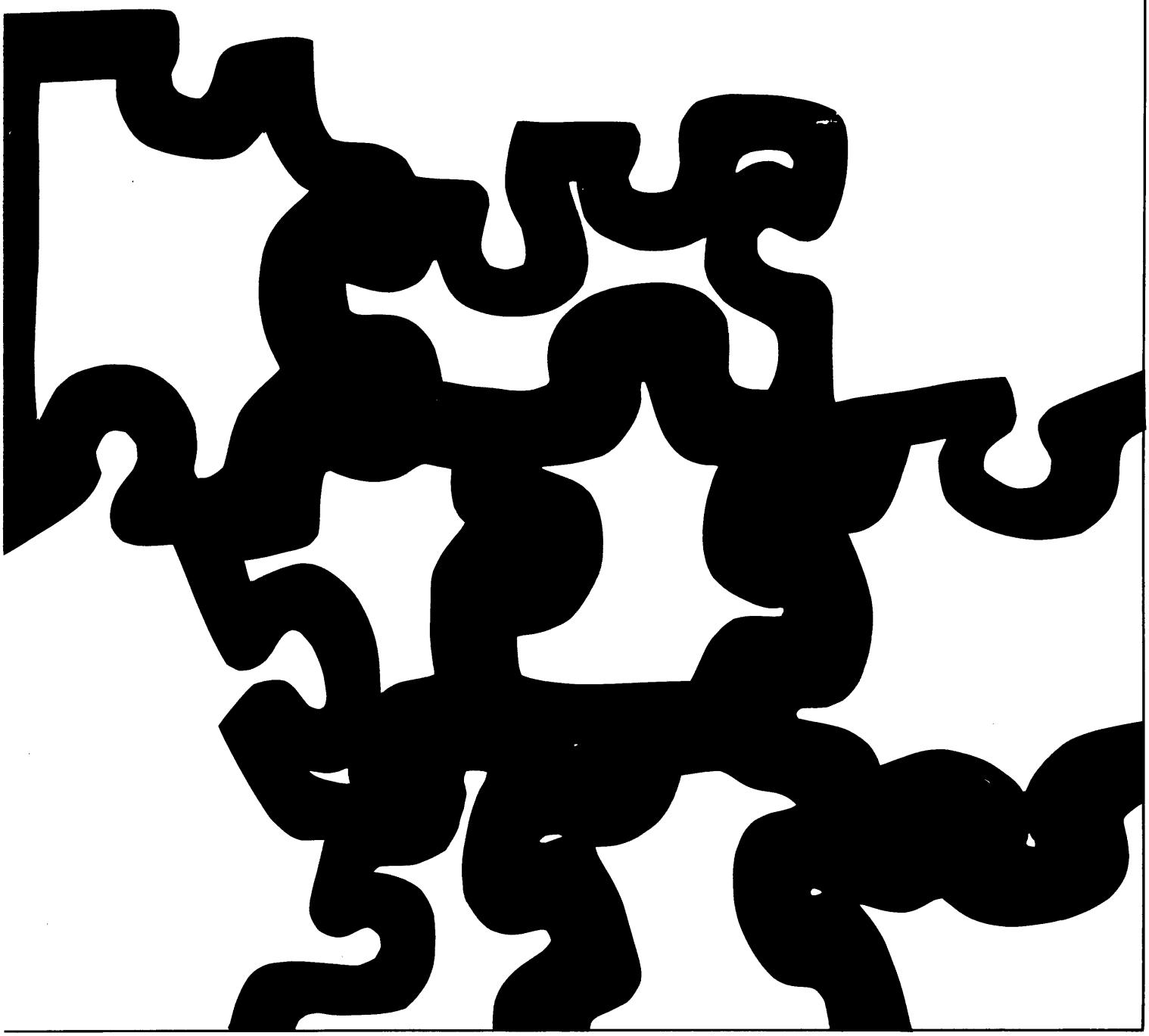
Design Engineer's Reference Series

Interface Designer's Manual for NOVA® and ECLIPSE® Line Computers



DataGeneral

Interface Designer's Manual for NOVA® and ECLIPSE® Line Computers

NOTICE

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS DOCUMENT OR THE INFORMATION CONTAINED IN IT, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

Ordering No. 014-000629

Rev. 01, September 1981

CONTENTS

Chapter 1
INTRODUCTION

Interfacing

One of the most valuable aspects of the modern digital computer is the variety of peripherals to which it can be connected, or interfaced. In Data General's computers, interfaces stand between the devices they control and the central processor, communicating with the peripherals individually and with the central processor over an *I/O bus*, or a set of wires carrying signals in parallel to all interfaces. The timing and functions of the I/O buses of Data General's NOVA and ECLIPSE computers are similar; boards are physically interchangeable, so that it is possible to build an interface to operate with both computer lines. (For those wishing to interface to the Microproducts' series of computers, see *Microproducts Hardware Systems Reference* (DGC No. 014-000636).)

It is easy to interface to Data General's computers. An interface can be built on a 15-inch square printed circuit board, which allows even large interfaces to be reliably constructed, with a minimum of off-board connections. Each of the control lines on the I/O bus is a dedicated line (used for a single function) with no additional timing signals needed. Thus, virtually no decoding of these control signals by the interface is necessary. Because all the control lines are dedicated, the bus is modular, and only the control lines corresponding to the implemented functions need be used. The bus is completely TTL-compatible and signals may be both transmitted and received using standard integrated circuit components. Data General also makes available general-purpose interface boards that simplify the job of designing and building an interface.

Scope of this Manual

The purpose of this manual is to describe the structure of the I/O bus and provide information for designing and building an interface assembly which can be used on NOVA and ECLIPSE lines. While very little knowledge of the input/output architecture of the computer is needed to use a peripheral sold by Data General Corporation, the design of I/O equipment requires a much greater understanding of this architecture. This manual discusses the I/O bus structure (it supports programmed I/O and data channel transfers only), explains specific functions, and provides information for designing and building an interface assembly. The reader should have a working knowledge of digital circuits and some experience with digital computers.

The manual is divided into chapters as follows:

Chapter 1	introduces this manual, defines terms and cites related documentation.
Chapter 2	covers the entire input/output facility from a programming perspective. It serves as an introduction to the facilities available and is primarily intended for a reader who is unfamiliar with the Data General input/output facilities.
Chapter 3	looks at the input/output facilities from the viewpoint of the I/O bus. It describes the various I/O functions and the bus signals that are used for each.
Chapter 4	considers the important electrical characteristics of the I/O bus.
Chapter 5	discusses the factors involved in packaging an interface and connecting it with the remainder of the computer system.
Chapter 6	provides information on the interfacing boards available from Data General.
Appendices	provide a number of reference tables for information about device codes, timing problems and character codes.

Overview of the Computer Lines

This manual describes how to interface to two of the lines of computers Data General manufactures: the NOVA line and the ECLIPSE line. These two lines differ primarily in their instruction sets, with the ECLIPSE computers featuring an expanded version of the set used by NOVA computers. The NOVA computers have a fixed instruction set to perform memory reference, arithmetic/logic and input/output functions. The ECLIPSE computers have an expanded instruction set which varies among the computer models. ECLIPSE instruction sets can contain instructions suited to business and scientific environments and, with the writeable control store feature, can include custom instructions. A full description of the instruction sets can be found in the programmer's reference manual for each computer model.

Definition of Terms

A peripheral generally consists of several units, a device, a controller and, sometimes, an adapter. The device, called a drive, a transport or a terminal, is the unit with which information is read, written, stored, or processed. For example, a terminal's keyboard *reads* information; a plotter *writes* information; a magnetic tape transport *stores* information; and an A/D converter *processes* information.

The controller is the interface between the computer and the device, interpreting commands from the computer to the device and passing information between them. Thus, a moving-head disc controller can translate the track address received from the computer into positional commands for the disc drive's access mechanism. Once the access mechanism positions the read/write heads, the controller translates the data words it receives from the computer into the sequence of bits required by the disc drive.

The adapter is an additional unit required by some peripherals in order to complete the communications link between the device and the controller.

The communications channel through which information passes between the computer and the controllers is called the Input/Output (I/O) bus. Since this bus is shared by all the controllers as well as by the CPU, it is, by necessity, a half-duplex bus; i.e., only one operation can occur at any time. The direction of all information transfers on the I/O bus is defined relative to the computer. *Output* always refers to moving information from the computer to a controller; *input* always refers to moving information from a controller to the computer.

Types of Information

The information transferred between a computer and a controller can be classified into three types: status, control, and data. Status information tells the computer about the state of the peripheral: is it busy?, is it ready?, is it operating properly? Control information is transferred by the computer to the controller to tell the peripheral what to do. Data is the information which originates from, or is sent to, the device during reading, writing, storing, or processing.

Types of Information Transfers

Information can be transferred between the computer and a peripheral in one of two ways: under direct program control, or under data channel control.

An information transfer occurring under direct program control moves a word or part of a word between an accumulator in the CPU and a register in the controller. A *word* is a 16-bit unit of data. This type of transfer occurs when an appropriate I/O instruction is executed in the program.

An information transfer under data channel control generally moves a block of data, one word at a time, between the computer's memory and the device, through a register in the controller. The block of data is transferred automatically via the data channel once the program, using I/O instructions, sets up the transfer for a particular peripheral.

Direct Program Control

Direct program control of information transfers, also called *programmed I/O*, is a way of transferring single words or parts of words to or from peripherals. Among the peripherals which transfer data in this way are terminals, paper tape readers and punches, card readers, line printers, and plotters. Since the data moves through an accumulator, it is readily available to the program for manipulation or decision making. In the case of input, for example, the program can decide whether to read another word or character based on the value of the word or character just read.

However, because at least one instruction--and most likely several, since the information must be stored in memory--must be executed for each character or word transferred, programmed I/O is slower and generally used only for peripherals which do not have to transfer large quantities of information quickly.

Data Channel Control

Some peripherals, such as disc and magnetic tape subsystems, are used to store large blocks of data. In order to reduce the amount of program overhead required, these blocks are transferred under data channel control. The commands used to set up the data channel transfer are assembled in accumulators and are transferred to the controller under direct program control. The block of data is then automatically transferred between memory and the controller via the data channel.

Once the program sets up and initiates the data channel transfer for a block of data, it does not have to take further action, and can proceed with other tasks while the block transfer is taking place. Each time the controller is ready to transfer a word from the block, it requests access to memory. When access is granted, the word is transferred. Because multiple instructions do not have to be executed for each word transferred, block transfers can occur at high rates, in some cases at more than a million words per second.

Program Interrupt Facility

When transferring information under either direct program control or data channel control, the program must be able to determine when the transfer is complete, so that it can start a new transfer, or proceed with a task that depends on the transfer just completed. Peripherals have status flags which can provide the program with this needed information. The I/O instruction set allows the program to check the status of these flags and make decisions based on the results of the checks. However, these status checks are time consuming, and to avoid the necessity of repeating them, all DGC computers have a program interrupt facility.

INTRODUCTION

The program interrupt facility provides a peripheral with a convenient means of notifying the processor that it requires service by the program. This is accomplished by allowing the peripherals to interrupt normal program flow on a priority basis. When a peripheral completes an operation, or encounters a situation requiring processor intervention, it can request a program interrupt from the processor. The processor honors such a request by interrupting the program in process, saves the address where the interruption occurred, and transfers control to the interrupt handling routine. The interrupt handling routine can identify which peripheral requires service and transfer control to the service routine for that peripheral. After servicing the peripheral, the routine can restore the system to the state it was in when the interrupt occurred.

For computer systems which require large amounts of I/O to many devices, a priority structure with as many as 16 levels can be established. This structure can be set up to provide rapid service to those devices which are crucial to the efficient operation of the computer system; the less critical devices are serviced as efficiently as possible. The priority interrupt structure, like the rest of the program interrupt facility, is under direct control of the program.

Chapter 2
INPUT/OUTPUT PROGRAMMING

I/O Instruction Set

Information transfers between the computer and the various peripherals are governed by the program by means of eight instructions, which constitute the I/O instruction set. These instructions allow the program to communicate with the peripherals' controllers and to control the program interrupt facility. This chapter covers only the I/O instructions necessary for these purposes; additional I/O instructions for special processor functions and options are fully described in the Programmer's Reference Manuals for the particular computer model. The chapter is meant as an introduction to the instruction set for those who have no experience with Data General's I/O system.

The effects of specific I/O instructions necessarily depend on the peripherals to which they are addressed. However, the general functions provided by the I/O instructions (loading and reading registers, issuing control signals, and testing flags) are the same for all peripherals; different peripherals merely use the available functions in different ways. In order to understand the general functions performed by the I/O instructions and how these functions are typically used by peripheral controllers, some knowledge of the architecture of a peripheral controller is required.

The Typical Controller

From the point of view of the program, a peripheral controller operates as a collection of data registers, control registers and status flags, with which communications are established. With these registers and flags, the program can route data between the computer and the device and monitor the operation of the device, as well.

The distinction made here between registers and flags is generally one of information content. A flag contains a single bit of information, while a register is made up of a number of bits. Groups of contiguous bits in a register which convey a single *piece* of information are referred to as *fields*. For example, in one of the magnetic tape controller's registers, bits 13-15 act together as a control field to select one of the eight possible tape transports in the subsystem.

The paragraphs below describe only the basic components of a typical controller. Other chapters will describe the additional structure required for a peripheral using the program interrupt facility or the data channel. What follows is meant only to typify the workings of a controller, since each controller is tailored to the specific device it controls, so that not all will fit the model given here.

The registers in a controller may be divided into three types according to the kind of information that is stored in them: there are data registers, control registers, and status registers.

Data registers (or data buffers) store data in the controller as it passes between the device and the computer. These buffers are needed because the computer and the device usually operate at different speeds. Since the operation of nearly all peripherals involves the transfer of a word, or part of a word, of data between the computer and the device, nearly all periherals controllers contain a data buffer. In the case of peripherals that transfer data under direct program control, the data buffer is directly accessible to the program. Data is transferred between the register in the controller and an accumulator in the central processor by an I/O instruction. In the case of a peripheral that transfers data under data channel control, the data is transferred between the register in the controller and memory. Data buffers in controllers that use the data channel need not be--and usually are not--accessible to the CPU through programmed I/O.

Control registers allow the program to supply the controller with information necessary for the operation of the device, such as drive or transport numbers, data block sizes, and command specification. A unit of control information is called a *control parameter*. Control parameters typically allow the program to select one of a number of peripheral units in a subsystem, the operation to be performed, and the initial values for flags and counters in the controller. The program specifies control parameters to the controller with an I/O instruction wherein the desired parameters are coded into the appropriate fields of the accumulator used in the transfer.

Status registers are used to indicate to the program the state of the peripheral. They consist primarily of status flags, but can also contain control parameters. The control parameters contained in status registers are commonly those which change during the operation of the peripheral. These are therefore important to the program, which must check on the progress of the peripheral's operation. For example, a program transferring consecutive sectors of

information to or from a disc in a single operation can read the current sector address and sector count during the operation to determine how far the operation is from completion. Status flags are set by the controller to indicate error conditions or inform the computer of the basic state of the peripheral.

The classification of controller registers into the three types described above is only a general one. A register may contain more than one type of information. This happens most often in a register that serves as a control register when loaded by the program and as a status register when read by the program. The disc sector address/sector counter register mentioned in the preceding paragraph is an example of such a combined control and status register.

The Busy and Done flags are the two fundamental flags in a controller and they serve a dual purpose.

Together they denote the basic state of the peripheral and can be tested by the program to determine that state. In addition, the program can manipulate these flags in order to control the operation of the peripheral. To place the peripheral in operation, the program sets the Busy flag to 1. The Busy flag remains in this state for the duration of the operation, indicating that the peripheral is in use and should not be disturbed by the program. When the peripheral completes its operation, the controller sets the Busy flag to 0 and the Done flag to 1 to indicate this fact. The setting of the Done flag to 1 can be used to trigger a program interrupt. Whether a program interrupt occurs depends on the state of the interrupt facility. However, no matter what the current state of the interrupt facility is, no interrupt can occur for that peripheral until its Done flag is set to 1. Therefore, the setting to 1 of the Done flag must *initiate a program interrupt request*. At this point, the program can either start the next operation by setting the Done flag to 0 and the Busy flag to 1, or it can idle (clear) the peripheral by setting both flags to 0.

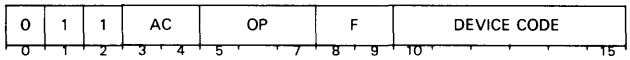
For a relatively simple peripheral, the Busy and Done flags alone may furnish enough status information to allow the program to service the peripheral adequately. However, a more complex peripheral will generally require additional status flags to specify its internal operating conditions more completely to the program. The difference between these additional status flags and the Busy and Done flags is that the Busy or Done flags may be tested directly with a single I/O instruction while any other status flag requires that its value first be read into an accumulator from the status register. Each status flag is assigned by the controller to one of the 16 available bit positions in a status register. The program may then perform any test it requires on the status word after it is read.

Status flags which indicate errors or malfunctions in the operation of a peripheral are termed *error flags*. Two types of error flags can be characterized, according to their effect on the operation of the peripheral when they are set. The first, or passive, type is merely set by the controller in the course of the operation when the associated error occurs. No immediate indication of this type of error is given to the program, and the operation is allowed to continue to completion. The second, or active, type of error flag is set by the controller when the program attempts to start an operation which is not allowed. In this case, the operation never begins and the Done flag is set to 1 immediately to notify the program. This type of error flag is used to prevent a severe and probably irrecoverable error from occurring. In either case, the program must respond, error or not, when it notices that a peripheral is *done*. It need only check the appropriate error flag (or flags) before assuming that the operation it initiated has been satisfactorily completed.

For example, among its many status flags, the controller for magnetic tape transports contains error flags to indicate parity errors and illegal operations. During a read operation, when a character is read with incorrect parity, the Parity Error flag is set to 1. No immediate notification of the error is given to the program, and the read operation is allowed to finish. The parity error can be detected at the completion of the operation, when the program should check for errors. At this time appropriate action can be taken, such as trying to read the misread record of tape again, or printing an error message on the console terminal. The Illegal flag, on the other hand, which is set when an illegal operation is attempted, prevents the operation from starting. The controller immediately sets both the Done and Illegal flags to 1 to notify the program. Illegal operations for a magnetic tape transport include writing on a tape that is write-protected and spacing backwards when the tape is at the beginning of tape marker.

Instruction Format

The general format of the I/O instructions is shown below.



Bits 0-2 are 011_2 and identify this as an I/O instruction; bits 3-4 specify an accumulator; bits 5-7 contain the operation code; bits 8-9 specify a flag control function or test condition; and bits 10-15 specify the code of the device.

Device Code Field

Bits 10-15 in an I/O instruction select the peripheral that is to respond to the instruction. The instruction format thus allows for 64_{10} device codes, numbered $0\text{-}77_8$. In all computers, device code 0 is not assigned to any peripheral, and device code 77_8 is used to implement a number of specific processor functions, such as controlling the program interrupt facility. Depending on the computer, a number of other specific device codes are reserved for processor options or features. The remaining device codes are available for referencing peripherals. Many of these codes have been assigned by Data General Corporation to standard peripherals, and the assembler recognizes convenient mnemonics for these codes. The list of the standard device code assignments and their associated mnemonics is given in Appendix A.

Flag Control Field

The Busy and Done flags are either manipulated or tested by the control functions or test conditions specified in bits 8 and 9 of the I/O instructions. In those instructions which allow flag manipulation, bits 8 and 9 are referred to as the F field. The flag control commands available, along with the associated mnemonics and bit configurations and the functions typically performed, are shown in Table 2.1.

F Field	Command	Mnemonic
00	(NONE) (OMITTED)	None
01	START S	Start the peripheral by setting the Busy flag to 1 and the Done flag to 0.
10	CLEAR C	Clear (idle) the peripheral by setting both the Busy and Done flags to 0.
11	PULSE P	Pulse the controller to achieve a special effect. The effect, if any, depends on the peripheral.

Table 2.1 Flag Control Functions

In the I/O instruction which allows flag testing, bits 8 and 9 are referred to as the T field. The bit configurations, mnemonics, and test conditions they select are shown in Table 2.2.

T Field	Mnemonic	Next Instruction is Skipped if:
00	BN	Busy flag is 1 (non-zero)
01	BZ	Busy flag is 0 (zero)
10	DN	Done flag is 1 (non-zero)
11	DZ	Done flag is 0 (zero)

Table 2.2 Test Functions

Two important features of the I/O instruction set result from the nature of the flag control field. First, because the flag control field is separate from the operation code field, a single I/O instruction can both transfer information between the controller and the computer and simultaneously control the operation of the peripheral. Secondly, the use of the flag control field as a T field allows the direct testing of a controller's Busy or Done flag in a single instruction, so that quick decisions based on the basic state of the peripheral can be made by the program.

Operation Code Field

The 3-bit operation code field selects one of the eight I/O instructions. In two of these instructions, no information transfer is specified; instead, bits 8 and 9 may specify either a control function or a flag test condition as described previously. The remaining six instructions involve an information transfer between the computer and the designated peripheral controller and may also specify a control function to be performed after the information transfer has been completed. The program can, therefore, access up to six registers in any one controller. Up to three of these six registers are output registers which can be loaded by the program with either data or control information. The other three are input registers from which the program can read either data or status information. Frequently, two different I/O instructions, one input and one output, reference the same register in a controller. However, this is not in any way required by the nature of the I/O instruction set.

In order to give names and mnemonics to the I/O instructions in their general form, the registers in a peripheral controller which are accessible to the program are referred to with letter designations. The three input registers are called the *A input buffer*, the *B input buffer*, and the *C input buffer*. Similarly, the three output registers are called the *A output buffer*, the *B output buffer*, and the *C output buffer*. Thus, for example, to read data from a peripheral controller's A input buffer, a *Data in A* instruction, with mnemonic **DIA**, is issued to that peripheral.

The eight operation codes, their associated mnemonics, and the instructions specified are shown in Table 2.3.

Operation Code Field	Mnemonic	Instruction
000	NIO	No Input or Output but perform the flag control function specified.
001	DIA	Read Data Into the computer from the A input buffer.
010	DOA	Write Data Out from the computer to the A output buffer.
011	DIB	Read Data Into the computer from the B input buffer.
100	DOB	Write Data Out from the computer to the B output buffer.
101	DIC	Read Data Into the computer from the C input buffer.
110	DOC	Write Data Out from the computer to the C output buffer.
111	SKP	SKiP the next instruction if the test selected for the Busy or Done flag is true.

Table 2.3 The eight I/O instructions

Accumulator Field

Bits 3 and 4 in an I/O instruction select one of the central processor's four accumulators: AC0, AC1, AC2, AC3. In those instructions which involve an information transfer between the processor and a peripheral controller, the specified accumulator either furnishes the information for an output transfer or receives the information in an input transfer. In the two I/O instructions which do not involve an information transfer, set bits 3 and 4 in these instructions to 0.

Instructions

A number of abbreviations and symbols are used in this manual to aid in describing how an instruction may be coded in assembly language. Abbreviations used are as follows:

AC or *ac*	accumulator
F or *f*	flag control command
T or *t*	flag test command
device	device code or mnemonic

The following conventions are used to indicate a specific type of instruction field:

[] *//*	Indicates that the enclosed symbol (e.g., *[f]*) is an optional operand or mnemonic.
BOLD	Indicates the operand or mnemonic printed in boldface is used exactly as shown. For example, code the mnemonic for the *Data In B* instruction: **DIB**.
italic	Indicates each operand or mnemonic in italics must be replaced with a number or symbol that provides the assembler value needed for that item.

When describing the format of a word involved in an information transfer between the computer and a controller, the various fields and bits in the word are labeled with names descriptive of their functions.

Data In A

DIA*[f]* *ac,device*
Places the contents of the A input buffer in the specified AC of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits transferred depends on the controller. Most controllers set unused bits to 0.

Data In B

DIB*[f]* *ac,device*
Places the contents of the B input buffer in the specified AC of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits transferred depends on the controller. Most controllers set unused bits to 0.

Data In C

DIC*[f]* *ac,device*
Places the contents of the C input buffer in the specified AC of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits transferred depends on the controller. Most controllers set unused bits to 0.

Data Out A

DOA*[f]* *ac,device*
Places the contents of the specified AC in the A output buffer of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits loaded into the buffer depends on the controller. The contents of the specified AC remain unchanged.

Data Out B

DOB*[f]* *ac,device*
Places the contents of the specified AC in the B output buffer of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits loaded into the buffer depends on the controller. The contents of the specified AC remain unchanged.

Data Out C

DOC*[f]* *ac,device*
Places the contents of the specified AC in the C output buffer of the designated controller. After the data transfer, the controller's Busy and Done flags are set according to the function specified by F. The number of bits loaded into the buffer depends on the controller. The contents of the specified AC remain unchanged.

No I/O Transfer

NIO*[f]* *device*
Sets the Busy and Done flags in the controller of the designated device according to the function specified by F.

I/O Skip

SKP *[t]* *device*
Skips the next sequential instruction if the test condition specified by *t* is true for the designated controller. The contents of all the accumulators and the Busy and Done flags for the specified device remain unchanged.

Program Interrupt Facility

When a peripheral completes an operation, the controller sets its Done flag to 1 to indicate that program service is required. The program can test the state of the Done flag repeatedly with *I/O Skip* instructions to determine when this occurs. However, continual interrogation of the Done flag by the program is generally wasteful of computing time, especially when flag checks need to be done frequently in order to ensure that service is not delayed so long that the peripheral loses data. The program interrupt facility provides a peripheral with a convenient means of notifying the processor that service is required.

All peripherals which use the program interrupt facility have access to a single direct line to the processor, called the Interrupt Request Line, along which their requests for service are communicated. An interrupt request can be generated by a peripheral when the perpipheral's Done

flag is set to 1. The processor can respond to, or *honor*, an interrupt request by halting the normal flow of program execution and transferring control to an interrupt handling routine. The programmer can control which peripherals may request interrupts, and when the processor may start an interrupt, by manipulating a number of flags, which are distributed among the processor and the peripherals.

The operation of the program interrupt facility, as controlled by these flags, is described below. Following portions of this chapter detail the instructions used to control the program interrupt scheme, offer further suggestions for programming an interrupt handler, and explain the operation of the vector instruction, which allows the ECLIPSE computer to perform much of its interrupt processing automatically.

Operation

Control Flags

The operation of the program interrupt facility is governed by the Interrupt On flag (*ION*) in the central processor and by the Done and Interrupt Disable flags in each peripheral using the facility. By manipulating these flags, the program can choose to disregard interrupt requests altogether, or it can selectively ignore certain peripherals.

The major control flag for the program interrupt facility is the Interrupt On flag in the central processor. To enable the interrupt facility, the program sets *ION* to 1, which allows the processor to respond to interrupt requests transmitted to it along the Interrupt Request Line. Setting *ION* to 0 disables the entire interrupt facility, and causes the processor to ignore all interrupt requests.

ION is manipulated by the program exactly like a Busy flag for the central processor. A Start command in any I/O instruction directed to the CPU (device code 77_8) sets *ION* to 1; a Clear command in such an instruction sets *ION* to 0. (*ION* is also set to 0 at power-up, and when RESET occurs.)

Each controller for a peripheral using the program interrupt facility contains an Interrupt Disable flag which allows the program to stop interrupts from that peripheral. When a peripheral's Interrupt Disable flag is set to 1, it cannot make an interrupt request.

The Interrupt Disable flags of all peripherals are manipulated at once with a single I/O instruction. This instruction, *Mask Out* (**MSKO**), sets up the Interrupt Disable flags of all peripherals connected to the program interrupt facility according to a mask contained in the accumulator specified by the instruction. Each peripheral is assigned by its hardware to a bit position in the mask. (Mask bit assignments for standard peripherals are given in Appendix A.) When a *Mask Out* instruction is given, each peripheral's Interrupt Disable flag is set to the value of the assigned bit of the mask. At power-up, however, and when RESET occurs, all Interrupt Disable flags are set to 0.

