specified by the expression in the operand field. The value of the expression may not exceed the maximum ROM page number for the chip being used. (See .CHIP Directive.) There are 64 locations in each ROM page.

Example:
```
.PAGE 2     ; SET LOCATION COUNTER TO
            ; X'80
```

### .LOCAL Directive

Syntax: .LOCAL [;comments]

Description: The .LOCAL directive establishes a new program section for local labels (labels beginning with a dollar sign [$]). All local labels

between two .LOCAL directive statements have their values assigned to them only within that particular section of the program. Note that a .LOCAL directive is assumed at the beginning and the end of a program; thus, one .LOCAL directive within a program divides the program into two local sections. Up to 58 .LOCAL directives may appear in one assembly.

Example:

```
$X:.WORD    1     ; FIRST LABEL $X
.LOCAL            ; ESTABLISH NEW LOCAL SYMBOL
                  ; SECTION
$X:.WORD    1     ; SECOND LABEL $X, NO CONFUSION
                  ; SINCE THEY ARE IN DIFFERENT
                  ; "LOCAL" BLOCKS
```

### .SET Directive

Syntax: .SET symbol,expression [;comments]

Description: The .SET directive is used to assign values to symbols. In contrast to an ASSIGNMENT statement, a symbol assigned a value with the .SET directive *can be assigned* different values an arbitrary number of times within an assembly language program, with each new value taking precedence over the previous value for a particular symbol.

Example:

```
.SET  A,100        ; SET A = 100
.SET  B,50         ; SET B = 50
.SET  C,A-25*B/4   ; SET C = A-25*B/4
```

Note: This expression is always evaluated from left to right regardless of the operators used between the variables and constants unless parentheses appear in the expression.

### .CHIP Directive

Syntax: .CHIP expression [;comments]

Description: The .CHIP directive specifies to the assembler the particular COP device for which the assembly source code is being written. This is necessary since different COP400 devices having a different number of COP400 instructions may use the COP Cross Assembler. The devices which may be specified with the .CHIP directive and the corresponding values for their operand field expressions are as follows:

| COP400 Device | Operand Expression |
|---|---|
| COP410L | 410 |
| COP411L | 411 |
| COP420/420L/420C | *420 |
| COP421/421L/421C | 421 |
| COP440 | 440 |
| COP444L | 444 |
| COP445L | 445 |

*Indicates default value.

A feature associated with the .CHIP directive is that the assembler allows for multiple .CHIP directives in the program. The assembler will treat the program as one written for the COP device specified by the last .CHIP directive (or if none, the default device, the COP420) until it encounters a new .CHIP directive. It will then treat the program as one written for a different device as specified by the new .CHIP directive.

Examples:

1. No .CHIP directive:

```
.PAGE   0
  .
  .
  .
.END
```

Assembler assumes default device, the COP420.

2. Multiple .CHIP directives:

```
.PAGE   0          ; ASSEMBLER ASSUMES COP420
  .
  .
  .
.CHIP   440        ; ASSEMBLER ASSUMES COP440 FOR
                   ; FOLLOWING CODE UNTIL NEXT .CHIP
  .
  .
  .
.END
```

### .OPT Directive

Syntax: .OPT $\text{expression}_1$,$\text{expression}_2$ [;comments]

Description: The .OPT directive specifies to the assembler which mask-programmable options have been selected for the device for which the program is written (as specified by the .CHIP directive). The first expression indicates the option number; the second expression indicates the value to be assigned to the specified option number. Values for the first expression (option numbers) must be within the range 1 through 56; values for the second expression (option values)

must be within the range 0 through 14. A value of 15 indicates an undefined option. Also, option numbers and values must be valid for the particular COP device for which the program is written. For specific information on the options and values associated with COP400 devices, see *COP400 Microcontroller Family Chip User's Manual.*

The .OPT directive does not convey information to the assembler for its own use. It is necessary to provide option information to be included in the assembler Load Module output file for mask programming the selected options into the COP part when fabricated.

Examples:

```
.OPT    1,3    ; SPECIFY OPTION 1=3
.OPT    2,1    ; SPECIFY OPTION 2=1
```

### .INCLD Directive

Syntax:

```
.INCLD filename [;comments]
```

Description: The .INCLD directive includes the symbolic file specified in the operand field of the directive in the current assembler source code. Specifically, it causes the assembler to read source code from the specified file on the current diskette until an end-of-file mark is reached, at which time it will again start reading source code from the assembly input file (see Figure 8.1 below). The file must be a symbolic file. The default modifier is SRC. Since the specified file is included in the source code at assembly time, the included file must, as mentioned above, be contained on the current diskette at assembly time. Expansion of the source code included by this directive on the assembler output listing is controlled by bit 5 in the operand field of the .LIST directive. A .LIST with bit 5 set to "1" must be contained in the assembly source code *prior* to the .INCLD directive in order for the contents of the included file to be expanded on the assembler output listing (see .LIST directive, above).

Example:

```
.LIST    X'21       ; EXPAND .INCLD SOURCE CODE ON
                    ; OUTPUT LISTING
.INCLD   BCDADD     ; INCLUDE 'BCDADD.SRC' FILE ON
                    ; CURRENT DISKETTE
```

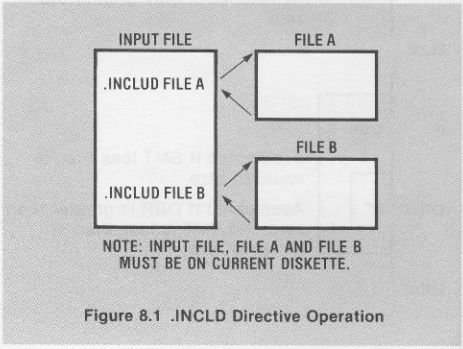



**Figure 8.1 .INCLD Directive Operation**

### Conditional Assembly Directives

Syntax:

```
[label] .IF      expression [;comments]
        .ELSE               [;comments]
        .ENDIF              [;comments]
```

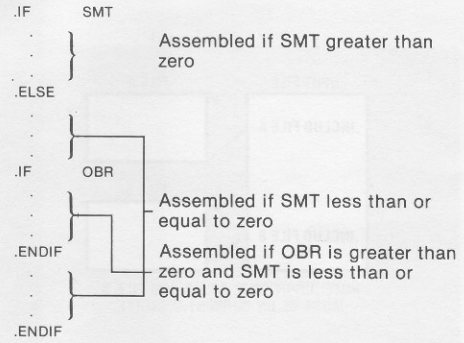
Description: The conditional assembly directives selectively assemble portions of a source program based on the value of the expression in the operand field of the .IF directive statement. All source statements between a .IF directive and its associated .ENDIF are defined as an .IF-.ENDIF block. These blocks may be nested to a depth of ten. The .ELSE directive can be optionally included in a .IF-.ENDIF block. The .ELSE directive divides the block into two parts. The first part of the source statement block is assembled if the .IF expression is greater than zero; otherwise, the second part is assembled. When the .ELSE directive is not included in a block, the block is assembled only if the .IF expression is greater than zero. If an error is detected in the expression, the assembler assumes a true value (greater than zero).

Examples:

1. Two-part conditional assembly:

```
.IF      COMPR
  .
  .      } Assembled if COMPR greater than zero
  .
.ELSE
  .
  .      } Assembled if COMPR less than or equal to zero
  .
.ENDIF
```

2. Nested .IF-.ENDIF block conditional assembly:




Labels appearing on .IF statements are assigned the address of the next assembled instruction. Labels cannot be used on .ELSE or .ENDIF directives.

Listing of conditional assembly code is controlled by the .LIST directive.

## 8.5 Macros

The primary use of macros is to make the assembly process easier, by inserting duplicative or similar assembly language statements into the program source code without the need to manually enter these statements into the program each time they are required. A macro, once defined, will automatically, during assembly time, place reiterative code or similar code with changed parameters into the assembler source code when called by its macro name. The following sections are devoted to explaining the process of defining and calling macros, with and without parameters, and describing assembler directives associated with the use of macros.

Using macros, a programmer can gradually build a library of basic routines, allowing variables unique to particular programming applications to be defined in and passed to a particular macro when called by main programs. Such macros can be automatically included in the assembly source code of main programs using the .INCLD directive (see Section 8.4.3) or read into the source code during an editing session using the *READ FROM <filename>* command (see Chapter 7).

### 8.5.1 Defining a Macro

The process of defining a macro involves preparing statements which perform the following functions:

- Give it a name
- Declare any parameters to be used
- Write the assembler statements it contains
- Establish its boundaries

Macros must be defined before their use in a program. Macro definitions within an assembly do not generate code. Code is generated only when macros are called by the main program. Macro definitions are formed as follows:

```
.MACRO      mname [,parameters]
.
.
.
macro body
.
.
.
.ENDM
```

where:

a. .MACRO is the directive mnemonic which initiates the macro definition. It must be terminated by at least one blank.

b. "mname" is the name of the macro. It is legal to define a macro with the same name as an already existing macro, in which case the latest definition is operative. Previous definitions are, however, retained in the macro definition table unless deleted from the buffer space by the .MDEL directive (see below). The macro name is used by the main program to call the macro, and must adhere to the rules given for symbol construction in Section 8.2.

c. [,parameters] is the optional list of parameters used in the macro definition. Each parameter must adhere to the symbol construction rules. Parameters are delimited from "mname" and successive parameters by commas.

The following are examples of legal and illegal .MACRO directives:

| Legal | Illegal | Reason Illegal |
|---|---|---|
| .MACRO MAC,A,B | .MACRO SUB,?1H | Special character used in parameter |
| .MACRO $ADD,OP1,OP2 | .MACRO 1MAC,C,D | First character of macro name numeric |
| .MACRO LIST,$1 | .MACRO MAC,25 | First character of parameter must be alphabetic |
| .MACRO MSG3 | .MACRO M$AC | Special character used in macro name |

d. The macro body consists of assembly language statements. The macro body may consist of simple text, text with parameters, and/or macro-time operators.

e. The .ENDM signifies the end of the macro and must be used to terminate a macro definition.

### Simple Macros

The simplest form of macro definition is one with no parameters or macro operators. The macro body is simply a sequence of assembly language statements which are substituted for each macro call. Of course such identical macro calls are inefficient if called repetitively within the same assembly program — a repeatedly used series of assembly language statements within a program should be coded as a subroutine. However, simple macros with no variables are useful in compiling a library of basic routines to be used within *different* programs, since, as mentioned above, they allow the programmer to simply call the macro within the program rather than repeatedly coding all the macro body statements into each program when needed. An example of a simple macro definition follows:

```
; MACRO "INC2" TO INCREMENT A 2-DIGIT BCD RAM COUNTER
; WHEN CALLED, B MUST POINT TO A LOW-ORDER DIGIT OF
;  COUNTER

.MACRO  INC2        ; BEGIN MACRO DEFINITION
SC                  ; INITIALIZE C TO 1 TO ADD LOW-ORDER
                    ;  DIGIT
CLRA                ; ZERO TO A
AISC    6           ; BCD ADJUST RESULT IF NECESSARY
ASC
ADT                 ; IF RESULT > 9, LOW-ORDER DIGIT = 0
XIS                 ; PLACE INCREMENTED DIGIT IN M,
                    ;  POINT TO HIGH-ORDER DIGIT
CLRA                ; ZERO TO A
AISC    6           ; ADD CARRY, IF PROPAGATED FROM
                    ;  LOW-ORDER DIGIT TO HIGH-ORDER
                    ;  DIGIT
ASC
ADT                 ; BCD ADJUST RESULT IF NECESSARY
X                   ; REPLACE DIGIT IN M
.ENDM
```

### Macros with Parameters

Obviously, the above macro could be made more flexible by the addition of parameters in the macro definition, allowing the programmer to specify the low-order digit of the RAM counter to be incremented in the macro call itself, rather than relying on the instruction in the main program which loads the B (RAM address) register with the proper value before calling the macro. The following is an example of the use of parameters within a macro definition to accomplish this result:

```
; MACRO "INC2A" TO INCREMENT A 2-DIGIT RAM COUNTER
; THE LOW-ORDER DIGIT OF THE COUNTER IS REPRESENTED
;  BY PARAMETERS "R","D"

.MACRO  INC2A,R,D ; R,D = REGISTER #, DIGIT # OF LOW-
                  ;  ORDER DIGIT COUNTER
LBI     R,D       ; POINT TO LOW-ORDER DIGIT OF
                  ;  COUNTER
SC                ; INITIALIZE C TO 1 TO ADD TO LOW-
                  ;  ORDER DIGIT
CLRA              ; ZERO TO A
AISC    6         ; BCD ADJUST RESULT IF NECESSARY
ASC
ADT               ; IF RESULT > 9, LOW-ORDER DIGIT = 0
XIS               ; PLACE INCREMENTED DIGIT IN M,
                  ;  POINT TO HIGH-ORDER DIGIT
CLRA              ; ZERO TO A
AISC    6         ; ADD CARRY, IF PROPAGATED FROM
                  ;  LOW-ORDER DIGIT TO HIGH-ORDER
                  ;  DIGIT
ASC
ADT               ; BCD ADJUST RESULT IF NECESSARY
X                 ; REPLACE DIGIT IN M
.ENDM             ; END MACRO DEFINITION
```

### 8.5.2 Calling a Macro

Once a macro has been defined, it may be called by a program to generate code. A macro is called by placing the macro name in the operation field of the assembly language statement and the actual value of parameters to be used (if any) by the symbolic macro definition parameters. The following form is used for a macro call:

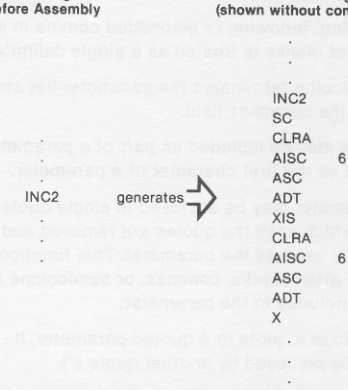
```
mname  [parameters]
```

where,

a. "mname" is the name previously assigned in the macro definition.

b. [parameters] is the list of input parameters. When a macro is defined without parameters, the parameter list is omitted from the call.

A call to the simple INC2 macro, defined above, would be expanded as follows:

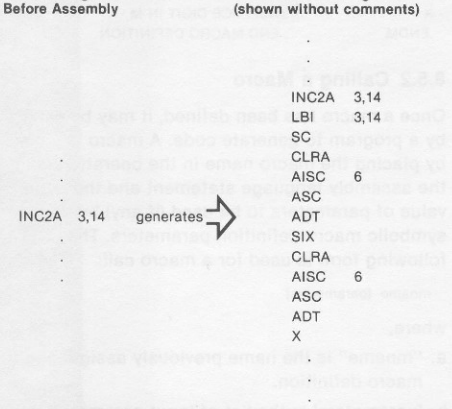



Note: The Macro call (INC2) as well as the expanded macro machine and source code will appear on the assembler output listing if a .LIST directive with bits 2 and 3 set is placed in the program's source code (see Section 8.4.3). The macro call statement (INC2) itself will not generate machine code.

### 8.5.3 Using Parameters

As already indicated, the power of a macro can be increased tremendously through the use of optional parameters. The parameters allow variable values to be declared when the macro is called.

For example, the "parameter" version of INC2, *INC2A* (Section 8.5.1), could be used to increment a 2-digit RAM based upon the parameter values specified in the macro call. The following macro call illustrates the use of the INC2A macro to increment a 2-digit RAM counter whose low-order digit is contained in RAM register 3, digit 14:




When parameters are included in a macro call, the following rules apply to the parameter list:

a. Commas or blanks delimit parameters.

b. Consecutive blanks are treated as a single delimiter.

c. A leading, following or embedded comma in a string of blanks is treated as a single delimiter.

d. A semicolon terminates the parameter list and starts the comment field.

e. Quotes may be included as part of a parameter except as the first character of a parameter.

f. A parameter may be enclosed in single quotes ('), in which case the quotes are removed and the string is used as the parameter. This function is useful when blanks, commas, or semicolons are to be included in the parameter.

g. To include a quote in a quoted parameter, it must be preceded by another quote (").

h. Missing or null parameters are treated as strings of length zero.

#### Parameters Referenced by Number

"#" is a macro operator that references the parameter list in the macro call. When used in an expression, it is replaced by the number of parameters in the macro call. The following .IF directive, for example, causes the conditional code to be expanded if there are more than 10 parameters in the macro call:

```
.IF     #>10
```

'#N' — Nth Parameter

When used in conjunction with a constant or variable, the '#' operator references individual parameters in the parameter list. The following example demonstrates how this function may be used in defining and calling a macro to establish a program memory data table:

```
.MACRO  X               ; MACRO DEFINITION
.WORD   #1,#2,#3
.ENDM
```

```
Macro Call                      Generated Code
.                               .
.                               .
.                               .
X       X'61,X'FF,X'90          .WORD  X'61,X'FF,X'90
.                               .
.                               .
.                               .
```

This technique eliminates the need for naming each parameter in the macro definition, particularly convenient when long parameter lists are to be used. It also allows powerful macros to be defined using an arbitrary number of parameters.

### 8.5.4 '^' — Concatenation Operator

The "^" macro operator is used for concatenation. When found, the "^" is removed from the output string and the strings on each side of the operator are compressed together after parameter substitution.

Example:

```
.MACRO      LABEL,X
R^X:        .WORD 1
I^X:        .WORD 1
```

The Macro call:

```
.
.
LABEL  0
.
.
.
```

generates:

```
.
.
.
R0:  .WORD  1
I0:  .WORD  1
.
.
.
```

Another example of the use of this operation is shown in Section 8.5.8 (Macro-Time Loop Example).

### 8.5.5 Local Symbols

```
.MLOC symbol[,symbol . . .] [;comments]
```

When a label is defined within a macro, a duplicate definition results with the second and each subsequent call of the macro. This problem can be avoided by using the .MLOC directive to declare labels local to the macro definition. In other words, if a macro definition containing fixed labels is to be called more than once during an assembly, duplicate definition errors will occur unless the .MLOC directive is used in the macro definition.

To illustrate this problem, consider the following macro definition, intended for multiple calls in an assembly, which does not use the .MLOC directive. Since it is a multiple loop routine, jumping back to the "CLRA" instruction, the inability to use a fixed label referencing this instruction requires the use of more complicated transfer of control instructions (JP) referenced to the assembly location counter (".") and not to one label:

```
; MACRO "CLRAM" TO CLEAR ALL DIGITS (0-15) OF ALL COP420
;  DATA MEMORY REGISTERS (0-3)

.MACRO  CLRAM
LBI     3,0         ; CLEAR REGISTER 3 FIRST
CLRA
XIS                 ; EXCHANGE ZEROS INTO MEMORY DIGIT
JP      -2          ; JUMP BACK TO "CLRA" UNTIL REGISTER
                    ;  CLEARED
XABR                ; REGISTER CLEARED, BR TO A
AISC    15          ; REGISTER 0 CLEARED?
JP      .+3         ; YES, JUMP TO FIRST INSTRUCTION
                    ;  AFTER ROUTINE
XABR                ; NO, BR -1 TO BR
JP      .-7         ; JUMP BACK TO "CLRA" TO CLEAR NEXT
                    ;  REGISTER
.ENDM
```

Now here is the same macro (without comments) using the .MLOC directive which allows the fixed label, "CLEAR," to be referenced by the JP instructions:

```
.MACRO  CLRAM
.MLOC   CLEAR
        LBI     3,0
CLEAR:  CLRA
        XIS
        JP      CLEAR
        XABR
        AISC    15
        JP      .+3
        XABR
        JP      CLEAR
        .ENDM
```

The .MLOC directive may occur at any point in a macro definition, but it must precede the first occurrence of the symbol(s) it declares local. If it does not, no error will be reported per se, but symbols used before the .MLOC will not be recognized as local. Local macro labels appear in the symbol table map at the end of the assembly listing as ZZXXXX, where XXXX is a particular hex number.

### 8.5.6 Conditional Expansion

The versatility and power of the macro assembler is enhanced by the conditional assembly directives. The conditional assembly directives (.IF, .ELSE and .ENDIF) allow the user to generate different lines of code from the same macro simply by varying the parameter values used in the macro calls. Three relational operators are provided:

= (equal)
< (less than)
> (greater than)

### .IF, .ELSE, .ENDIF Directives

When the macro assembler encounters a .IF directive within a macro expansion, it evaluates the relational operation that follows. If the expression is satisfied (evaluated greater than 0), the lines following the .IF are expanded until a .ELSE or a .ENDIF directive is encountered. If the expression is not satisfied (evaluated less than or equal to 0), only the lines from the .ELSE to the .ENDIF are expanded. See Section 8.4.3 for additional information on the conditional assembly directives.

Example:

```
; SHIFT THE CONTENTS OF RAM ADDRESS R,D RIGHT IF
;  N>0, LEFT OTHERWISE
 .MACRO  SHIFT,R,D,N
 LBI     R,D             ; POINT TO RAM DIGIT R,D
 .IF     N>0
 CLRA                    ; SHIFT RIGHT IF N>0
 SKMBZ   3
 AISC    4
 SKMBZ   2
 AISC    2
 SKMBZ   1
 AISC    1
 .ELSE                   ; SHIFT LEFT IF N≤0
 LD
 ADD
 .ENDIF
 X                       ; EXCHANGE SHIFTED DIGIT IN A
                         ;  BACK INTO RAM
 .ENDM
```

### .IFC Directive

Syntax: [label] .IFC $string_1$ operator $string_2$ [;comments]

Description: The .IFC directive allows conditional assembly based on character strings rather than the value of an expression as in the .IF directive. $String_1$ and $string_2$ are the character strings to be compared. Operator is the relational operator between the strings. Two operators are allowed: EQ (equal) and NE (not equal). If the relational operator is satisfied, the lines followng the .IFC are assembled until a .ELSE or a .ENDIF is encountered. The .ELSE and .ENDIF directives have the same effect with the .IFC directive as they do with the .IF directive.

The primary application of the .IFC is to compare a parameter value such as #3 against a specific string.

Example:

```
.IFC #3 NE INTEGER
```

### 8.5.7 Useful Directives

#### .SET Directive

Syntax: [label] .SET symbol, expression [;comments]

Description: The .SET directive is used to assign values to symbols (variables). A variable assigned a value with the .SET directive can be reassigned different values an arbitrary number of times (see Section 8.4.3). Set variables are useful during macro expansion to control macro-time looping and macro communication. To insure value correspondence between pass 1 and pass 2 of the assembler, all values in the expression must be defined before use in a .SET directive. If a value is not previously defined, an error is reported and a value of zero is returned. For an example of use of the .SET directive in a macro-time loop, see Section 8.5.8.

#### .MDEL Directive

Syntax: [label] .MDEL mname[,mname . . .] [;comments]

Description: The .MDEL directive deletes macro definitions from the macro definition table and frees the buffer space used by the definitions.

Example:

```
.MDEL INC2
```

#### .ERROR Directive

Syntax: [label] .ERROR ['string'] [;comments]

Description: The .ERROR directive generates an error message and an assembly error that is included in the error count at the end of the program. The directive is useful for parameter checking in macros. For example, the INC2A macro, defined in Section 8.5.1, will put out erroneous code, if written for a COP420 program, if R>3 or D>15, since the COP420 has 4 RAM registers (0-3) containing 16 digits (0-15) each. To flag this condition with an error message, the following .ERROR directives may be included in the INC2A macro definition:

```
.MACRO   INC2A,R,D
.IF      D>15
.ERROR   'LBI WILL NOT WORK WITH D VALUE>15'
.ELSE
.IF      R>3
.ERROR   'LBI WILL NOT WORK WITH R VALUE>3'
.ELSE
LBI      R,D
SC
CLRA
AISC     6
ASC
ADT
XIS
CLRA
AISC     6
ASC
ADT
X
.ENDIF
.ENDIF
.ENDM
```

### 8.5.8 Macro-Time Looping

#### .DO and .ENDDO Directives

Syntax:

```
[label] .DO      count [;comments]
[label] .ENDDO         [;comments]
```

Description: Macro-time looping is facilitated through the .DO and .ENDDO directives. These directives are used to delimit a block of statements which are repeatedly assembled. The number of times the block will be assembled is specified by the .DO directive "count" value. Following is the format of a .DO-.ENDDO block:

```
.DO      count
          .
          .
          .
         source
          .
          .
          .
        .ENDDO
```

Note: .DO, .ENDDO, and .EXIT are defined only within a macro definition.

The "X" macro described in the section on "#" could be modified to generate a variable number of words, using .DO and a loop counter.

#### .EXIT Directive

Syntax: [label] .EXIT [;comments]

Description: Early termination of looping in a .DO-.ENDOO block can be effected with the .EXIT directive. This directive allows the current loop to finish and then terminates looping. The .EXIT directive is commonly used in conjunction with a conditional test within a macro loop which will exit from the loop if a variable is equal to a particular value. In such cases the .DO "count" value is not crucial, provided it exceeds the maximum number of times the .DO loop will be required or expected to be

executed for a particular macro definition or for possible macro calls.

**Example of a Macro-Time Loop**

The following examples show the use of the .DO, .ENDDO, and .EXIT directives. The macro CTAB generates a constant table from 0 to MAX where MAX is a parameter of the macro call. Each word has a label D0X:, where X is the value of the data word.

```
        .MACRO   CTAB,MAX
        .SET     X,0
        .DO      MAX + 1
D0  X:  .WORD    X
        .SET     X,X + 1
        .ENDDO
        .ENDM
```

Now a call of the form:

```
        .CTAB    10
```

generates code equivalent to:

```
        .SET     X,0
D00:    .WORD    X
        .SET     X,X + 1
D01:    .WORD    X
        .SET     X,X + 1
D02:    .WORD    X
         .
         .
         .
        .SET     X,X + 1
D09:    .WORD    X
        .SET     X,X + 1
D0A:    .WORD    X
```

Note: Care must be taken when writing macros that generate a variable number of data words through the use of the .IF or the .DO directives. If the operands on these directives are forward referenced, their values change between pass 1 and pass 2 and the number of generated words may change. Should this be the case, all labels defined after the macro call that has changed values generate numerous assembly errors of the following form:

```
ERROR DUP. DEF.
```

### 8.5.9 Nested Macro Calls

Nested macro calls are allowed; that is, a macro definition may contain a call to another macro. When a macro call is encountered during macro expansion, the state of the macro currently being expanded is saved and expansion begins on the nested macro. Upon completing expansion of the nested macro, expansion of the original macro continues. Depth of nesting allowed will depend on the parameter list sizes, but on the average about 10 levels of nesting will be allowed.

A logical extension of a nested macro call is a recursive macro call, that is, a macro that calls itself. This is allowed, but care must be taken that an infinite loop is not generated.

### 8.5.10 Nested Macro Definitions

A macro definition can be nested within another macro. Such a macro is not defined until the outer macro is expanded and the nested .MACRO statement is executed. This allows the creation of special-purpose macros based on the outer macro parameters and, when used with the .MDEL directive, allows a macro to be defined only within the range of the macro that uses it.

## 8.6. Example of Creating and Assembling a User Program

The following example illustrates the basic process of creating an assembly language file and, after checking for errors, assembling the user program file. The use of a diskette containing the PDS main programs EDIT, LIST and ASM with the volume name "1" is assumed. The user program given is a sample display/keyboard debounce-decode program similar in most respects to a program treated in detail in Section 5.3 of the *COP400 Microcontroller Family Chip User's Manual*. This program illustrates typical usage of some of the most commonly used assembler directives. The use of a CRT console and high speed printer is assumed. The assembler input file, DSPLY.SRC, as well as the output file, DSPLY.LM, are written to and read from the same diskette containing the system main programs mentioned above, disk 1.

**Creating File DSPLY.SRC**

Assuming PDS has been initialized and the EXEC program is currently in use, the user creates the DSPLY.SRC assembly as follows:

1. Invoke the EDIT program:

   ```
   X>@EDIT (CR)
   EDIT,REV:A
   E>
   ```

2. Next, enter the disk-edit mode, creating a new filename, DSPLY.SRC. (The EDIT program displays the number of available sectors on the DISK.):

   ```
   E>E DSPLY (CR)
   CREATE NEW FILE  (Y/N,CR = YES)? (CR)
   AVAILABLE SECTORS  (# of sectors)
   E>
   ```

3. Enter input mode and insert assembly language statements as shown in Figure 8.2. After entering a line and pressing a carriage return,

EDIT will re-prompt with the next line number followed by a "?":

```
E>CR
  1?      .TITLE DSPLY,'COP420 DISPLAY DEMO' CR
  2?
  .
  .
  .       source code
  .
  .
  .
229?      .END CR
230?      CR
E>
```

4. Exit input mode by pressing CR and finish the edit, writing the assembly language file to the disk, catalogued as DSPLY.SRC:

```
E>F CR
FINISH CURRENT EDIT (Y/N,CR = YES)? CR
E>
```

5. If the user desires, the first debug of the entered code may be performed by using the LIST system program to obtain a listing of the source code to verify proper format and content of the assembly language statements prior to an assembly. The following command calls the LIST program and outputs a listing to the line printer as shown in Figure 8.2, unformatted except for line numbers:

```
E>@LIST DSPLY.SRC PR NE NH NP CR
LIST,REV:A
```

(Listing now begins.)

6. Having verified the assembly language statements contained in DSPLY.SRC, the user may perform a limited assembly of the program, obtaining only an error listing on the CRT to determine if any edits are required as indicated by the error message output to the CRT. The following command *calls and invokes* an assembly of DSPLY.SRC, outputting an error message listing as shown on the CRT. (If any errors had been encountered during the assembly, the line numbers, assembly language statements and type of errors would be displayed as well as the count of the total number of assembly errors.):

```
L>@ASM I = DSPLY,O = DSPLY,L = *CN,EL CR
ASM,REV:A
END PASS 1  (Error message listing follows.)
COP CROSS ASSEMBLER PAGE 1
DSPLY COP420 DISPLAY DEMO
NO ERROR LINES
  227 ROM WORDS USED
END PASS 2
SOURCE CHECKSUM = D7A7
OBJECT CHECKSUM = 0529
INPUT FILE 1:DSPLY.SRC
A>
```

If error lines had been displayed, the user may have been able to determine from the error message definitions the proper edits to make to the source code without the need for a complete assembler output listing. If not, to obtain a complete assembler output listing, the following command would be entered on the console:

```
A>I = DSPLY,L = *PR CR
```

7. After obtaining an error-free assembly, the user can create a load module file for loading into PDS shared memory for debugging and obtain a full output listing on the printer as follows:

```
A>I = DSPLY,O = DSPLY,L = *PR CR
CREATING FILE 1:DSPLY.LM
END PASS 1  (Listing begins to printer.)
END PASS 4
A>
```

Figure 8.3 provides a complete assembler output listing for an assembly of DSPLY.SRC.

```
1             .TITLE  DSPLY,'COP420 DISPLAY DEMO'
2   ;COP 420 DISPLAY/KEYBOARD DEBOUNCE/DECODE ROUTINE.
3   ;DISPLAYS 14 BCD DIGITS CONTAINED IN M(0,14) THROUGH
4   ;M(0,1), HIGH-ORDER TO LOW-ORDER, RESPECTIVELY.
5   ;DECIMAL POINT POSITION VALUE CONTAINED IN M(0,15).
6   ;DIGIT POSITION VALUE CONTAINED IN M(1,15).
7   ;TEMPORARY STORAGE OF 4 BITS OF SEGMENT DATA IN M(3,14).
8   ;KEYBOARD DEBOUNCE COUNTER (KBC) CONTAINED IN M(3,15).
9   ;SEVEN-SEGMENT DECODE ROM LOOKUP DATA CONAINED IN PAGE
10  ;4, WORDS 0 - F.
11  ;KEYSTRAP DATA TIED TO D14-D12 LINES PLACED IN M(1,14)
12  ;THROUGH M(1,12).
13  ;EXIT TO KEYDECODE ROUTINE AFTER DEBOUNCING KEYSWITCH
14  ;CLOSURES WITH DIGIT VALUE IN M(1,15) AND G PORT DATA
15  ;IN A.
16            .SPACE  5           ;LEAVE 5 BLANK LINES ON LISTING
17            .PAGE   0
18            DIGIT = 1,15        ;ASSIGN VALUE 1,15 TO "DIGIT"
19            STORE = 3,14        ;ASSIGN VALUE 3,14 TO "STORE"
20            KBC   = 3,15        ;ASSIGN VALUE 3,15 TO "KBC"
21            CLRA                ;FIRST INSTRUCTION MUST BE A "CLRA"
22  START:    JSR     CLRAM       ;CLEAR ALL RAM
23            LBI     0,14
24  LDRAM:    CBA
25            XDS                 ;LOAD DISPLAY REGISTER WITH NUMBERS
26                                ;14 - 1
27            JP      LDRAM
28  DSPLY:    OGI     15          ;SET ALL G PORTS HIGH
29            LBI     KBC         ;POINT TO M(3,15))
30            STII    15          ;15 TO KBC
31  DSP1:
32            LBI     0,14        ;NO, START DISPLAY AT DIGIT 14
33  DSP2:     CBA                 ;DIGIT POSITION TO A
34            XAD     DIGIT       ;STORE IN M(1,15)
35            CLRA
36            AISC    4           ;SET A2 TO FLIP TO PAGE 1 FOR LOOKUP
37            LEI     0           ;BLANK SEGMENTS (RESET EN2)
38            LQID                ;LOOKUP TABLE SEGMENT DATA TO Q
39            LBI     DIGIT       ;POINT TO DIGIT POSITION
40            LD      1           ;DIGIT POSITION TO A, POINT TO
41                                ;DECIMAL POINT POSITION DIGIT TO A
42            SKE                 ;DECIMAL POINT = DIGIT POSITION?
43            JMP     NODP        ;NO, RESET DECIMAL POINT BIT IN Q
44            CLRA
45            AISC    4
46            JP      .-1         ;DELAY 9 INSTR. CYCLE TIMES
47  DIGOUT:   LBI     DIGIT       ;POINT TO DIGIT POSITIONN
48            LD                  ;DIGIT POSITION TO A
49            CAB                 ;DIGIT POSITION TO BD
50            OBD                 ;OUTPUT DIGIT VALUE
```

**Figure 8.2. DSPLY.SRC Source Code**

```
51            LEI     4         ;OUTPUT SEGMENT DATA (SET ENZ)
52            LBI     KBC       ;POINT TO KBC
53            ING               ; G PORTS TO A
54            AISC    1         ;ALL G PORTS STILL HIGH (= 15)?
55            JMP     KEYDWN    ;NO, JUMP TO "KEYDOWN" ROUTINE
56            CLRA
57            AISC    3         ;YES, DELAY 13 INSTR. CYCLE TIMES
58            JP      .-1       ;BACK TO PREVIOUS INSTR. UNTIL SKIP
59            LBI     KBC
60            JMP     NRDY
61            .FORM             ;FORM FEED
62            .PAGE   1
63    ;WORDS 0-F EQUAL SEVEN-SEGMENT DECODE LOOKUP DATA TABLE
64    ;I(0) - I(7) = D.P.,SG - SA
65    ;SENT UPON LOOKUP TO Q(7) - Q(0), RESPECTIVELY.
66    ;HEX VALUE FOR ASCII CHARACTERS 0 - 9,P,A,U,C,F,BLANK
67    ;PLACED IN SUCCESSIVE LOCATIONS BY ".WORD" DIRECTIVE.
68            .SPACE  5         ;LEAVE 5 BLANK LINES ON LISTING
69            .WORD   X'FD      ;=0      (7-SEGMENT DECODE HEX VALUES)
70            .WORD   X'61      ;=1
71            .WORD   X'DB      ;=2
72            .WORD   X'F3      ;=3
73            .WORD   X'67      ;=4
74            .WORD   X'B7      ;=5
75            .WORD   X'3F      ;=6
76            .WORD   X'E1      ;=7
77            .WORD   X'FF      ;=8
78            .WORD   X'E7      ;=9
79            .WORD   X'CF      ;=P
80            .WORD   X'EF      ;=A
81            .WORD   X'7D      ;=U
82            .WORD   X'9D      ;=C
83            .WORD   X'8F      ;=F
84            .WORD   X'00      ;=BLANK
85    DEBOUN:
86            SKMBZ   3         ;UP BIT = 1?
87            JP      ALLUP     ;YES
88            SKMBZ   2         ;NO, NRB = 1?
89            JP      STR       ;YES, A = 15 SO STORE IT IN KBC
90    DECKBC: ADD               ;DECREMENT KBC
91    STR:    X                 ;PLACE A IN KBC
92            SMB     3         ;SET UP BIT OF KBC
93            JMP     DSP1      ;DO DISPLAY LOOP OVER AGAIN
94    ALLUP:  SKMBZ   2         ;NRB = 1?
95            JP      DECKBC    ;YES, DECREMENT KBC (A=15)
96            AISC    4         ;NO, SET KBC = 11
97            NOP               ;DEFEAT "AISC" SKIP
98            JP      STR
99    KEYDWN: LDD     DIGIT     ;DIGIT POSITION TO A
100           AISC    4         ;DIGIT POSITION > 11 (STRAP DATA)?
```