Interrupt Requests

Interrupt requests by a peripheral are governed by its Done and Interrupt Disable flags. When a peripheral completes an operation, it sets its Done flag to 1, and this action initiates a program interrupt request. If its Interrupt Disable flag is 0, the request is communicated to the CPU. If the *ION* flag is 1, the processor has to honor the interrupt request as soon as it is able. If the Interrupt Disable flag is 1, the request is not communicated to the CPU; it is blocked until the Interrupt Disable flag is set back to 0.

The processor is able to interrupt the sequential flow of program instructions if all the following conditions hold:

1. The processor has just completed an instruction, or a data channel transfer occurring between two instructions.
2. At least one peripheral is requesting an interrupt.
3. Interrupts are enabled: that is *ION* is 1.
4. No peripheral is waiting for a data channel transfer; that is, there are no outstanding data channel requests. The data channel has priority over program interrupts.

When the processor finishes an instruction, it takes care of all data channel requests before it starts an interrupt; this includes any additional data channel requests that are initiated while data channel transfers are being made. When no more peripherals are waiting for data channel transfers, the processor starts an interrupt if *ION* is 1 and at least one peripheral is requesting an interrupt.

The processor starts an interrupt by automatically executing the following sequence:

1. It sets *ION* to 0 so that no further interrupts may be started.
2. It disables any address translation methods.
3. It stores the contents of the program counter (which points to the next instruction in the interrupted program) in location 0, so that a return to the interrupted program can be made after the interrupt service routine has finished.
4. It simulates a **JMP @1** instruction to transfer control to the interrupt service routine. Location 1 should contain the address of the routine, or the first part of an indirect address chain that points to the routine.

Servicing An Interrupt

The interrupt service routine (or handler) must save the state of the processor, identify which peripheral requires service, and service the peripheral.

Saving the state of the processor involves saving the contents of any accumulators that will be used in the interrupt service routine, saving the carry bit if it will be used, and saving the stack and frame pointers. The state of the processor must be saved so that it may be restored before the interrupted program is allowed to resume.

There are three ways in which the interrupt handler can identify which peripheral requires service.

1. On the NOVA and ECLIPSE lines, the interrupt handler can execute a polling routine. This routine is merely a sequence of *I/O Skip* instructions which test the states of the Done flags of all peripherals in use. With this method peripheral priorities are determined by the order in which the tests are performed. Note that the polling technique disregards the state of the Interrupt Disable flags. Peripherals which are masked out will be recognized if their Done flags are 1, even though these peripherals could not have caused the interrupt.
2. On the NOVA and ECLIPSE lines, the interrupt handler can issue an *Interrupt Acknowledge* instruction (**INTA**). This instruction reads the device code of the first peripheral on the I/O bus that is requesting an interrupt, into a specified accumulator. Note that with this method the Interrupt Disable flags are significant. Peripherals which are masked out cannot request an interrupt and, therefore, cannot respond to the *Interrupt Acknowledge* instruction.
3. On the ECLIPSE computer, the interrupt handler can issue a *Vector* instruction (**VCT**). This instruction determines which peripheral requires service in exactly the same way as the *Interrupt Acknowledge* instruction. However, the device code obtained is not placed in an accumulator but is used as an index into a table of addresses. Besides vectoring automatically to the correct peripheral service routine, the *Vector* instruction can perform other operations necessary to the handling of priority interrupts. Because the *Vector* instruction is available only on the ECLIPSE computer, and because its operation is relatively complex, it is described later.

After determining which peripheral requires service, the interrupt handler generally transfers control to a peripheral service routine. This routine performs the information transfer to or from that peripheral (if required) and either starts the peripheral on a new operation or idles the peripheral if it has no more operations pending.

When all service for the peripheral has been completed, either the peripheral service routine or the main interrupt handler should perform the following sequence to dismiss the interrupt:

1. Set the peripheral's Done flag to 0 to dismiss the interrupt request which was just honored. If this is not done, the undismissed interrupt request will cause another interrupt--this time incorrectly-- as soon as the interrupt handler finishes and attempts to return control to the interrupted program.
2. Restore the pre-interrupt states of the accumulators, the carry bit, the stack pointers, and address translation.
3. Set *ION* to 1 enable interrupts again.
4. Jump back to the interrupted program. (Usually a **JMP @0** instruction is given.)

The instruction that enables interrupts (usually **INTEN**) sets the Interrupt On flag to 1, but the processor does not allow the state of the *ION* flag to change to 1 until the next instruction begins. Thus, after the instruction that turns interrupts back on, the processor always executes one more instruction (assumed to be the return to the interrupted program) before another interrupt can start. The program must give this final return instruction immediately after enabling interrupts in order to ensure that no waiting interrupt can overwrite the contents of location 0 before they are used to return control to the interrupted program.

The following diagram, Fig. 2.1, shows how normal program flow is altered during a program interrupt. The interrupt handler is shaded to indicate that this block of instructions is not interruptable, since the processor sets the *ION* flag to 0 to disable further interrupts when the interrupt occurs. Interrupts are not enabled again until the interrupt handler executes its *Interrupt Enable* instruction just prior to returning control to the interrupted program.

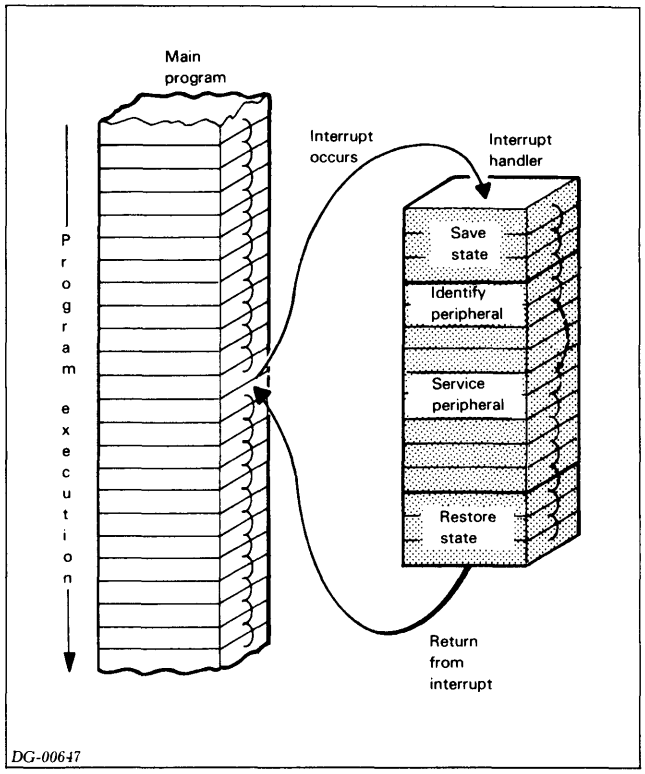


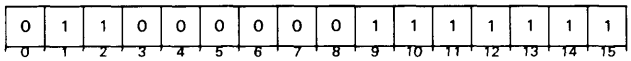
Figure 2.1 Program flow with interrupt

Instructions

The instructions which control the program interrupt facility use special device code 77_8 (mnemonic **CPU**). When this device code is used, bits 8 and 9 of the skip instructions test the state of *ION* and the power fail flag; in the other instructions these bits turn interrupts on or off by setting *ION* to 1 (Start command) or 0 (Clear command).

Interrupt Enable

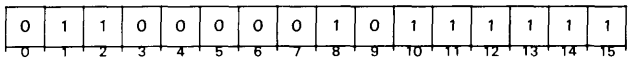
INTEN
NIOS CPU



Set the Interrupt On flag to 1 to allow the processor to respond to interrupt requests. If the Interrupt On flag actually changes state (from 0 to 1), the processor will execute one more instruction before it can start an interrupt. On the ECLIPSE computer, the processor will execute one more instruction before starting an interrupt even if the Interrupt On flag is already 1. However, if that instruction is one that is interruptible, an interrupt can occur as soon as the instruction begins to execute. The assembler recognizes the mnemonic **INTEN** as equivalent to **NIOS CPU**.

Interrupt Disable

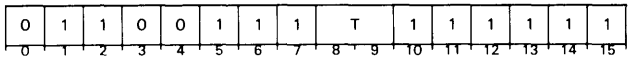
INTDS
NIOC CPU



Set the Interrupt On flag to 0 to prevent the processor from responding to interrupt requests. The assembler recognizes the mnemonic **INTDS** as equivalent to **NIOC CPU**.

CPU Skip

SKP,*t* **CPU**



If the test condition specified by *T* is true, the next sequential word is skipped.

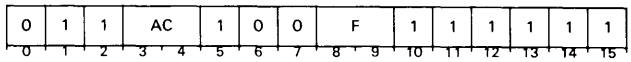
The *CPU Skip* instruction enables the programmer to make decisions based upon the value of the Interrupt On flag or the Power Fail flag. Which test is performed is based upon the value of bits 8-9 in the instruction. Bits 8-9 can be set by appending an optional mnemonic to the *CPU Skip* mnemonic. The optional mnemonics and their results are given in Table 2.4.

Class Abbreviation	Coded Character	Result Bits	Operation
t	BN	00	Tests for Interrupt On = 1.
	BZ	01	Tests for Interrupt On = 0.
	DN	10	Tests for Power Fail = 1.
	DZ	11	Tests for Power Fail = 0.

Table 2.4 CPU Skip Conditions

Mask Out

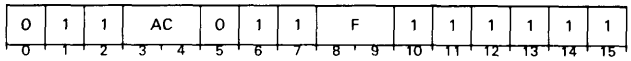
MSKO *ac*
DOB*[f]* *ac*,**CPU**



Set the Interrupt Disable flags in all peripherals according to the mask contained in the specified AC. (A 1 in a mask bit sets the flags in all peripherals assigned to that bit to 1, a 0 sets them to 0.) The Interrupt On flag is set according to the function specified by F. The contents of the specified AC remain unchanged. Mask bit assignments for standard peripherals are given in Appendix A. The assembler recognizes the instruction **MSKO** *ac* as equivalent to **DOB** *ac*,**CPU**.

Interrupt Acknowledge

INTA *ac*
DIB*[f]* *ac*,**CPU**

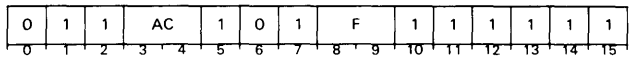


The peripheral with the highest priority interrupt request places its device code in bits 10-15 of the specified AC. Bits 0-9 are set to 0. After the data transfer, the Interrupt On flag is set according to the function specified by F. If no peripheral is requesting an interrupt, the specified AC is set to 0. The assembler recognizes the instruction **INTA** *ac* as equivalent to **DIB** *ac*,**CPU**.

> **NOTE:** *If* $F = 11_2$ *a* Vector *instruction is performed in ECLIPSE computers.*

I/O Reset

IORST
DIC*[f]* *ac*,**CPU**



Resets all peripherals connected to the I/O bus: sets their

Busy, Done, and Interrupt Disable flags to 0 and, depending on the peripheral, performs any other required initialization. After the peripherals' flags are altered, the Interrupt On flag is set according to the function specified by F. The assembler recognizes the mnemonic **IORST** as equivalent to **DICC** 0,**CPU** --that is, as the instruction defined here with F set to 10_2.

If the mnemonic **DIC** is used to perform this function, an accumulator must be coded to avoid assembly errors. Regardless of how it is coded, during execution the AC field is ignored and the contents of the specfied AC remain unchanged. At power-up and when RESET occurs, the processor performs the equivalent of an **IORST** instruction.

The assembler recognizes a number of convenient mnemonics for instructions that control the program interrupt, as shown in Table 2.5.

Mnemonic	Instruction	Mnemonic Equivalent	Octal Equivalent
INTEN	INTERRUPT ENABLE	NIOS 0,CPU	060177
INTDS	INTERRUPT DISABLE	NIOC 0,CPU	060277
MSKO ac	MASK OUT	DOB ac,CPU	062077
INTA ac	INTERRUPT ACKNOWLEDGE	DIB ac,CPU	061477
IORST	I/O RESET	DICC 0,CPU	062677

Table 2.5 Program Interrupt Mnemonics

To set up the Interrupt Disable flags according to the mask contained in AC2, give **MSKO** 2 or **DOB** 2,**CPU**.

However, there is one important difference between these special mnemonics and the standard ones: mnemonics for enabling and disabling interrupts cannot be appended to them. Thus, to set the Interrupt On flag to 0 while performing a *Mask Out* instruction using AC2, use **DOBC** 2,**CPU**.

Note that use of the mnemonic **IORST** sets the Interrupt On flag to 0. To set the flag to 1 while resetting the peripherals, use **DICS** 0,**CPU**.

Priority Interrupts

If the Interrupt On flag remains 0 throughout the interrupt service routine, the routine cannot be interrupted, and there is only one level of peripheral priority. All peripherals that have not been disabled by the program are, for the most part, equally able to request interrupts and receive interrupt service. Only when two or more peripherals are requesting an interrupt at exactly the same time is a priority distinction made. When this happens, priority is determined either by the order in which *I/O Skip* instructions are given or, if the *Interrupt Acknowledge* or *Vector* instruction is used, by the order of peripherals along the I/O bus. In a system with peripherals of widely differing speeds and/or service requirements, a more extensive priority structure may be required. The program interrupt facility hardware and instructions allow the program to implement up to 16 interrupt priority levels.

For example, suppose that a card reader and a terminal are being operated at the same time. While a card is being read, an interrupt is requested as each new column of data is available, and the program must read this data within 430 microseconds, typically, before it is overwritten in the Data Buffer by the data from the next column. If the terminal service routine takes 300 microseconds, card reader service will never be delayed longer than this, and a single-level program interrupt scheme will suffice.

However, this interrupt scheme will not work if the terminal service routine takes 600 microseconds, since a card reader interrupt request initiated soon after terminal service is begun will not be honored in time, and a column of data will be lost. In order to avoid losing data, the program interrupt scheme used must allow the card reader to interrupt the lengthy terminal service routine. This involves creating a two-level priority structure and assigning the card reader to the higher priority level.

In general, a multiple-level priority interrupt scheme is used to allow higher-priority peripherals to interrupt the service routines of lower-priority peripherals. A hierarchy of priority levels can be established through program manipulation of the Interrupt Disable Flags of all peripherals in the system. When the interrupt request from a peripheral of a certain priority is honored, the interrupt handler sets up the new priority level by establishing new values for the Interrupt Disable flags of all peripherals according to an appropriate *Interrupt Priority Mask* used with the *Mask Out* instruction. Peripherals whose interrupt Disable flags are set to 1 by the corresponding bit of this priority mask are *masked out*, or disabled, and are thereby regarded as being of lower priority than the peripheral being serviced. Before proceeding with the peripheral service routine, the Interrupt On flag is set to 1 so that the higher-priority peripherals may interrupt the current service routine.

Interrupt Priority Mask

The bit of the priority mask that governs the Interrupt Disable flag for a given peripheral is assigned to that peripheral by the hardware and cannot be changed by the program. Although lower-speed devices are generally assigned to higher-numbered mask bits, no implicit priority ordering is intended. The manner in which these priority levels are ordered is completely up to the programmer. By means of the priority mask the program can establish any desired priority structure, with one limitation: in the cases in which two or more peripherals are assigned to the same bit of the priority mask, these peripherals are constrained to be at the same priority level. When a peripheral causes an interrupt, a decision must be made whether to place all other peripherals which share the same mask bit with the interrupting peripheral at a higher or lower priority level. If a decision is made to mask out all peripherals which share that priority mask bit, the interrupting peripheral is also masked out.

Priority Interrupt Handler

A priority interrupt handler differs from a single-level interrupt handler in several ways. The handler must be *re-entrant*. This means that if a peripheral service routine is interrupted by another, higher priority peripheral, no information is lost which the handler will need to restore the state of the machine. The two items of information which should be saved, in addition to those saved by a single-level interrupt handler, are the return address (the contents of location 0) and the current priority mask. This information must be stored in different locations each time the interrupt handler is entered at a higher level. Doing this ensures that the necessary return information belonging to an earlier interrupt is not overwritten by a higher level interrupt. A common method of storing return information for a re-entrant interrupt handler is through the use of a push-down stack.

The interrupt handler (including the peripheral service routines) for a multi-level priority scheme should perform the following tasks:

1. Save the state of the processor, that is, the contents of the accumulators, the carry bit, location 0, the stack parameters, and the current priority mask.
2. Identify the peripheral that requested the interrupt.
3. Transfer control to the service routine for that peripheral.
4. Establish the new priority mask with a *Mask Out* instruction for that peripheral's service routine and store it in memory at the location reserved for the current priority mask at that level of interrupt.
5. Enable interrupts. Now, any peripheral not masked out can interrupt this service routine.
6. Service the peripheral that requested the interrupt.
7. Disable interrupts in preparation for dismissal of this interrupt level, so that no interrupts will occur during the transition to the next lower level.
8. Restore the state of the processor, including the former contents of the accumulators and the carry bit and reinstitute the pre-interrupt priority mask with a *Mask Out* instruction.
9. Enable interrupts.
10. Transfer control to the return address which was saved from location 0.

Fig. 2.2 is a simplified representation of program flow in priority interrupt environment. Shaded areas indicate non-interruptable chapters of instructions. Additional higher-priority interrupts could increase the depth of interrupts still further.

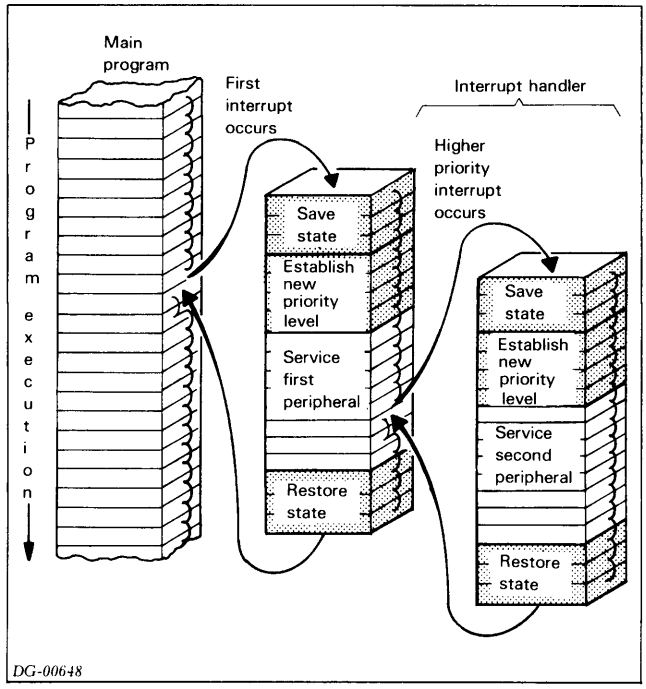


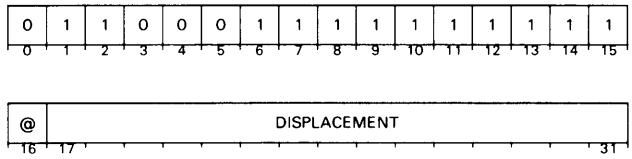
Figure 2.2 Program flow with priority interrupts

The Vector Instruction

The ECLIPSE line of computers incorporates an instruction which greatly reduces the burden of programming a priority interrupt system. Since this instruction is available only on the ECLIPSE line of computers, it is described separately below. In effect, the *Vector* instruction (**VCT**) can be used to perform the first five tasks listed above for the multilevel priority interrupt handler.

Vector On Interrupting Device Code

VCT *[@]displacement*



This instruction provides a fast and efficient method for transferring control from the main I/O interrupt handler to the correct interrupt service routine for the interrupting device. Bit 0 of the second word of the instruction is the *stack change bit* though labeled @ (the indirect sign). Bits 1-15 contain the address of a 64-word vector table. Vector table entries are one word in length and consist of a *direct* bit in bit 0 followed by an address in bits 1-15.

The *Vector* instruction performs an Interrupt Acknowledge function. It adds the device code return to the address of the vector table and fetches the vector table entry at that address. If the direct bit in the fetched vector table entry is 0, the instruction takes the address in bits

1-15 to be the address of the device handler routine for the interrupting device and control immediately transfers there by placing the address in the program counter.

If the direct bit is 1, the instruction takes the address in bits 1-15 of the vector table entry to be the address of the device control table (DCT) for the interrupting device. At this point, it examines the stack change bit. If the stack change bit is 0, it does not perform a stack change. If the stack change bit is 1, the instruction creates a new stack by placing the contents of memory location 6 in the stack limit, and the contents of memory location 7 in the stack fault. It then pushes the previous contents of memory locations 40_8-43_8 onto this new stack.

Device control tables must consist of at least two words. The first word of a DCT consists of a *push bit* in bit 0 followed by the address of the device handler routine for the interrupting device in bits 1-15. The second word of a DCT contains a mask that will be used to construct the new interrupt priority mask. Succeeding words in a DCT may contain information that is to be used by the device interrupt handler.

After the *Vector* instruction performs the stack change procedure, it fetches and inspects the first word of the DCT. If the push bit is 1, it pushes a standard return block onto the stack, with bits 1-15 of physical location 0 placed in bits 1-15 of the last word pushed. If the push bit is 0, the instruction does not push a return block.

Following this procedure, it places the address of the DCT in bits 1-15 of AC2 and sets bit 0 of AC2 to 0.

Next, the instruction pushes the current interrupt priority mask on the stack. It logically ORs the contents of the second word of DCT with the current interrupt priority mask and places the result in both AC0 and memory location 5. This constructs the new interrupt priority mask and places it in AC0 and the save location for the mask. A Mask Out function that also enables the interrupt system is now performed.

After establishing a new interrupt priority mask and enabling the interrupt system, the instruction transfers control to the device handler by placing bits 1-15 of the first word of the DCT in the program counter.

Use of the Vector Instruction

The *Vector* instruction is an extremely powerful instruction. Because of the impact of interrupt latency on overall system performance, and the impact of the *Vector* instruction on interrupt latency, you should understand this instruction very well before you use it.

The *Vector* instruction can operate in any one of five modes: A, B, C, D, and E. In general, as one goes through the modes, from A to E, the instruction performs more work, giving the user more power, but also taking more time to execute.

For all modes, the *Vector* instruction uses bits 1-15 of the second word to address the vector table. It performs an *Interrupt Acknowledge* instruction and adds the device code received to the address of the vector table and fetches the word at that location. At this point, the mode selection process begins.

Deciding which mode is to be used for execution is a function of the direct bit in the vector table entry, the stack change bit in the second word of the *Vector* instruction and the push bit in the first word of the DCT. The table below, Table 2.6, gives the relationship.

Direct	Stack Change	Push	Mode
0	X	X	A
1	0	0	B
1	0	1	C
1	1	0	D
1	1	1	E

Table 2.6 Vector Instruction Modes

X = Don't care

For mode A, the state of the stack change bit doesn't matter, because it is never checked.

Mode A is used when no time can be wasted in getting to the interrupt handler for a device. A device requiring mode A service would typically be a nonbuffered device with a very small latency time. Alternately, a real time process that must receive control immediately after an event could be serviced using mode A. The programmer pays for the speed realized through mode A by giving up the state saving and priority masking features of the other modes.

Modes B, C, D, and E are used to implement a priority interrupt structure. They all build a new priority mask and save the old priority mask before issuing a *Mask Out* instruction that enables the interrupt system. These modes differ in the amount of time and work that they devote to saving the state of the machine.

In a priority system, there are typically two types of processes: those that operate at *base* level, and those that do not. Base level is defined as operating with all levels of interrupt enabled and no interrupt processing in progress. Non-base level is defined as operating with some interrupt processing in progress. In general, the user programs operate at base level and the various interrupt handlers in the system operate at non-base level.

While a process is operating at base level, one of the first things that the supervisor program should do when it receives an interrupt is to change the stack environment. There are two reasons for this. First, the supervisor has no control over whether or not the user has defined a stack by placing meaningful information in the stack control words. Additionally, even if the user has initialized a stack, the supervisor has no control over the size of it. If the user

has defined a stack, but is very close to his stack limit, it would not be acceptable for a supervisor interrupt routine to fill the user's stack to overflowing. By using either mode D or E, the *Vector* instruction will change the stack environment and initialize a stack over which the supervisor has full control. At the same time, the *Vector* instruction will save the stack environment of the user so that it may be restored before control is returned to the user.

If an interrupt handler is already processing when another interrupt is received, then the stack environment has already been changed by the interrupt that occurred at base level and should not be changed again. For interrupts that occur at non-base level, modes B and C of the *Vector* instruction can be used.

The difference between modes D and E is the same as the difference between modes B and C: modes B and D do not push a return block onto the stack. While this makes them slightly faster than modes C and E, it complicates returning control to the interrupted program.

All modes of the *Vector* instruction can be combined in one vector table. Devices that require mode A service will have bit 0 set to 0 in their vector table entry. The other devices will have bit 0 set to 1 in their vector table entries, and the user can control their modes of service by setting the push bit in their DCT's.

Note the *Vector* instruction uses four locations in memory (locations 4-7). These locations contain the interrupt priority mask and the vector stack parameters. Location 4 contains the vector stack pointer. Location 5 contains the interrupt priority mask. Location 6 contains the vector stack limit. Location 7 contains the address of the vector stack fault handler.

Data Channel Facility

Peripherals which need to transfer large blocks of data quickly generally accomplish their data transfers via the data channel facility. The actual data channel transfers do not disturb the state of the processor, since the data is transferred directly between registers in the controller and memory. This greatly reduces the amount of program overhead in the form of executing I/O instructions and loading or storing data. In some CPUs, it increases the time required for program execution, however, since the processor pauses each time a word is to be transferred. After the transfer occurs, the processor continues. The program needs only to set up the peripheral for the transfer, and can then perform other unrelated tasks.

Table B.2 in Appendix B includes the maximum transfer rates for all types of transfer.

The data channel allows many peripherals to be active at the same time, providing access to memory for individual controllers on demand. Peripherals which use the data channel operate under a priority structure imposed on them by the channel. In cases where more than one controller requests access to the data channel at the same time, priority is determined by a serial priority chain.

Controller Structure

Understanding the operation of the data channel requires a knowledge of the structure of the controllers which use it. The controllers usually contain the normal Busy and Done flags, status, control, and data registers, and the program interrupt components. Additional components are added to handle the functions necessary to operate the data channel. Some of these components, generally available to the program, are in the form of additional control and status registers.

A Word Counter and a Memory Address Counter are usually added. The Word Counter is used by the program to specify the size of the block of data to be transferred (number of words). The Memory Address Counter is used to specify the address in memory which is used in the data transfer.

Word Counter

The program loads the Word Counter with the two's complement of the number of words in the block. Each time a word is transferred, the controller automatically increments the counter by 1. When the counter overflows, the controller terminates data channel transfers.

The size of the Word Counter varies from peripheral to peripheral, depending on the block size associated with the peripheral. A typical Word Counter has either 12 or 15 bits, allowing for up to 4096 or 32,768 word blocks, respectively. Although the Word Counter specifies the negative of the desired block size, the most significant bit of the register need not be a 1, since it is not a sign bit for the number. No sign bit is necessary because the word count is treated as negative by the controller, by virtue of being incremented instead of decremented. Thus, a word count of 0 is valid; in fact, it specifies the largest possible block size. Table 2.7 further illustrates the correspondence between the desired word count and the value which must be loaded into a 12-bit or 15-bit Word Counter.

(Negative) Word Count (Decimal)	15-Bit Value (Octal)	12-Bit Value (Octal)
−1	77777	7777
−2	77776	7776
−8	77770	7770
−100	77634	7634
−2047	74001	4001
−2048	74000	4000
−2049	73777	3777
−4095	70001	0001
−4096	70000	0000
−4097	67777	
−8192	60000	
−32767	00001	
−32768	00000	

Table 2.7 Word Counter Values

Memory Address Counter

The Memory Address Counter always contains the address in memory that the controller will use for the next data transfer. The program loads it with the address of the first word in the block to be transferred. During each transfer, the controller increments the Memory Address Counter by 1. Therefore, successive transfers are to or from consecutive memory locations.