**Figure 8.2. DSPLY.SRC Source Code (continued)**

```
101             JP      KBCTST  ;NO
102             AISC    12      ;YES, RESTORE STRAP DIGIT VALUE
103             CAB             ;STRAP DIGIT POSITION TO BD
104             CLRA
105             AISC    1
106             XABR            ;1 TO BR(POINT TO STRAP DATA REG. 1)
107             ING             ;STRAP DATA TO A
108             X               ;PLACE IN APPROPRIATE DIGIT, REG. 1
109             JMP     NRDY
110     KBCTST: RMB     3       ;RESET UP BIT OF KBC
111             CLRA
112             AISC    8       ;DELAY 5 INSTR. CYCLE TIMES
113             JP      .-1     ;REPEAT PREVIOUS INSTR. UNTIL SKIP
114             CLRA            ;0 TO A
115             SKE             ;KBC = 0?
116             JMP     NRDY    ;NO
117             LEI     0       ;YES, BLANK SEGMENTS
118             ING             ; G PORTS TO A
119             LBI     DIGIT   ;POINT TO DIGIT VALUE
120             JMP     KEYDEC  ;JUMP TO KEYDECODE ROUTINE
121             .FORM
122             .PAGE   2
123     CLRAM:  LBI     3,0
124     CLEAR:  CLRA
125             XIS
126             JP      CLEAR
127             XABR
128             AISC    15      ;REGISTER 0 CLEARED?
129             RET             ;YES, RETURN
130             XABR            ;NO, BR -1 TO BR
131             JP      CLEAR
132     BLANK:  CLRA            ;PLACE "F"S (DISPLAY BLANKS) IN A
133                             ;RAM REGISTER
134             AISC    15
135             XIS
136             JP      BLANK
137             RET
138             .FORM           ;FORM FEED
139             .PAGE   4
140     ;FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH COLUMN
141     ;AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT FROM DISPLAY
142     ;ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO KEY1 - KEY16
143     ;DECODE ROUTINES. LABELS "KEY1" THROUGH "KEY16" MUST
144     ;BE LOCATED WITHIN PAGES 4 THROUGH 7.
145              SPACE  5       ;FIVE BLANK LINES ON LISTING
146     KEYDEC:  COMP           ;COMPLEMENT A SO THAT BIT=1
147                             ;INDICATES KEY CLOSURE
148             JID             ;JUMP TO KEYDECODE ROUTINE FOR
149                             ;PARTICULAR KEY CLOSURE
150               = X'111       ;MOVE ASSEMBLER LOCATION
```

**Figure 8.2. DSPLY.SRC Source Code (continued)**

```
                                    ;COUNTER TO KEY1 ROM POINTER ADDRESS
          .ADDR    KEY1             ;PLACE KEY1 POINTER IN ADDRESS X'111
          .ADDR    KEY2             ;PLACE KEY2-KEY4 POINTERS IN NEXT
          .ADDR    KEY3             ;ROM LOCATIONS
          .ADDR    KEY4
          . = X'121                 ;MOVE TO KEY5 POINTER LOCATION
          .ADDR    KEY5
          .ADDR    KEY6
          .ADDR    KEY7
          .ADDR    KEY8
          . = X'141                 ;MOVE TO KEY9 POINTER LOCATION
          .ADDR    KEY9             ;(PAGE 5)
          .ADDR    KEY10
          .ADDR    KEY11
          .ADDR    KEY12
          . = X'181                 ;MOVE TO KEY13 POINTER LOCATION
          .ADDR    KEY13            ;(PAGE 6)
          .ADDR    KEY14
          .ADDR    KEY15
          .ADDR    KEY16
KEY1:     JMP      ONE              ;G0,D1 KEY
KEY2:     JMP      TWO              ;G0,D2 KEY
KEY3:     JMP      THREE            ;G0,D3 KEY
KEY4:     JMP      FOUR             ;G0,D4 KEY
KEY5:     JMP      FIVE             ;G1,D1 KEY
KEY6:     JMP      SIX              ;G1,D2 KEY
KEY7:     JMP      SEVEN            ;G1,D3 KEY
KEY8:     JMP      EIGHT            ;G1,D4 KEY
KEY9:     JMP      NINE             ;G2,D1 KEY
KEY10:    JMP      TEN              ;G2,D2 KEY
KEY11:    JMP      ELEVEN           ;G2,D3 KEY
KEY12:    JMP      TWELVE           ;G2,D4 KEY
KEY13:    JMP      THIRTN           ;G3,D1 KEY
KEY14:    JMP      FOURTN           ;G3,D2 KEY
KEY15:    JMP      START            ;G3,D3 KEY
KEY16:    JMP      START            ;G3,D4 KEY
NRDY:     LDD      DIGIT            ;DIGIT POSITION TO A
          AISC     14               ;LAST DIGIT DONE (A = 1)?
          JMP      DEBOUN           ;YES, JUMP TO DEBOUNCE ROUTINE (A=15)
          AISC     1                ;NO, DECREMENT DIGIT POSITION VALUE
          LBI      0,0              ;POINT TO DISPLAY REGISTER 0
          CAB                       ;DIGIT POSITION VALUE TO BD
          CLRA
          AISC     4                ;DELAY 9 INSTR. TIMES
          JP       .-1              ;REPEAT PREVIOUS INSTR. UNTIL SKIP
          JMP      DSP2             ;DISPLAY NEXT DIGIT
NODP:     LBI      STORE            ;POINT TO M(2,15)
          COMA                      ;SE-SG,D.P. TO A
          X                         ;EXCHANGE INTO M(2,15)
          RMB      0                ;RESETD.P. BIT (DECIMAL POINT OFF)
```

**Figure 8.2. DSPLY.SRC Source Code (continued)**

```
201             CAMQ                ;SEGMENT DATA BACK TO Q
202             JMP     DIGOUT
203
204             .FORM
205             .PAGE   7
206     ONE:    LBI     0,1
207     TWO:    LBI     0,2
208     THREE:  LBI     0,3
209     FOUR:   LBI     0,4
210     FIVE:   LBI     0,5
211     SIX:    LBI     0,6
212     SEVEN:  LBI     0,7
213     EIGHT:  LBI     0,8
214     NINE:   LBI     0,9
215     TEN:    LBI     0,10
216     ELEVEN: LBI     0,11
217     TWELVE: LBI     0,12
218     THIRTN: LBI     0,13
219     FOURTN: LBI     0,14
220             CBA
221             XAD     1,9         ;SAVE KEY NUMBER
222             LBI     0,0
223             JSR     BLANK       ;BLANK DISPLAY REGISTER
224             LBI     0,0
225             LDD     1,9         ;KEY NUMBER TO A
226             CAB
227             X                   ;KEY NUMBER TO DISPLAY REGISTER
228             JMP     DSPLY
229             .END
```

Figure 8.2. DSPLY.SRC Source Code (continued)

```
46 01B DA              JP      -1      ;DELAY 9 INSTR. CYCLE TIMES
47 01C 1E     DIGOUT:  LBI     DIGIT   ;POINT TO DIGIT POSITIONN
48 01D 05              LD              ;DIGIT POSITION TO A
49 01E 50              CAB             ;DIGIT POSITION TO BD
50 01F 333E            OBD             ;OUTPUT DIGIT VALUE
51 021 3364            LEI     4       ;OUTPUT SEGMENT DATA (SET EN2)
52 023 3E              LBI     KBC     ;POINT TO KBC
53 024 332A            ING             ; G PORTS TO A
54 026 51              AISC    1       ;ALL G PORTS STILL HIGH (= 15)?
55 027 605E            JMP     KEYDWN  ;NO, JUMP TO "KEYDOWN" ROUTINE
56 029 00              CLRA
57 02A 53              AISC    3       ;YES, DELAY 13 INSTR. CYCLE TIMES
58 02B EA              JP      -1      ;BACK TO PREVIOUS INSTR. UNTIL SKIP
59 02C 3E              LBI     KBC
60 02D 61A5            JMP     NRDY
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

```
61                      .FORM           ;FORM FEED


62      0040            .PAGE   1
63                  ;WORDS 0-F EQUAL SEVEN-SEGMENT DECODE LOOKUP DATA TABLE
64                  ;I(0) - I(7) = D.P.,SG - SA
65                  ;SENT UPON LOOKUP TO Q(7) - Q(0), RESPECTIVELY.
66                  ;HEX VALUE FOR ASCII CHARACTERS 0 - 9,P,A,U,C,F,BLANK
67                  ;PLACED IN SUCCESSIVE LOCATIONS BY ".WORD" DIRECTIVE.
68      0005            .SPACE  5       ;LEAVE 5 BLANK LINES ON LISTING


69 040 FD               .WORD   X'FD    ;=0     (7-SEGMENT DECODE HEX VALUES)
70 041 61               .WORD   X'61    ;=1
71 042 DB               .WORD   X'DB    ;=2
72 043 F3               .WORD   X'F3    ;=3
73 044 67               .WORD   X'67    ;=4
74 045 B7               .WORD   X'B7    ;=5
75 046 3F               .WORD   X'3F    ;=6
76 047 E1               .WORD   X'E1    ;=7
77 048 FF               .WORD   X'FF    ;=8
78 049 E7               .WORD   X'E7    ;=9
79 04A CF               .WORD   X'CF    ;=P
80 04B EF               .WORD   X'EF    ;=A
81 04C 7D               .WORD   X'7D    ;=U
82 04D 9D               .WORD   X'9D    ;=C
83 04E 8F               .WORD   X'8F    ;=F
84 04F 00               .WORD   X'00    ;=BLANK
85              DEBOUN:
86 050 13               SKMBZ   3       ;UP BIT = 1?
87 051 D9               JP      ALLUP   ;YES
88 052 03               SKMBZ   2       ;NO, NRB = 1?
89 053 D5               JP      STR     ;YES, A = 15 SO STORE IT IN KBC
90 054 31       DECKBC: ADD             ;DECREMENT KBC
91 055 06       STR:    X               ;PLACE A IN KBC
92 056 4B               SMB     3       ;SET UP BIT OF KBC
93 057 600B             JMP     DSP1    ;DO DISPLAY LOOP OVER AGAIN
94 059 03       ALLUP:  SKMBZ   2       ;NRB = 1?
95 05A D4               JP      DECKBC  ;YES, DECREMENT KBC (A=15)
96 05B 54               AISC    4       ;NO, SET KBC = 11
97 05C 44               NOP             ;DEFEAT "AISC" SKIP
98 05D D5               JP      STR
99 05E 231F     KEYDWN: LDD     DIGIT   ;DIGIT POSITION TO A
100 060 54              AISC    4       ;DIGIT POSITION > 11 (STRAP DATA)?
101 061 EC              JP      KBCTST  ;NO
102 062 5C              AISC    12      ;YES, RESTORE STRAP DIGIT VALUE
103 063 50              CAB             ;STRAP DIGIT POSITION TO BD
104 064 00              CLRA
105 065 51              AISC    1
106 066 12              XABR            ;1 TO BR(POINT TO STRAP DATA REG.1)
107 067 332A            ING             ;STRAP DATA TO A
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

```
108 069 06                  X                   ;PLACE IN APPROPRIATE DIGIT, REG. 1
109 06A 61A5                JMP     NRDY
110 06C 43          KBCTST: RMB     3           ;RESET UP BIT OF KBC
111 06D 00                  CLRA
112 06E 58                  AISC    8           ;DELAY 5 INSTR. CYCLE TIMES
113 06F EE                  JP      -1          ;REPEAT PREVIOUS INSTR. UNTIL SKIP
114 070 00                  CLRA                ;0 TO A
115 071 21                  SKE                 ;KBC = 0?
116 072 61A5                JMP     NRDY        ;NO
117 074 3360                LEI     0           ;YES, BLANK SEGMENTS
118 076 332A                ING                 ; G PORTS TO A
119 078 1E                  LBI     DIGIT       ;POINT TO DIGIT VALUE
120 079 6100                JMP     KEYDEC      ;JUMP TO KEYDECODE ROUTINE


121                          FORM

122         0080            .PAGE   2
123 080 3F          CLRAM:  LBI     3,0
124 081 00          CLEAR:  CLRA
125 082 04                  XIS
126 083 81                  JP      CLEAR
127 084 12                  XABR
128 085 5F                  AISC    15          ;REGISTER 0 CLEARED?
129 086 48                  RET                 ;YES, RETURN
130 087 12                  XABR                ;NO, BR -1 TO BR
131 088 81                  JP      CLEAR
132 089 00          BLANK:  CLRA                ;PLACE "F"S (DISPLAY BLANKS) IN A
133                                             ;RAM REGISTER
134 08A 5F                  AISC    15
135 08B 04                  XIS
136 08C 89                  JP      BLANK
137 08D 48                  RET
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

```
138                     .FORM                   ;FORM FEED


139     0100            .PAGE   4
140                     ;FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH COLUMN
141                     ;AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT FROM DISPLAY
142                     ;ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO KEY1 - KEY16
143                     ;DECODE ROUTINES.  LABELS "KEY1" THROUGH "KEY16" MUST
144                     ;BE LOCATED WITHIN PAGES 4 THROUGH 7.
145     0005            .SPACE  5               ;FIVE BLANK LINES ON LISTING


146 100 40      KEYDEC: COMP                    ;COMPLEMENT A SO THAT BIT=1
147                                             ;INDICATES KEY CLOSURE
148 101 FF              JID                     ;JUMP TO KEYDECODE ROUTINE FOR
149                                             ;PARTICULAR KEY CLOSURE
150     0111            .   = X'111             ;MOVE ASSEMBLER LOCATION
151                                             ;COUNTER TO KEY1 ROM POINTER ADDRESS
152 111 85              .ADDR   KEY1            ;PLACE KEY1 POINTER IN ADDRESS X'111
153 112 87              .ADDR   KEY2            ;PLACE KEY2-KEY4 POINTERS IN NEXT
154 113 89              .ADDR   KEY3            ;ROM LOCATIONS
155 114 8B              .ADDR   KEY4
156     0121            .   = X'121             ;MOVE TO KEY5 POINTER LOCATION
157 121 8D              .ADDR   KEY5
158 122 8F              .ADDR   KEY6
159 123 91              .ADDR   KEY7
160 124 93              .ADDR   KEY8
161     0141            .   = X'141             ;MOVE TO KEY9 POINTER LOCATION
162 141 95              .ADDR   KEY9            ;(PAGE 5)
163 142 97              .ADDR   KEY10
164 143 99              .ADDR   KEY11
165 144 9B              .ADDR   KEY12
166     0181            .   = X'181             ;MOVE TO KEY13 POINTER LOCATION
167 181 9D              .ADDR   KEY13           ;(PAGE 6)
168 182 9F              .ADDR   KEY14
169 183 A1              .ADDR   KEY15
170 184 A3              .ADDR   KEY16
171 185 61C0    KEY1:   JMP     ONE             ;G0,D1 KEY
172 187 61C2    KEY2:   JMP     TWO             ;G0,D2 KEY
173 189 61C4    KEY3:   JMP     THREE           ;G0,D3 KEY
174 18B 61C6    KEY4:   JMP     FOUR            ;G0,D4 KEY
175 18D 61C8    KEY5:   JMP     FIVE            ;G1,D1 KEY
176 18F 61CA    KEY6:   JMP     SIX             ;G1,D2 KEY
177 191 61CC    KEY7:   JMP     SEVEN           ;G1,D3 KEY
178 193 61CE    KEY8:   JMP     EIGHT           ;G1,D4 KEY
179 195 61D0    KEY9:   JMP     NINE            ;G2,D1 KEY
180 197 61D1    KEY10:  JMP     TEN             ;G2,D2 KEY
181 199 61D2    KEY11:  JMP     ELEVEN          ;G2,D3 KEY
182 19B 61D3    KEY12:  JMP     TWELVE          ;G2,D4 KEY
183 19D 61D4    KEY13:  JMP     THIRTN          ;G3,D1 KEY
184 19F 61D5    KEY14:  JMP     FOURTN          ;G3,D2 KEY
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

```
185 1A1 6001   KEY15:  JMP     START   ;G3,D3 KEY
186 1A3 6001   KEY16:  JMP     START   ;G3,D4 KEY
187 1A5 231F   NRDY:   LDD     DIGIT   ;DIGIT POSITION TO A
188 1A7 5E             AISC    14      ;LAST DIGIT DONE (A = 1)?
189 1A8 6050           JMP     DEBOUN  ;YES, JUMP TO DEBOUNCE ROUTINE (A=15)
190 1AA 51             AISC    1       ;NO, DECREMENT DIGIT POSITION VALUE
191 1AB 0F             LBI     0,0     ;POINT TO DISPLAY REGISTER 0
192 1AC 50             CAB             ;DIGIT POSITION VALUE TO BD
193 1AD 00             CLRA
194 1AE 54             AISC    4       ;DELAY 9 INSTR. TIMES
195 1AF EE             JP      .-1     ;REPEAT PREVIOUS INSTR. UNTIL SKIP
196 1B0 600C           JMP     DSP2    ;DISPLAY NEXT DIGIT
197 1B2 3D     NODP:   LBI     STORE   ;POINT TO M(2,15)
198 1B3 332C           CQMA            ;SE-SG,D.P. TO A
199 1B5 06             X               ;EXCHANGE INTO M(2,15)
200 1B6 4C             RMB     0       ;RESETD.P. BIT (DECIMAL POINT OFF)
201 1B7 333C           CAMQ            ;SEGMENT DATA BACK TO Q
202 1B9 601C           JMP     DIGOUT
203


204                    .FORM

205     01C0           .PAGE   7
206 1C0 3381   ONE:    LBI     0,1
207 1C2 3382   TWO:    LBI     0,2
208 1C4 3383   THREE:  LBI     0,3
209 1C6 3384   FOUR:   LBI     0,4
210 1C8 3385   FIVE:   LBI     0,5
211 1CA 3386   SIX:    LBI     0,6
212 1CC 3387   SEVEN:  LBI     0,7
213 1CE 3388   EIGHT:  LBI     0,8
214 1D0 08     NINE:   LBI     0,9
215 1D1 09     TEN:    LBI     0,10
216 1D2 0A     ELEVEN: LBI     0,11
217 1D3 0B     TWELVE: LBI     0,12
218 1D4 0C     THIRTN: LBI     0,13
219 1D5 0D     FOURTN: LBI     0,14
220 1D6 4E             CBA
221 1D7 2399           XAD     1,9     ;SAVE KEY NUMBER
222 1D9 0F             LBI     0,0
223 1DA 6889           JSR     BLANK   ;BLANK DISPLAY REGISTER
224 1DC 0F             LBI     0,0
225 1DD 2319           LDD     1,9     ;KEY NUMBER TO A
226 1DF 50             CAB
227 1E0 06             X               ;KEY NUMBER TO DISPLAY REGISTER
228 1E1 6007           JMP     DSPLY
229                    END
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

```
ALLUP   0059    BLANK   0089    CLEAR   0081    CLRAM   0080
DEBOUN  0050    DECKBC  0054    DIGIT   001F    DIGOUT  001C
DSP1    000B    DSP2    000C    DSPLY   0007    EIGHT   01CE
ELEVEN  01D2    FIVE    01C8    FOUR    01C6    FOURTN  01D5
KBC     003F    KBCTST  006C    KEY1    0185    KEY10   0197
KEY11   0199    KEY12   019B    KEY13   019D    KEY14   019F
KEY15   01A1    KEY16   01A3    KEY2    0187    KEY3    0189
KEY4    018B    KEY5    018D    KEY6    018F    KEY7    0191
KEY8    0193    KEY9    0195    KEYDEC  0100    KEYDWN  005E
LDRAM   0004    NINE    01D0    NODP    01B2    NRDY    01A5
ONE     01C0    SEVEN   01CC    SIX     01CA    START   0001
STORE   003E    STR     0055    TEN     01D1    THIRTN  01D4
THREE   01C4    TWELVE  01D3    TWO     01C2

NO ERROR LINES

  227  ROM WORDS USED

SOURCE CHECKSUM = D7F0


INPUT FILE     13:DSPLY.SRC

LISTING FILE   13:DSPLY.LST
```

**Figure 8.3. DSPLY.SRC Assembly Output Listing (continued)**

# 9 COP Monitor and Debugger (COPMON)

The COP Monitor and Debugger (COPMON) is a PDS system program which contains an extensive set of debugging commands. These allow the user to easily check out and develop COP programs and systems using the COP Emulator Card and shared memory. (See Chapter 2.)

## 9.1 COPMON Capabilities

COPMON allows the user to interrupt the COP's execution of a program on one of several conditions and examine all COP internal register data at each interruption. This is called a *breakpoint*. COPMON also allows the user to examine the program path of the COP, recording up to 253 steps (adresses) before or after a specified condition. This is called a *trace*. Conditions for a breakpoint or a trace may be a specific address (address), the next value of program counter (immediate), or any combination of two external inputs on the emulation card called EXT2 and EXT1 (EXT EVENT). For a complete description of the emulator card see Section 2.3.

The TRACE command allows the user to specify which condition the TRACE will begin on and how many program steps previous to that condition to record. The GO command then executes the TRACE operation. After a TRACE, the user may examine trace data with the TYPE command or search for an address in trace memory with the SEARCH command. These commands allow the user to compare the actual step sequence with the expected step sequence, and to easily spot all occurrences of any given address.

The BREAKPOINT command allows the user to specify up to ten conditions on which to interrupt the COP's execution. Only one condition is tested for at a time (see BREAKPOINT Command). When the COP is breakpointed, the user may examine all the COP registers and RAM, as well as the values of the input lines. This information is also available after each single-step. Thus, the user may compare the actual breakpoint information to the expected values at any step or occurrence of any condition in the program. Such information allows the user to easily identify errors and check logic. For further information on the system software and hardware associated with COPMON, see Section 2.3.

A typical debugging session may go as follows:

1. The faulty program is loaded into PDS shared memory either from the disk or from PROMs. The user may wish to examine the contents of shared memory at this time.
2. A TRACE operation is performed and trace memory is examined to determine where the actual execution sequence deviated from the desired execution sequence.
3. Breakpoints are performed around the address in question and breakpoint data is examined to discover the cause of the error. The user may wish to single-step through the problem area, or set breakpoints on specific conditions.
4. If a possible error is discovered, shared memory may be altered to correct the error or to aid in identifying the error. Steps 2-4 of this example may be repeated until the program executes properly.
5. New PROMs may be programmed with the revised program in shared memory. For a concise list of the alterations, the COMPARE command may be used to compare the old PROMs or disk file to the new program in shared memory.

COPMON allows the user to reset the chip or restart the chip at any address, allowing the breakpoint and trace operations to be used easily and effectively. Breakpoint and trace may not operate simultaneously. The COP may run from shared memory or the PROMs on the emulator card. When running from the PROMs, shared memory must have the same data as the PROMs in order for breakpoint to operate correctly. COPMON allows the user to select either of these run modes. The STATUS command allows easy inspection of COPMON's basic operation parameters. COPMON can be loaded and executed from the console or the front panel. The first part of this chapter will describe operation from the console; the second part will describe operation from the front panel.

## 9.2 Console Operation

To call COPMON from the console, the user types in the @ command, obtaining the following responses:

```
X>@COPMON

COPMON,REV:C DATE
CHIP NUMBER (DEFAULT = 420)? XXX
```

The user then enters a three-digit number ("XXX" indicated above) which represents one of the valid chip numbers listed in Table 9.1.

**Table 9.1. Valid Chip Numbers**

| Chip # | Memory Size | Register Address | Digit Address | Block # Range |
|---|---|---|---|---|
| 410/411 | 0-X'1FF | X'0-X'3 | X'0, X'9-X'15 | 0 |
| 420/421 | 0-X'3FF | X'0-X'3 | X'0-X'15 | 0,1 |
| 440/444/445 | 0-X'7FF | X'0-X'7 | X'0-X'15 | 0-3 |

The chip number is used by COPMON to select the correct instruction subset, memory size and register size. If no number is typed after the chip number prompt, COPMON defaults to the COP420 after the carriage return. Otherwise, the *valid* chip number typed by the user is used. Each *subsequent* time COPMON is loaded, it will use the last valid chip number specified by the user or, if no number was previously specified, the default COP420 number. The chip number may also be changed with the CHIP command described later. After the user responds to the initial chip number prompt, COPMON responds with the COPMON prompt symbol, "C>."

Example:

```
CHIP NUMBER (DEFAULT = 420)? 444
C>
```

COPMON responds with the prompt after the execution of each command. Prompt mode is also indicated on the front panel by the presence of the blinking rightmost decimal point light. From the prompt mode the user may enter commands from either the console or the front panel.

## 9.3 COPMON Console Commands

The COPMON console commands are summarized in Table 9.2 and are described in detail below. Commands may be abbreviated to one or two characters as indicated by the underlined characters in Table 9.2. Command options are defined in Table 9.3.

**Table 9.2. COPMON Console Command Summary**

| Command Name | Operand Syntax | Description | Page |
|---|---|---|---|
| ALTER | [<ADDR>][,[<VALUE>] . . .] | Alter Shared Memory | 9-3 |
| AUTOPRINT | [<PRINT OPT>[<PRINT OPT> . . .]] | Set Print Options | 9-4 |
| BREAKPOINT | [<COND>[/<COND> . . .]] [,<OCCUR #>[,<GOPT>]] | Set Breakpoint | 9-4 |
| CLEAR | | Clear Trace and Breakpoint Enable Flags | 9-4 |
| CHIP | <CHIP #> | Set or Display Chip Number | 9-4 |
| COMPARE | [<FILENAME>,][<BLOCK #>,<PROM TYPE> | Compare Data | 9-4 |
| DUMP | <PROM TYPE>,<BLOCK #> | Dump PROM | 9-5 |
| DEPOSIT | <VALUE>,<ADDR RANGE> | Deposit Value in Shared Memory | 9-5 |
| FIND | <VALUE>[,<ADDR RANGE>[,<MASK>]] | Find Value in Shared Memory | 9-5 |
| GO | [<ADDR>] | Begin Execution | 9-5 |
| LIST | [<ADDR RANGE>],<ADDR RANGE> . . .]] | List Shared Memory | 9-6 |
| LOAD | <FILENAME> [,<0>] | Load Shared Memory from File | 9-6 |
| MODIFY | <print opt>,<value>[,<value1> . . . ] | Modify Chip Registers | 9-6 |
| NEXT | [<GOPT>] | Breakpoint on Next Instruction | 9.6 |
| PROGRAM | [<FILENAME>,][<BLOCK #>,<PROM TYPE> | Program PROM | 9-6 |
| PUT | [<ADDR>][,<INSTRUCT>[,<INSTRUCT> . . .]] | Put Instruction (Assemble) | 9-7 |
| RESET | | Reset Chip | 9-7 |
| SINGLESTEP | [<GOPT>] | Single Step | 9-7 |
| SEARCH | <ADDR> | Search for Address in Trace Memory | 9-7 |
| SHARED MEM | <YN> | Set/Clear Shared Memory Mode | 9-7 |
| STATUS | | Display COP Status | 9-7 |
| TIME | [<cond1>[,<occur1>][/<cond2>[,<occur2>]]] | Measure Elapsed Time | 9-7 |
| TYPE | [<PRINT OPT>],<PRINT OPT> . . .]] | Type Breakpoint or Trace Data | 9-8 |
| TRACE | [<COND>[,<OCCUR #>],<PRIOR>],<GOPT>]]]] | Set Trace Conditions | 9-8 |
| UNASSEMBLY | <Y/N> | Disassemble Shared Memory | 9-9 |

**Table 9.3. COPMON Console Command Option Summary**

| | | | |
|---|---|---|---|
| <ADDR> | = One to three hex digits, less than or equal to the maximum address of the chip defined by <CHIP #>. (See Table 9.1.) | <PRINT OPT> | = A — Accumulator (BR) |
| | = 'P' — Previous. | | = ALL — All breakpoint data (BR) |
| | = '.' — Current address, i.e. last address altered or typed. | | = B — RAM addr reg B (BR) |
| | = 'N' — Next Address. | | = C — Carry bit (BR) |
| | = "L" — Last address defined by <CHIP #> | | = G — G I/O port (BR) |
| <ADDR RANGE> | = <ADDR>[/<ADDR> | | = I — I input port (BR) |
| <BLOCK #> | = Decimal number, defines consecutive 512-byte blocks of shared memory or load module. Valid range determined by shared memory range for COP device. (See Table 9.1.) | | = L — L I/O port (BR) |
| | | | = M — All RAM on chip (BR) |
| <CHIP #> | = 410 OR 411 OR | | = M <REG #> — RAM registers <REG #> (BR) |
| | = 420 OR 421 OR | | = M <REG #><DIG #> — RAM digit <REG #>, <DIG#> (BR) |
| | = 440 OR 444 OR 445 | | = P — Program counter (BR) |
| <DIG #> | = Hex digit in range X'F down to the minimum digit address of the chip defined by <CHIP #>. (See Table 9.1.) | | = Q — Q latches (BR) |
| | | | = S — Serial I/O register (BR) |
| <END> | = Decimal number in range 0-253, last location of trace memory desired. | | = T — Trace Memory 0-253 (TR) |
| | | | = <START> — Trace Memory <START> (TR) |
| <EVT COND> | = EV00 | | = <START>/<END> — Trace memory <START> through <END> (TR) |
| | = EV01 | | |
| | = EV10 FORMAT: EV<EXT2><EXT1> | <PRIOR> | = Decimal number in range 0-253, number of addresses traced prior to address on which <COND> occurred. |
| | = EV11 '1' = Logic 1 | | |
| | = EVX0 '0' = Logic 0 | <PROM TYPE> | = E — MM5204 Erasable PROM. |
| | = EVX1 'X' = Don't care | | = B — 87S296 Bipolar PROM. |
| | = EV0X | | |
| | = EV1X | <REG #> | = Hex digit in range 0 to the maximum register address of the chip defined by <CHIP #>. (See Table 9.1.) |
| <FILENAME> | = Valid PDS filename. | | |
| <GOPT> | = G (GO immediately after printing). | <START> | = Decimal number in range 0-253, first location in trace memory desired. |
| <INSTRUCT> | = Valid COP400 instruction mnemonic with operand. | | |
| | | <COND> | = <ADDR> |
| <MASK> | = Hex number in range 0-X'FF | | = <EVT COND> |
| <OCCUR #> | = Decimal number in range 1-256, number of times <COND> occurs before initiating trace of breakpoint. | | = I (immediate trace or breakpoint). |
| | | <VALUE> | = Hex number in range 0-X'FF. |
| | | <VOLUME NAME> | = Valid volume name. |

### ALTER SHARED MEMORY Command

Syntax: ALTER [<ADDR>][<VALUE> . . .]

Description: Alter the contents of consecutive shared memory locations to the specified hexadecimal values beginning at the specified address. Consecutive commas will increment the current address pointer, leaving the data at these locations unaltered. If no address is specified, the command begins at the last altered or listed location. (See LIST command.) If two or more values separated by spaces are given for <value>, the last of these values will be the one stored. The alterable range of shared memory is determined by the chip number. (See Table 9.1.) *The COP chip is automatically reset.*

Example: C>A 1CF,D0,,D1

Place D0 in location 1CF, leave 1D0 unchanged, and place D1 in location 1D1.

### AUTOPRINT Command

Syntax: AUTO [<PRINT OPT>[,<PRINT OPT> . . .]]

Description: Set or clear Autoprint options.

Each autoprint option included causes its corresponding value to be printed during the breakpoint, trace, and single step operations. Table 9.3 is a list of the print options with their description and the operations to which they apply — (BR) for Breakpoints and Single-Steps; (TR) for Traces. If no options are specified, all are cleared. A "*PR" at the end of the line will cause the autoprint output to go to the printer. The 16-digit contents of any specified RAM register will be printed, *left to right*, most significant digit to least significant digit.

Example: C>AU A,P

Causes the contents of the accumulator and the program counter to be printed after each breakpoint and single-step operation.

### BREAKPOINT Command

Syntax: BREAKPOINT [<COND>[/<COND>[/<COND> . . .]]] [,<OCCUR #>[,<GOPT>]]

Description: The breakpoint command sets and prints the conditions which determine breakpoint operation. A maximum of ten conditions are allowed. If one or more conditions are specified, all previous conditions are cleared. If no conditions are specified, previous conditions are retained. The breakpoint command puts the specified condition(s) in a circular list. Only the condition at the top of the list is looked for during breakpoint operation, but the list is rotated one position each time a breakpoint operation occurs. If <OCCUR #> is not specified, it defaults to the last specified value. If <GOPT> is specified, the breakpoint operation will occur repeatedly on successive conditions in the circular list. This will continue until a break is received from the console or panel. Data specified by the AUTOPRINT command will be printed, thus providing an automatic map of relevant data during COP execution.

The breakpoint command sets the breakpoint enable flag, but does not initiate breakpoint. If the breakpoint enable flag is set, breakpoint will be initiated by the next GO command. For proper breakpoint operation while running from PROMs, shared memory must have the same data as the PROMs.

Example:
```
C>BR 2/35/I/EVX1/26, 4, G
BRKPT ENABLED
A:2 A:35 IMMD EVX1 A:20 OCCUR:4 GO:Y
```

Breakflag is enabled, the next GO will cause successive breakpoints on the fourth occurrence of each of the five conditions, circling through the list until interrupted.

### CLEAR Command

Syntax: CLEAR

Description: The command clears the breakpoint enable and trace enable flags, but leaves their conditions unaltered. (See BREAKPOINT, TRACE and GO commands.) It has no operands.

Example:
```
C>C
BRKPT AND TRACE DISABLED
```

### CHIP Command

Syntax: CHIP <chip #>

Description: This command allows the user to change and display the chip number. The chip number determines memory and register size. (See Table 9.1.) If no option is specified, the current chip number is displayed.

Example:
```
C>CH 440
CHIP NUMBER 440
```

### COMPARE Command

Syntax:
```
COmpare <Filename>
COmpare <Filename>, <Block #>,
  <PROM type>
COmpare <Block #>, <PROM type>
```

Description: Three types of syntax are shown. The first compares a file to shared memory. The second compares the specified block of the specified file to the specified PROM. The third compares the specified block of shared memory to the specified PROM. These are the only valid syntax for the COMPARE command. Each pair of values that does not compare is displayed with the address of shared memory where that data would be located, and with identifiers indicating which

device the data is from. Non-compares are listed until the entire range has been compared or a break is received from the console.

Example:
```
C>CO 3, E
6E1    S:BA    P:6A
77F    S:B6    P:B7
COMPARE DONE
```

### DUMP Command

Syntax: DUMP <PROM type>, <Block #>

Description: This command dumps the contents of the specified PROM into the specified block in shared memory. The checksum is displayed when the dump is completed.

Example:
```
C>D E, 3
CKSM = 3AFC
```

### DEPOSIT Command

Syntax: DEPOSIT <VALUE>, <ADDR RANGE>

Description: This command puts the specified value into each location of the specified address range.

Example: C>DE F6, 11/1E

F6 is put in locations 11 through 1E.

### FIND Command

Syntax: FIND <VALUE> [,<ADDR RANGE> [,<MASK>]]

Description: This command searches for the specified value in the specified address range of shared memory. If the mask option is present, each shared memory word and <value> will be "ANDed" with the value of <mask> before it is tested. This allows the user to find parts of bytes in shared memory. If <mask> is not specified, it defaults to X'FF. Each occurrence of <VALUE> is printed on the console until the search is done or it is interrupted from the console.

Example:
```
C>F 8E,200/3FF
2CC   8E   2B0   8E   3FF   8E
FIND DONE
```

### GO Command

Syntax: GO [<ADDR>]

Description: The function of this command depends on the status of the COP chip and the breakpoint and trace enable flag. (See BREAKPOINT and TRACE commands.) The function of the GO command for each combination of relevant parameters is described in Table 9.4. Generally a breakpoint will be initiated if the breakpoint enable is set, a trace will be done if the trace enable flag is set, and the chip will be started running in a normal manner if neither flag is set. Breakpoint and trace flags remain unchanged after the GO command. For example, if the breakpoint flag is enabled, the first condition in the list is EVOX, the autoprint options are B, P, and <GOPT>is not set, the following sequence will occur:

Example:
```
C>GO
BRKPTD ON EVOX AT A:XXX
B:01  P:XXX
```

Here XXX stands for the address at which EVOX occurred. A similar message would appear if TRACE were enabled instead of Breakpoint.

**Table 9.4. GO Operation Summary**

The function of the "GO" command depends on the mode that the COP chip is in, whether or not BRKPT or TRACE is enabled, and whether or not <ADDR> is given:

| Addr Given | BRKPT or TRACE Enabled | COP Chip Status | Function Performed |
|---|---|---|---|
| No | No | Reset | Start chip at addr 000. |
| | | Breakpointed | Start chip at BRK addr. |
| | | Running | "COP ALREADY RUNNING' |
| | BRKPT | Reset | Start chip at addr 000, enter breakpoint mode. |
| | | Breakpointed | Start chip at BRK addr, enter breakpoint mode. |
| | | Running | Enter breakpoint mode. |
| | TRACE | Reset | Start chip at addr 000, enter trace mode. |
| | | Breakpointed | 'CANT TRACE WHEN BRKPTED' |
| | | Running | Enter trace mode. |
| Yes | No | Reset | Start chip at <ADDR>. |
| | | Breakpointed | Start chip at <ADDR>. |
| | | Running | 'COP ALREADY RUNNING'. |
| | BRKPT | Reset | Start chip at <ADDR>, enter breakpoint mode. |
| | | Breakpointed | Start chip at <ADDR>, enter breakpoint mode. |
| | | Running | Breakpoint immediate, start chip at <ADDR>, enter breakpoint mode. |
| | TRACE | Reset | 'CANT GO TO ADR AND TRACE'. |
| | | Breakpointed | 'CANT GO TO ADR AND TRACE'. |
| | | Running | 'CANT GO TO ADR AND TRACE'. |

### LIST Command

Syntax: LIST [<ADDR Range> [,<ADDR Range> . . .]]

Description: This command lists the contents of the specified address ranges starting from the next lower multiple of X'10. If <ADDR range> is just one address, just that location is listed. If no address range is specified, 256 locations are listed starting from the multiple of X'10 below the current address. The current address is the last address printed or altered. Subsequent LIST commands with no operands will list the next 256 locations. *This command automatically resets the COP chip.*

Example:
```
C>L 4/8
000 3A C2 00 F2 03 29 76 AA D0
```

### LOAD Command

Syntax: LOAD <FILENAME> [,<0>]

Description: This command loads the specified load module (.LM) file into shared memory. If the 0 is specified the memory is *not* zeroed before loading.

Example:
```
C>LO DEMO
FINISHED LOADING
```

### MODIFY Command

Syntax: MODIFY <print opt>,<value>[,<value 1> . . . ]

Description: This is the command structure for the MODIFY registers and COP RAM routine. This routine should be helpful in debugging hardware prior to debugging software on that hardware. This function allows the user to directly alter internal registers, RAM, and I/O ports on the COPS chip while breakpointed.

*Only one (1) register may be modified in a single command line.*

Examples:
```
C>BR 1
BRKPT ENABLED
A:001 OCCUR  1 GO:N
C>R
CHIP IS RESET
C>GO
BREAKPOINTED ON A:001 AT A:001
C>M M0,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
```

This command sets memory register 0 digit 0 to 0, memory register 0 digit 1 to 1, etc.