Transfer Sequence

The actual data channel transfer sequence is a two-way communication between processor and controller and proceeds as follows. When a peripheral has a word of data ready to be transferred to memory or wants to receive a word from memory, it issues a data channel request to the processor. The processor begins the data channel cycle by acknowledging peripheral's data channel request. The acknowledgment signal dismisses the peripheral's data channel request. It then causes the peripheral to send back to the processor the address of the memory location and direction of transfer involved in the sequence. Following the receipt of the address, the data itself is transferred in the appropriate direction.

Each time data transfer is completed, the processor/controller interaction is over. The controller carries out any tasks necessary to complete the data transfer, such as transferring the data to the device itself for an output operation.

The controller increments both the Memory Address Counter and the Word Counter during the transfer. If the word count becomes 0, the controller terminates further transfers, sets the Busy flag to 0, the Done flag to 1, and initiates a program interrupt request. If the Word Counter has not yet overflowed, the peripheral continues its operation, issuing another data channel request when it is ready for the next transfer.

Processor Pauses

Data General computers implement the data channel to memory interface in several ways. Some computers briefly suspend program execution to effect the data transfer. These suspensions can occur only at well defined points in program execution. Some computers allow data channel transfers only between instructions. Other computers allow data channel operation between instructions and at certain other points in most instructions (I/O instructions are among those during which data channel transfers cannot occur.)

Priorities

In terms of priorities, in those processors that suspend program execution for data channel transfers, program execution has priority over the data channel. This is true except at certain points in the processor's operation, at which times the data channel has absolute priority (not only over normal program execution, but also any pending program interrupt requests). At these points, the processor will handle all existing data channel requests, including those which are generated while data channel transfers are in progress, before starting a program interrupt or resuming normal instruction execution. Thus, if data channel requests are being generated by a number of peripherals as fast as (or faster than) the processor can handle them, all processing time will be spent handling data channel transfers. Therefore, program execution will stop until all the data channel transfers are made. When the data channel is being used at less than its maximum rate, however, processing time is shared between the data channel, which receives as much as it needs, and the program, which uses the rest.

When the processor honors a data channel request and more than one controller is requesting a data channel transfer, priority is given to the controller which is closest to the processor on the I/O bus.

Programming

Programming a peripheral for a data channel block transfer typically involves the following steps:

1. The program checks the peripheral's status, usually by testing the Busy flag and/or reading a status word and checking one or more error or ready bits. If an error has occurred, ideally, the program should take appropriate action. If no error has occurred but the peripheral is not yet ready, the program should wait for the peripheral to complete its operation. When the peripheral is ready, the program may proceed.
2. The program locates the data block in the device, usually by giving a peripheral *address*; by specifying a unit number, channel number, sector number, or the like.
3. The program locates the data block in memory by loading the Memory Address Counter with the address of the first word of the block.
4. The program then loads the proper value into the Word Counter to specify the size of the data blocks.

5. The program specifies the type of transfer and initiates the operation. If the peripheral is capable of several different operations, specifying the type of transfer usually involves loading a control register in the controller. The operation itself is usually initiated by one of the I/O control functions, Start or Pulse.

The above five steps may be accomplished with programmed I/O instructions or by setting up a channel control block whose address is passed to the peripheral with programmed I/O instructions.

Setting up and initiating the data channel operation is the major part of programming a data channel block transfer. However, if any errors could have occurred during the operation, the program should check for these errors when the operation is complete and take appropriate action if one or more have occurred.

Timing

On large systems which depend heavily on input/output, both the direct program control and data channel facilities can be badly overloaded. This overloading means that certain peripherals are seriously compromised because they lose data or perform poorly, since the system cannot respond to them in time.

This section explains how a system can be overloaded and what steps can be taken to minimize the detrimental effects.

Direct Program Control

Nearly all peripherals operating under direct program control request program service by setting their Done flags to 1. Whether the CPU determines that the Done flag is set to 1 by repeatedly checking it or by responding to interrupt requests, there may be a significant delay between the time when the peripheral requests program service and the time when the CPU carries out that service. This delay is called *programmed I/O latency.*

When the program interrupt facility is not used, programmed I/O latency has two components which can be calculated from the tables in Appendix B.

1. The interval between the time the Done flag is set to 1 by the peripheral and the time the flag is checked by the CPU.
2. The time required by the peripheral service routine to transfer data to/from the periperal and set the Done flag to 0 (by idling the peripheral or instructing it to begin a new operation).

The first component can be diminished by performing frequent checks on the Done flag; the second can be diminished by writing an efficient peripheral service routine.

When the program interrupt facility is used, the programmed I/O latency has at least four components:

1. The time from the setting of the Done flag to 1, to the end of the instruction being executed by the CPU.
2. The time the interrupt facility needs to store the program counter in location 0 and simulate a **JMP @1** instruction.
3. The time required by the interrupt handler to save the state of the machine, identify the peripheral and transfer control to the service routine.
4. The time required by the service routine to transfer data to/from the peripheral and set the Done flag to 0.

The programmed I/O latency may be extended by three other time periods:

5. When CPU operation is suspended because of other system activity.
6. When the CPU does not respond to the peripheral's interrupt request because the interrupt system is disabled. (For example, during the servicing of an interrupt from another peripheral.)
7. When peripheral's Interrupt Disable flag is set to 1 during the servicing of an interrupt of a higher priority peripheral.

The first component is determined by the longest non-interruptable instruction that the CPU can execute. On the NOVA line computers, this is usually a few microseconds (unless long indirect address chains are used in several processors); on some ECLIPSE computers it can be considerably longer due to the presence of the WCS feature, which enables the programmer to code long instructions that do not allow program interrupts to occur during their execution.

The second component is also machine dependent; in general it is approximately two or three times as long as a memory reference. The third, fourth, sixth and seventh components are determined by software and account for the bulk of the programmed I/O latency. The fifth component is determined by the nature and the number of other activities in progress in the system.

Programmed I/O latency is important because a peripheral that must wait too long for program service from the CPU may suffer from degraded performance. The longest allowable delay between the time when a peripheral sets its Done flag to 1 and the time when the CPU transfers data to/from that peripheral and sets the Done flag to 0 is called the *maximum programmed I/O latency* of the peripheral. When the actual programmed I/O latency for a peripheral exceeds the maximum programmed I/O latency, the specific effects depend on the device in question. In the worst case, data may be incorrectly read or written. The maximum allowable programmed I/O latencies for each peripheral can be found in the *Peripherals Manual*, DGC No. 014-000632.

A peripheral service routine must usually perform certain computations (updating pointers to buffers, byte counters, etc.), but rarely are these computations so complex that they cannot be accomplished within the constraints of the maximum allowable programmed I/O latency. However,

if several peripherals are competing for service at the same time, it may be necessary to jeopardize the performance of some of them by deferring their requests for program service until the CPU has serviced the higher priority requests. For this reason, all DGC computers incorporate the priority interrupt facility described earlier.

The object of the priority interrupt facility is to minimize the loss of important data. In order for the programmer to achieve this end, the assignment of the software priority levels should be made in the light of the following considerations:

1. The maximum allowable programmed I/O latency for each peripheral.
2. The result of exceeding the maximum allowable programmed I/O latency for each peripheral (slowdown or data loss),
3. The cost of losing data.

Data Channel Control

Problems with time constraints may also be encountered when transferring data via the data channel. When a peripheral needs data channel service, it makes a data channel request. However, some CPUs can only allow data channel peripherals to access memory at certain times. (At such times, it is said that data channel breaks are enabled.) In addition, there may be more than one peripheral waiting to access memory at any one time. Consequently, there may be a significant delay between the time when a peripheral requests access to memory and the time when the transfer actually occurs. This delay is called data channel latency and has the following components:

1. The time between the peripheral's request for memory access and the next data channel break,
2. The time required to complete data channel transfers to/from higher priority (closer) peripherals that are also requesting memory access.

The length of the first component depends on the design of the CPU. In some computers, data channel breaks are enabled only between instructions so that long instructions (**MUL, DIV**) and long indirect address chains can have a significant effect on data channel latency. In other computers, data channel breaks may be enabled during most instructions (but not during I/O instructions), so that data channel latency is reduced. See Appendix B for further information on the differences between CPUs.

The length of the second component depends on the number of data channel peripherals operating in the system at a higher priority and the frequency of their use.

Most peripherals using the data channel control operate under fixed time constraints. Disc drives and magnetic tape transports are typical data channel peripherals. In each of these devices, a magnetic medium moves past a read or write head at constant velocity. If data is not read or written at the correct instant, it will be transferred to or from the wrong place on the magnetic medium. Consequently, on input, such devices must be allowed to write a word into memory before the next word is assembled by the controller, and on output, the controller must be able to read a word from memory before the surface is positioned under the write head. In either case, if the data channel latency is too long, data cannot be properly transferred. Most peripherals operating under data channel control set an error flag when this happens, so the service routine can take appropriate action to recover from the error, if possible.

The maximum allowable data channel latency of a peripheral is the longest time the peripheral can wait for a data channel transfer. The range of times is from a few microseconds to several hundred microseconds. At the time the system is configured, data channel priorities should be assigned to peripherals on the basis of the following considerations:

1. The maximum allowable data channel latency of the peripheral. A peripheral with a short allowable latency should usually receive a higher priority than one with a long allowable latency.
2. The recovery time of a peripheral (i.e., how long before it can repeat a transfer that failed because of excessive data channel latency) if the peripheral can recover.
3. The cost of losing data from the peripheral if the peripheral cannot recover.

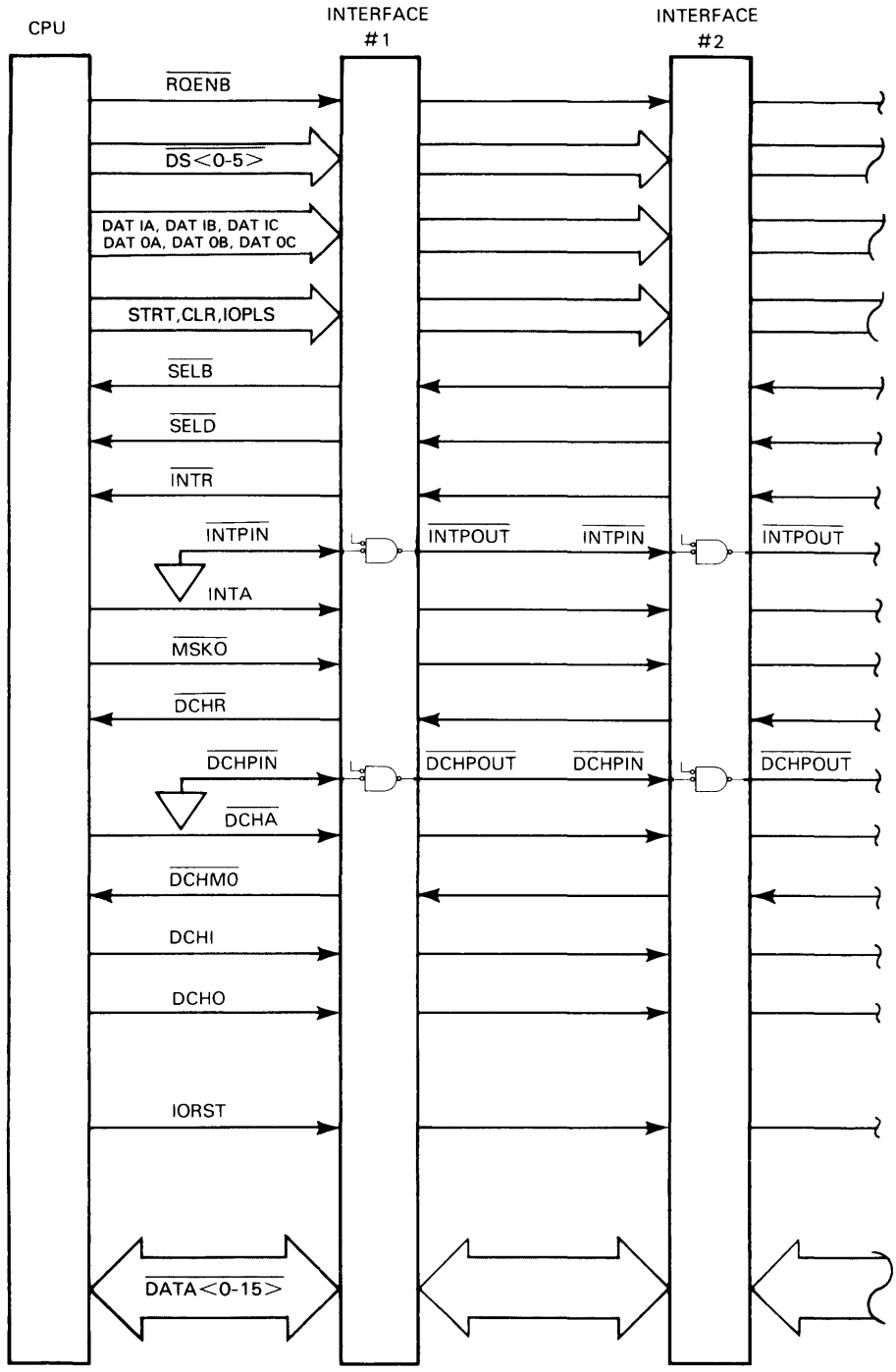
If data channel latency seems to be a problem in a system, latency might be improved by changing programs; less frequent use of long instructions and lengthy indirect chains, as well as less frequent use of I/O instructions. In addition, there is an upper limit on the number of data channel transfers per second that a computer can support. In cases where this limit is exceeded, the only solution is to reduce the number of peripherals using the data channel at the same time.

A final consideration is that high use of the data channel in some CPUs reduces the speed of program execution, since the processor pauses for each transfer. This may adversely affect the CPU's capacity to respond to interrupts and service those peripherals operating under direct program control.

Chapter 3
I/O BUS

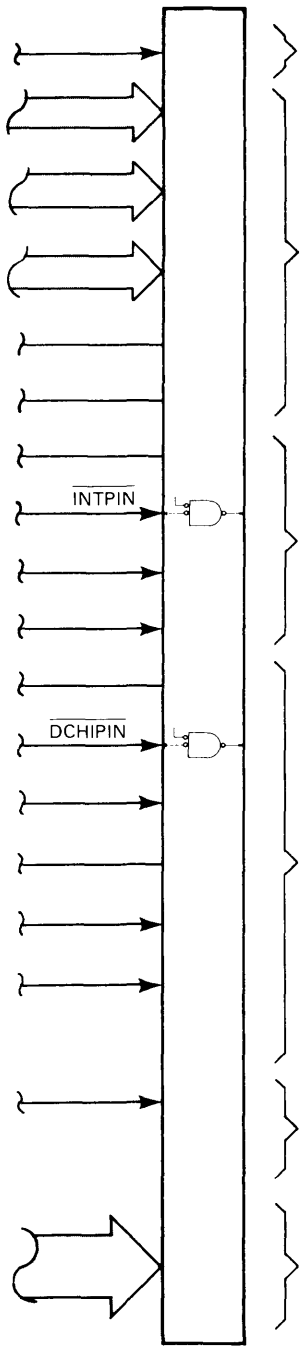
Introduction

Despite minor differences among the I/O facilities of the various computers, the structure of the I/O bus itself is the same for all NOVA and ECLIPSE computers. This structure embodies a single bus connecting the central processor to all interfaces as shown in Fig. 3.1. Data is transferred on the bus along 16 parallel, bidirectional, data lines. Control signals are carried along dedicated, unidirectional, control lines. In addition to specifying a unique function, each control signal generated by the central processor provides all timing necessary to perform that function. Data transfers are synchronous; no *hand-shaking* occurs between the interface and the central processor. The data channel and program interrupt facilities each use their own single request and priority lines. The two request lines are run in parallel to all interfaces, so that an interface requiring either data channel or program interrupt service need only assert the appropriate line and wait for the processor to respond. The serial priority lines are independent and are chained from interface to interface, so that priority for service is granted to the interface closest on the chain to the central processor.



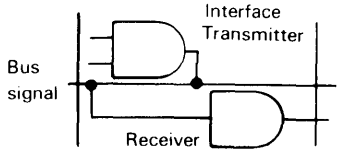
DG-08002

Figure 3.1 The I/O bus



GROUP	SIGNALS	DESCRIPTION
ENABLE	RQENB	Enables interrupt request and data channel requests
PROGRAMMED I/O	DS<0-5>	Programmed I/O device select
	DATIA, DATIB, DATIC DATOA, DATOB, DATOC	Data input or data output command strobe
	STRT, CLR, IOPLS	Flag control or flag test command
	SELB	Selected Busy flag
	SELD	Selected Done flag
PROGRAM INTERRUPT	INTR	Interrupt request
	INTPIN/INTPOUT	Serial interrupt priority chain
	INTA	Interrupt acknowledge
	MSKO	Interrupt mask out strobe
DATA CHANNEL	DCHR	Data channel request
	DCHPIN/DCHPOUT	Serial data channel priority chain
	DCHA	Data channel acknowledge
	DCHM0	Data channel mode select
	DCHI	Data channel input strobe
	DCHO	Data channel output strobe
SYSTEM CONTROL	IORST	I/O reset command
DATA & ADDRESS	DATA<0-15>	Transfers: Programmed I/O data Interrupt acknowledge device code Data channel address Data channel data

NOTE: All interface to bus connections are parallel connections (see diagram) unless shown otherwise.



Logic Conventions

Drawings

Data General logic prints are drawn in close accordance with Mil-Std-806C. With this convention, logical functions are drawn as physically implemented. That is, where discrete gates are used to implement a function, these gates are shown. On the other hand, where a more complex integrated circuit is used, for instance a multiplexor, that function is shown as a rectangular box instead of the gates comprising the function.

Signal Levels

Throughout this manual, a distinction is frequently made between electrical levels and logical values. To minimize confusion, electrical levels are always indicated by an *H* or *L*, and logical values by a *1* or *0*. As an electrical level, an *H* indicates that the signal is high (greater than +2.0 volts) and an *L* indicates that it is low (less than +0.7 volts). An asserted, or true, signal is indicated by a logical *1* and a false signal by a *0*.

Signal Names

The voltage level at which a signal is said to be *asserted* (*true*) is a matter of definition. To distinguish between signals that are asserted high (0=L, 1=H) and those that are asserted low (0=H, 1=L), a naming convention has been adopted in Data General's documentation which defines the relationship between the logical value and electrical level of a signal. If the signal name includes a horizontal bar over the name, as **WRITE**, then that signal is asserted when it is at a low electrical level; conversely, a signal without the bar, **WRITE**, is asserted when high.

To be expressed, logical functions may often require more than one binary signal. For instance, three lines are required to express an octal digit. Generally, these closely related signals are individually identified by effectively subscripting a common label. For instance, suppose that **BUS0** through **BUS5** are all required to completely specify a function. All or part of such a group of signals is identified by placing brackets around the range of subscripts included, as **BUS<0-5>**. In this case, the suffix carries the information that there are six BUS lines under discussion, from **BUS0** through **BUS5**, inclusive.

Summary of I/O Bus Signals

The forty-eight signals which comprise the I/O bus can be divided functionally into five groups. The following list shows this grouping, along with a brief description of the function of each signal. Timing information for these signals is shown in Chapter 4. Note that, with the exception of the two priority lines, all I/O bus lines are run in parallel to all interfaces.

Data

DATA<0-15> *Data.* All data and addresses, for both data channel and programmed I/O, are transferred between the processor and interfaces attached to the I/O bus via these 16 bidirectional lines. The interrupt disable mask and interrupt acknowledge information are also carried on these lines.

Programmed I/O

DS<0-5> *Device Select.* These lines carry the low-order six bits of an I/O instruction; that is, the device code. Only the interface whose device code corresponds to that carried on these lines should respond to control signals generated on the I/O bus.

DATIA *Data In A.* Asserted by the processor during the execution of a **DIA** instruction. Should cause the interface selected by **DS<0-5>** to place the contents of its A input buffer on **DATA<0-15>**.

DATIB *Data In B* Asserted by the processor during the execution of a **DIB** instruction. Should cause the interface selected by **DS<0-5>** to place the contents of its B input buffer on **DATA<0-15>**.

DATIC *Data In C.* Asserted by the processor during the execution of a **DIC** instruction. Should cause the interface selected by **DS<0-5>** to place the contents of its C input buffer on **DATA<0-15>**.

DATOA *Data Out A.* Asserted by the processor during the execution of a **DOA** instruction, after the processor has placed the contents of the specified accumulator on **DATA<0-15>**. Should cause the interface selected by **DS<0-5>** to load its A output buffer with the data on **DATA<0-15>**.

DATOB *Data Out B.* Asserted by the processor during the execution of a **DOB** instruction, after the processor has placed the contents of the specified accumulator on **DATA<0-15>**. Should cause the interface selected by **DS<0-5>** to load its B output buffer with the data on **DATA<0-15>**.

DATOC *Data Out C.* Asserted by the processor during the execution of a **DOC** instruction, after the processor has placed the contents of the specified accumulator on **DATA<0-15>**. Should cause the interface selected by **DS<0-5>** to load its C output buffer with the data on **DATA<0-15>**.

STRT *Start.* Asserted by the processor during the execution of any I/O instruction (except an *I/O Skip* instruction) in which bits 8 and 9=01 (i.e., any I/O instruction in which the *Start (S)* control function is specified). Not asserted during **DIA**, **DIB**, **DIC**, **DOA**, **DOB**, **DOC** instructions until after the data transfer has occurred. Usually used to initiate peripheral operation by setting the Busy flag to 1 and the Done flag to 0.

CLR *Clear.* Asserted by the processor during the execution of any I/O instruction (except an *I/O Skip* instruction) in which bits 8 and 9=10 (i.e.,

any I/O instruction in which the *Clear (C)* control function is specified). Not asserted during **DIA**, **DIB**, **DIC**, **DOA**, **DOB**, **DOC** instructions until after the data transfer has occurred. Usually used to terminate peripheral operation by setting the Busy and Done flags to 0.

IOPLS *I/O Pulse.* Asserted by the processor during the execution of any I/O instruction (except an *I/O Skip* instruction) in which bits 8 and 9=11 (i.e., any I/O instruction in which the *Pulse (P)* control function is specified). Not asserted during **DIA**, **DIB**, **DIC**, **DOA**, **DOB**, **DOC** instructions until after the data transfer has occurred. Usually used to initiate special peripheral operations.

SELB *Selected Busy.* Asserted low by the interface selected by the device select lines if its Busy flag is set to one.

SELD *Selected Done.* Asserted low by the interface selected by the device select lines if its Done flag is set to one.

Program Interrupt

INTR *Interrupt Request.* Asserted low by an interface to request program interrupt service.

MSKO *Mask Out.* Asserted low by the processor during the execution of the **MSKO** instruction, after the contents of the designated accumulator have been placed on **DATA<0-15>**. Used to load the contents of **DATA<0-15>** into the interrupt disable flip-flops of all interfaces using the interrupt system.

INTP *Interrupt Priority.* Seen asserted by the first interface on the I/O bus using the program interrupt facility, and transmitted in series through each successive interface. An interface should not issue an asserted **INTP OUT** unless it is receiving an asserted **INTP IN** and is not requesting interrupt service.

INTA *Interrupt Acknowledge.* Asserted by the processor during the execution of the **INTA** instruction. If an interface receives **INTA** while it is also receiving **INTP IN** asserted and while it is requesting interrupt service, it should place its device code on **DATA<10-15>**.

Data Channel

DCHR *Data Channel Request.* Asserted low by an interface when it requires data channel service.

DCHP *Data Channel Priority.* Seen asserted by the first data channel interface on the I/O bus, and transmitted in series through each interface. An interface should not issue an asserted **DCHP OUT** unless it is receiving an asserted **DCHP IN** and it is not requesting data channel service. Also used by some processors to determine data channel speed.

DCHA *Data Channel Acknowledge.* Asserted low by the processor at the beginning of each data channel cycle. Should cause the interface that is receiving an asserted **DCHP IN** signal and whose DCH REQ flip-flop is set, to set its DCH SEL flip-flop and place the memory address to be used for this transfer on **DATA<0-15>** and the mode on the data channel mode lines of the I/O bus.

DCHM0 *Data Channel Mode.* Asserted low by the interface whose DCH SEL flip-flop is set to inform the processor of the type of data channel cycle to be performed, as follows:

DCHM0	Function
0=H	Output
1=L	Input

Table 3.1 Data channel modes

DCHI *Data Channel Input.* Asserted by the processor for data channel input (**DCHM0**=1). Should cause the interface whose DCH SEL flip-flop is set to place the contents of its input register on **DATA<0-15>**.

DCHO *Data Channel Output.* Asserted by the processor for data channel output (**DCHM0**=0), after the data word has been placed on **DATA<0-15>**. Should cause the priority-selected interface to load the data from **DATA<0-15>**.

System Control

IORST *I/O Reset.* Asserted by the processor during the **IORST** instruction or when RESET occurs. **IORST** is also issued prior to processor operation at power turn-on. This signal should be used to initialize the machine state of all interfaces in the system.

RQENB *Request Enable.* Asserted low by the processor to synchronize program interrupt and data channel requests from all interfaces. In any interface, **INTR** and **DCHR** should be clocked only on the leading edge of **RQENB**.

Programmed I/O Protocol

Device Selection

Every programmed input/output instruction includes a six-bit device code which uniquely references the interface which is to be involved in the transfer. During the execution of an input/output instruction, the device select lines, **$\overline{\text{DS0}}$** through **$\overline{\text{DS5}}$**, carry the contents of the low-order six bits of the instruction, the device code. The interface, therefore, should not assert the DATA lines of the bus or initiate any other function as a result solely of the device select lines. Rather, the selected interface should respond only to the assertion of control signals on the I/O bus.

An interface can decode the device select lines in a number of ways. The device select lines corresponding to the bit positions containing a one in the interface's device code could be inverted, and an AND function performed on the resultant six lines, as shown in Figure 3.2 for device code 13_8.

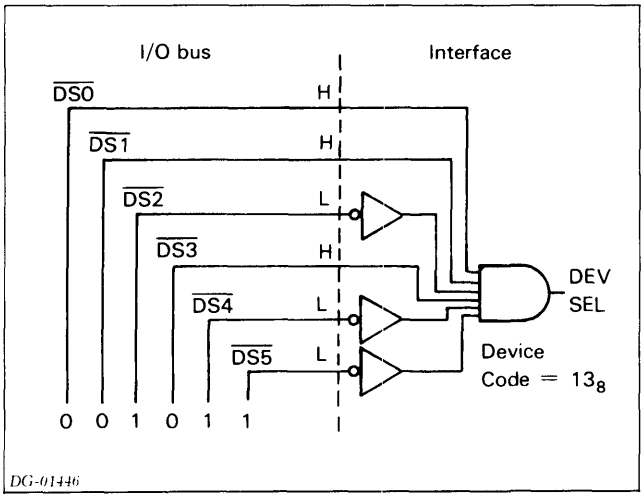


Figure 3.2 Single device code selection network

Similarly, the lines corresponding to zeros could be inverted and the six lines applied to a NOR gate. There are other possibilities; the two important details are that the lines are asserted low and that **$\overline{\text{DS0}}$** is the high-order bit of the device code. Note that it is possible to have the interface respond to more than one device code by decoding fewer than six lines. The remaining lines might be clocked into a register to select a particular interface mode as shown in Figure 3.3.