```
C>M15,5,6,7,8 . . .
```

This command sets memory register 1 digit 5 to 5, etc.

```
C>M E, 4
```

This command loads the E register with 4 (enable Q register to L bus).

```
C>M Q, AA
```

This command in conjunction with the previous command loads the Q register with AA and thus puts AA on the L bus.

### NEXT Command

Syntax: Next [<GOPT>]

Description: This command is identical to Single Step except at a JSR/JSRP instruction where it will set the Breakpoint to the address of the following instruction and stop there after executing the subroutine in real time.

### PROGRAM Command

Syntax: PROGRAM [<FILENAME>,] <BLOCK #>, <PROM TYPE>

Description: This command allows the user to program the PROM specified by <PROM TYPE> from the specified block of the specified file. If <FILENAME> is omitted, the PROM will be programmed from shared memory. To program a Bipolar 87S296 PROM a special pin scrambler must be inserted (see Section 2.1 and Figure 2.3). The command prompts the user to insert the correct PROM. To begin programming the user inserts the correct PROM (and disc if programming from a file), then presses a carriage return in response to the insert prompt. PDS messages to the console indicate the state of the programming procedure.

Example:
```
C>P DEMO, 1, E
INSERT EPROM, PROGRAM (Y/N,CR = YES)?(CR)

PROGRAMMING VERIFYING
CKSM = XXXX
```

### PUT Command

Syntax: PUT [<ADDR>][,<INSTRUCT>[,<INSTRUCT>...]]

Description: Replace the contents of shared memory starting at the address specified with the opcodes of the specified instructions. If no address is given, placement begins at the current address. *This command automatically resets the COP chip.* Instruction opcodes may be directly specified in the operand field. Instructions with double operands may only be specified in hex format and, unlike the Assembler format, double operands may not be separated by commas (e.g., LBI 23 is OK; LBI 2,3 is not allowed).

Example:
```
C>PU 130, CLRA, AISC 15 LBI 39
C>
```

### RESET Command

Syntax: RESET

Description: This command resets the COP chip and sets the reset flag, which determines operation of the GO command. (See Table 9.4.)

Example:
```
C>R
CHIP IS RESET
```

### SINGLE STEP Command

Syntax: SINGLESTEP [<GOPT>]

Description: This command performs a breakpoint on the next instruction. If the COP is reset, it breakpoints at address 1. If it is running, it is breakpointed on the next instruction. If it has already breakpointed, it steps one instruction. After each single-step, information specified in the AUTOPRINT command is printed. If <GOPT> is included, it will automatically step and print data until interrupted by the console.

Example:
```
C>S G
Step
A : 0 P : 10
Step
A : 1 P : 11
.
.
.
```

### CARRIAGE RETURN SINGLE STEP

Syntax: (CR)

Description: If the COP is breakpointed a carriage return is identical to singlestep without <GOPT>.

Example:
```
C>(CR)
STEP
P : 81 B : 6
```

### SEARCH Command

Syntax: SEARCH <ADDR>

Description: This command searches Trace Memory for the specified address. Each occurrence is displayed and it searches until finished or interrupted by the console.

Example:
```
C>SE 2FE
 0 0 A : 2FE SKIP E : 1111
 8 8 A : 2FE      E : 1101
SEARCH DONE
```

Each line of output from the SEARCH command and the TYPE (trace memory) command contains the following information, from left to right:

1. Trace Memory Location
2. Location relative to TRACE condition location
3. Program Counter
4. Skip indication
5. Value of external event inputs E4-E1, left to right

In the above example, the Trace Memory locations are the same as their location relative to the Trace Condition location, since no prior number was specified. Therefore, the Trace Condition location is the first Trace Memory location.

### SHARED MEMORY Command

Syntax: SHARED MEMORY [Y/N]

Description: This command allows the user to specify whether the COP chip runs from shared memory or PROM. If "Y" is typed, the COP will run from shared memory. If "N" is typed, the COP will run from PROM. *The COP chip is automatically reset.*

Example:
```
C>SH Y
SHARED MEMORY MODE
C>SH N
PROM MODE
C>
```

### STATUS Command

Syntax: STATUS

Description: Prints status of COP chip and various other internal conditions.

Example:
```
C>ST
CHIP NUMBER 420
CHIP IS RESET
BRKPT AND TRACE DATA NOT VALID
SHARED MEMORY MODE
BRKPT CONDITIONS:
A : 005  OCCUR:  1  GO : N
TRACE CONDITIONS:
EVX1 OCCUR:  1 PRIOR : 0  GO : N
```

### TIME Command

This command is similar to the TRACE command. It allows the measurement of elapsed time between addresses and/or external events.

Syntax: TIME
[<cond1>(,<occur1>][/<cond2>[,<occur2>]]]

Description: The TIME command sets and prints the conditions which control the time measurement. The timer is started when the first set of conditions is met and the timer is stopped when the second set of conditions is met. The second set of conditions is invoked only after the first set of conditions is satisfied, and it is only looked for

from that time. If those conditions have been encountered prior to the first set of conditions having been met they are ignored.

As in the TRACE command the TIME command is not initiated until a GO command is issued. The TIME, TRACE, and BREAKPOINT commands are mutually exclusive.

The limits on the TIME command are: The time between the events must be more than 500 microseconds and less than 7 minutes. If the time is less than 500 microseconds the conditions may not be recognized or if they are recognized the time reported will be in error. If the time is longer than 7 minutes a timer overflow message will be reported. The resolution of the TIME command is ±100 microseconds.

If only COND1 is specified, COND2 is set to COND1 and the occurrences are both set to the last value of OCCUR1. If only COND1 and OCCUR1 are specified, COND2 is set the same as COND1 and OCCUR2 is set the same as OCCUR1. If COND1 and COND2 are specified, OCCUR1 and OCCUR2 are left at their previous values.

Examples:

```
C>TI EVX1, 2/234, 3

TIME ENABLED:
EVX1 OCCUR:  2 TO  A:234 OCCUR:  3
```

This command will measure the time from the second positive transition on EXTERNAL EVENT 1 (high on 1, don't care on 2) to the third occurrence of address 234 after the EXTERNAL EVENT condition has been met.

```
C>TI 350, 1/24, 2

A: 350 OCCUR:  1 TO A:024 OCCUR:  2
```

This command will measure the elapsed time from the first occurrence of address 350 to the second occurrence of address 24 after the occurrence of address 350.

```
C>TI 44

A:044 OCCUR:  1 TO A:044 OCCUR:  1
C>GO

TIME ON  A:044 TO  A:044  16.8 MS
```

This example shows the default conditions of the command. Used with the previous example this command will measure the elapsed time between the first occurrence of address 44 and the next occurrence of address 44.

Notes: None of the time command examples are invoked unless they are followed by a GO command.

Time is reported in milliseconds.

### TYPE Command

Syntax: TYPE [<PRINT OPT> [,<PRINT OPT> . . .]]

Description: This command types out the information specified to the printer or console. As with the AUTOPRINT command, if a RAM register is specified, its 16-digit contents will be listed, left to right, most significant digit to least significant digit. If no options are specified, trace memory will be displayed in blocks of 16.

Example:

```
C>T P, Q, B, M14, M2
B : 10  Q : FF  P : 004  M14 : 0
 M2:00000000120F120E
```

### TRACE Command

Syntax: TRACE [<COND>[,<OCCUR#>[,<PRIOR> [,<GOPT>]]]]

Description: This command allows the user to set the print trace conditions. During a Trace operation COPMON stores each consecutive value of the COP program counter in a 254-byte circular buffer, so that at any time during trace operation, the buffer has the previous 254 values of program counter. When <COND> has been met the number of times specified by <OCCUR#>, COPMON saves the number of values of the program counter prior to <COND> specified by <PRIOR>, and fills the rest of the buffer with the subsequent values of the program counter. It then prints the <COND> specified and the address where <COND> was recognized, followed by any Trace data specified by the AUTOPRINT command. If <COND>, <OCCUR#> or <PRIOR> are omitted, they retain their previous values. If <GOPT> is included, then each time a trace operation is finished, another GO command is performed with the same conditions, continuing until interrupted by the panel or console. The TRACE command does not initiate trace operation, but sets the trace enable flag so that trace

operation is initiated on the next GO command. (See Table 9.4.)

Example:

```
C>TR EVOX, 2, 22
TRACE ENABLED:
EVOX OCCUR: 2 PRIOR: 22 G: N
```

### UNASSEMBLE Command

Syntax: UNASSEMBLE <Y/N>

Description: The UNASSEMBLE mode will give an opcode and mnemonic for each instruction. The UNASSEMBLE command is of the same type as the AUTOPRINT command and when invoked will work with the LIST, TRACE, BREAKPOINT, and SINGLE STEP commands.

*When the UNASSEMBLE mode is selected the chip will be reset on TYPE, contrary to the way trace data is handled when running normally.*

IF A LIST IS STARTED ON THE SECOND BYTE OF A TWO-BYTE INSTRUCTION THE UNASSEMBLY WILL BE INCORRECT UNTIL TWO SUCCESSIVE ONE-BYTE INSTRUCTIONS ARE ENCOUNTERED.

The STATUS command will give the status of the UNASSEMBLE command along with the rest of the status of the chip.

Example:

```
C>UN Y

DO UNASSEMBLY
```

## 9.4 Front Panel Conventions

**Display:** The blinking rightmost decimal point light indicates prompt mode. While in prompt mode, new commands may be entered from the panel or console. A steady decimal point light over the word "run" indicates that the COP is running.

**Command Format:** The panel command format is similar to the console command format in that the command is entered first, followed by the operand. Also, the nature of each command and its operands generally parallels its counterpart on the console. To minimize keystrokes, commas are not always used to separate options but are used when necesasary to delimit numbers. Each specific command automatically determines the number of digits displayed and used when entering numeric operands. Digits entered are shifted into the rightmost digit and the previous number displayed is shifted left. If invalid numbers for a specific operand are shifted through the display, the words "err" or "er" may be displayed either in place of the invalid number or elsewhere on the display. The user may continue to press new digits, and, when the resulting number is valid, the error message will disappear from the display. Digits may be shifted indefinitely, until a legal or illegal delimiter is pressed. At this point the number displayed is used by the command.

Upper case keys are selected with the shift key. Unlike the console, the shift key is not held down while the other key is pressed. Instead, the shift key is pushed *first*, and the following key has the upper case value.

Similarly, for special function keys, "SFUNC" is pressed first, and the following key has the special function value. Whenever the "SFUNC" key is pressed, "SFNC" will be displayed in the left or right half of the display.

**Command Illustration:** The following is a sample of the format which will be used to illustrate each command:

### BREAKPOINT Command

Syntax: BKPT [<cond>[(+/-)<cond> . . .]][,[<occur>]]

Description: Here the function and details of the command are described.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| | .* |
| SHIFT | |
| BKPT | BP 0. 001 |
| 5 | BP 0. 005 |
| 4 | BP 0. 054 |
| INCR | BP 1. NONE |
| SFUNC | BP 1. SFNC |
| E | BP 1. E |
| 2 | BP 1. E 02 |
| 1 | BP 1. E 21 |
| , | OCCU. R001 |
| 5 | OCCU. R005 |
| TERM | .* |

The syntax line contains the command key label followed by the options. In the syntax line, upper case labels represent the sequence "shift" and then that key; special function key sequences are labeled "SFNCX" where "X" is the label of the second key. Options may include key labels and the general options defined in Table 9.6. In option punctuation, a comma represents the comma key; "+" and "–" denote the "INCR" and "DECR" keys respectively, and a period (".") represents the "TERM" key.

Each line under "Example:" represents the key pressed and the resulting display. Each key pressed is shown, including "SHIFT" and "SFUNC." The asterisk in the far right place represents the flashing prompt light. The period in the middle (right side of left half) represents the run light, which stays on while the chip is running.

Table 9.5. COPMON Panel Command Table

| Command Name | Syntax | Description | Page |
|---|---|---|---|
| BREAKPOINT | BKPT [<cond>[+/−]<cond> . . .]] [, [<occur>]] | Read, and set breakpoint conditions and flags. | 9-10 |
| | | The following commands display COP data when the COP is breakpointed: | |
| | | Displays — Syntax: Accumulator — ACC; B register — BREG; Carry & skip flags — CARRY; G I/O ports — GIN; I input ports — IIN; L input ports — LIN; Program counter — PC; Q register — QREG; RAM register — REG <reg #>; S register — SREG | 9-11 |
| CHIP | C [<chip #> | Display/altering number. | 9-11 |
| CLEAR | CLR | Clear breakpoint and trace enable flags. | 9-11 |
| COMPARE | SFNC1 SMEM <block #><PROM type> [+] | Compare shared memory to PROM. | 9-11 |
| | SFNC1 SFNC <file #>, <block #> <PROM type> [+] | Compare file to PROM. | |
| | SFNC1 SFNCD <file #> SMEM [+] | Compare file to shared memory. | |
| CONSOLE | SFNCC | Console baud rate setup. | 9-12 |
| GO | GO [<addr>] | Begin execution of breakpoint or trace. | 9-12 |
| LOAD | SFNC0 <PROM type> SMEM <block #> | Load shared memory from PROM. | 9-12 |
| | SFNC0 SMEM <block #> <PROM type> | Program PROM from shared memory. | |
| | SFNC0 SFNCD <file #>, <block #> <PROM type> | Program PROM from file. | |
| | SFNC0 SFNCD <file #> SMEM | Load shared memory form file. | |
| MODE | SFNC4 [<mode>] | Set/display mode. | 9-13 |
| RESET | SFNCCLR | Reset chip. | 9-13 |
| SEARCH | SFNC2 SMEM <value> [+] | Search for value in shared memory. | 9-13 |
| | SFNC2 TRACE <addr> [+] | Search for address in trace memory. | |
| SHARED MEMORY | SMEM [<addr>][,<value>[,<value> . . .]] | Display/alter shared memory. | 9-13 |
| SINGLE STEP | SS | Single step. | 9-14 |
| TRACE | TRACE [<cond>][<occur>[[<prior>]]] | Set, and display trace conditions and flag. | 9-14 |
| TRACE MEMORY | SFNC3 [<trace #> | Display trace data. | 9-14 |
| ABORT | ABORT [?] | Abort. | 9-14 |

| SPECIAL FUNCTION | DEFINITION | DISPLAY |
|---|---|---|
| SFCN 0 | LOAD | LOAD |
| SFCN 1 | COMPARE | CH |
| SFCN 2 | SEARCH | SEARCH |
| SFCN 3 | RD TRC MEM | T. 000 S. 001 |
| SFCN 4 | SET MODE | RUN SHAR (RUN PRO) |
| SFCN A | ABORT | ABORTED |
| SFCN C | CONSOLE | CONS OLE? |
| SFCN CLR | RESET | RESE T? |

## 9.5 Front Panel Operation

COPMON may be loaded by inserting the disk and pressing the switch labeled "MONITOR." When it finishes loading COPMON the chip number prompt is displayed on the panel. The chip number is used by COPMON to select the correct memory size and register size (see Table 9.1). The user may either change the chip number by entering a new number or use the one displayed. If the number in the display is a valid chip number when "TERM" is pushed, it becomes the new chip number. Otherwise, the chip number prompt is displayed again with the old number. The first time COPMON is loaded after power-on, the chip number will be 420. After each subsequent loading of COPMON the chip number will be the valid chip number specified by the user.

| Example: | KEY SEQUENCE | DISPLAY |
|---|---|---|
| | | CHIP ? 420 |
| | 4 | CHIP ? 204 |
| | 4 | CHIP ? 044 |
| | 0 | CHIP ? 440 |
| | TERM | * |

After the user has responded to the chip number prompt, and after each subsequent command, COPMON returns to prompt mode. This is indicated by the flashing rightmost decimal point. While in prompt mode the user may enter commands from the panel or console.

**Table 9.6. COPMON Panel Operand Definitions**

| Operand | Definition |
|---|---|
| <addr> | = Shared memory address, range defined by chip number. (See Table 9.1.) |
| <block #> | = Decimal number, defines consecutive 512-byte blocks of shared memory or load module. Valid range determined by shared memory range of COP device. (See Table 9.1.) |
| <chip #> | = Chip number "410" or "411" or "420" or "421" or "440" or "444" or "445" |
| <cond.> | = <addr> — address<br><ev. cond.> — external event<br>SS — immediate |
| <ev. cond> | = SFNCE 00<br>= SFNCE 01<br>= SFNCE 02<br>= SFNCE 10<br>= SFNCE 20<br>= SFNCE 11<br>= SFNCE 12<br>= SFNCE 21<br>Format is SFNCE <EXT2> <EXT1><br>"0" = Logic 0<br>"1" = Logic 1<br>"2" = Don't care |
| <file #> | = Decimal file number from 1-99 as displayed by "D" command in file manager. (See Chapter 5.) |
| <mode> | = SMEM Shared memory<br>= SFNCE EPROM (MM5204)<br>= SFNCB BPROM (87S296) |
| <occur> | = Decimal number, 1-256, number of times <cond> occurs before initiating breakpoint or trace. |
| <prior> | = Decimal number, 0-253, number address traced prior to address on which <cond> was found. |
| <PROM type> | = SFNCE — MM5204 Erasable PROM<br>= SFNCB — 87S296 Bipolar PROM |
| <reg #> | = Hex digit in range 0 to the maximum register of the chip defined by <chip #>. (See Table 9.1.) |
| <trace #> | = Decimal number, 0-253, location in trace memory to be displayed. |
| <value> | = Hex number, 0-X'FF. |

**Table 9.7. COPMON Panel Error Messages**

| Panel Message | Meaning |
|---|---|
| ABORTED | COMMAND ABORTED |
| SEQ ERR | ILLEGAL COMMAND |
| OP ERR | ILLEGAL OPERAND |
| NOT ERAS | NOT ERASED |
| FILE ERR | FORMAT ERROR IN FILE |
| CHSU ERR | CHECKSUM ERROR |
| BAD PRO | BAD PROM |
| RNG ERR | ERROR — LOAD ADDRESS TOO LARGE |
| BP ABORT | BREAKPOINT ABORTED |
| TR ABORT | TRACE ABORTED |
| TRACE ER | CANT TRACE WHEN BRKPTED |
| CANTADDR | ADDR ILLEGAL WHEN TRACING |
| RUNNING | COP ALREADY RUNNING |
| TRACE EN | CANNOT SS WHEN TRACE ENABLED |

## COPMON Panel Commands

The COPMON panel commands are summarized in Table 9.5 and are described in detail below. Panel Operand definitions are listed in Table 9.6. Table 9.7 is a list of Panel Error Messages.

### BREAKPOINT Command

Syntax: BKPT [<cond>[(+/-)<cond> . . .]][,[<occur>]

Description: The BREAKPOINT command sets and displays the conditions which determine breakpoint operation. A maximum of 10 conditions is allowed. After a breakpoint command, conditions up through the largest number of conditions altered or examined are retained, others are cleared. The BREAKPOINT command puts the specified conditions in a circular list. Only the condition at the top of the list is looked for during breakpoint operation, but the list is rotated once each time a breakpoint is finished. The program counter and breakpoint condition appear from left to right on the display each time a breakpoint is finished. If <occur> or any conditions are not altered, then they are used again for the next breakpoint operation. The BREAKPOINT command sets the breakpoint enable flag when "TERM" is pushed. Breakpoint operation is initiated by the "GO" command when the breakpoint flag is set. (See GO Command and Table 9.4.) For proper breakpoint operation while running from PROMs, shared memory must have the same data as the PROMs.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SHIFT | |
| BKPT | BP 0 001 |
| SFUNC | BP 0 SFNC |
| E | BP 0 E |
| 2 | BP 0 E 02 |
| 1 | BP 0 E 21 |
| INCR | BP 1 NONE |
| 6 | BP 1 006 |
| , | OCCU R001 |
| 5 | OCCU R005 |
| TERM | * |

Above, the user has set two conditions, E21 and address 006, and has changed the occurrence to 5.

### DISPLAY BREAKPOINT DATA Commands

Syntax:

ACC
BREG
CARRY
GIN
IIN
LIN
PC
QREG
REG <reg. #>
SREG

Description: Each command displays the data corresponding to its label and returns to prompt mode with the exception of the REG command. The REG command has one operand. The REG command looks for this operand or an executable command. The CARRY command displays the carry and skip from left to right. These commands are only valid if the COP is breakpointed. If not, the message "NOT BPTD" is displayed.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SHIFT | |
| ACC | AREG E |
| SHIFT | AREG E |
| PC | PC 20A |
| SHIFT | PC 20A |
| REG | R 00 6 |
| 2 | R 02 B |
| 5 | R 25 9 |
| TERM | * |

### CHIP Command

Syntax: C<chip #>

Description: The chip command allows the user to display and change the chip number exactly as the chip number prompt does. The chip number determines memory and register size (see Table 9.1). *This command resets the COP.*

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| C | CHIP ? 420 |
| 4 | CHIP ? 204 |
| 1 | CHIP ? 041 |
| 0 | CHIP ? 410 |
| TERM | * |

### CLEAR Command

Syntax: CLR.

Description: This command clears the breakpoint and trace enable flags.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| CLR | CLEA R? |
| TERM | BPTN CLR* |

### COMPARE Command

Syntax:

SFNC 1 SMEM <block #> <PROM type>. [+].
SFNC1 SFNCD <file #>,<block #> <PROM type>. [+].
SFNC1 SFNCD <file #> SMEM. [+].

Description: Three examples of syntax are shown. The first compares the specified block of shared memory to the specified PROM. The second compares the specified block of the specified file to the specified PROM. The third compares the specified file to shared memory. These are the only valid syntax for the COMPARE command. The first value that does not compare is displayed with the address of shared memory where the data would be located. The address is displayed on the left side and the disagreeing values are separated by a period on the right side. The values from left to right are from the source device and the destination device. The source and destination devices are the first and second in the command string, respectively. When the first non-compare is found, the user can display subsequent non-compares by pressing "INCR" until the last one.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| 1 | CH |
| SFUNC | SRC SFNC |
| D | SRC DISC. |
| 3 | SRC D.03 |
| , | SRC D.0.3.0 |
| 2 | SRC D.0.3.2 |
| SFUNC | SRC SFNC |
| E | DEST EPRO |
| TERM | ?.4A6 66.46 |
| INCR | ?.4F1 3A.3B |
| INCR | CH DONE* |

In the above example, block 2 of file 3 had two non-compares with the EPROM. At address 4A6, the disk had 66 while the PROM had 46. At 4F1, the disk had 3A and the PROM had 3B.

### CONSOLE Command

Syntax: SFNCC

Description: This command allows the user to select a new baud rate. After the command is entered, a carriage return will allow automatic selection of baud rate.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| C | CONS OLE? |
| TERM | CR? |
| | (carriage return) |

### GO Command

Syntax: GO [<addr.>]

Description: The function of the command depends on the status of the COP chip and the breakpoint and trace enable flags. (See BREAKPOINT and TRACE Commands.) Each possible function of the GO command is described in Table 9.4. Generally, a breakpoint will be initiated if the breakpoint enable flag is set, a trace will be done if the trace enable flag is set, and the chip will be started. Breakpoint and trace flags remain unchanged after the GO command. Suppose the breakpoint flag were enabled and the first condition on the list were E 21. The following sequence would occur.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| | . |
| SHIFT | . |
| GO | GO . |
| TERM | (.) |
| | B.56F E 21 |

In the above example, the chip was running until it found the breakpoint condition, at which point the run light went off and the message appeared. The message contains the program counter and breakpoint condition from left to right. At this time other breakpoint data is available also. (See DISPLAY BREAKPOINT DATA Commands.) Of course, this example assumes a program in shared memory which eventually finds EXT1 high at address X'56F. The run light in parentheses represents the temporary blank panel between "TERM" and finding the condition. This may or may not be too short for the user to see. If trace were enabled instead of breakpoint, with a trace condition of E 21, the following sequence would occur:

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| | . |
| SHIFT | . |
| GO | GO . |
| TERM | T.56F. E 21* |

In the above example, COPMON has stored the 254 consecutive program counter values including skip information. Assuming the same program as in the previous example, EXT1 again went high at address X'56F. The message contains the program counter value when the event was found and the event (EVENT 2 DON'T CARE/EVENT 1 = "1"). When the COP is traced, as in this example, the user may examine trace memory with the SEARCH and TRACE MEMORY commands.

### LOAD Command

Syntax:
SFNC0 <PROM type> SMEM <block #>
SFNC0 SMEM <block #> <PROM type>. [.]
SFNC0 SFNCD <file #>, <block #> <PROM type>. [.]
SFNC0 SFNCD <file #> SMEM

Description: The LOAD command loads data from the first device in the command sequence (source) to the second device in the command sequence (destination). The first syntax line dumps the contents of the specified PROM into the specified block of shared memory. The second syntax line programs the specified PROM from the specified block of the specified file. The last syntax line loads shared memory from the specified file. These are the only valid syntax for this command. When programming a PROM, a PROM-type verification prompt is displayed. Bipolar PROMs (87S296) require a special pin scrambler (see Section 2.3 and Figure 2.3). If "TERM" is pressed after the PROM-type prompt, the PROM is programmed. Because programming voltages are different for the two types of PROMs, it is important that the correct device be inserted before programming. Programming a PROM takes a couple of minutes, and during this time a message is displayed indicating the PROM is being

programmed. After the two programming commands, the checksum is displayed. After the other two commands, a message is displayed indicating that the load is finished.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| 0 | LOAD |
| SFUNC | SRC SFNC |
| D | SRC DISC |
| 1 | SRC D.01 |
| 2 | SRC D.12 |
| , | SRC D.12.0 |
| 2 | SRC D.12.2 |
| SFUNC | SRC SFNC |
| B | DEST BPRO |
| TERM | BPRO ? |
| TERM | |
| | PROG - ING |
| | CHSU XXXX* |

**MODE Command**

Syntax: SFNC4 [<mode>]

Description: This command allows the user to examine or alter the running mode of the COP. The COP may run from shared memory or from a PROM. *This command resets the COP chip.*

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| 4 | RUN SHAR |
| SFUNC | RUN SFNC |
| E | RUN PRO |
| TERM | * |

**RESET Command**

Syntax: SFNCCLR.

Description: This command resets the COP chip.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| CLR | RESE T? |
| TERM | RESE T * |

**SEARCH Command**

Syntax: SFNC2 SMEM <value> [+].
SFNC2 Trace <addr> [+].

Description: The first line of syntax searches shared memory for the value specified. *This command resets the COP chip.* The second line searches trace memory for the specified address. After the initial value or address is found, subsequent occurrences can be examined by pressing "INCR." In the example below, only two occurrences of 33 were found in shared memory.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| 2 | SEAR. CH |
| SHIFT | SEAR. CH |
| SMEM | S. |
| 3 | S.16A 03 |
| 3 | S.120 33 |
| INCR | S.134 33 |
| INCR | S. NO 33 |
| TERM | * |

In the above example, X'33 was found at address X'120 and X'134. The first occurrence of X'03 was inadvertently found along the way. In the next example, address 120 is found twice, in trace memory locations 17 and 27 (decimal).

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC |
| 2 | SEAR. CH |
| SHIFT | SEAR. CH |
| TRACE | T. |
| 1 | T. NO S.001 |
| 2 | T. NO S.012 |
| 0 | T.017 S.120 |
| INCR | T.027 S.120 |
| INCR | T.027 S.120 |
| TERM | T. NO.S.120 |
| | * |

**SHARED MEMORY Command**

Syntax: SMEM [<addr>] [,<value>[,<value> . . .]]

Description: This command allows the user to examine and alter the contents of shared memory. *This command resets the chip.* The first comma separates the address specification and value specification. Each subsequent comma increments the address. The value stored is the last value displayed for each address. In the example below X'C7 is stored in address X'011 and X'013,X'012 is left unaltered.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SHIFT | . |
| SMEM | S.000 00 |
| 1 | S.001 33 |
| 1 | S.011 23 |
| , | S.011 23 |
| C | S.011 3C |
| 7 | S.011 C7 |
| , | S.012 03 |
| , | S.013 23 |
| C | S.013 3C |
| 7 | S.013 C7 |
| TERM | * |

### SINGLE STEP Command

Syntax: SS

Description: This command allows the user to single step through a program. After each step some piece of breakpoint data is displayed. This data will be whichever register was examined during the previous breakpoint or single step. If none is specified it defaults to display the program counter. If the program counter advances by more than one in a step, it may mean that the undisplayed step(s) were skipped. Any other command may be executed immediately after single step.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SS | PC 3C7* |
| SS | PC 3C8* |
| SHIFT | PC 3C8* |
| ACC | AREG 4* |
| SS | AREG 8* |
| SS | AREG C* |

(Next command.)

### TRACE Command

Syntax: TRACE [<cond>][,[<occur>[,[<prior>]]]]

Description: This command allows the user to examine and alter the conditions which affect trace operation. If any of the options are not specified, they remain unaltered from the previous specification. The default values on entry to COPMON are X'001, 1, 0 respectively for <cond>, <occur> and <prior>. During trace operation COPMON stores each consecutive value of the COP program counter in a 254-word circular buffer, so that at any time during trace operation the buffer has the 254 previous values of the program counter. When <cond> has been met the number of times specified by <occur>, COPMON saves the number of values of program counter prior to <cond> specified by <prior>, and fills the rest of the buffer with the subsequent values of the program counter. It then displays a message which contains from left to right the address where <cond> is recognized and the <cond> specified in the TRACE command. The TRACE command does not initiate trace operation but sets the trace enable flag. If that flag is set, trace operation is initiated on the next GO command. (See GO Command and Table 9.4.)

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SHIFT | . |
| TRACE | TR . 001 |
| 4 | TR . 004 |
| 5 | TR . 045 |
| , | OCCU . R001 |
| 4 | OCCU . R004 |
| , | PR10 . R000 |
| 6 | PR10 . R006 |
| 4 | PR10 . R064 |
| TERM | . * |

### TRACE MEMORY Command

Syntax: SFNC 3 [<trace #>]

Description: This command displays trace memory. The first location displayed is the last location displayed by the previous TRACE MEMORY command or the location where <cond> was recognized.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SFUNC | SFNC. |
| 3 | T.064. S.045 |
| 2 | T.642. ERR |
| 1 | T.421. ERR |
| 0 | T.210. S.16C |
| TERM | . * |

### ABORT Command

Description: Any command may be aborted before pressing "TERM" by pressing SFNCA. The prompt mode is returned.

Example:

| KEY SEQUENCE | DISPLAY |
|---|---|
| SHIFT | |
| GO | GO |
| 1 | GO 001 |
| SFUNC | GO SFNC |
| A | ABOR TED * |

# 10 File List Program (LIST)

The File List program (LIST) is a PDS system program that provides the user with a means of listing any type of file on the system console or printer. LIST has several print options available that allow setting of page headings, control of page and line numbers, etc. Files that are not symbolic are listed in hexadecimal and ASCII.

## 10.1 Using LIST

To call LIST, the user types in the @ command:

```
X>@LIST filename [options]
LIST,REV:A
(Listing now begins.)
```

or

```
X>@LIST
LIST,REV:A
L>filename [options]
(Listing now begins.)
```

where "filename" is the name of the file to be listed, and "options" are print options described below. Table 10.1 summarizes all LIST options. Each may be abbreviated to the first two characters. The default modifier for the filename is "SRC."

Upon completion of all copies of the listing, LIST prompts the user for another filename and options.

### Options

Options are scanned left to right and are separated from the @LIST command and other options by spaces. The order is significant to the extent that later options may change those previously specified. Otherwise, the only other requirement is that the HD option must be last (since it is followed by text rather than more options). All option names can be abbreviated to the first two characters. In the option parameters, "n" is a decimal number and "char" is a single ASCII character.

**Table 10.1. Summary of LIST Options**

| Option | Meaning |
|---|---|
| ANSI | File contains ANSI carriage-control character in column 1. |
| CO n | Select number of copies to print. |
| DBL | Double space. |
| FO char | Set form feed character. |
| HD text | Set heading text. |
| HEX | Print file in unformatted hex/ASCII. |
| IN n | Indent left margin n spaces. |
| LI n/n | Select line range to print. |
| LP n/n | Set number of lines per page. |
| NE | Suppress page eject after 58 lines. |
| NF | Suppress formatting (same as combination of NE, NH, NP and UN). |
| NH | Suppress heading at top of page. |
| NL n | Set number of nulls following a carriage return. |
| NP | Suppress page eject when encountering assembler directive .FORM. |
| N or NU | Print line numbers on listing. |
| PA n/n | Select page range to print. |
| PG | Eject page when encountering assembler directive .FORM. |
| P or PR | Select printer output. |
| QU | Compress blanks in the output (quick mode). |
| TAB | Print the tab character. |
| TR n | Set right margin at n characters and truncate characters exceeding the margin. |
| TTY | Send form feed on new pages. |
| UN | Suppress line numbers on listing. |
| WAIT | Wait for any input character before resuming listing on new pages. |
| WI n | Set right margin at n characters and start new line for characters exceeding the margin. |

*Device Selection Options* specify the output device, line width, and spacing between pages.

| Option | Description |
|---|---|
| (default) | Six line feeds between page on output listing. |
| NL n | Set number of nulls to be output after a carriage return; used for generating paper tapes to be read on other systems. |
| P or PR | Select Printer output and generate form feed between pages. |
| TTY | TTY new page: send form feeds instead of line feeds. |
| WAIT | TTY new page: wait for input character to resume. |

*Text Formatting Options* control the actual text that is printed.

| Option | Description |
|---|---|
| ANSI | Treat the first character of the text line as an ANSI carriage-control character. The control character is not printed but controls the line spacing instead. The control characters are:<br>+ = No spacing (overprint)<br>– = Triple space<br>0 = Double space<br>1 = New page<br>Anything else = Single space |
| DBL | Double space. If used together with the ANSI option, the specified spacing is doubled. |
| FO char | Set form feed character. If this character occurs in column 1 of the text line, a new page occurs (instead of printing the character). |
| HEX | Print file in unformatted hexadecimal/ASCII. Other formatting options will not apply. This option is always taken for files other than Symbolic or Data files. |
| N or NU | Print line numbers on listing (default for files with modifier of ".SRC"). |
| QU | Quick Mode: compress blanks in the printout. Any sequence of consecutive blanks in a line image is printed as a single blank. This option will negate any indent that may be set. |
| TAB | Print the tab character (09). Otherwise the tab is printed as the number of spaces required to move to the next fixed tab stop (every 8 columns in the text field). |
| UN | Do not print line numbers (default for all files except for those with a modifier of ".SRC"). |

*Line Size Options* control indentation and margin.

| Option | Description |
|---|---|
| IN n | Indent left margin n spaces (default = 0). |
| TR n | Set right margin at n characters; any additional characters are truncated and not printed. |
| WI n | Set right margin at n characters; any additional characters are folded over onto a new line (default = 72). |

Note: The PR Option sets the width to 80, but does not change the fold/truncate mode.

*Print Selection Options* determine what part of the data is to be printed.

| Option | Description |
|---|---|
| CO n | Select number of copies to print. |
| LI n/n | Select line range to print. May be specified as n or n/n; the first number is the first line to be printed and the second number is the last line to be printed. |
| PA n/n | Select page range to print; action is the same as LI option except that the range is by page number. |

Both LI and PA may be specified; the action is as follows: if either range starts at 1, the starting number of the other option determines the first line to be listed. The first end specification encountered stops the listing. If a single line or page number is specified, printing begins with that line or page and continues to the end of the file.

*Page/Heading Options* control the number of lines on a page, page heading, and page printing.

| Option | Description |
|---|---|
| (default) | Lines are counted so that a new page occurs after every 58 text lines, thus providing proper formatting on 8½-by-11 pages. If the file modifier is .LST, the NF option is assumed. If the file modifier is SRC, the PG option is assumed; otherwise NP is assumed. |
| HD text | Set heading text. The default is the name of the file, e.g., LIST.SRC. This must be the last option on the calling line since all characters following the HD up to the carriage return are used for the heading text. |
| LP n/n | Set number of lines per page. The first number is the number of text lines on a page; the second number is the number of lines on a sheet of paper. The default is 58/66, which is correct for normal 8½-by-11 pages. It is not necessary to specify both numbers if only the number of text lines is to be changed. The text will be automatically centered vertically on the page. |
| NE | Suppress page eject on counted line; page ejects occur only where explicitly determined by the text (either form feeds or assembler directive .FORM, if being checked). |
| NF | No formatting (same as NE, NH, NP, and UN). This option is the default for files with a modifier of ".LST." |
| NH | Suppress heading at top of page. |
| NP | Suppress page eject when encountering assembler directive .FORM. |
| PG | Check for assembler directive .FORM. This option is the default for files with a modifier of ".SRC." |

Example: List on printer ADD.SRC file, no formatting except for line number printout:

```
X>@LIST ADD.SRC PR NE NH NP
```

# 11 Cross Reference Program (XREF)

The Cross Reference program (XREF) is a PDS system program that provides the user with a means for printing a "symbol map" of COP assembly language programs. The symbol map shows the name of every symbol in the program, the line number where the symbol is defined, and all of the line numbers where the symbol is used.

## 11.1 Using XREF

To call XREF, the user types in the @ command:

```
X>@REF filename
XREF,REV:A
```

(Cross referencing now begins.)

or

```
X>@XREF
XREF,REV:A
R>filename
```

(Cross referencing now begins.)

where "filename" is the name of the COP assembly language (SYMBOLIC) file to be cross referenced. If the filename is followed by "*PR," the listing will be printed on the printer.

Figure 11.1 shows a typical cross reference listing. Local symbols (i.e., those starting with a $ sign) are listed first. Symbols are listed in alphabetical order. The numbers listed beside each symbol are the numbers of the lines in which the symbol appears. A "-" beside a line number indicates that the symbol is assigned or otherwise given a value in that line. A "*" beside a symbol means that the symbol appears in only one line.