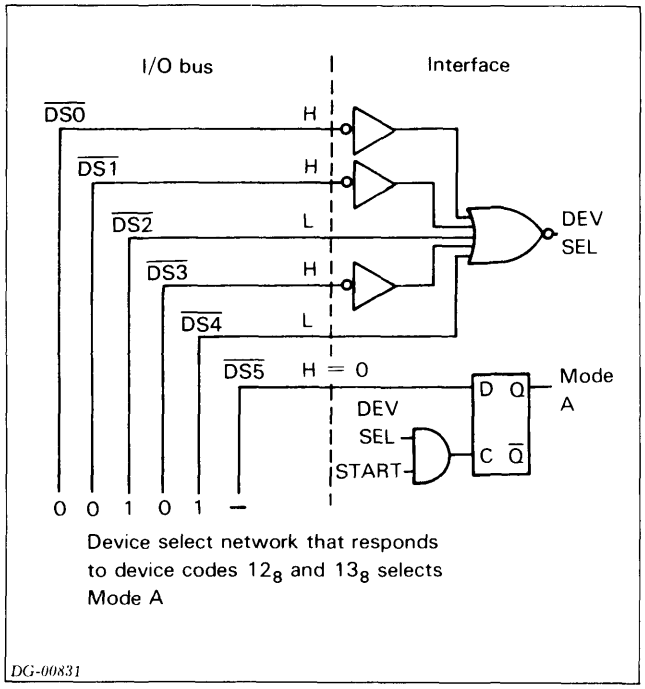


Figure 3.3 Multiple device code selection network

Assigning Device Codes

There are a number of factors to be considered when assigning a device code to an interface. A six-bit device code allows sixty-four possible codes, 0 to 77_8. In all machines, device code 77_8 is assigned to the central processor, to implement such special functions as mask out and interrupt acknowledge, and hence can not be assigned to an interface. Similarly, a number of programming considerations restrict the use of device codes as assigned by Data General and used in all Data General software (see Appendix A). When assigning a device code to a custom interface, it is important to consider what other devices are currently in the system or may be installed at a future date.

Data Transfer Signals

Data Input

The three programmed data input instructions - **DIA**, **DIB**, and **DIC** (*Data In A, B, and C*) - allow data to be transferred from up to three distinct sources per device code in the interface and loaded into any one of the four accumulators of the processor. Additionally, if specified in the instruction, the **STRT**, **CLR**, or **IOPLS** signal will be issued by the processor to control the status flags or other interface functions.

The execution of a data input instruction consists of two parts. The first is the data transfer, followed by the (optional) control pulse. There are three signals used for the data input transfer, one for each of the three possible data sources in the interface. Through the use of three separate signals, the need for a decoding network in the interface is avoided.

During the first half of the execution of the data input instruction, one of the three transfer control signals - **DATIA**, **DATIB**, or **DATIC** - is asserted, as shown in the sequence diagram below. This signal should be used by the selected interface to cause data from the proper sources to be asserted on the DATA lines of the bus. After a time delay, the processor will load the information on the DATA lines into the accumulator specified in the instruction and drop the transfer control signal. Because all interfaces are wired to the I/O bus in parallel, it is extremely important that only the interface referenced by the device select code assert any of the DATA lines, and then only in response to the transfer control signal.

The second portion of the instruction execution consists of the assertion of the appropriate control pulse - **STRT**, **CLR**, or **IOPLS** - as specified in the instruction (see Figure 3.4). The response of the interface to each of these signals will be discussed later.

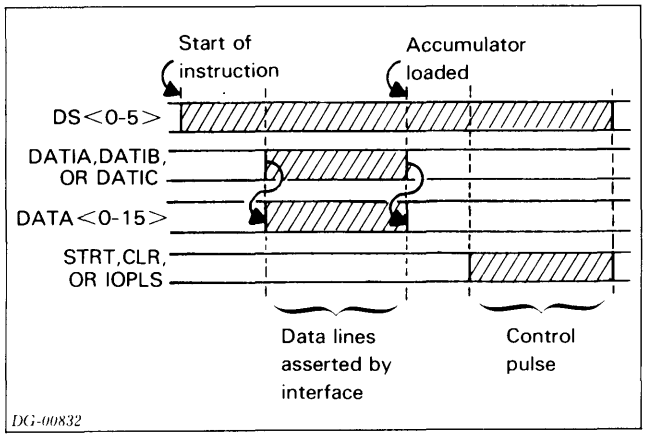


Figure 3.4 Data in sequence

Data Output

The three programmed data output instructions - **DOA**, **DOB**, and **DOC** (*Data Out A, B, and C*) - allow data to be transferred from any one of the four accumulators of the processor to one of up to three destinations per device code in the interface. Additionally, if specified in the instruction, the **STRT**, **CLR**, or **IOPLS** signal will be issued by the processor to control the status flags or other interface functions.

The execution of a data output instruction consists of two parts, the data transfer followed by the (optional) control pulse. There are three signals used for the data output transfer, one for each of the three possible data destinations in the interface. Through the use of three separate signals, the need for a decoding network in the interface is avoided. Note that the three destinations used for data output need have no relation to the three data sources used for data input.

As shown in the sequence diagram below, the processor asserts the DATA lines with the contents of the specified accumulator. After allowing time for the data to propagate down the I/O bus and for the lines to settle, the processor asserts one of the three transfer control signals - **DATOA**, **DATOB**, or **DATOC**. This signal should be used by the selected interface to gate the contents of the DATA lines to the proper destination and to load a register. After dropping the transfer control signal, the processor will drop the data from the DATA lines. Because all interfaces are wired to the I/O bus in parallel, it is extremely important that no device, including that referenced by the device select code, be allowed to assert any of the DATA lines during the data output instruction.

The second portion of the instruction execution consists of the assertion of the appropriate control pulse - **STRT**, **CLR**, or **IOPLS** - as specified in the instruction (see Figure 3.5). The response of the interface to each of these signals will be discussed later.

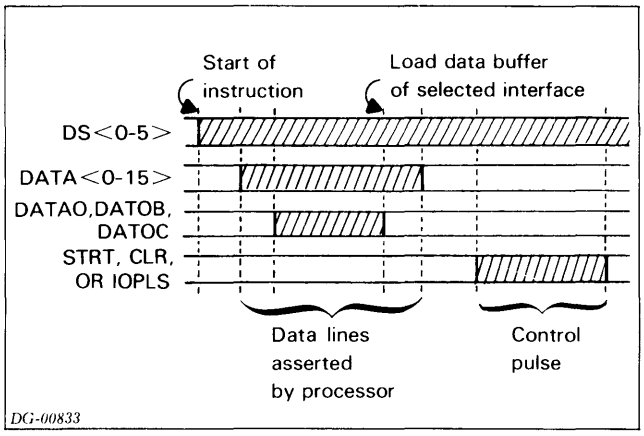


Figure 3.5 Data out sequence

I/O Skip

The operation of most peripherals is not synchronous to the operation of the processor. Because of its faster processing rate, the computer will generally have to wait for the completion of a peripheral's operation. It is usually important that the processor not issue any new instructions to the peripheral until it has completed its previous operation. This asynchronous operation of the processor and peripheral requires that the CPU be able to test the status of the peripheral.

The *I/O Skip* instruction allows the program to test the state of two I/O bus lines, **SELB** and **SELD**. Whenever an interface recognizes its device code on the device select lines, as discussed above, it should assert the **SELB** and/or **SELD** lines depending on the internal state of the interface. (Ordinarily, **SELB** will be asserted if the Busy flag is set and **SELD** will be asserted if the Done flag is set.) During the execution of the *I/O Skip* instruction, the processor checks the state of the appropriate line and skips the next sequential instruction if the line matches the condition specified in the instruction.

The test condition is specified in the T field of the instruction as shown in Table 3.2:

T Field	Mnemonic	Next Instruction Skipped If:
00	BN	$\overline{\text{SELB}}$ = L
01	BZ	$\overline{\text{SELB}}$ = H
10	DN	$\overline{\text{SELD}}$ = L
11	DZ	$\overline{\text{SELD}}$ = H

Table 3.2 Test conditions

Start, Clear, I/O Pulse

During the second portion of the execution of any I/O instruction except the *I/O Skip* instruction, the processor can issue one of three control signals - **STRT**, **CLR**, or **IOPLS** - as coded in the instruction. Though a convention is followed in the use of these signals in Data General interfaces, as explained below, the designer should realize that they may be used for virtually any purpose.

Busy/Done Network

In Data General interfaces, two flags, the Busy flag and the Done flag, carry elementary status information needed by the program. Whenever the interface detects its device code on the device select lines it asserts the $\overline{\text{SELB}}$ line if its Busy flag is set and $\overline{\text{SELD}}$ if its Done flag is set. These lines are tested by the processor during the *I/O Skip* instruction.

Although the significance of the flags may vary somewhat depending on the particular interface in question, they do carry a similar meaning in many cases. The Busy flag generally indicates that the device is currently processing data or waiting for some response from an external system. When this flag is set, any interference from the program, such as an attempt to transfer data to this device, may produce unpredictable results. The Done flag indicates that the device has completed an operation and is awaiting a response by the program. In many devices, it is important that this response come within a maximum time period, to prevent a degradation of system performance. See Fig. 3.6 for Busy/Done sequencing.

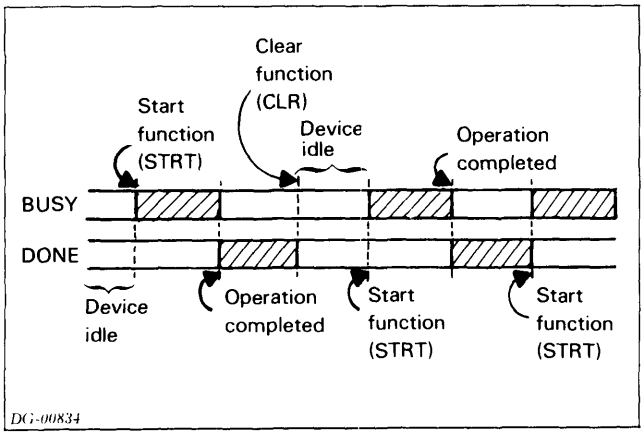


Figure 3.6 Busy/done sequencing

In addition to conveying status information to the program, these flags can be used as switches by the program to control the interface. Generally, the *Start* pulse causes the Busy flag to be set and the Done flag to be cleared, initiating interface operation. At the completion of the operation, a signal originating in the interface sets DONE and clears BUSY. If at any time the *Clear* pulse is issued by the processor, both flags should be cleared.

Figure 3.7 shows one implementation of the Busy/Done network. The **IORST** signal clears both the Busy and Done flags directly. Signals generated by the control function part of an I/O instruction affect the flags only if the device has recognized its device code on the device select lines. When the device completes its operation it generates a completion signal, **DEV COMPLETE**, that clears the Busy flag and sets the Done flag. The signal need not act on both flags directly; it can just as well clear the Busy flag, whose state change sets the Done flag. Note that in the configuration shown here, the data input to the Done flag is the output of the Busy flag. Therefore, the completion signal will not set the Done flag if the program has previously cleared the Busy flag.

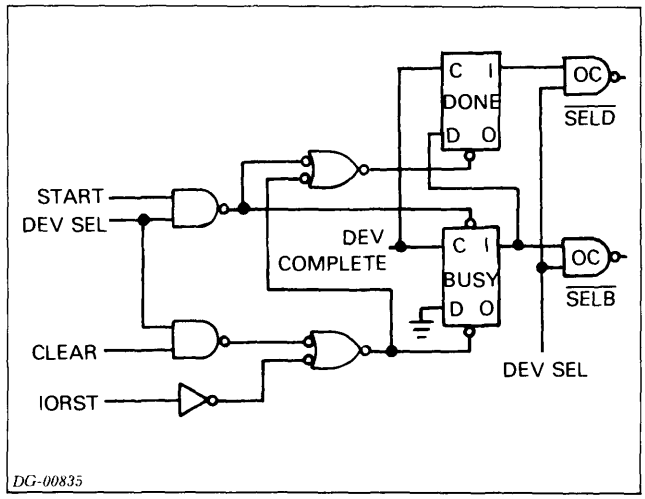


Figure 3.7 Typical busy/done network

Program Interrupt System

Interrupt Request

An interface issues a program interrupt request by asserting the $\overline{\text{INTR}}$ line of the I/O bus. The central processor checks this line at the end of every instruction, and if it is asserted (and *ION* is 1), executes the program interrupt function. The interrupt in no way affects the interface itself; any action to actually service the device must be the result of the software interrupt handler.

Interrupt Disable

In addition to the three data output transfer control signals, there is a signal, $\overline{\text{MSKO}}$, which functions in much the same way. This signal allows a 16-level priority system to be established for the program interrupt. $\overline{\text{MSKO}}$ is issued

during the data transfer portion of a *Data Out B* instruction, when a device code of 77_8 is specified (**CPU**).

Conventionally, each interface using the interrupt system is assigned a hardware priority level corresponding to one of the sixteen bits of a data word. When the **MSKO** signal is received by the interface (regardless of device code), an interrupt disable flag (INT DISABLE) should be loaded from the **DATA** signal corresponding to the priority assignment of that interface, as shown in Figure 3.8. Whenever this INT DISABLE flag is set, the interface should be inhibited from issuing an interrupt request. Additionally, if the interface was issuing an interrupt request and the INT DISABLE flag is then set, the request should be dropped on the next **RQENB** pulse.

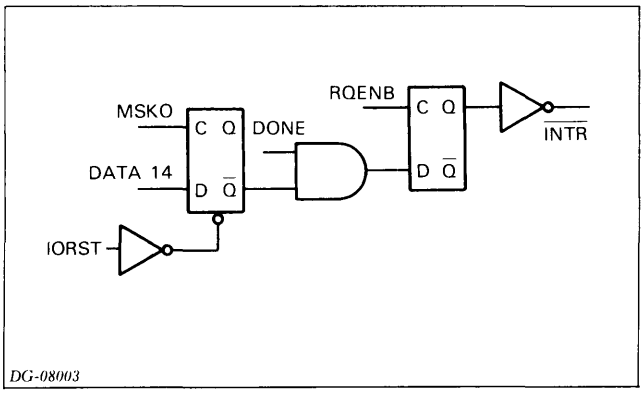


Figure 3.8 Typical priority mask bit circuit

Interrupt Priority

There is a program interrupt priority signal on the I/O bus which should be passed through a priority network in every interface that uses the program interrupt. This signal, which must be passed undisturbed by other interfaces (and memories), is called **INTP IN** as it enters each interface and **INTP OUT** as it leaves. **INTP IN** starts on the computer back panel, immediately adjacent to the central processor boards, as a low (asserted) signal, but is pulled high to succeeding interfaces by any interface that requests interrupt service. Any interface that receives a high **INTP IN** signal should pass a high **INTP OUT** to the following interfaces and on down the bus.

The circuit in Figure 3.9 shows the suggested implementation of this priority network. In many cases, more than one interface using the program interrupt will be built on a single board. In such cases, each interface will require its own priority network. As many elements as needed, each similar to that shown below, would be chained together, with the **INTP OUT** signal of one feeding **INTP IN** of the next. Note that the terminating resistors shown are used on the signals that enter and leave the board.

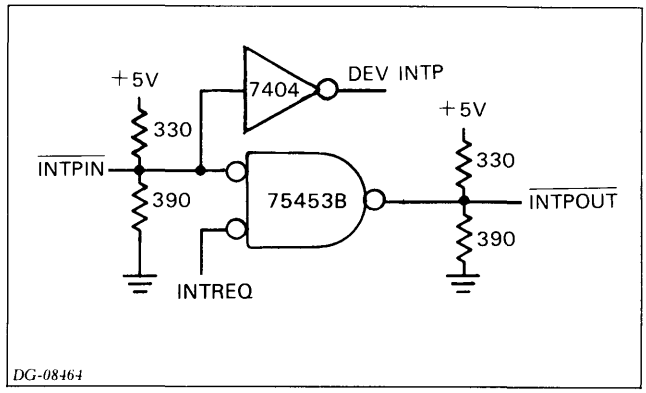


Figure 3.9 Typical interrupt priority chain

Timing on this interrupt priority chain can be critical and becomes especially so for large systems with many interfaces. Often it is possible to build several interfaces on a single board. In these cases the propagation time can be significantly reduced by replacing the priority chain on such a board with two separate chains. One, consisting of a single network element, determines the priority of the entire board and quickly passes the **INTP** signal on to the next board. A separate chain determines the priority of the various interfaces on the board. See Figure 3.10.

Figure 3.11 shows a typical interrupt control circuit as implemented in each interface.

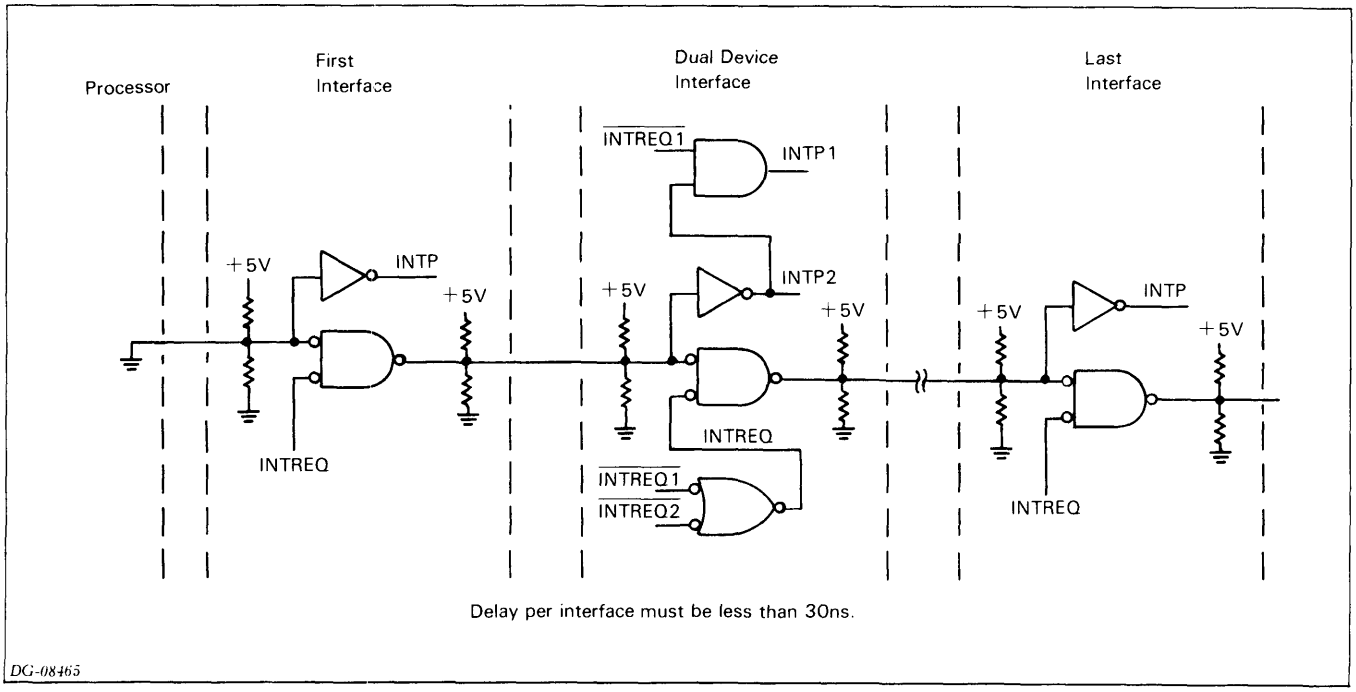


Figure 3.10 Interrupt priority chain

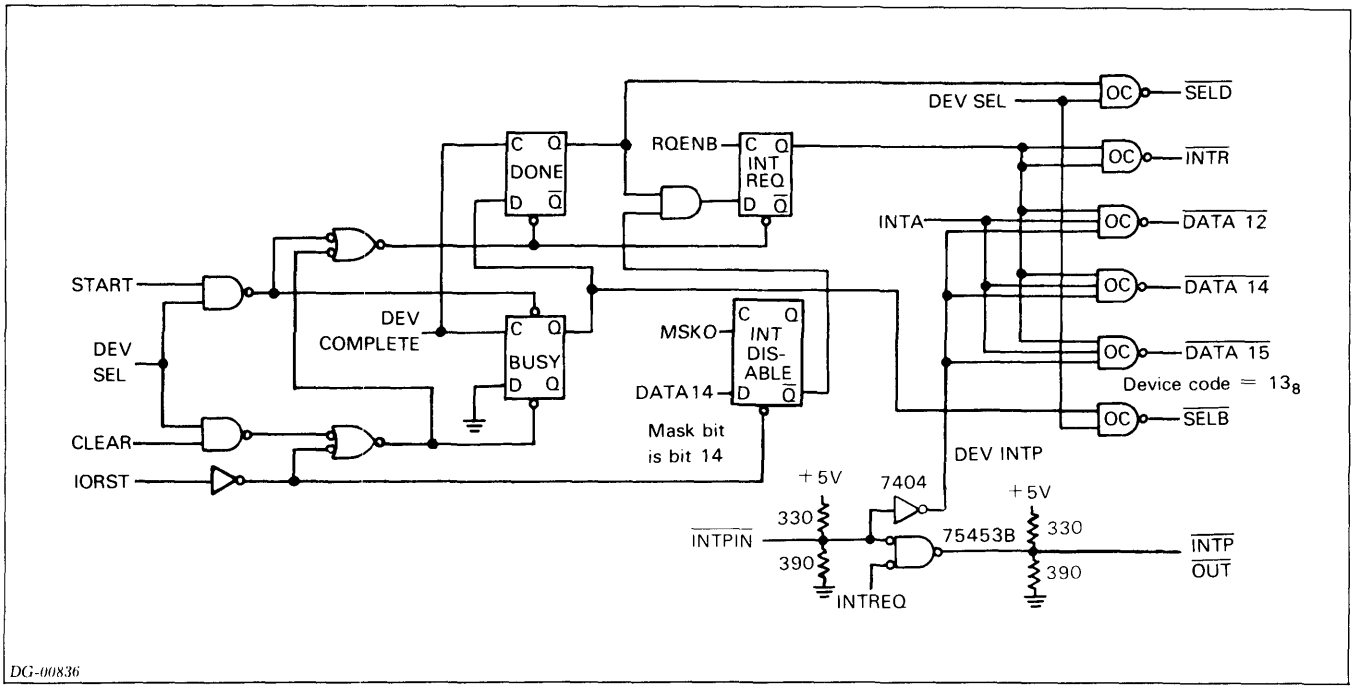


Figure 3.11 Typical interrupt control circuit

Interrupt Acknowledge

Once the processor has received an interrupt request and transferred control to the interrupt service routine, the software must service the interface that caused the interrupt. Before the program can even attempt to service the interface, it must determine which interface did, in fact, cause the interrupt. The simplest way that this can be achieved is by the use of the *Interrupt Acknowledge* instruction (**INTA**). This instruction is equivalent to a *Data Input B* instruction with a device code of 77_8 (**CPU**). This instruction executes as a data input transfer instruction, but the processor asserts the **INTA** signal during the data transfer portion of the instruction.

When the **INTA** signal is received by an interface (regardless of device code), the condition of the **INTP IN** line to that interface should be checked. If this line is asserted and the interface is currently issuing an interrupt request, it should assert its device code on **DATA<10-15>** of the I/O bus, for duration of the **INTA** signal. At the end of the **INTA** period, the processor loads the selected accumulator with the contents of the DATA lines.

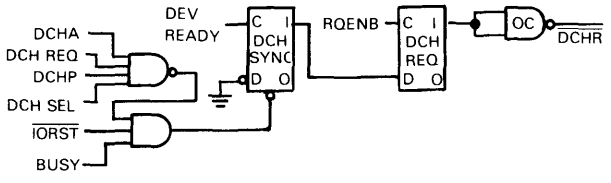
Data Channel

Unlike the programmed input/output transfers, which are each controlled by one unique signal on the bus, the data channel transfers are somewhat more complex. The interplay between the central processor and the interface is much more involved, due to the number of functions performed for each transfer. Unlike the design of a programmed I/O interface, where certain liberties can be taken in the designed response to a bus signal, the design of a data channel interface is relatively restricted, and it is recommended that such a design correspond to the following description.

Data Channel Request

An interface issues a data channel request by asserting the **DCHR** line of the I/O bus. The central processor checks this line at certain points in its operation and, if it is asserted, executes a data channel function. Unlike interrupts, the processor services a data channel request completely without software intervention. On some machines, normal program execution is suspended for the duration of the data channel transfer. This delay is virtually transparent to the program as each transfer takes from 1 to 2 microseconds.

The signal **RQENB**, generated by the processor, is used to time the data channel requests in the same manner as the interrupt requests. The usual convention is to use **RQENB** to clock the state of an interface-controlled flip-flop (DCH SYNC in Figure 3.12) into a data channel request (DCH REQ) flip-flop, which in turn drives the **DCHR** line. It is very important that **DCHR** be clocked only on the leading edge of **RQENB**. (See Appendix D.)



Typical data channel request network. DEV READY is generated by device or other interface circuitry. DCH SEL is asserted when interface is currently being serviced.

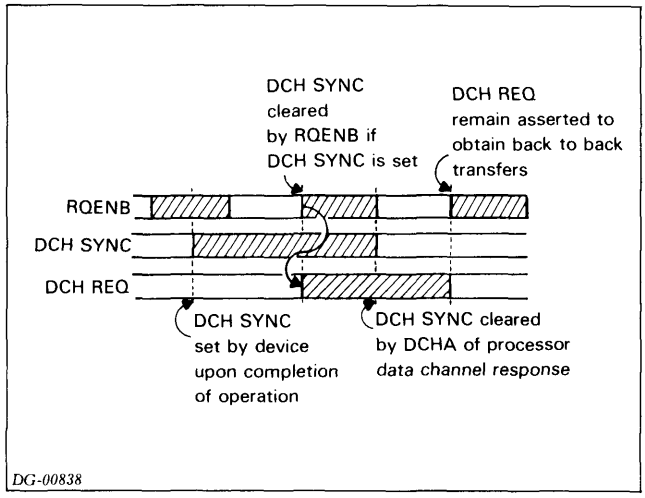
DG-00840

Figure 3.12 Typical data channel request circuit

When the processor sees **DCHR** asserted, at the next convenient point in its operation, it transfers data to or from the highest priority controller requesting service. Just before the end of every data channel transfer, the processor asserts the **RQENB** signal again and if the **DCHR** signal is still asserted at the end of the current transfer, the processor performs data channel transfer to the highest priority controller that is still requesting data channel service. The processor continues to perform data channel transfers in this way until no controller on the I/O bus is requesting data channel service.

Multiple data channel transfers are allowed in all computers. In those computers that pause to allow data channel transfers, multiple transfers can occur in a single processor pause, or data channel break. These multiple transfers are referred to as back-to-back transfers.

Figure 3.13 shows the timing sequence required for data channel transfers.



DG-00838

Figure 3.13 Typical data channel sequence

Data Channel Priority

There is an elementary hardware priority system on the I/O bus which serves to arbitrate between two or more interfaces requesting data channel service at the same time. In the event of simultaneous data channel requests, this priority system causes service to be granted to the interface that is requesting service and is closest to the processor on the I/O bus.

The data channel priority signal should be passed through a priority network in every interface that uses the data channel. This signal, which must be passed undisturbed by other interfaces (and memories), is called **DCHP IN** as it enters each interface and **DCHP OUT** as it leaves. **DCHP IN** starts on the computer back panel, immediately adjacent to the central processor boards, as a low (asserted) signal, but is pulled high to succeeding interfaces by any interface that requests data channel service. Any interface that receives a high **DCHP IN** signal should pass a high **DCHP OUT** to the following interface and on down the I/O bus.

The only interface that should respond to the processor's data channel signals is that which is requesting data channel service and is receiving a low level **DCHP IN**; that is, the interface closest to the processor that is requesting data channel service.

The circuit shown in Figure 3.14 shows the suggested implementation of this priority network. In some cases, more than one interface using the data channel facility will be built on a single board. In such cases, each interface will require its own priority network. As many elements as needed, each similar to that shown below, would be chained together, with the **DCHP OUT** signal of one feeding **DCHP IN** of the next. Note that the terminating resistors shown are used on the signals that enter or leave the board.