```
X>@XREF PDS:COPPGM.SRC

FILE PDS:COPPGM.SRC

$5          653     703-
$8          538     557     560-
$DF1        258     287-
$DF2        260      88-
$END        536     563-
$LOOP       533-    552
$SAVE       418     437     450-    577     592     595-
A            71-    103     104     112     113     133     134
ADD1         88     326-
ADDSYM      338     391-
AFST         47-    257     525
B           103     105     112     113     783-    789     790     791     792     793
            795     796     797     798     799     801     803     804     805     809
            820     821     822     824     825     829     830     832     835     837
            843     844     846     847     851     852     854     855     857     861
            863     864     867     869     872     874
BEG         154-    170     171     172     173     190     200     204     205     205
            206     206     207     208
IQT         157-    169     189     189     190     191     191     192     192     193
            193     194     194
JTAB         81-    312
L            66-    136     137     137     137     137     138     138     138     138
            139     139     139     139     140     140     140     140     141     141
            144     145     145     145     145     146     146     146     146     147
            147     147     147     148     148     148     148     149     149
LAST         28-    237     336     391     400     420     652     722
LBYT        468     472-
LINCNT       41-    648     708     713
LINE         39-    233     432     529     532
MAIN        269     296-    313
MAXRAM       60-    238
MESG         51-    253     271     396     446
MTOP         61-    239
N            68-    131     131     131     131     132     132     132     132     133
            133
SYM         155-    168     170     171     173     175     175     176     176     180     180
            182     183     187     203
T            77-    127
TEMP         43-    368     369     539     545     673     690
TOVFLW      397     447     451-
X            67-    141     149
*XREF         1
XX          104     107
YY          105     107
ZRO          19-    466     583     685
ZZ          106     107
```

**Figure 11.1. Typical Cross Reference Listing**

# 12 Mask Transmittal Program (MASKTR)

## 12.1 Use of Mask Transmittal Program

MASKTR is a PDS system program which is used to generate a "Transmittal File" that NSC uses for creating the COP chip ROM/OPTIONS mask and the functional test program.

The transmittal filename will be the same as the LOAD MODULE filename, the modifier will be .TRN, and the internal file type is SYT.

The transmittal file contains:

1. Name and phone number of the responsible person.
2. Company name and address.
3. Date.
4. Chip number.
5. Listing of options showing option number, option name, and option value.
6. ROM data including addresses, unused addresses are set to OP CODE zero (0), which is a CLRA instruction.
7. Source, object, and transmittal file checksums.

To enter any information for the TRANSMITTAL file, MASKTR must first be in the TRANSMITTAL mode. This mode may be entered with the TRANSMITTAL command (T) or by typing the filename on the end of the '@MASKTR' line.

When MASKTR is in the TRANSMITTAL mode the user is requested to provide the necessary information:

1. LOAD MODULE FILENAME
2. CHIP NUMBER
3. NAME AND PHONE NUMBER OF RESPONSIBLE PERSON
4. COMPANY NAME AND ADDRESS
5. DATE
6. OPTION VALUES

MASKTR prompts the user with a description of the desired item required by the program, the current value of the data item (as last entered by the user), and then asks for the new value from the user. If no change is required a carriage return will leave the value unchanged; if a change is requested for the chip number or options the value entered is checked for validity. Entering a blank line causes an advance to the next item to be entered.

The items are arranged in a circular order such that the user will be prompted for responsible person (name/phone), company (name/address), date, chip number, options, and then back to responsible person in that order.

NOTE: A CNTL Q in column 1 causes a return to the prompt mode.

To call MASKTR, type:

```
X>@MASKTR

MASKTR, REV:B, DATE
T>
```

MASKTR is then ready to accept one of the commands listed in Table 11.1 and described in detail below.

**Table 12.1. MASKTR Command Summary**

| Command Name | Operand Syntax | Description |
|---|---|---|
| ABORT | A | Abort transmittal file creation |
| CHIP | C | Enter chip number |
| COMPANY | CO | Enter company name/address |
| DATE | D | Enter date |
| FINISH | F | Finish transmittal file creation |
| LIST | L | List transmittal file |
| NAME | N | Enter responsible person name/phone |
| OPTION | O [<opt#>] | Enter chip options |
| PRINT | P <chip#> | Print available options for chip |
| TRANSMITTAL | T <filename> | Begin creation of transmittal file |

where:

<opt#> = Number of valid option for current chip number. "O" may be left off of command call if <opt#> is used. This number causes entry mode to be entered at the specified option number. If "O" alone is used entry is at the beginning of the option list.

<chip#> = Valid chip number and letter.

<filename> = Standard PDS filename.

### 12.2. ABORT Command

Syntax: ABORT

Description: This command will abort the creation of a transmittal file and return the program to the PROMPT mode.

Example:

```
T>A
ABORT TRANSMITTAL FILE CREATION (Y/N, CR = YES) CR
TRANSMITTAL FILE CREATION ABORTED
T>
```

### 12.3. CHIP Command

Syntax: CHIP

Description: This command causes the program to prompt the user for the chip number.

Example:

```
T>C
CHIP NUMBER: 420L
CHIP NUMBER: 320L
EXTENDED TEMPERATURE RANGE (Y/N,
CR = Y)? CR
```

Note: The chip number must be specified in the above manner if parts with extended temperature range are desired.

### 12.4. COMPANY Command

Syntax: COMPANY

Description: This command causes the program to prompt the user for the company name and address. Eight lines are allowed for this entry.

Example:

```
T>CO
COMPANY NAME AND ADDRESS:
UNSPECIFIED
COMPANY NAME/ADDRESS:
NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DR.
SANTA CLARA, CA 95051
CR
DATE: UNSPECIFIED
```

### 12.5. DATE Command

Syntax: DATE

Description: This command causes the program to prompt the user for the date. One line is allowed for this entry.

Example:

```
T>D
DATE: UNSPECIFIED
DATE: 1 JANUARY, 1979
CHIP NUMBER: 420
```

### 12.6. FINISH Command

Syntax: FINISH

Description: This command finishes creation of the transmittal file, and writes it to the disk. There is a prompt for the disk to be sent to NSC to be placed in the drive. NOTE: The disk must be an initialized disk.

Example:

```
T>F
FINISH CREATION OF TRANSMITTAL FILE (Y/N, CR = YES) CR
INCOMPLETE OPTION SPECIFICATION
T>
```

Note: The user must completely define all options before the program will allow a transmittal file to be written to the disk.

The FINISH command also checks for conflicting CKO-CKI option selections and option selections which are illegal for a bonding option value of 2.

### 12.7. LIST Command

Syntax: LIST

Description: This command causes the transmittal file to be listed as it will appear on the form that will be returned to the customer from NSC for verification and sign-off before a mask will be generated from the customer's transmittal disk.

NOTE: An *PR at the end of this command will cause the listing to go to the printer.

Example: T>L

This example will list the transmittal file on the console in blocks that will fit on the screen. A CR will advance to the next block of data. Any other key followed by a CR will return to the PROMPT. A CNTLQ will also return to the PROMPT mode.

### 12.8. NAME Command

Syntax: NAME

Description: This command prompts the user for the name/phone number of the person responsible for this program. Two lines are allowed for this entry.

Example:

```
T>N
RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE USER
123 456 7890
COMPANY NAME/ADDRESS:
```

## 12.9. OPTION Command

Syntax: [Option] [<opt#>]

Description: This command causes the program to prompt the user for the valid options for the chip specified. If the "O" is omitted the <opt#> must be specified. If the "O" is inserted and <opt#> is omitted the program prompts for options from the first option.

Example:

```
T>O 12

OPTION 12: L3 DRIVER = UNSPECIFIED

  00 = STANDARD OUTPUT
  01 = OPEN DRAIN
  02 = HI CURRENT LED SEG OUT
  03 = HI CURRENT TRI-STATE
  04 = LOW CURRENT LED SEG OUT
  05 = LOW CURRENT TRI-STATE

OPTION 12: L3 DRIVER 01

OPTION 12: L3 DRIVER = 01 (Y/N, CR = YES) CR

OPTION 13: L2 DRIVER = UNSPECIFIED
```

## 12.10. PRINT Command

Syntax: PRINT <CHIP#>

Description: This command prints out the allowable options for the chip specified in the command.

NOTE: If an *PR is entered at the end of the line the options are sent to the printer.

Example:

```
T>P 420

ABORT TRANSMITTAL FILE CREATION (Y/N, CR = YES); CR

TRANSMITTAL FILE CREATION ABORTED
T>P 420

CHIP NUMBER: 420

OPTION 1: GROUND

  NOT AN OPTION

OPTION 2: CKO OUTPUT

  00 = CLOCK GEN OUT XTAL/RES
  01 = RAM KEEP ALIVE
  02 = GENERAL INPUT, VCC LOAD
  03 = MULTICOP SYNC IN
  04 = GENERAL INPUT, HI-Z
```

This example will print the COP420 options on the console. As in the LIST command, the block of data is sent to the screen and a CR return will advance through the options. An *PR will put the options to the printer. The print command can NOT be used wile in the TRANSMITTAL MODE.

## 12.11. TRANSMITTAL Command

Syntax: TRANSMITTAL < filename>

Description: When the TRANSMITTAL command is invoked, the chip number prompt is given. The LOAD MODULE is read into memory, and the entered chip number is checked against the chip number contained in the LOAD MODULE (assembled with REV B ASM). If the chip numbers do not match the program aborts the TRANSMITTAL command and returns to prompt. If the chip numbers agree, the valid chip number is entered into the data table and used to determine which options are valid and available. The ROM data and option values (if any) from the LOAD MODULE are also entered into the data table.

The TRANSMITTAL command may also be invoked by typing the filename on the @MASKTR line.

Example:

```
EXEC, REV:A

X>@MASKTR

MASKTR, REV:B,DATE

M>T MASKEX

DISK WITH LOAD MODULE IN DRIVE (Y/N, CR = YES)? CR

CHIP NUMBER: UNSPECIFIED
CHIP NUMBER: 421

CHIP NUMBER: 421
CHIP NUMBER: CR

ERROR: PROGRAM ASSEMBLED FOR 420
M>T MASKEX

DISK WITH LOAD MODULE IN DRIVE (Y/N, CR = YES)? CR

CHIP NUMBER: UNSPECIFIED
CHIP NUMBER: 420

CHIP NUMBER: 420
CHIP NUMBER: CR

RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE COPUSER
(415) 777-6234

COMPANY NAME/ADDRESS:
UNSPECIFIED
```

COMPANY NAME/ADDRESS:
NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051
CR

DATE: UNSPECIFIED
DATE: JANUARY 5, 1979

CHIP NUMBER: 420
CHIP NUMBER: CR

OPTION 01: GROUND = 00

NOT AN OPTION

OPTION 02: CKO OUTPUT = 02

00 = CLOCK GEN OUT XTAL/RES
01 = RAM KEEP ALIVE
02 = GENERAL INPUT, VCC LOAD
03 = MULTICOP SYNC IN
04 = GENERAL INPUT, HI-Z

OPTION 02: CKO OUTPUT CR

OPTION 03: CKI INPUT = 04

00 = XTAL /16
01 = XTAL /8
02 = TTL /16
03 = TTL /8
04 = RC/4
05 = OSC (SCHMITT IN) /4

OPTION 03: CKI INPUT CR

OPTION 04: RESET INPUT = 00

00 = LOAD VCC
01 = HI-Z

OPTION 04: RESET INPUT 1

OPTION 04: RESET INPUT = 01 (Y/N, CR = YES)? CR

OPTION 05: L7 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 05: L7 DRIVER CR

OPTION 06: L6 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 06: L6 DRIVER CR

OPTION 07: L5 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 07: L5 DRIVER CR

OPTION 08: L4 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 08: L4 DRIVER CR

OPTION 09: IN 1 INPUT = 00

00 = TTL LOAD
01 = TTL HI-Z

OPTION 09: IN 1 INPUT CR

OPTION 10: IN 2 INPUT = 00

00 = TTL LOAD
01 = TTL HI-Z

OPTION 10: IN 2 INPUT CR

OPTION 11: VCC = 00

NOT AN OPTION

OPTION 12: L3 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 12: L3 DRIVER CR

OPTION 13: L2 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 13: L2 DRIVER CR

OPTION 14: L1 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 14: L1 DRIVER CR

OPTION 15: L0 DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE

OPTION 15: L0 DRIVER CR

OPTION 16: SI INPUT = 00

00 = LOAD VCC
01 = HI-Z

OPTION 16: SI INPUT CR

OPTION 17: SO DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = PUSH/PULL

OPTION 17: SO DRIVER CR

OPTION 18: SK DRIVER = 02

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = PUSH/PULL

OPTION 18: SK DRIVER CR

OPTION 19: IN 0 INPUT = 00

00 = TTL LOAD
01 = TTL HI-Z

OPTION 19: IN 0 INPUT CR

OPTION 20:: IN 3 INPUT = 00

00 = TTL LOAD
01 = TTL HI-Z

OPTION 20: IN 3 INPUT CR

OPTION 21: G0 I/O PORT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = STANDARD OUTPUT SMALL DRIVER
03 = OPEN DRAIN SMALL DRIVER

OPTION 21: G0 I/O PORT CR

OPTION 22: G1 I/O PORT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = STANDARD OUTPUT SMALL DRIVER
03 = OPEN DRAIN SMALL DRIVER

OPTION 22: G1 I/O PORT CR

OPTION 23: G2 I/O PORT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = STANDARD OUTPUT SMALL DRIVER
03 = OPEN DRAIN SMALL DRIVER

OPTION 23: G2 I/O PORT CR

OPTION 24: G3 I/O PORT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = STANDARD OUTPUT SMALL DRIVER
03 = OPEN DRAIN SMALL DRIVER

OPTION 24: G3 I/O PORT CR

OPTION 25: D3 OUTPUT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN

OPTION 25: D3 OUTPUT CR

OPTION 26: D2 OUTPUT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN

OPTION 26: D2 OUTPUT CR

OPTION 27: D1 OUTPUT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN

OPTION 27: D1 OUTPUT CR

OPTION 28: D0 OUTPUT = 00

00 = STANDARD OUTPUT
01 = OPEN DRAIN

OPTION 28: D0 OUTPUT CR

OPTION 29: COP FUNCTION = 00

00 = NORMAL
01 = MICROBUS

OPTION 29: COP FUNCTION CR

OPTION 30: COP BONDING = UNSPECIFIED

00 = 28 PIN PACKAGE
02 = 28 AND 24 PIN PACKAGES

OPTION 30: COP BONDING 2

OPTION 30: COP BONDING = 02 (Y/N, CR = Y) CR

OPTION 31: IN INPUT LEVEL = 00

00 = STANDARD TTL
01 = HIGH TRIP POINT

OPTION 31: IN INPUT LEVEL CR

OPTION 32: G INPUT LEVEL = UNSPECIFIED

00 = STANDARD TTL
01 = HIGH TRIP POINT

OPTION 32: G INPUT LEVEL 1

OPTION 32: G INPUT LEVEL = 01 (Y/N, CR = YES)? CR

OPTION 33: L INPUT LEVEL = UNSPECIFIED

00 = STANDARD TTL
01 = HIGH TRIP POINT

OPTION 33: L INPUT LEVEL 1

OPTION 33: L INPUT LEVEL = 01 (Y/N, CR = YES)? CR

OPTION 34: CKO INPUT LEVEL = UNSPECIFIED

00 = STANDARD TTL
01 = HIGH TRIP POINT

OPTION 34: CKO INPUT LEVEL 0

OPTION 34: CKO INPUT LEVEL = 00 (Y/N, CR = YES)? CR

OPTION 35: SI INPUT LEVEL = UNSPECIFIED

00 = STANDARD TTL
01 = HIGH TRIP POINT

OPTION 35: SI INPUT LEVEL 0

OPTION 35: SI INPUT LEVEL = 00 (Y/N, CR = YES)? CR

RESPONSIBLE NAME/PHONE:

JOE COPUSER
(415) 777-6234

RESPONSIBLE NAME/PHONE: CR

COMPANY NAME/ADDRESS:

NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051
USA

COMPANY NAME/ADDRESS: CNTLQ

#

M>L

PDS TRANSMITTAL FILE

RESPONSIBLE NAME/PHONE:

JOE COPUSER
(415) 777-6234

COMPANY NAME/ADDRESS:

NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051

DATE: JANUARY 5, 1979

FILE NUMBER B8A7 62A0 102B

CHIP NUMBER: 420

| OPTION | VALUE | OPTION | VALUE |
|---|---|---|---|
| 01: GROUND | = 00 | 10: IN 2 INPUT | = 00 |
| 02: CKO OUTPUT | = 02 | 11: VCC | = 00 |
| 03: CKI INPUT | = 04 | 12: L3 DRIVER | = 02 |
| 04: RESET INPUT | = 01 | 13: L2 DRIVER | = 02 |
| 05: L7 DRIVER | = 02 | 14: L1 DRIVER | = 02 |
| 06: L6 DRIVER | = 02 | 15: L0 DRIVER | = 02 |
| 07: L5 DRIVER | = 02 | 16: SI INPUT | = 00 |
| 08: L4 DRIVER | = 02 | 17: SO DRIVER | = 02 |
| 09: IN 1 INPUT | = 00 | 18: SK DRIVER | = 02 |
| 19: IN 0 INPUT | = 00 | 28: D0 OUTPUT | = 00 |
| 20: IN 3 INPUT | = 00 | 29: COP FUNCTION | = 00 |
| 21: G0 I/O PORT | = 00 | 30: COP BONDING | = 02 |
| 22: G1 I/O PORT | = 00 | 31: IN INPUT LEVEL | = 00 |
| 23: G2 I/O PORT | = 00 | 32: G INPUT LEVEL | = 01 |
| 24: G3 I/O PORT | = 00 | 33: L INPUT LEVEL | = 01 |
| 25: D3 OUTPUT | = 00 | 34: CKO INPUT LEVEL | = 00 |
| 26: D2 OUTPUT | = 00 | 35: SI INPUT LEVEL | = 00 |
| 27: D1 OUTPUT | = 00 | | |

ROM VALUES:

```
000   00 33 5E 33 6C 2E 8D 3E 8D 91 3A 70 3E 7D 33 A8
010   7F 33 B8 7F 2E 7D 61 80 00 01 51 11 51 03 51 13
020   51 5E 49 48 00 00 00 00 00 00 00 00 00 00 00 00
030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
040   33 B8 15 5F CC 5F DA 5B 68 60 63 C6 00 58 21 F1
050   2C 05 5F 00 26 50 00 16 72 CA 00 58 21 EF 91 CA
060   2C 05 52 5F 48 06 25 50 23 28 16 23 38 06 48 00
070   52 55 21 CA 3A 46 CA 00 00 00 00 00 00 00 00 00
080   15 23 B9 05 23 A9 48 05 23 B9 05 04 83 00 07 8D
090   48 0E 68 8D 1D 00 52 07 95 1E 70 70 2C 70 48 3C
0A0   33 2A 40 06 4C 32 4F 5F 4D C8 05 51 51 5F AB 48
0B0   3F 04 04 04 04 04 04 04 C7 0E 33 3E 48 22 00 56
0C0   30 4A 07 00 56 30 4A 06 05 48 00 00 00 00 00 00
0D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

100   43 01 4B 03 4B 03 01 03 00 4B 4B 30 02 14 24 03
110   01 23 21 03 49 02 90 A0 B4 54 93 02 24 01 A0 02
120   00 CA 08 4B 4B D8 3B 10 30 84 FC 48 80 00 C2 90
130   4A 48 0A 4A 48 42 42 48 4A 4A CE 88 10 02 04 41
140   5F F7 8F 39 0F 79 71 BD F6 09 11 70 38 36 36 3F
150   F3 3F F3 ED 01 3E 30 36 00 00 09 31 00 0E 00 08
160   00 00 20 FF ED ED 58 00 00 00 C0 C0 00 C0 00 00
170   31 00 51 41 60 61 71 01 71 61 01 00 80 C8 40 83
180   1E 15 54 BF 33 2C 16 06 0F BF 33 2C 16 06 38 15
190   33 3C 33 5F 1F 22 05 B9 4F 44 0F 05 4F 44 1E 05
1A0   4F 0E 05 3E 4F 35 50 32 4F 41 ED 6A 80 BA 33 5F
1B0   3E 35 AB 50 05 23 8F 15 23 80 05 1E 06 43 42 9F
1C0   6B 40 33 5E 3E 05 52 D0 23 3D 2A 17 05 5C DB D7
1D0   51 DE 23 3D 2B 68 B8 A9 A9 32 F4 6B 4D FB 23 3D
1E0   5F E8 2E 05 5E E9 63 C0 A9 2D 05 3E 21 D8 3D 05
1F0   3C 32 21 22 68 18 32 2E 00 30 06 3E AA 06 61 80
```

ROM VALUES

```
200   30 31 32 33 34 35 36 37 38 39 30 2A 2D 00 00 FF
210   00 7D 51 57 45 52 54 59 55 40 4F 50 0A 00 00 00
220   00 41 53 44 46 47 48 4A 4B 4C 3B 7F 0D 00 00 00
230   00 5A 58 43 56 42 4E 4D 2C 2E 2F 20 08 00 00 00
240   00 21 22 23 24 25 26 27 28 29 40 3A 3D 00 00 00
250   00 7D 51 57 45 52 54 59 55 49 5F 40 0A 00 00 00
260   00 41 53 44 46 47 48 4A 5B 5C 2B 7F 0D 00 00 00
270   00 5A 58 43 56 42 5E 5D 3C 3E 3F 20 08 00 00 00
280   33 A1 05 5F C7 06 F0 07 C2 3A 11 CD D6 33 A2 25
290   16 73 35 4E 58 CF 2F 7D 7A 33 A7 BD 5A F4 07 BD
2A0   5A F0 07 BD 5E ED 33 A3 05 5C ED 07 70 2F 7C 77
2B0   3E 05 50 48 33 A7 01 C0 F0 00 00 00 00 00 00 00
2C0   0D 00 07 C2 0F 06 1D 00 52 07 C9 1F 06 48 22 00
2D0   2B 11 32 03 D6 13 54 3D 13 53 03 52 11 51 2D BF
2E0   33 A8 33 2C 16 06 20 42 48 00 00 00 00 00 00 00
2F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

300   0A 0D 0F 13 18 2B 38 3A 35 35 33 B8 D6 2A DF 29
310   43 4C DD 29 35 50 80 F5 3B 05 5E E6 06 05 50 87
320   F5 33 B8 05 5E D6 28 7F 38 7F F5 2C 05 5F E1 06
330   33 91 80 6A C0 33 6C 48 91 E6 29 15 70 06 63 0A
340   33 01 48 33 68 39 13 DF 29 33 2C 16 06 39 05 56
```

```
350   DD 15 23 B8 05 23 A8 68 60 39 76 63 26 00 FF 01
360   E6 73 29 25 50 C9 72 29 43 4D 05 50 33 2C 07 CC
370   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
380   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
390   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3C0   9F 5F C6 51 68 18 61 FB B9 3E 05 2D 06 3C 05 3D
3D0   06 2E 70 3A 03 C6 6A CE 3B 13 E9 05 52 06 23 28
3E0   B0 23 38 B0 3A 01 60 40 C6 3F 4B E4 00 00 00 00
3F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

SOURCE CHECKSUM  62A0

OBJECT CHECKSUM 102B

TRANSMIT CHECKSUM B8A7
M>FI

FINISH CREATION OF TRANSMITTAL FILE (Y/N, CR = YES)? CR

DISK TO BE MAILED IN DRIVE (Y/N, CR = YES)? CR

CREATING FILE JOEUSER:MASKEX. TRN

M>
```

The disk is now ready to be sent to:

National Semiconductor Corp.
2900 Semiconductor Dr.
Santa Clara, CA 95051

ATTN:
COP Control Customer Service
DISK/DISK/DISK/DISK/DISK/DISK

NOTE: A mailing package, which includes a label with this information, is available from:

COPS Marketing, M/S C2385
National Semiconductor
2900 Semiconductor Dr.
Santa Clara, CA 95051
Phone: (408) 737-5883

# 13 Memory Diagnostic (MDIAG)

## 13.1. Use of Memory Diagnostics

MDIAG is a PDS system program which allows the user to run diagnostics on the PDS memory. This program will run ADDRESS, BIT, WORD, AND GALPAT tests. The program will test the area in which it resides by moving the entire program just before the tests are run on that section of memory. After the tests are run on that section of memory the program is restored to its original location.

All reports of errors or passes are sent to the console unless the *PR is invoked, in which case all reports will be routed to the printer (i.e., overnite runs).

### ADDRESS Test

In the ADDRESS test, the address of each memory location in the test range is stored in that memory location. Each location is then checked for the proper value. This test checks the addressing capability of the PDS CPU and MEMORY boards.

### WORD Test

In the WORD test, a value is stored in successive memory locations. As each value is stored, that memory location is checked for the proper value. The value is then complemented, stored again, and the location is rechecked. Finally, the value is recomplemented, stored again, and the location is rechecked once again. Then the next memory location is tested in the same manner, until the end of the test range is reached. Then, the entire test is repeated for a total of two passes. This test checks whether memory words within the test range can save the given values and their complements.

### BIT Test

In the BIT test, each bit of each word in the test range is tested in the same manner as in the WORD test above. This test is identical to the WORD test except that a single "1"-BIT MASK is used for memory store and compare operations, and this mask is changed (after completing testing of the bit) to the next bit of the word. When all bits of one word are exhausted, the test advances to the next word of the test range.

### GALPAT Test

In the GALPAT test, a background word (X'AAAA) is stored in each memory location in the test range. Each location in the test range is then checked for the proper value. Next, a test word (X'5555) is stored in the first memory location of the test range. All the background locations are tested sequentially, with the test location being tested between each sequential background location. Then, the location where the testword was loaded is restored to the background word, and the testword is moved to the next location following the one it was stored in previously. All locations in the test range are checked again, in the same manner. This process continues until the testword has been stored once in every location of the test range. This completes PASS 1. Then, the background word and the test word are swapped (BACKGROUND = X'5555, TESTWORD = X'AAAA) and the entire test is repeated for PASS 2. This test tests for "crosstalk" between memory locations. Note that the test time is proportional to the square of the range to be tested. Testing a 2K range takes about four times as long as testing a 1K range.

The diagnostics are broken into ADDRESS/WORD/BIT tests on 0/4K, 4/16K, A000/AFFF, and DC00/DFFF. The GALPAT tests are broken into 1K increments. The parameter routine prompts for all required inputs: addresses to be tested, tests to be run, and the mode in which the tests are to be run. The tests allowed are ADDRESS (A), WORD (W), BIT (B), GALPAT (G), or ALL (CR).

The modes allowed are CONTINUOUS (C) and HALT (H).

The CONTINUOUS (C) MODE continues testing until interrupted by a keyboard entry. If an error is encountered the error is reported and the next block of memory is then tested. If no errors occur, then the block tested is reported and the next block is tested.

The HALT (H) MODE tests the memory until there is an error, in which case the error is reported and the test is terminated, or until the entire requested test range is tested, in which case the addresses are reported and the test halts.

To call MDIAG, type:

```
X>@MDIAG

MDIAG, REV A, DATE
M>
```

MDIAG is then ready to accept one of the commands described below.

The commands accepted by this program are:

PARAMETER — This command gets all the parameters required to run this program. The user is prompted for the type of input required and illegal responses are rejected.

RUN — This command runs the tests as specified by the input to the PARAMETER command.

Examples:

```
M>PA

ADDRESS RANGES 0/3FFF, A000/AFFF, DC00/DFFF
MAY BE SPECIFIED 0/AFFF OR 100/FFFF ETC.

ADDRESS RANGE TO BE TESTED? (CR = ALL) 0/3FFF

TESTS? (CR = ALL) A, B, W

MODE FOR RUNNING TESTS, C = CONTINUOUS, H = HALT

MODE (CR = H) C

M>RUN

ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 0000/0FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 0000/0FFF
CONTINUE TEST? (Y/N, CR = YES)? CR
  (keyboard interrupt)

ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF
CONTINUE TEST? (Y/N, CR = YES)? N

M>
```

This example ran the ADDRESS, WORD, and BIT tests on the system program memory space.

```
M>PA

ADDRESS RANGES 0/3FFF, A000/AFFF, DC00/DFFF
MAY BE SPECIFIED 0/AFFF OR 100/FFFF ETC.

ADDRESS RANGE TO BE TESTED (CR = ALL) A000/A080

TESTS TO BE RUN
A = ADDRESS, B = BIT, W = WORD, G = GALPAT

TESTS? (CR = ALL) CR

MODE FOR RUNNING TESTS, C = CONTINUOUS, H = HALT

MODE? (CR = H) C

M>RU

ADDRESS, WORD, BIT TEST(S) PASSED AT A000/A080
GALPAT BACKGROUND ERROR TEST FAILED AT A010
DATA SHOULD BE AAAA/DATA IS AAEA
ADDRESS, WORD, BIT TEST(S) PASSED AT A000/A080
CONTINUE TEST? (Y/N, CR = YES)? N

M>
```

This example runs all tests on addresses A000/A080 reporting pass/fail information until a keyboard interrupt. (NOTE: A KEYBOARD INTERRUPT IS ONLY TESTED DURING A MESSAGE OUTPUT.)


**National Semiconductor Corporation**
2900 Semiconductor Drive
Santa Clara, California 95051
Tel: (408) 737-5000
TWX: (910) 339-9240

**National Semiconductor**
District Sales Office
345 Wilson Avenue, Suite 404
Downsview, Ontario M3H 5W1
Canada
Tel: (416) 635-7260

**Mexicana de Electronica**
Industrial S.A.
Tlacoquemecatl No. 139-401
Esquina Adolfo Prieto
Mexico 12, D.F.
Tel: 575-78-68, 575-79-24

**NS Electronics Do Brasil**
Avda Brigadeiro Faria Lima 844
11 Andar Conjunto 1104
Jardim Paulistano
Sao Paulo, Brasil
Telex: 1121008 CABINE SAO PAULO

**National Semiconductor GmbH**
Eisenheimerstrasse 61/2
8000 München 21
West Germany
Tel: 089/9 15027
Telex: 522772

**National Semiconductor (UK) Ltd.**
301 Harpur Centre
Horne Lane
Bedford MK40 1TR
United Kingdom
Tel: 0234-47147
Telex: 826 209

**National Semiconductor Benelux**
789 Ave. Houba de Strooper
1020 Bruxelles
Belgium
Tel: (02) 4783400
Telex: 61007

**National Semiconductor Ltd.**
Vodroffsvej 44
1900 Copenhagen V
Denmark
Tel: (01) 356533
Telex: 15179

**National Semiconductor**
Expansion 10000
28, Rue de la Redoute
92 260 Fontenay-aux-Roses
France
Tel: (01) 660-8140
Telex: 250956

**National Semiconductor S.p.A.**
Via Solferino 19
20121 Milano
Italy
Tel: (02) 345-2046/7/8/9
Telex: 332835

**National Semiconductor AB**
Box 2016
12702 Skarholmen
Sweden
Tel: (08) 970190
Telex: 10731

**National Semiconductor**
Calle Nunez Morgado 9
Esc. DCHA. 1-A
Madrid 16
Spain
Tel: (01) 215-8076/215-8434
Telex: 46642

**National Semiconductor Switzerland**
Alte Winterthurerstrasse 53
Postfach 567
CH-8304 Wallisellen-Zürich
Tel: (01) 830-2727
Telex: 59000

**NS International Inc., Japan**
47F Shinjuku Center Building
1-25-1 Nishishinjuku, Shinjuku-ku
Tokyo, Japan
Tel: (04) 355-5711
TWX: 232-2015 JSCJ-J

**National Semiconductor (Hong Kong) Ltd.**
8th Floor,
Cheung Kong Electronic Bldg.
4 Hing Yip Street
Kwun Tong
Kowloon, Hong Kong
Tel: 3-411241-8
Telex: 73866 NSEHK HX
Cable: NATSEMI

**NS Electronics Pty. Ltd.**
Cnr. Stud Rd. & Mtn. Highway
Bayswater, Victoria 3153
Australia
Tel: 03-729-6333
Telex: 32096

**National Semiconductor (Pty.) Ltd.**
10th Floor
Pub Building
Somerset Road
Singapore 0923
Tel: 7379338/7379339
Telex: NAT SEMI RS 21402

**National Semiconductor (Taiwan) Ltd.**
Rm. B, 3rd Floor
Ching Lin Bldg.
No. 9, Ching Tao E. Road
P.O. Box 68-332 or 39-1176 Taipei
Tel: 3917324-6
Telex: 22837 NSTW
Cable: NSTW TAIPEI

**National Semiconductor (Hong Kong) Ltd.**
Korea Liaison Office
6th Floor, Kunwon Bldg.
1-2 Mookjung-Dong
Choong-Ku, Seoul
C.P.O. Box 7941 Seoul
Tel: 267-9473
Telex: K24942




DA-FL5M50/Printed in U.S.A.

National Semiconductor

Semiconductor
Division

2900 Semiconductor Drive
Santa Clara, California 95051
(408) 737-5000

Cable: NATSEMICON
Telex: 346353
TWX: 910 339 9240

Dear Customer:

If the need ever arises to ship your COP-400PDS to another location, there are certain shipping precautions that should be employed to ensure the safe arrival of your system. These precautions are as follows:

- Remove the cover of the COPs system and check to see if there is a tie wrap around the card cage. If none is visable, use the attached instructions to secure your own tie wraps around the card cage. If no tie wraps are available, remove the cards, wrap in anti-static bags, and ship separately packed in foam.

- Remove the keys from the lock and tape to the top of the system.

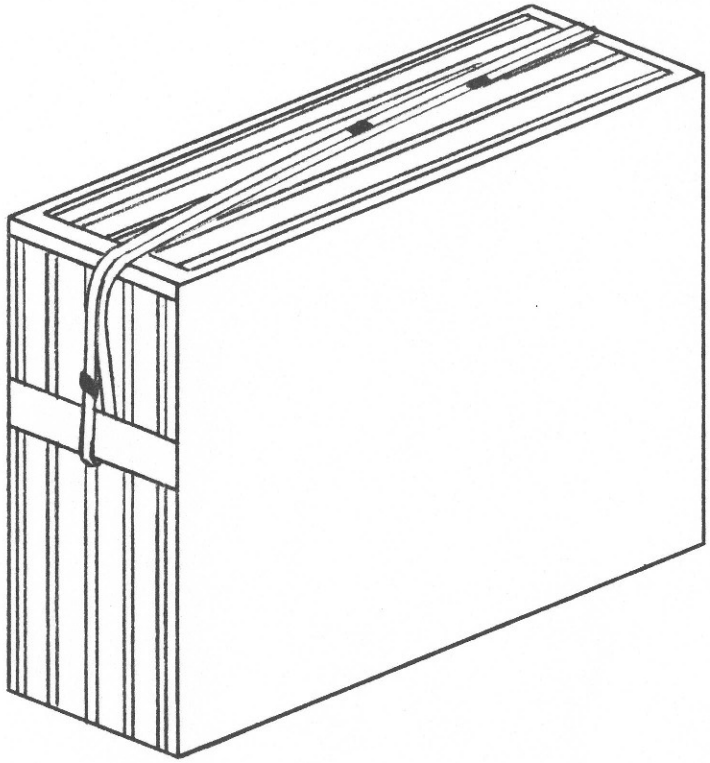
- Open the door to the floppy disc drive and insert a piece of foam or other material to prevent any movement.

- If the original box is no longer available, use a box slightly larger that the system so that the potential impact planes can be covered with two inches of Ethafoam$^{tm}$ or another form of semi-rigid packaging. We advise against the use of popcorn type filler due to its tendency to compress.

If you have any questions on shipping your COP-400PDS, please call the Microcomputer Service Department at (408)737-6270.

M. Antuna
Microcomputer
Service Manager

422305548-001
Rev. A



COPS CARD CAGE

1. ATTACH 3 OR 4 LARGE SIZE TIE WRAPS TOGETHER
2. LOOP AROUND CARD CAGE BRACKETS AS SHOWN
3. TIGHTEN TILL SNUG
4. CLIP OFF LOOSE TIE WRAP ENDS