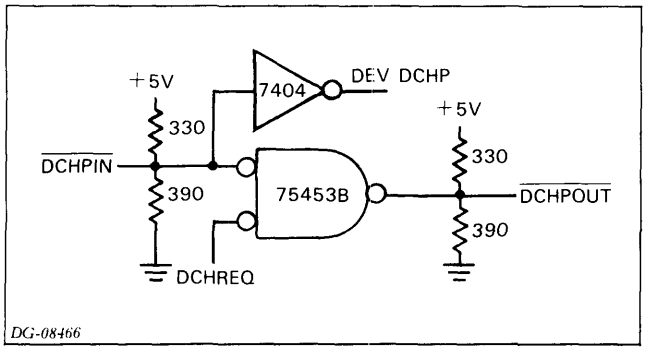


Figure 3.14 Typical data channel priority circuit

Timing on this data channel priority chain can be critical and become especially so for large systems with many interfaces. Often it is possible to build several interfaces on a single board or in a single external chassis. In these cases the propagation time can be significantly reduced by replacing the priority chain on such a board with two separate chains. One, consisting of a single network element, determines the priority of the entire board and quickly passes the **DCHP** signal on to the next board. A separate chain determines the priority of the various interfaces on the board (see Figure 3.15).

Acknowledge

As the first step in any data channel transfer the central processor will issue the data channel acknowledge signal, **DCHA**, to all interfaces on the I/O bus. The processor expects two types of information from the interface in response to this signal, the memory address and the mode of this transfer. Beyond this however, the **DCHA** signal alerts the interface to the beginning of a transfer, allowing it to perform some additional functions.

All of the data channel signals are issued on the I/O bus without an accompanying code on the device select lines. The priority network is used to determine which interface is to respond to these signals. Before one data channel cycle is complete, the data channel request from the interface being serviced must be cleared, to prevent an immediate second transfer. Therefore, a storage unit must be provided to maintain an interface's selected state even after a change in **DCH REQ** or **DCHP IN**.

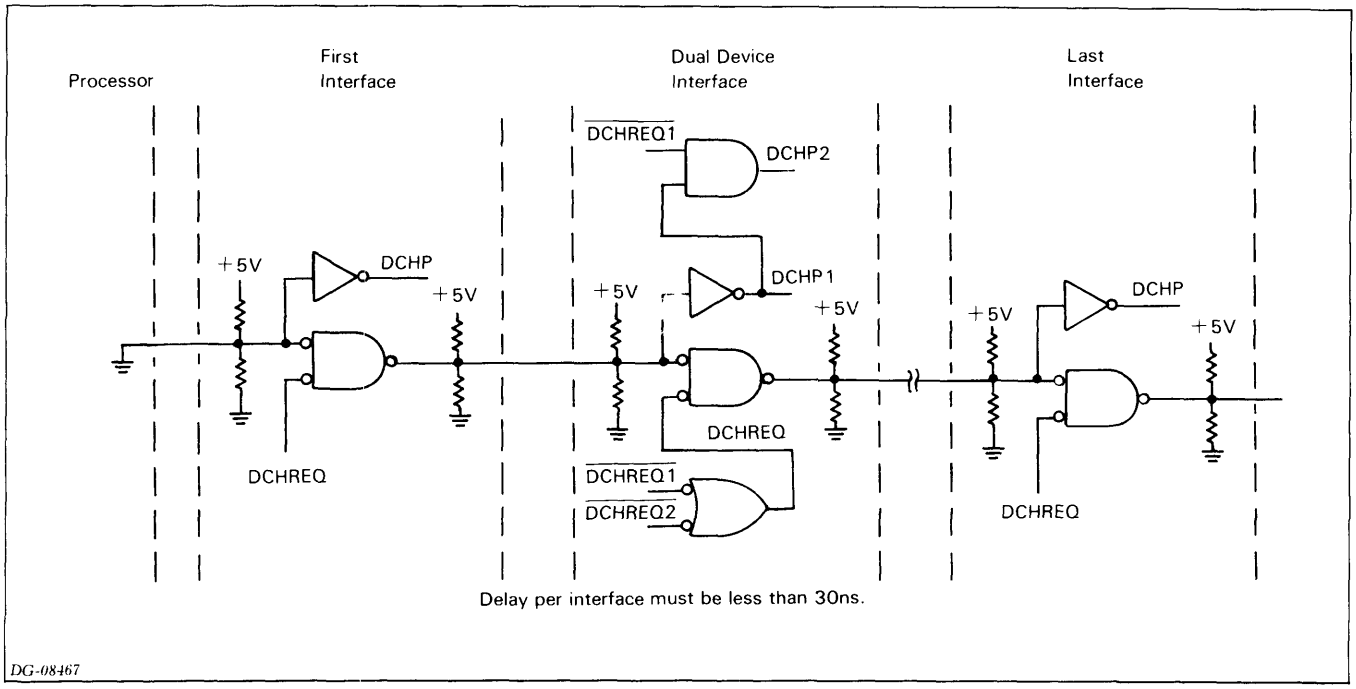


Figure 3.15 Data channel priority chain

Figure 3.16 shows the use of a flip-flop, called DCH SEL, which serves this purpose. On the leading edge of **DCHA**, this flip-flop will be set if **DCH REQ** is set and the interface is receiving an asserted **DCHP IN**. Otherwise, it will be cleared on this signal. Thus DCH SEL of the proper interface will remain set from the beginning of one data channel cycle until the beginning of the next. An interface should not respond to data channel control signals unless its DCH SEL flag is set.

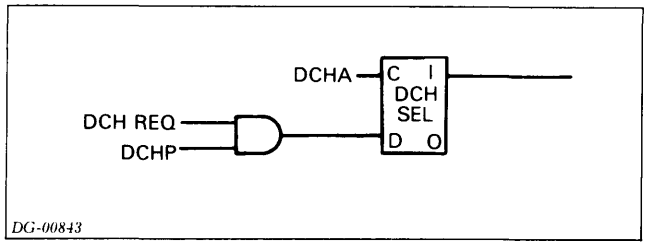


Figure 3.16 Typical data channel select circuit

While it is receiving **DCHA**, the interface whose DCH SEL flag is set should place the memory address for this transfer on the DATA lines of the I/O bus. Finally, unless back-to-back transfers are desired, the **DCHA** signal should be used to clear the DCH SYNC flag, so that **DCH REQ** will be cleared on the next **RQENB**. Figure 3.17 shows typical data channel sequencing. Figure 3.18 shows a complete typical data channel control circuit.

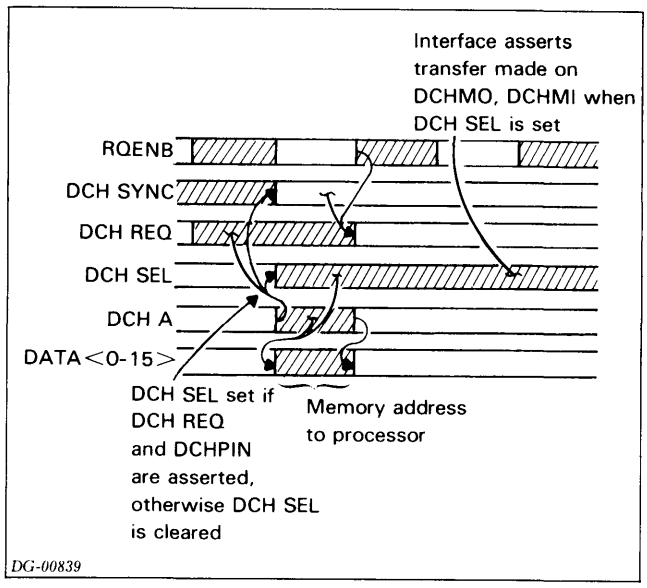


Figure 3.17 Data channel sequencing

Data Channel Map Selection

Data channel MAP selection is performed only by certain peripheral controllers in Data General computer systems that contain more than one data channel MAP. (The number and capabilities of each computer's data channel MAP is discussed in the programmer's reference manual for each computer model.) Data channel map selection occurs at data channel acknowledge time, **DATA0** and an extended map select bit selects one of the data channel maps. See Appendix C.

Data Channel Transfer Modes

As part of the interface's response to **DCHA** it should return to the central processor the mode of the transfer. The mode is designated by **DCHM0**, which should be asserted whenever **DCH SEL** (or its equivalent) is set. The modes are listed in Table 3.3, followed by a description of the transfers in each mode.

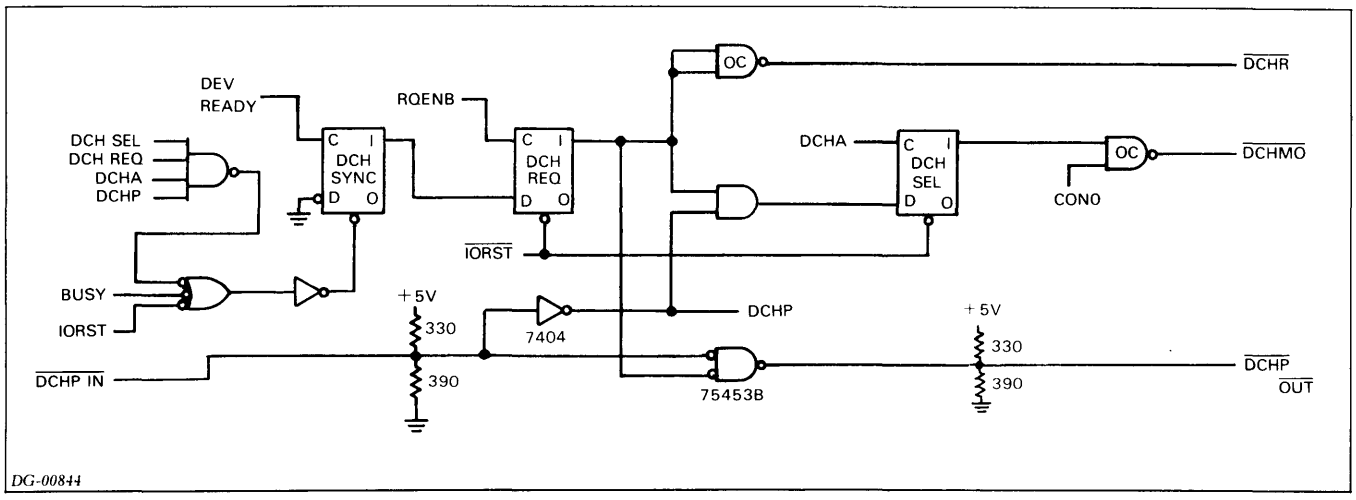


Figure 3.18 Typical data channel control circuit (complete)

$\overline{\text{DCHM0}}$	Function
0=H	Output
1=L	Input

Table 3.3 Data channel modes

Input

Following the end of **$\overline{\text{DCHA}}$**, the processor will assert **DCHI** as shown in Figure 3.19. The interface whose priority conditions are satisfied; that is, whose DCH SEL flag is set, should respond to this by asserting the DATA lines with the data word to be transferred. At the end of the **DCHI** period, the processor will load the contents of the DATA lines and write it into the previously addressed memory location.

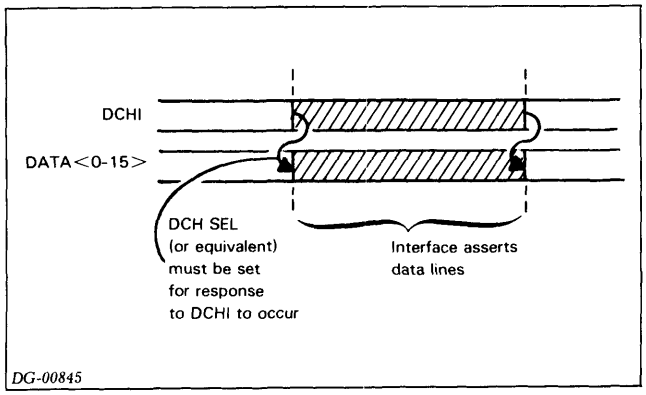


Figure 3.19 Data channel input

Output

Following the end of **$\overline{\text{DCHA}}$**, the central processor will assert the DATA lines of the bus with the data word read from the referenced memory location as shown in Figure 3.20. After allowing time for the lines to settle, **DCHO** will be asserted by the processor to signal the interface to load the data from the DATA lines. After the end of **DCHO**, the data will be dropped from the DATA lines.

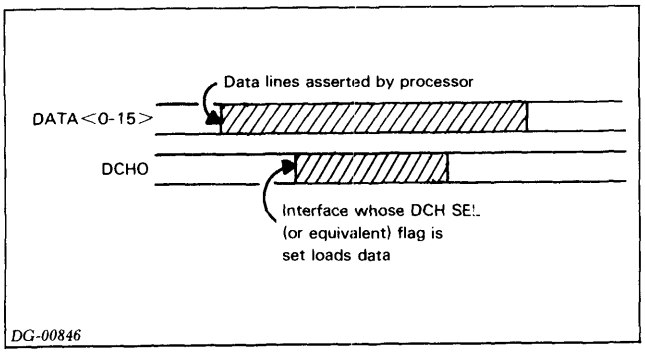


Figure 3.20 Data channel output

Examples

The two examples which follow illustrate some of the interface functions discussed earlier. The first, a switch register and relay buffer, illustrates the basic idea of input/output. The second interface, that of a paper tape punch, shows the use of the Busy/Done/Interrupt Network.

Switch Register/Relay Buffer

The switch register and relay buffer to be considered first is a very simple interface that uses neither the program interrupt nor data channel facilities. This device/interface shown in Fig. 3.21, allows the program to read three switches and to control three relays through a buffer. Of the basic control circuits discussed earlier, the only one this device has is the NAND gate to decode the device select lines. When a *Data Input A* instruction with device code 37_8 (**DIA** *ac*,37) is executed, the contents of the switches are loaded into the high-order three bits of the selected accumulator (an open switch is read as a *0*). Note the use of open collector gates to drive the data lines, and the use of a resistive voltage divider to generate standard logic levels from the switch contacts. When a *Data Output A* instruction with device code 37_8 (**DOA** *ac*,37) is executed, the contents of the high-order three bits of the selected accumulator are loaded into the relay buffer to control the three relays (a *1* from the accumulator causes the relay contact to close). Initially the contacts are open as the buffer is cleared by an *I/O Reset* instruction.

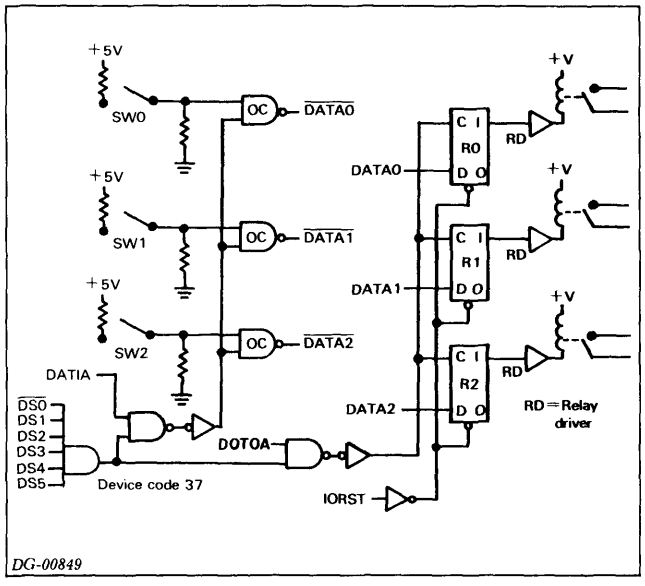


Figure 3.21 Example of simple interface

Note that in the figure many of the inputs from the bus are not in the polarities listed for bus signals. Invariably, any but the most complex interface would be mounted with a number of others on a single board. In this case, all of the interfaces on the board should share the same set of receivers, so that the board draws only one load from any given bus line. All DGC-supplied boards are designed this

way, and it is strongly recommended that the user do likewise. Only an interface for a very complex device would require an entire board.

Paper Tape Punch

The interface for the high speed paper tape punch, which is illustrated in Figure 3.22, shares a single board with the interface for an asynchronous terminal, tape reader and real time clock. The lower half of the drawing contains circuits for functions common to all of the interfaces. At the left are receivers for the data lines and other signals. At the right are networks that generate a **COMMON SELECT** signal by decoding **DS<0-2>** and generate common device code digits for the *Interrupt Acknowledge* instruction. The codes are 10_8 through 14_8 so accumulator bits 10 and 11 are 0, bit 12 is 1, but bit 13 is 1 only for the clock. (Both nets can be jumpered so the codes can be 50_8 through 54_8 instead.) Across the bottom is a chain that receives **INTP IN**, generates an individual acknowledgment signal for each device, and passes the priority signal along the bus only if no device on the board has an INT REQ flip-flop set.

In the middle are the standard circuits specifically for the punch. At the left is the gate that determines when the punch is called by decoding **DS<3-5>** and the **COMMON SELECT** signal. At the right is the network that places the low order two bits of the punch code on the bus when an interrupt is acknowledged for it. The remainder of the center section is taken up by the state/interrupt network which is as described earlier (in this specific case, **PUN INT DISABLE** is controlled by mask bit 13).

The upper part of the drawing contains the 8-bit punch buffer and logic to turn on the solenoid drivers in the punch at the appropriate time. A *Data Output A* instruction that selects the punch (**DOA** *ac*,**PTP**) loads the buffer from bits 8-15 of an accumulator. If BUSY is set, the advent of the proper position in the punch operating cycle triggers a one-shot that allows 1's in the punch buffer to drive the lines to the punch for 4.5 milliseconds. Note that the left-most driver always goes on - it punches the feed hold. The termination of the delay generates a completion signal that clears BUSY and sets DONE.

At the left is an input from the punch feed switch. Holding this switch on keeps the buffer clear and allows every synchronizing signal from the punch to trigger the one-shot, and thus produce a length of blank tape (i.e., tape with only feed holes punched).

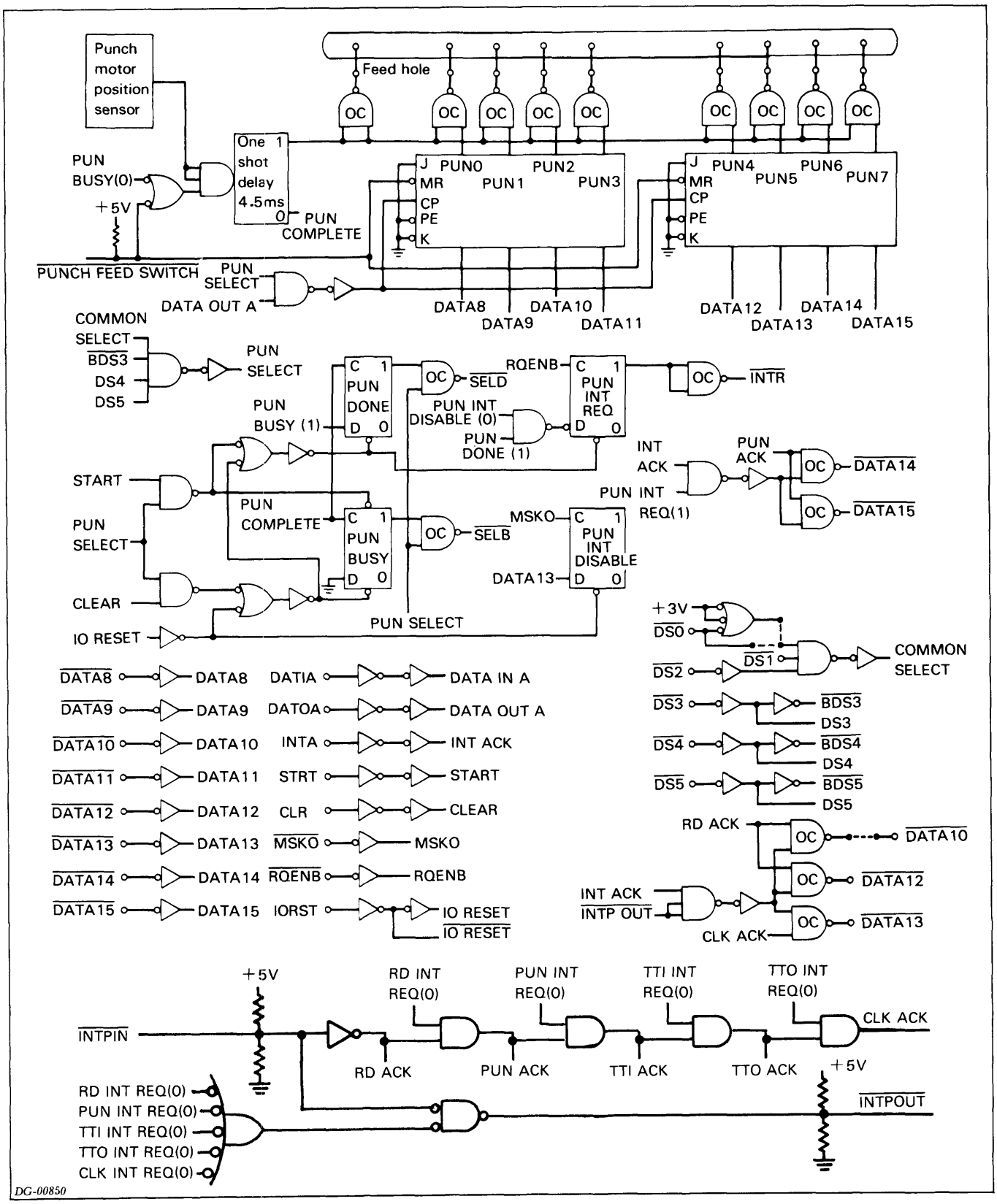


Figure 3.22 Example of paper tape punch interface

Chapter 4
I/O BUS TIMING AND ELECTRICAL SPECIFICATIONS

Timing

The following figures show the timing characteristics of the various I/O bus functions. There are some minor timing variations among the different processor types, but these can be neglected if an interface is properly designed. In order to facilitate the implementation of such a compatible interface, these timing diagrams show the limiting timing characteristics of all NOVA and ECLIPSE computers as a group. Timing data for a specific machine can be found in its respective Technical Manual.

The timing diagrams, Fig. 4.1 through Fig. 4.15, show two types of timing information. The times shown for the processor originated functions (e.g., **DCHI** pulse) are the times for which the interface must be designed. The times between processor bus functions and the interface responses on the bus to those functions are shown as the maximum allowable times. An interface designed to operate with these function times and maximum response times will operate with any NOVA or ECLIPSE processor. All times are in nanoseconds. All times are measured at the backpanel pins of the interface slot.

NOTE: *Signals that have no maximum duration or delay may have an 'indefinite' duration or delay.*

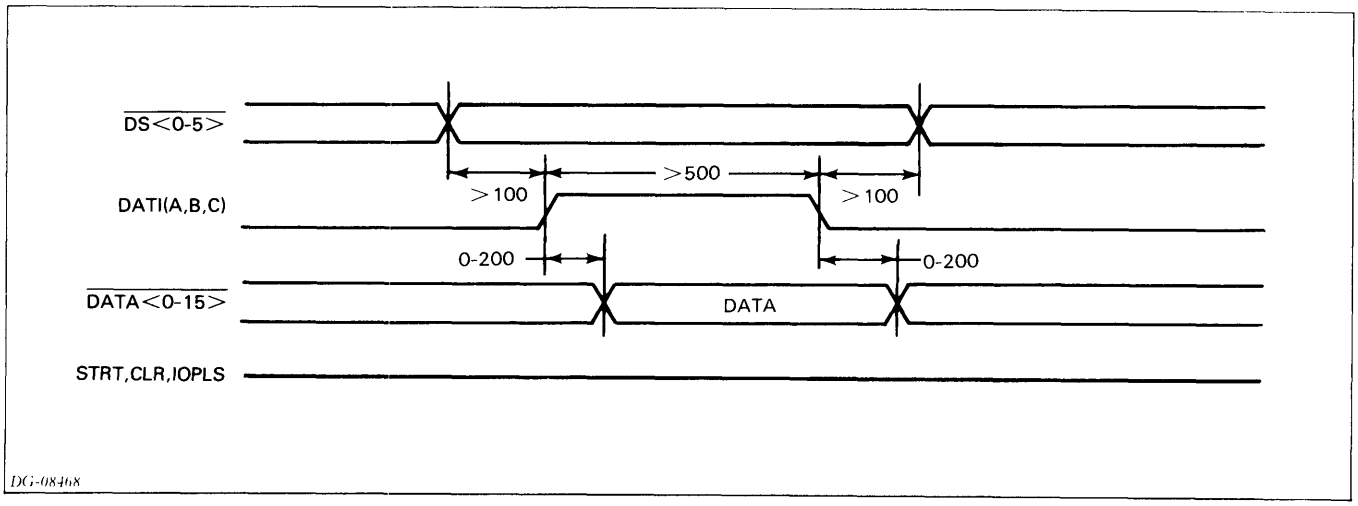


Figure 4.1 Programmed I/O input without S, C, or P pulse

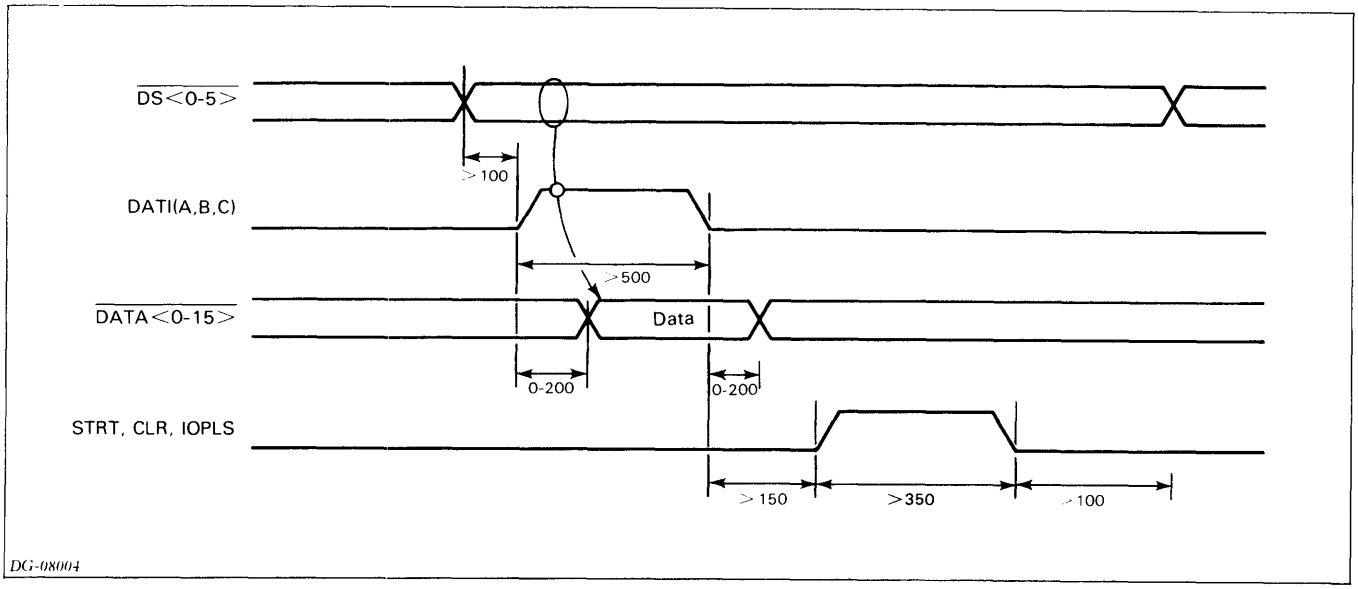


Figure 4.2 Programmed I/O input with S, C, or P pulse

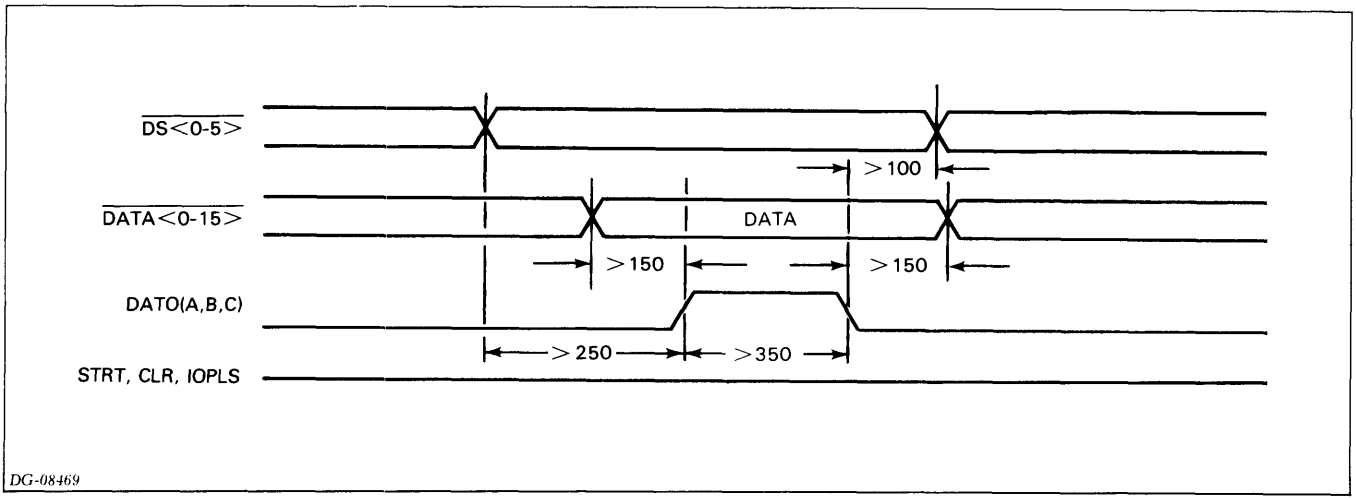


Figure 4.3 Programmed I/O output without S, C, or P pulse

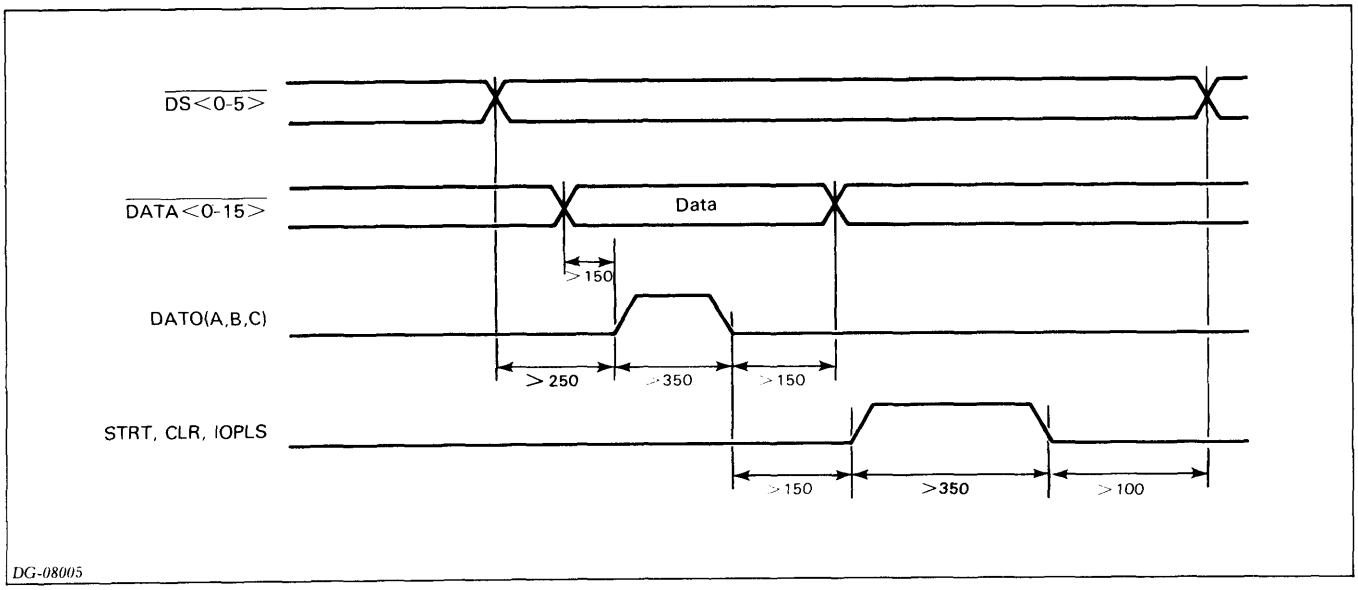


Figure 4.4 Programmed I/O output with S, C, or P pulse

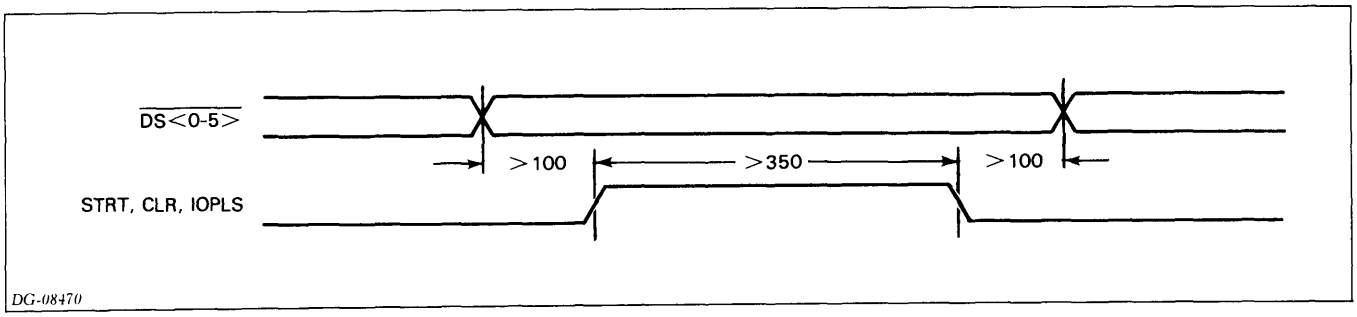


Figure 4.5 Programmed I/O no data transfer with S, C, or P pulse

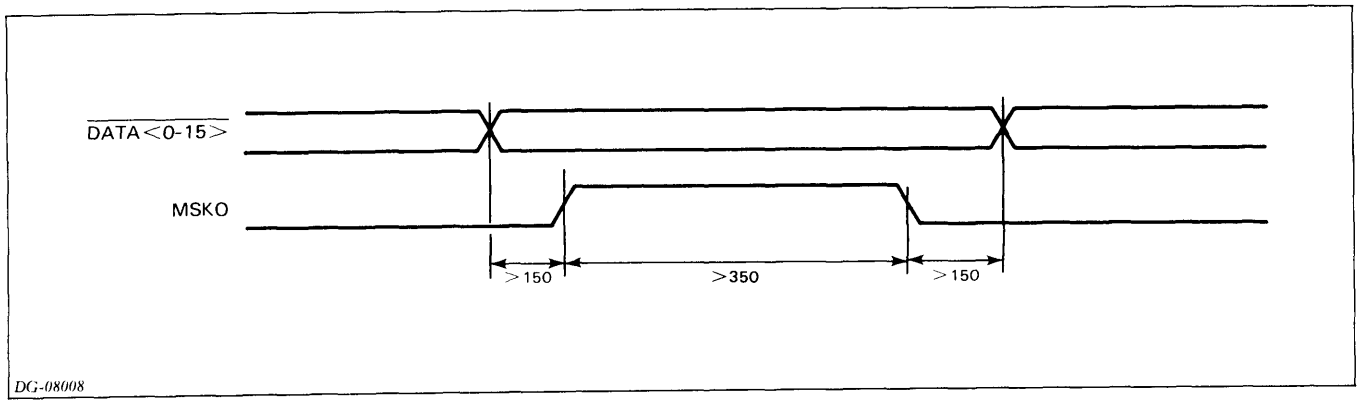


Figure 4.6 Mask out

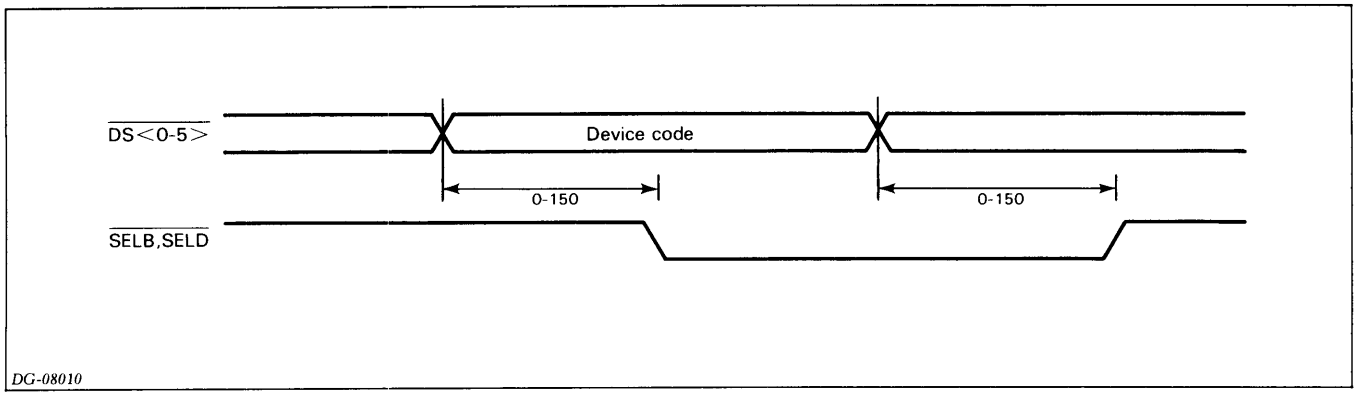


Figure 4.7 Skip test

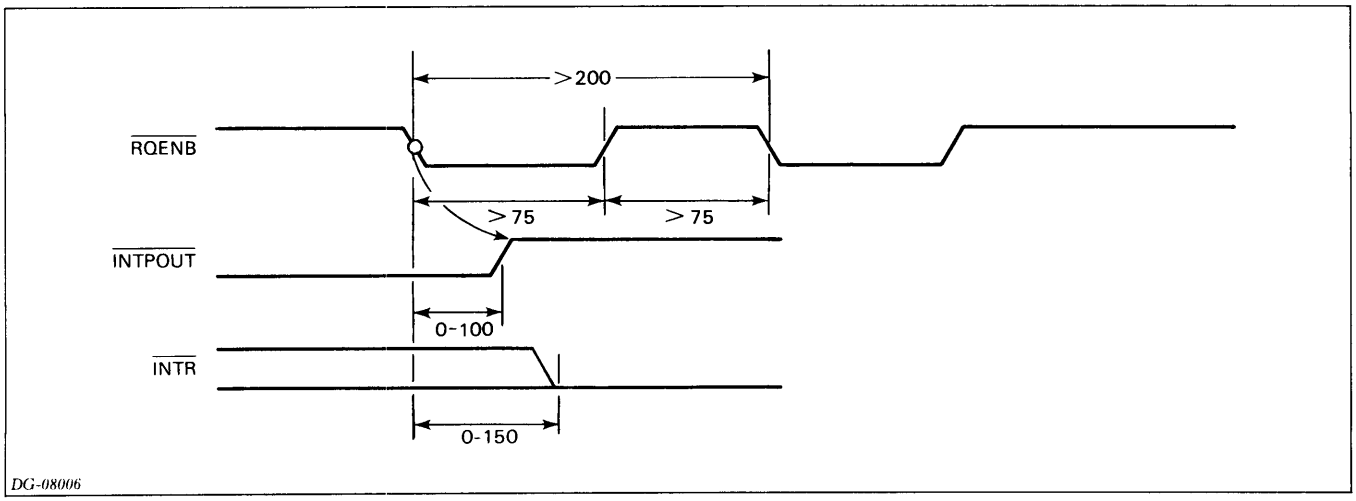


Figure 4.8 Program interrupt

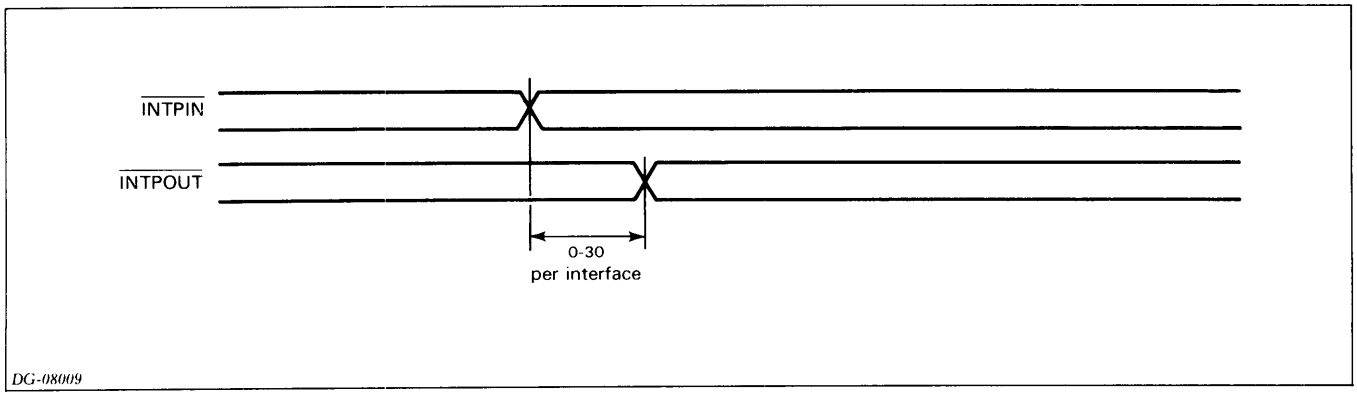


Figure 4.9 Interrupt priority chain

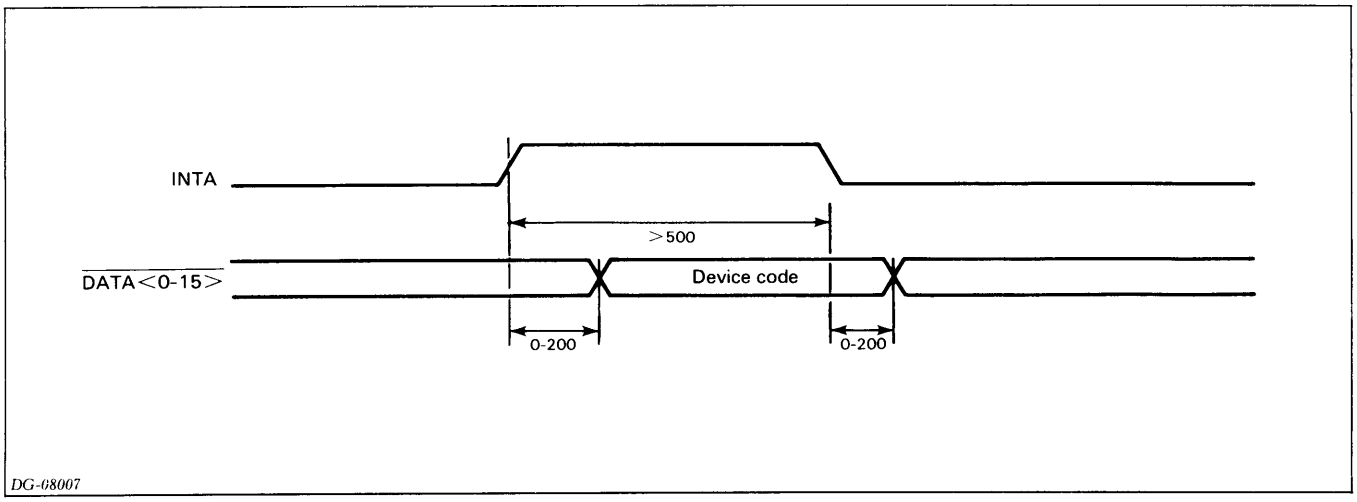


Figure 4.10 Interrupt acknowledge

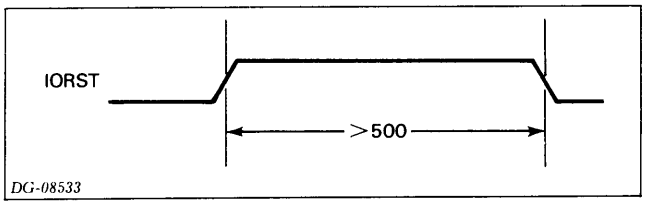


Figure 4.11 I/O reset

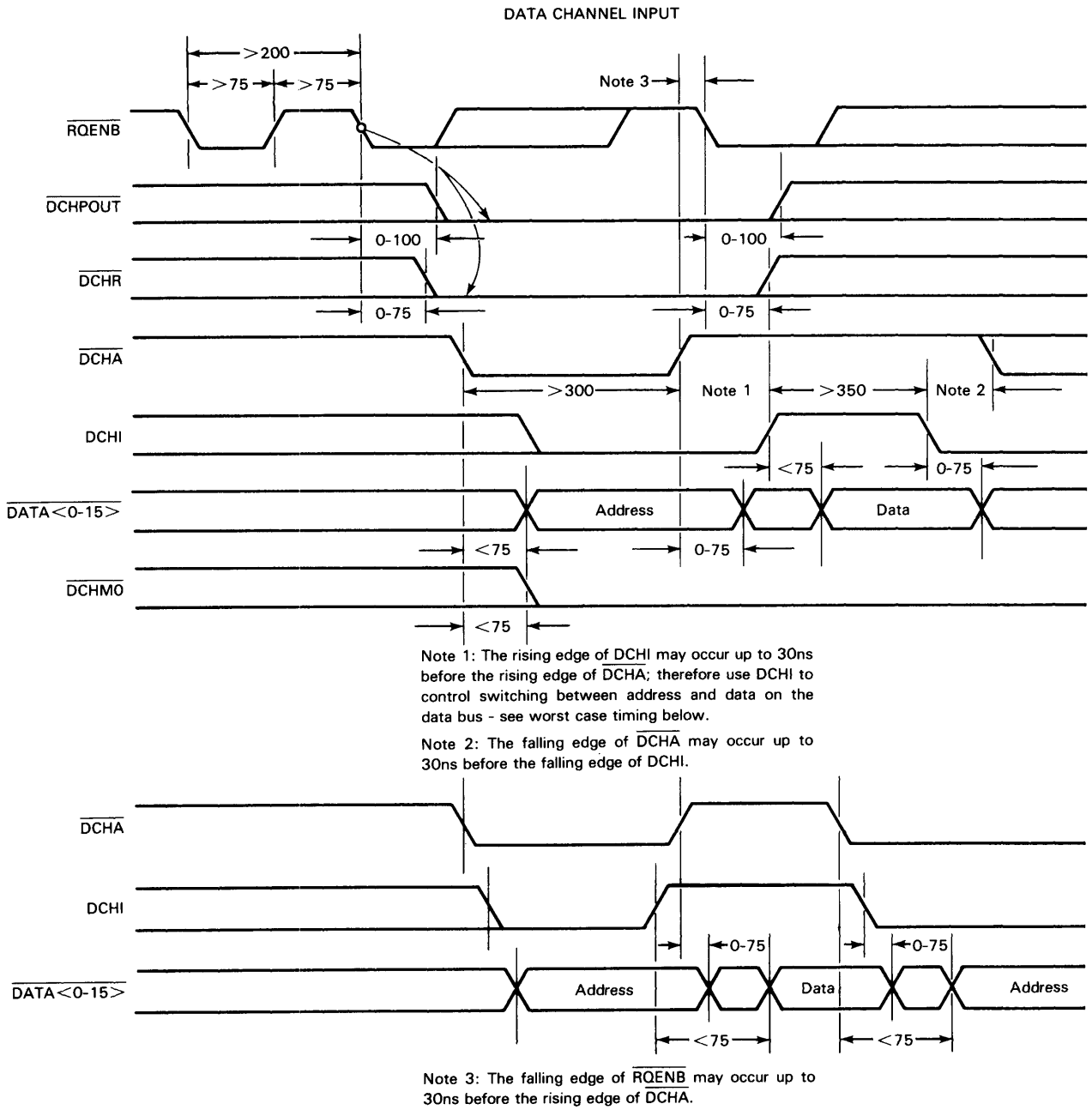


Figure 4.12 Data channel input

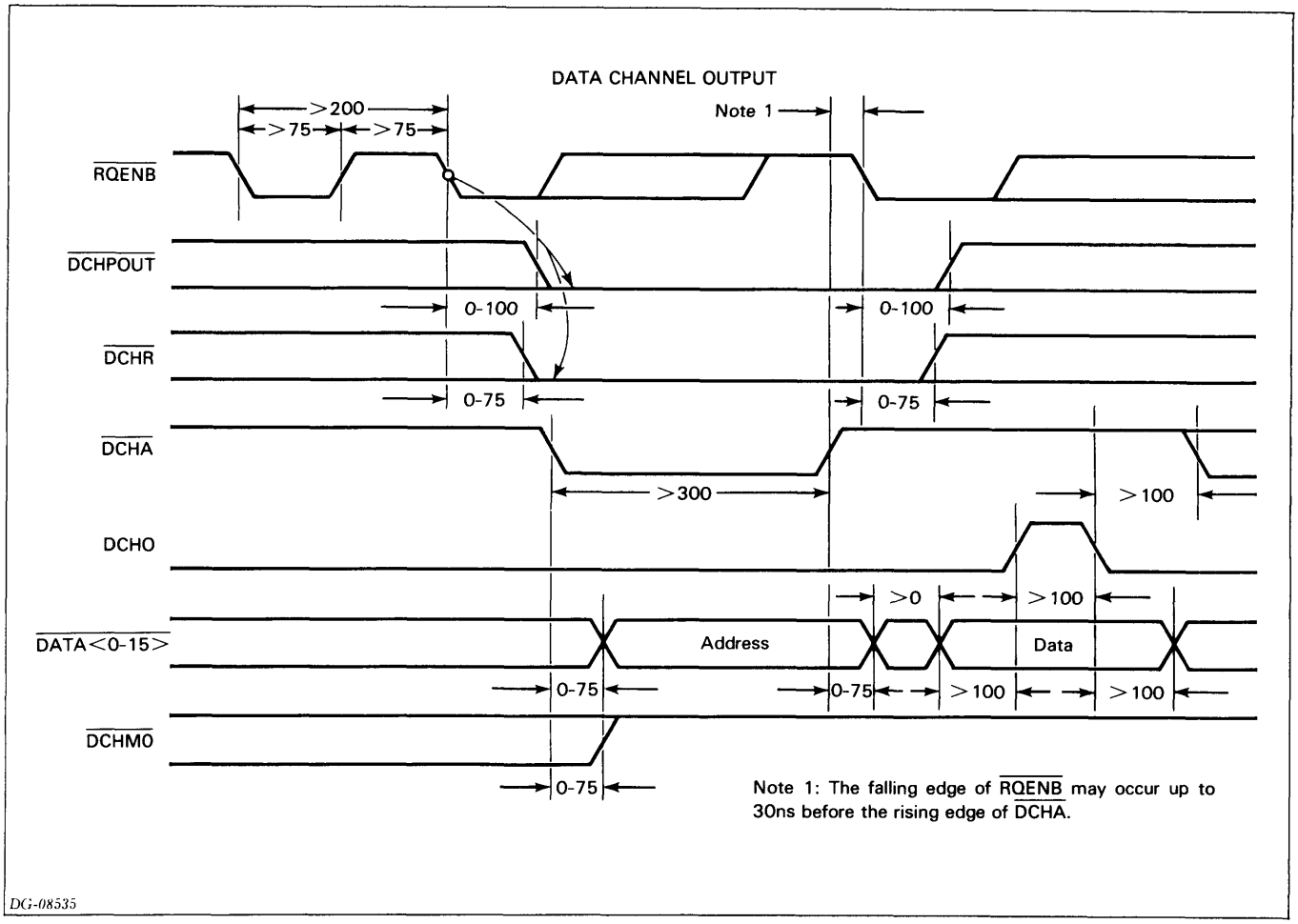


Figure 4.13 Data channel output

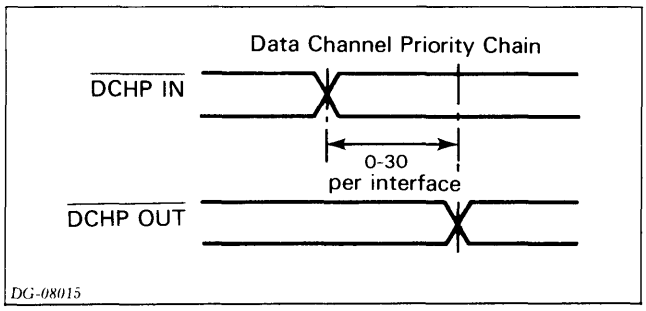


Figure 4.14 Data channel priority chain

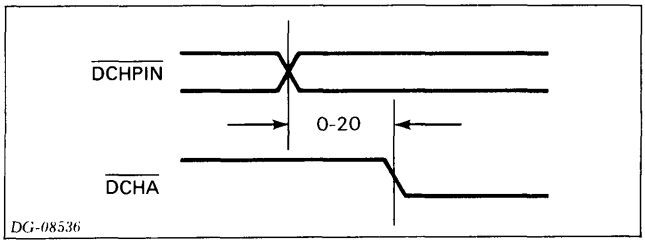


Figure 4.15 Priority to acknowledge delay

Signal Levels

All of the signals on the I/O bus are digital signals and thus have two electrical states. Any voltage between 0 and +0.7 volts is considered low while any voltage greater than +2.0 volts is considered high. The normal voltages for low and high levels are 0.5 and +2.7 to +3 volts, respectively. The relation between the electrical level of a signal and its logical value depends on the particular signal in question. Signals that are asserted when low are identified by a bar over the signal name, as explained in Chapter 3.

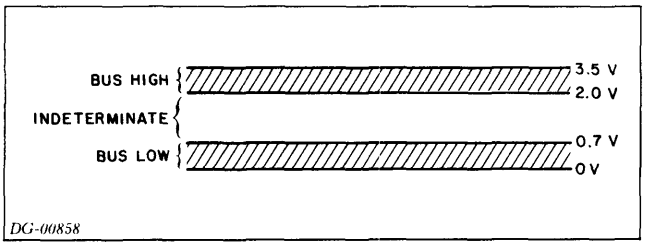


Figure 4.16 Bus electrical levels

Drivers

All signals that may be generated by the interface plus several other signals normally float high and are driven by pulling them towards the 0 volt level. This is done by causing enough current to flow through the line's terminating resistor from the +5 volt source so that the full five volts is dropped across this resistor. This is usually accomplished by using the collector of an NPN transistor, as shown in Fig. 4.17. Generally, a discrete transistor is not used for this purpose; rather the output of an open-collector TTL gate would be used. In any case, the device must be able to sink at least 70mA with a saturated collector emitter voltage not greater than 0.5 volts. The Data General #100-000117 or #100-000627 drivers are recommended for this application.

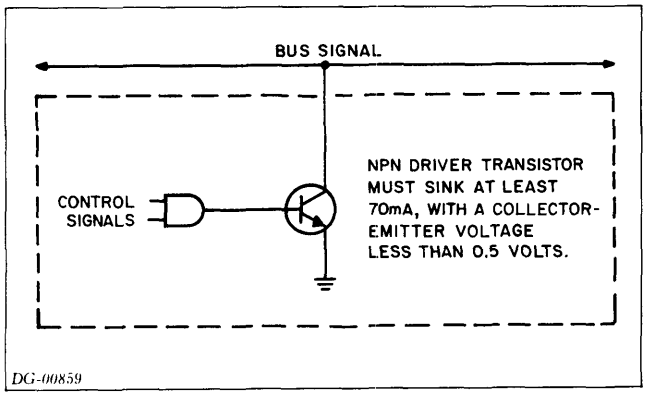


Figure 4.17 Typical line driver

Receivers

Receivers for the I/O bus signals may be either standard TTL gates or special purpose interface receivers. The net load that can be attached to the bus must be limited to 1.6mA per signal per board.

Noise

In any high-speed design, there should be a concern for noise generation and protection. There are many types of noise that should be of concern. For example crosstalk between adjacent high-speed signals, ringing of improperly or un-terminated signals, effects of ac (capacitive) loading, and transmission line effects are all noise sources. Several suggestions come out of these concerns:

- Use noise resistant receivers and filters for all high-speed signals especially those signals that travel a long distance.
- Keep all I/O bus signals very short. That is, keep the receivers and drivers for those signals physically close to the edge connector where they come off the backpanel. Lay out all bus signal etch to minimize capacitance and crosstalk.
- Synchronize all data, address, or status information so it does not change once enabled onto the I/O bus.
- Use only one receiver for each signal. That is, if a signal is used at more than one point on a single board or interface assembly, that board should nevertheless cause only one load on the bus for that signal.
- These signals should be treated as transmission lines; their physical as well as electrical characteristics are important.
- The effect on signal delays and pulse widths caused by bus receivers, transmitters, repeaters, and filters must be taken into account.

The logical, electrical, and mechanical design of bus interfaces and interconnects must minimize the generation of, and vulnerability to, electrical noise. Nevertheless, the noise immunity of 7400-series logic circuits may not be adequate, in all cases, to receive I/O bus signals. In the case of the signals listed in Table 4.1, the use of a filter is recommended to protect logic gates from bus noise. All other signals should use Schmitt trigger receivers.

DATIA	DATOA	$\overline{\text{DCHA}}$	INTA	OVFLO
DATIB	DATOB	DCHI	IOPLS	
DATIC	DATOC	DCHO	IORST	STRT
CLR	$\overline{\text{MSKO}}$			

Table 4.1 Signals to be filtered

For example, the circuit of Fig. 4.18 is recommended for receiving active-high signals suffering leading-edge noise.

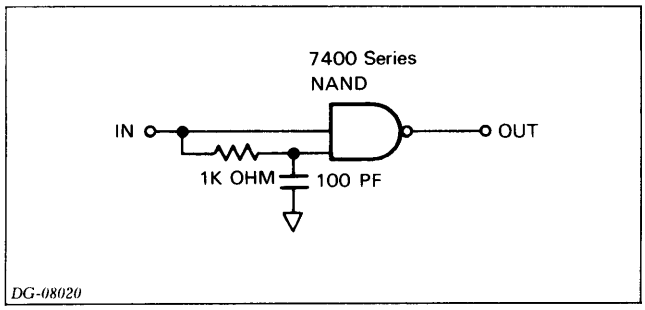


Figure 4.18 Bus receiver

Chapter 5
CONNECTIONS, CONNECTORS AND TERMINATORS

Introduction

This chapter explains how to connect controllers in a NOVA or ECLIPSE line computer chassis to adapters and devices outside the chassis, how to install external I/O bus cables, and how to terminate the I/O bus.

The chapter begins with a discussion of how connections to DGC computers are made. Each computer connection scheme is described in detail in the pages that follow. For quick reference, Table 5.1 gives the computer model and the page on which its connection scheme is described.

Table 5.1 Computer models

Cables and Connections

A controller must be connected both to the I/O bus and to the device it controls. The I/O bus brings command information and data from the computer to the controller. The controller must connect to the device through a series of cables and connectors.

The connections to the device are made in two parts: first through an internal cable, then through an external (or device) cable. The internal cable brings the signals from the controller to a convenient connection panel. The external cable takes the device signals from the computer chassis directly to the device.

The internal cable and its attached connectors are of four types (shown in Figures 5.1 through 5.4): a pin-connector type cable (plugs directly onto backpanel pins); a wire-wrapped socket-connector type cable; a wire-wrapped edge-connector (paddleboard) cable; or a push-on edge-connector (paddleboard) cable. Some computer models also have one or more integral internal cables (part of the backpanel). These are used for either a controller to device cable or for an I/O expansion chassis cable.

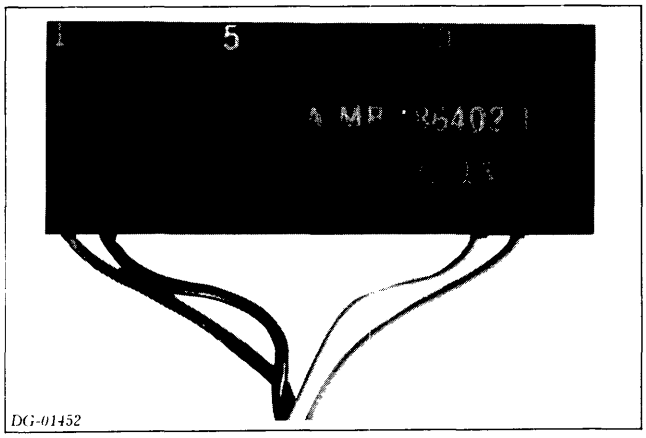


Figure 5.1 Typical pin type connector

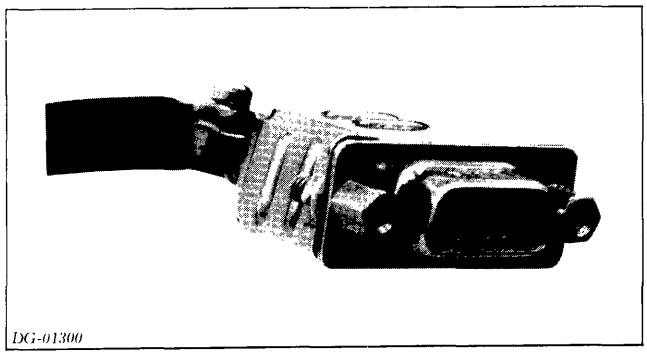


Figure 5.2 Typical socket connector

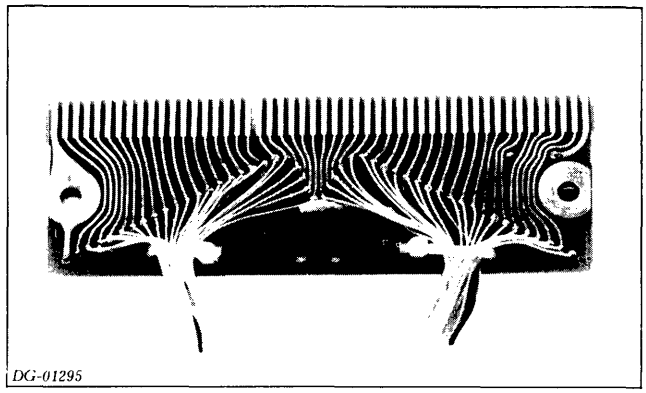


Figure 5.3 Typical wire-wrapped edge-connector

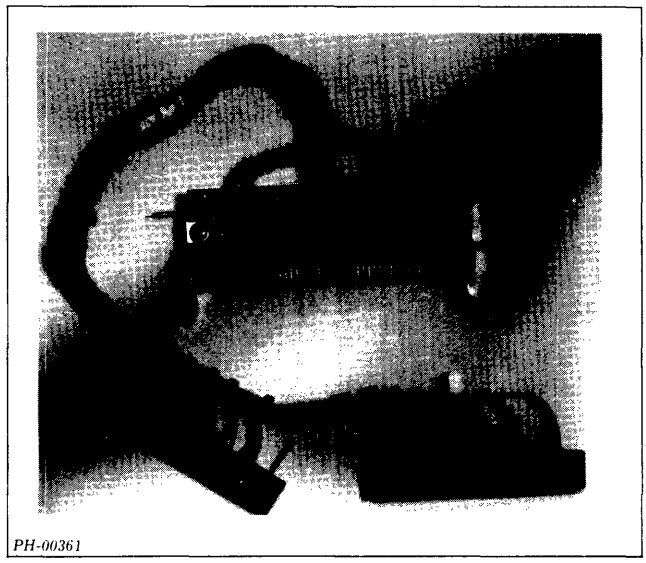


Figure 5.4 Typical push-on edge-connector

The external cable is typically a 50 wire round (or bundled) cable or a flat ribbon cable. This cable is what directly connects to the device (e.g., disk drive). It will use either a cannon-type connector for that type of internal cable, or an edge-connector for that type internal cable (see Figure 5.5).

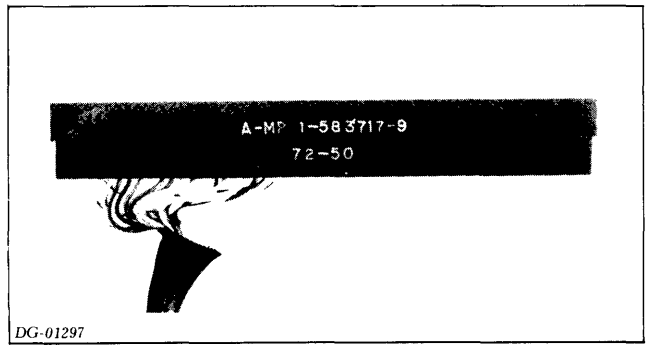


Figure 5.5 Typical edge-connector external cable

Backpanel Connections

Connections are made to a printed circuit board in a NOVA or ECLIPSE line chassis via pins on the backpanel. These pins are extensions of the contacts in the two 100-pin female edge connectors, into which the board is inserted. The pins in each connector are arranged in two rows of fifty pins each, one row making contact with fingers on the component side of the board and the other row making contact with fingers on the solder side of the board.

Backpanel pins within a slot are designated by a letter indicating one of the two female edge connectors and a two-digit, decimal number indicating the position of the pin within the connector. The letters "A" and "B" designate the female connectors. The odd numbered pins make contact with the component side of the board, and the even numbered pins with the solder side. (See Figures 5.6, 5.7, and 5.8.)

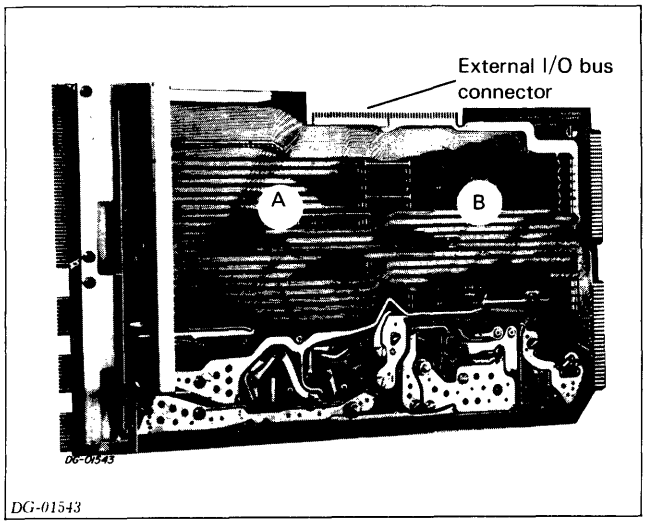


Figure 5.6 Backpanel (pin side)

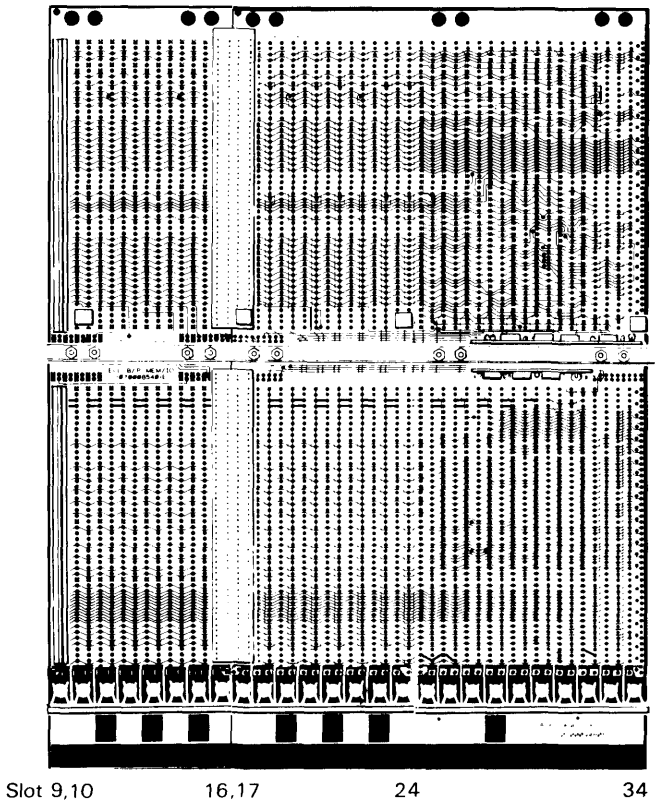


Figure 5.7 Backpanel connectors, vertical boards (pin side)

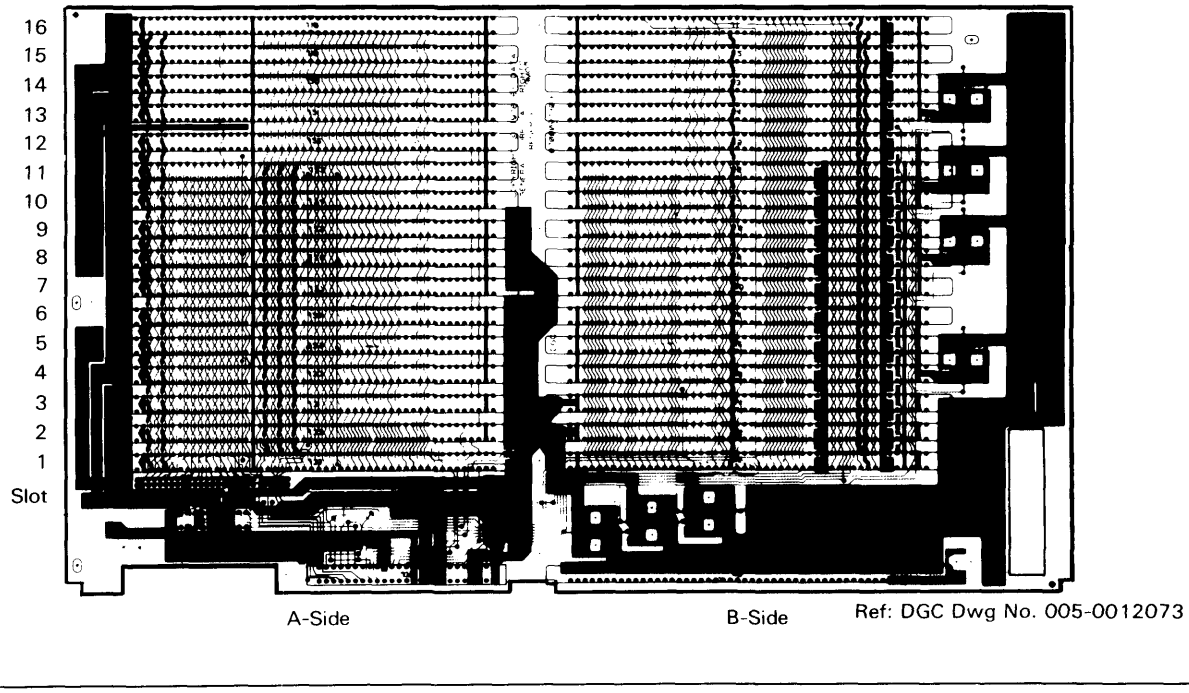


Figure 5.8 Backpanel connectors, horizontal boards (pin side)

I/O Bus Connections

The I/O bus carries signals which contain very high frequency components in their rising and falling edges. Any cable carrying these signals must be treated as a high-frequency transmission line. Within the computer chassis itself, the I/O bus is carried in etch on the backpanel; line lengths are short enough so that the propagation times are low compared to the rise time of the signals. Once the I/O bus is brought out of the chassis via cable, line lengths are extended so that reflections, settling times, and crosstalk become significant problems.

Whenever possible, a controller should be mounted inside a main or expansion chassis. The next chapter describes prefabricated boards available from Data General Corporation for this purpose. However, when the controller must be mounted outside these chassis, it should be connected to the I/O bus via the 95 ohm, twisted pair I/O bus cable sold by Data General Corporation. The total length of the I/O bus must never exceed 50 feet, including etch and wires within chassis.

I/O Bus Connections Within a Chassis

Minimal I/O bus cabling is required when a controller is installed in a NOVA or ECLIPSE line chassis because most I/O bus signals are carried to every slot available for I/O controllers via etch on the backpanel. However, some connections may be required in order to maintain the integrity of the priority signals **INTP** and **DCHP**.

Because the signals **INTP** and **DCHP** are chained from one controller to the next, jumpers must carry these signals across any unused slot or user-manufactured board that does not properly pass them along the bus. (See Chapter 3) **INTP** is jumpered across a slot by connecting pins A95 and A96 of the slot. Similarly, **DCHP** is jumpered across a slot by connecting pins A93 and A94 of the slot.

I/O Bus Connections Outside a Chassis

Controllers that are not mounted in a NOVA or ECLIPSE line chassis must be connected to the I/O bus by an external I/O bus cable. (Some models do not allow an external I/O bus; see each particular model.) One end of the I/O bus cable is connected to the controller, and the other end to an edge or socket connector on the computer chassis (the external I/O bus connector). The I/O bus is connected to the external I/O bus connector by wires or etch on the backpanel. Appendix C shows the assignments of I/O bus signals to pins on the backpanel and to pins in the external I/O bus connector.

Cabling to an Adapter or Device

A controller in a NOVA or ECLIPSE line chassis is connected to an adapter or device outside the chassis by an internal cable and an external cable. The internal cable, composed of etch or wires, connects pins in one slot on the backpanel with pins in an edge or socket connector mounted on the back or side of the chassis. One end of the external cable is then plugged into that edge or socket connector and the other end is plugged into the adapter or device. Appendix E shows which backpanel pins are used to connect to the I/O bus, and which pins are available to connect to external devices or adapters.

Cabling to User Designed Interfaces

When an internal cable must be installed for a user-designed interface, it is recommended that DGC type 4192 be used for wire-wrap, and DGC No. 005-012496 for plug-on. These are general purpose internal cables, compatible with many standard Data General interfaces. They are supplied with the proper connector for the machine specified. The 4192 cable comprises fifty wires, each of which is tagged with a backpanel pin number. When each wire is connected to the pin designated by the attached tag, that cable will be compatible with the etched connector of slot 9 of the NOVA 820, 1200 and 2/10 backpanels. The 005-012496 cable also comprises fifty wires that are installed in connectors which can be plugged directly onto the required backpanel slot. The correlation between backpanel pins and connector pins for these cables are shown in Table 5.2.

Edge Connector	Socket Connector	Backpanel Side Pin Number
A thru AF	Shell	GND
1	2, 4	GND
2	31	A92
3	30	A91
4	19	A78
5	18	A77
6	17	A76
7	16	A75
8	15	A73
9	14	A71
10	13	A69
11	12	A67
12	11	A65
13	10	A63
14	9	A61
15	8	A59
16	7	A57
17	5	A47
18	6	A49
19	20	A79
20	21	A81
21	23	A84
22	22	A83
23	25	A86
24	24	A85
25	27	A88
26	26	A87
27	28	A89
28	29	A90
29	32	B6
30	33	B11
31	34	B13
32	35	B15
33	36	B19
34	37	B23
35	38	B25
36	39	B27
37	40	B31
38	41	B34
39	42	B36
40	43	B38
41	44	B40
42	45	B48
43	46	B49
44	47	B51
45	48	B52
46	49	B53
47	50	B54
48	51	B67
49	52	B69
50	3	+5V

Table 5.2 General purpose internal cable, DGC model no. 4192

Machine Specific Connections

The following diagrams illustrate the different internal cables used in Data General computers. Socket connectors are used on the NOVA and SUPERNOVA, NOVA 800, 830, 840, 1200 Jumbo and 800 Jumbo computers. All other machines use edge type connectors. On all edge connector machines except the ECLIPSE MV/8000, the first TTY cable is plugged directly onto the backpanel through a pin connector similar to DGC model 1051G. On all socket connector machines, the signals for the TTY are carried through backpanel etch to a socket connector. All other device cables plug onto connectors mounted at the rear of the backpanel. These connectors, in turn, are either wire-wrapped or plugged onto the appropriate backpanel pins.

In the following diagrams we use P-numbers to distinquish the various positions for cables and connectors.

NOVA and SUPERNOVA

Figure 5.9 shows the connector configuration for the NOVA computer; cabling for the SUPERNOVA computer is quite similar. The internal cables are wire-wrapped to pins on the backpanel. These cables are brought under the chassis to the rear, where the connectors are attached to a mounting area on the power supply. The NOVA and SUPERNOVA computers use socket type connectors for their device cables.

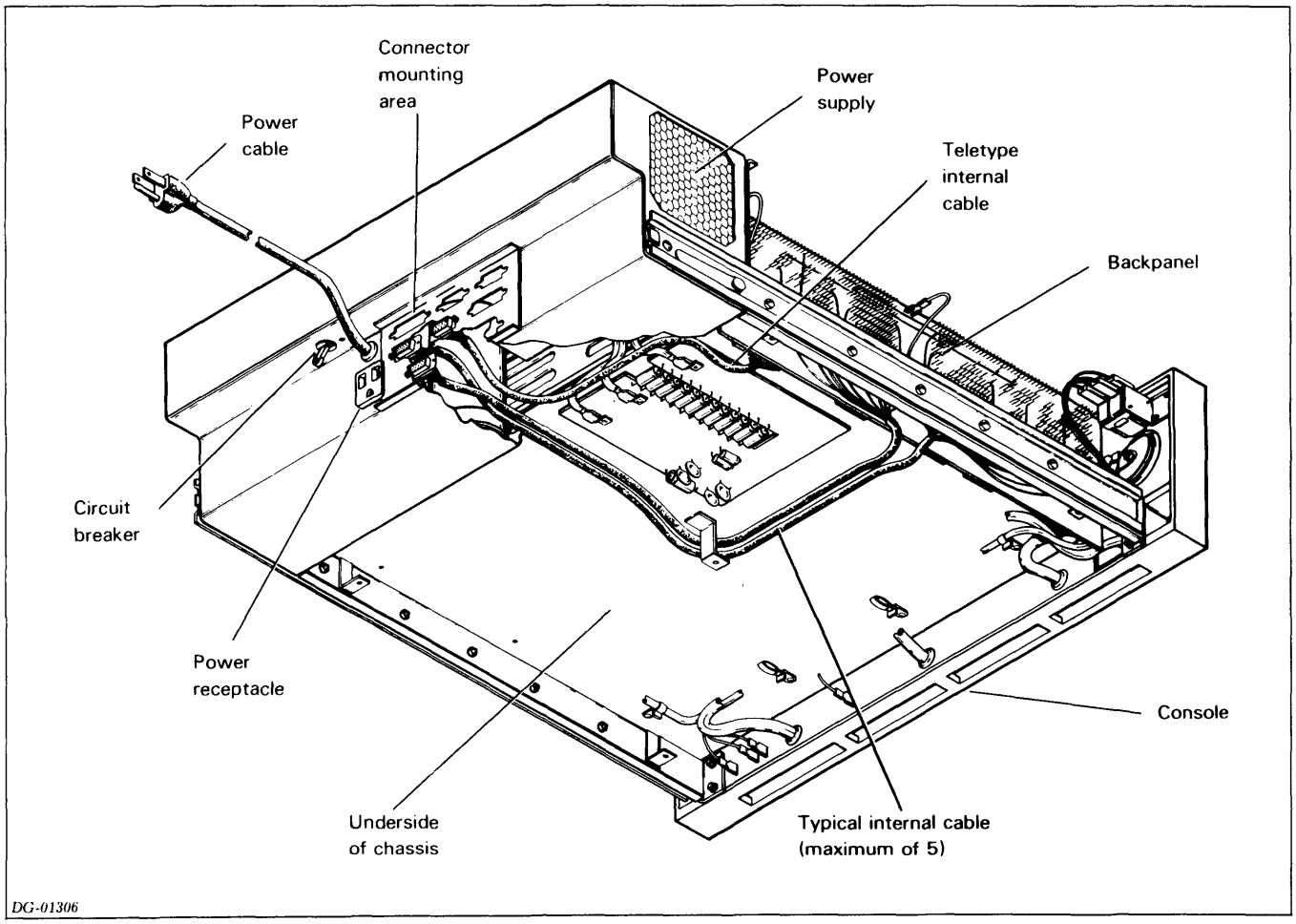


Figure 5.9 NOVA and SUPERNOVA connector configurations

NOVA 800, 1200, 1200 Jumbo, 830, 840 Computers and Their Expansion Chassis

Socket connectors are used exclusively on these machines. Of the four connectors mounting locations at the bottom of the connector bracket, P2, P3, and P4 are assigned to the terminal, paper tape punch, and paper tape reader respectively. Socket P2 is connected by etch on the backpanel to slot three. Internal cables for the paper tape reader and punch may be connected by attaching pin connectors from the external socket to P6 and P8, mounted at the bottom of the backpanel. P6 and P8 are also connected by etch on the backpanel to slot three. Similarly, an I/O bus socket installed in one of the horizontal positions would be connected to the appropriate locations on the backpanel. To connect other device cables, an internal cable should be run from an external cable connector to the appropriate backpanel pins.

NOTE: *These machines use the 4050 or 4051 junction box instead of the 4083 connector panel used on edge connector type machines.*

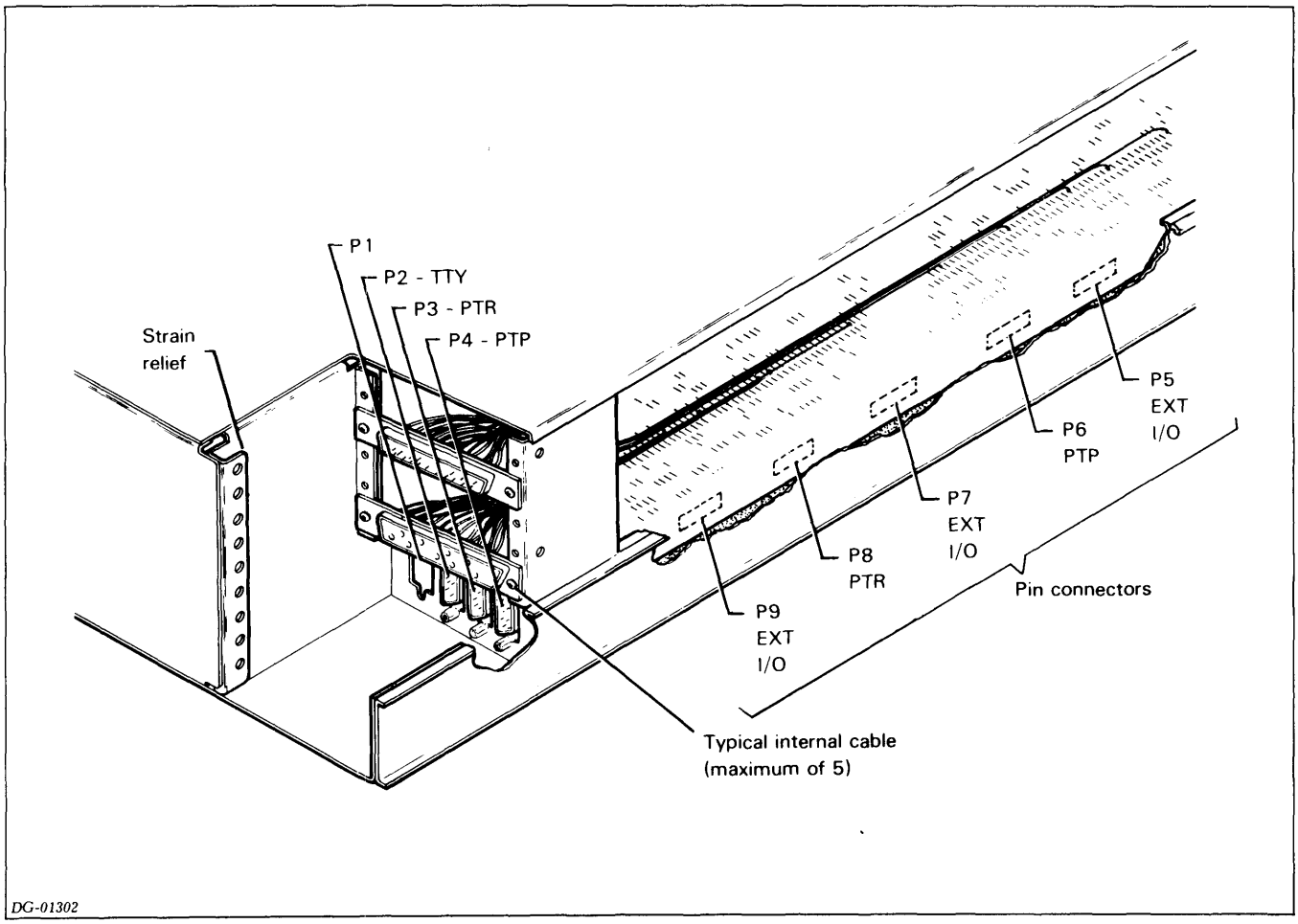


Figure 5.10 Connectors for NOVA 800, 1200, 1200 JUMBO, 830, and 840 computers

NOVA 1210 Computers

The TTY cable is plugged onto the backpanel at P2, on slot 3. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers on this fifty-pair connector. Device cables other than that of the TTY are plugged onto connectors mounted, as needed, at the rear of the backpanel and wire-wrapped (via internal cables) to the proper backpanel pins. P4 and P5, each a ten-pair edge connector, and P6, a thirteen-pin connector, make up a single unit which is installed only when the paper tape reader, the paper tape punch, or a second teletype is installed. P7, a fifty-pair edge connector, is mounted on standoffs beside P4-6, and wire-wrapped to backpanel pins when it is needed. The necessary connectors are furnished when DGC standard interfaces are purchased.

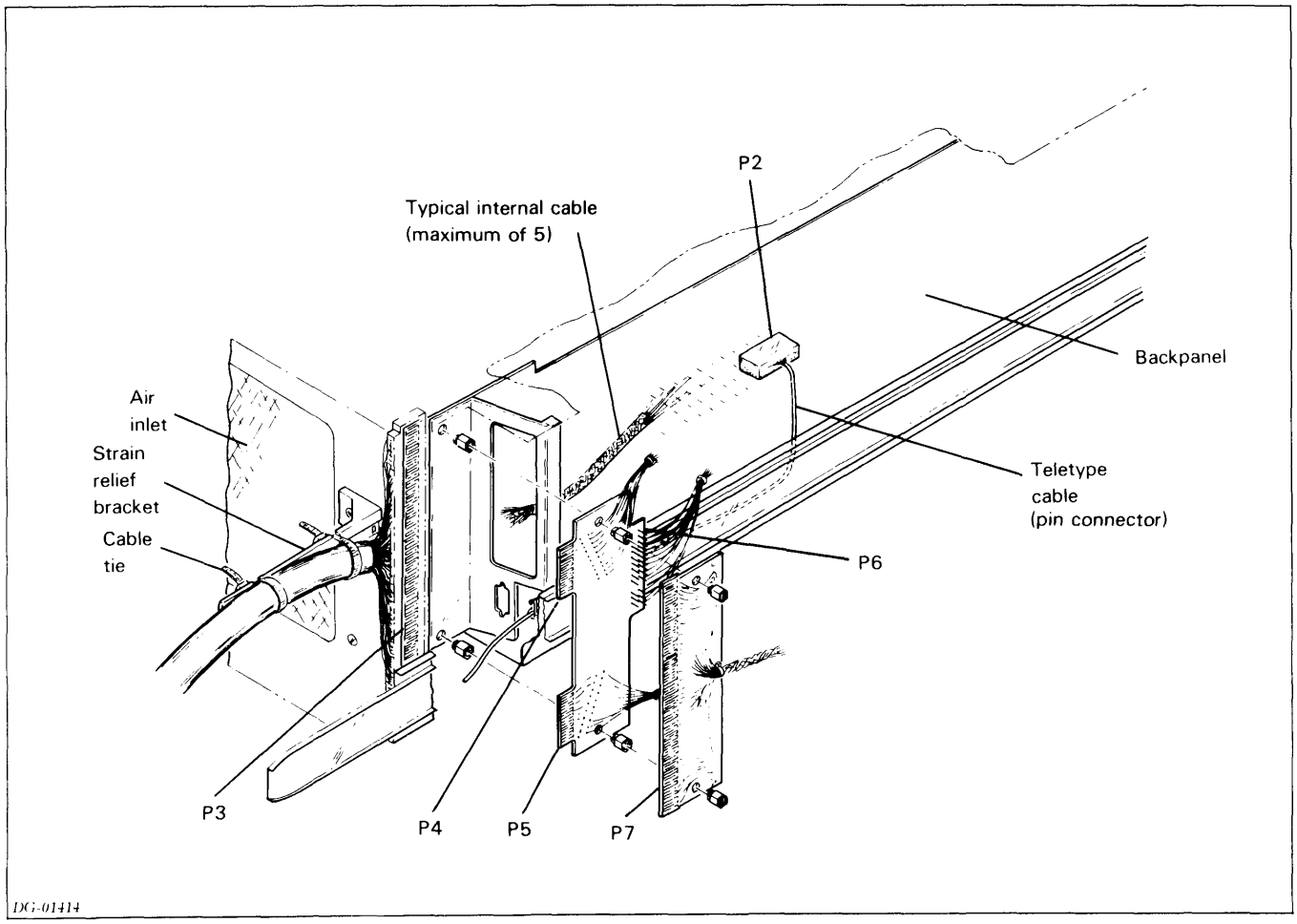


Figure 5.11 NOVA 1210 connectors

NOVA 820, 1220 Computers and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2, on slot 3. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. P4 is a fifty-pair edge connector whose fingers are permanently connected to the backpanel pins of slot 9. Similarly, P5, P6, and P7 (ten-pair connectors), are permanently connected for use when the paper tape reader or punch, cassette, or EIA interfaces are installed in slot 3.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. P8 through P12 may be either connector shown in Figure 5.13. If required, socket-type connectors can be installed in the spaces provided in the I/O bracket, shown in Figure 5.14. The necessary connectors are furnished when DGC standard interfaces are purchased.

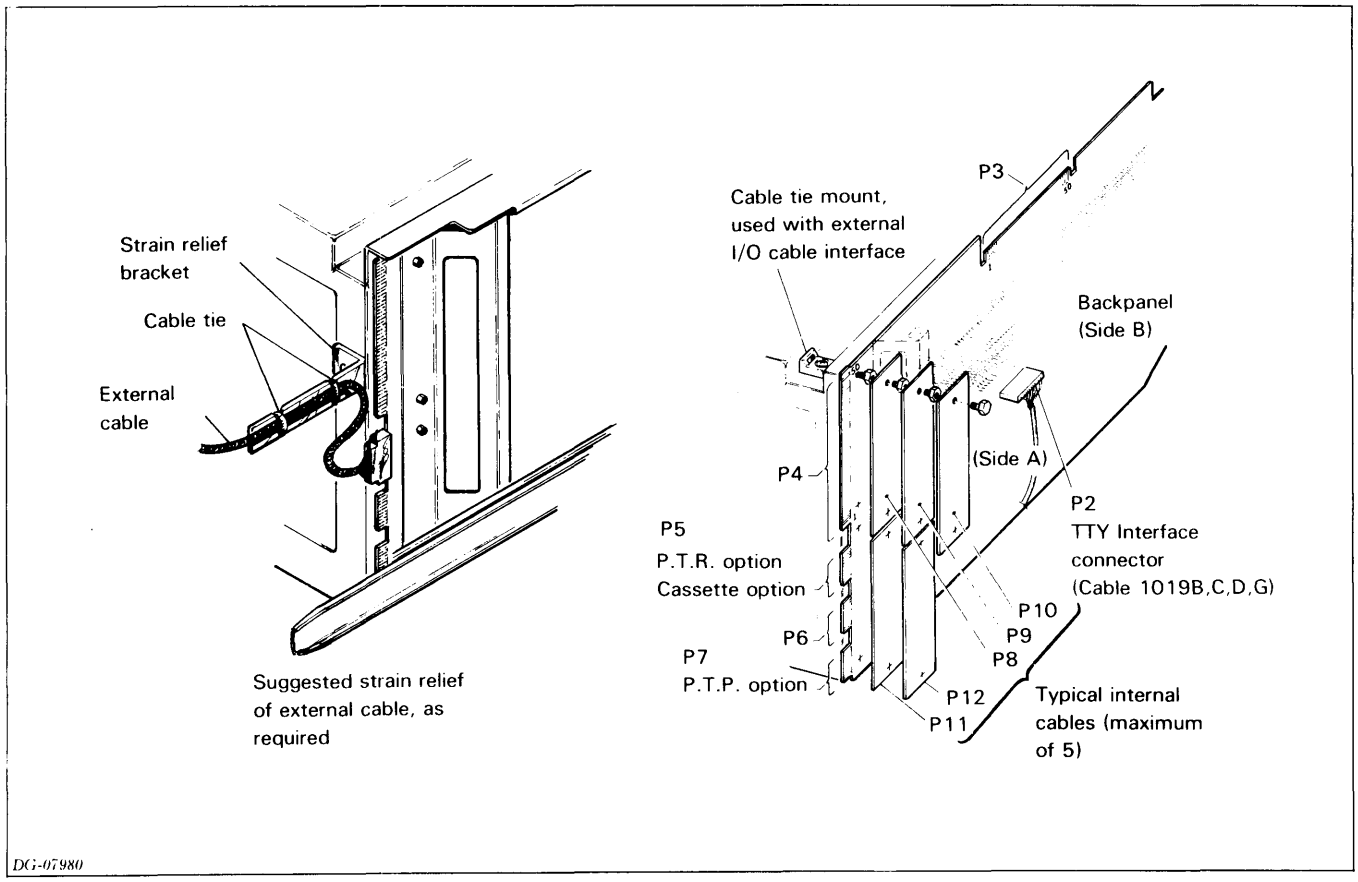


Figure 5.12 NOVA 820, 1220 and expansion chassis connectors

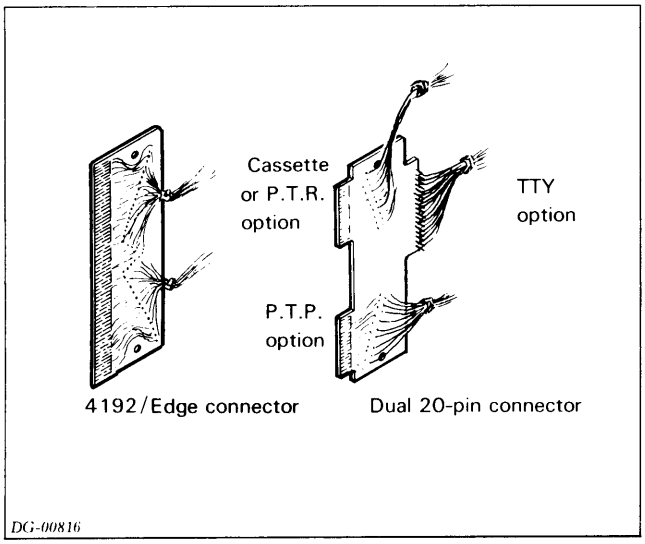


Figure 5.13 Additional internal cables

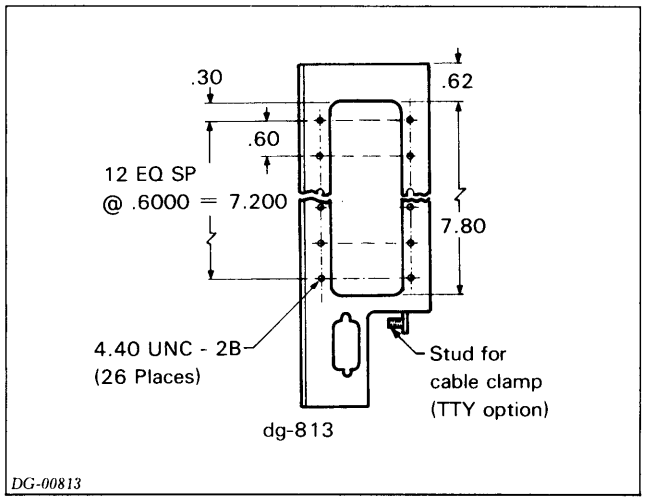


Figure 5.14 I/O bracket

NOVA 2/4 Computers

The TTY cable is plugged onto the backpanel at P2, on slot 3. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this fifty-pair connector. Device cables other than that of the TTY are plugged onto connectors mounted, as needed, at the rear of the backpanel and wire-wrapped to the proper backpanel pins. P4 and P5 optional connectors may be either connector shown in Figure 5.16. Additionally, the 4083 connector panel may be mounted as shown in Figure 5.17. The necessary connectors are furnished when DGC standard interfaces are purchased.

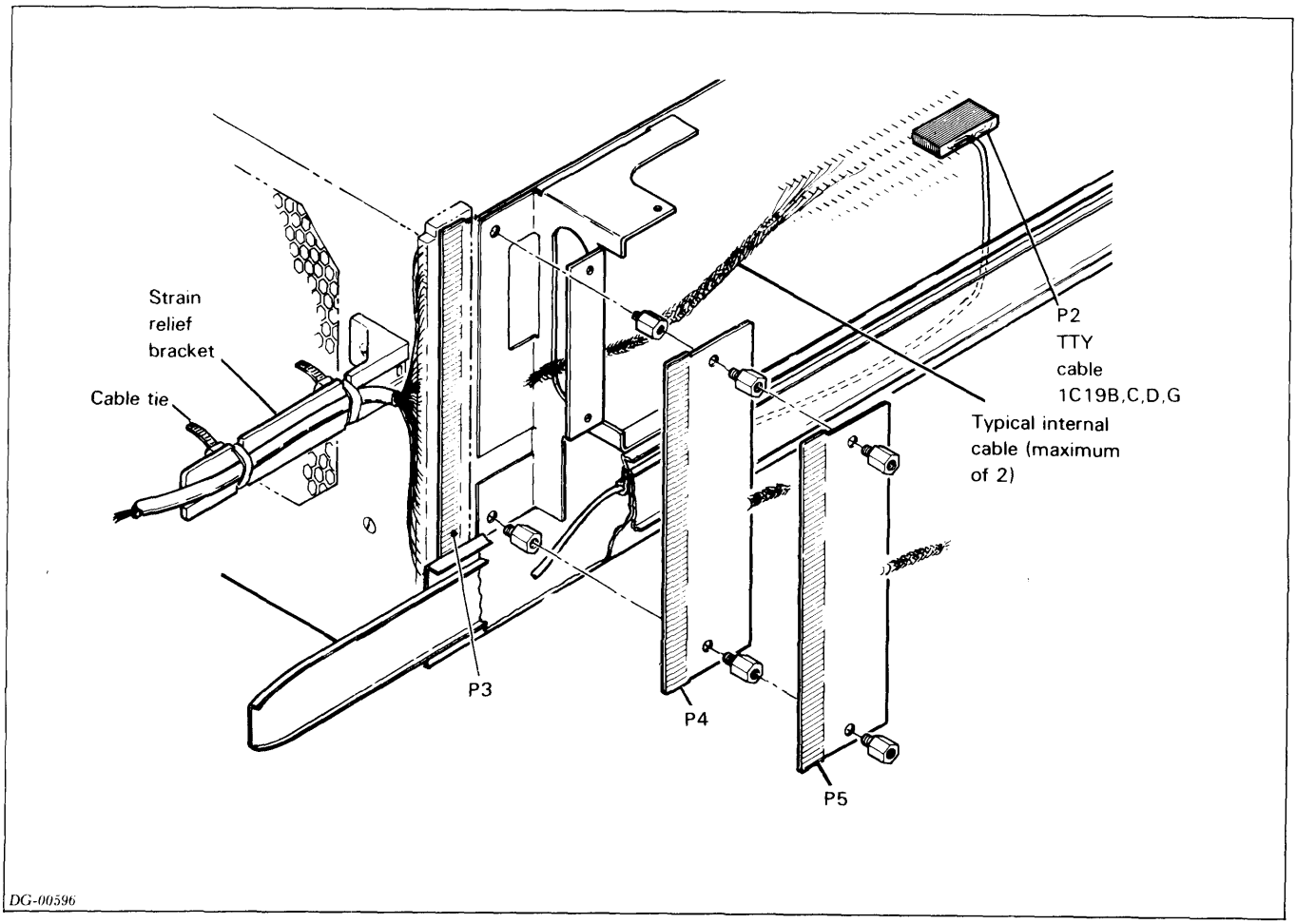


Figure 5.15 NOVA 2/4 connectors

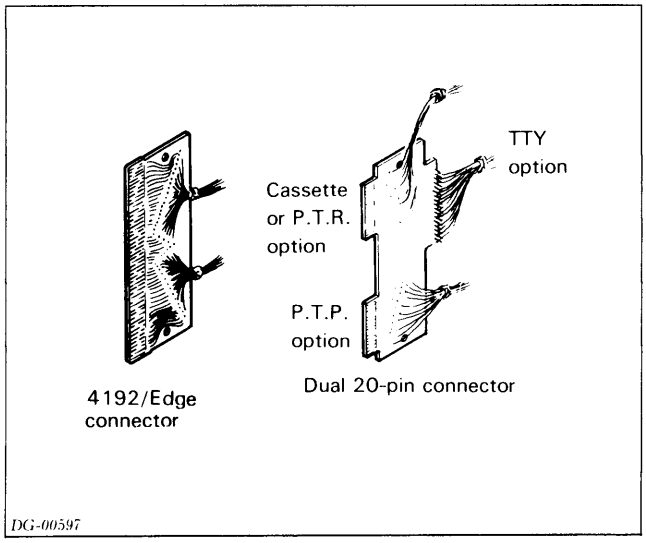


Figure 5.16 Additional internal cables

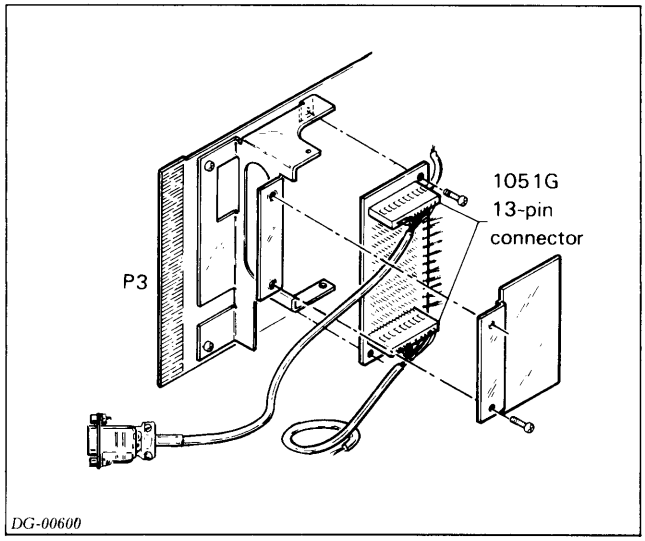


Figure 5.17 4083 connector panel

NOVA 2/10 Computer and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2, on slot 3. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. P4 is a fifty-pair edge connector whose fingers are permanently connected to backpanel pins of slot 9. Similarly, P5 and P6 ten-pair connectors, are permanently connected for use when the DGC Paper Tape Reader or Punch or DGC cassette interfaces are installed in slot 3.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. P7 through P11 may be either connector shown in Figure 5.19, while P12 and P13 must be the 4192 (fifty-pair) type. The 4083 connector panel may be mounted as shown in Figure 5.20. The necessary connectors are furnished when DGC standard interfaces are purchased.

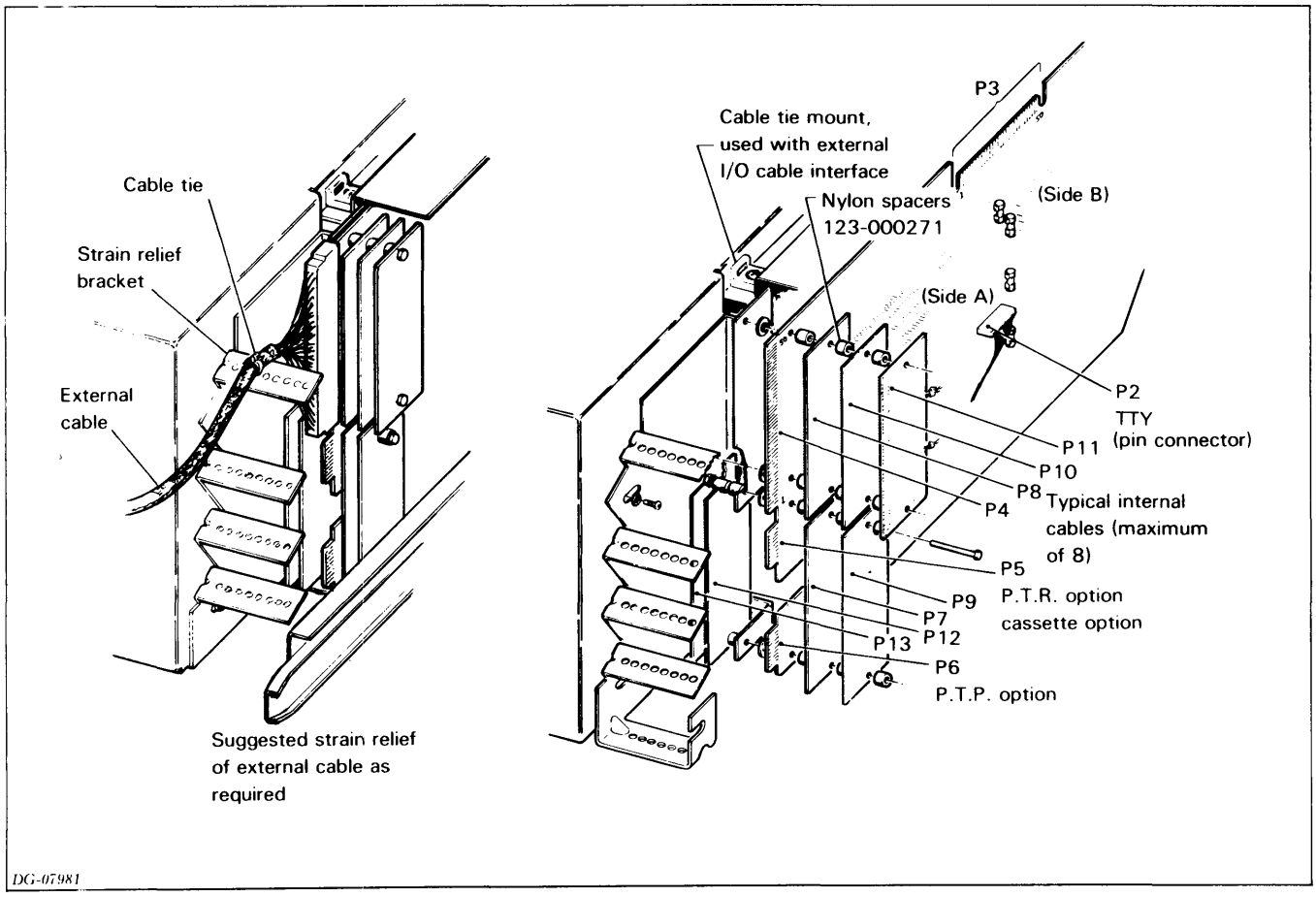


Figure 5.18 NOVA 2/10 and expansion chassis connectors

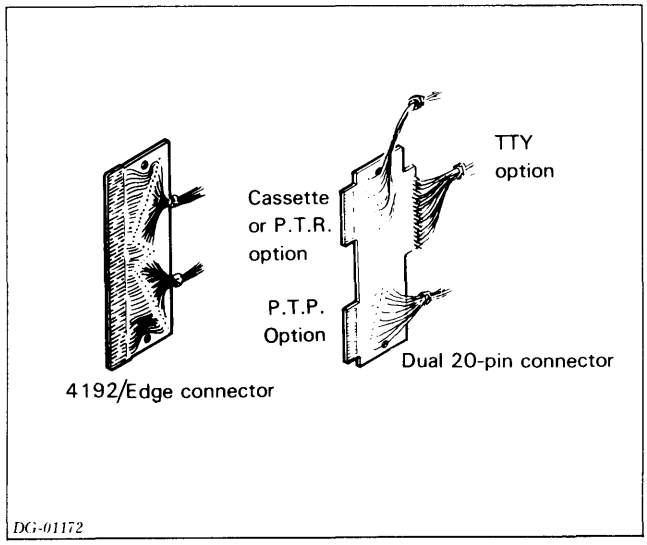


Figure 5.19 Additional internal cables

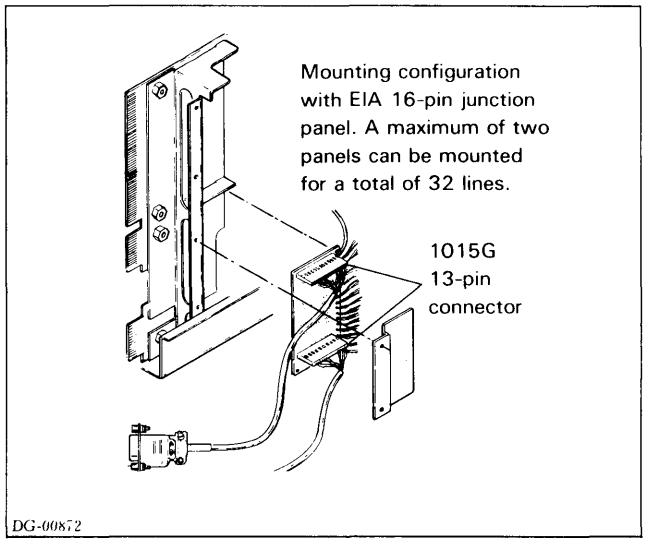


Figure 5.20 4083 connector panel

NOVA 3/4 Computer and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2, on slot 3. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to fingers on this connector. Another device cable may be plugged onto connector P4 at the rear of the backpanel, and wire-wrapped to the proper backpanel pins.

P3 must have a terminator installed if an external I/O cable is not used. P4 may be any of the cables or options shown in Figures 5.22 and 5.23. If an expansion chassis is used, a daisy chain cable connects P3 of the main chassis to PX3 of the expansion chassis. Another internal cable must then be wire-wrapped to the proper backpanel pins on the expansion chassis to accept the I/O bus terminator.

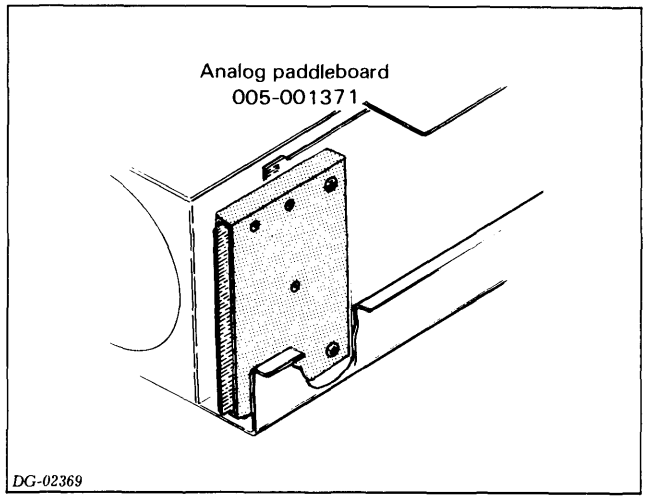


Figure 5.22 Analog paddleboard

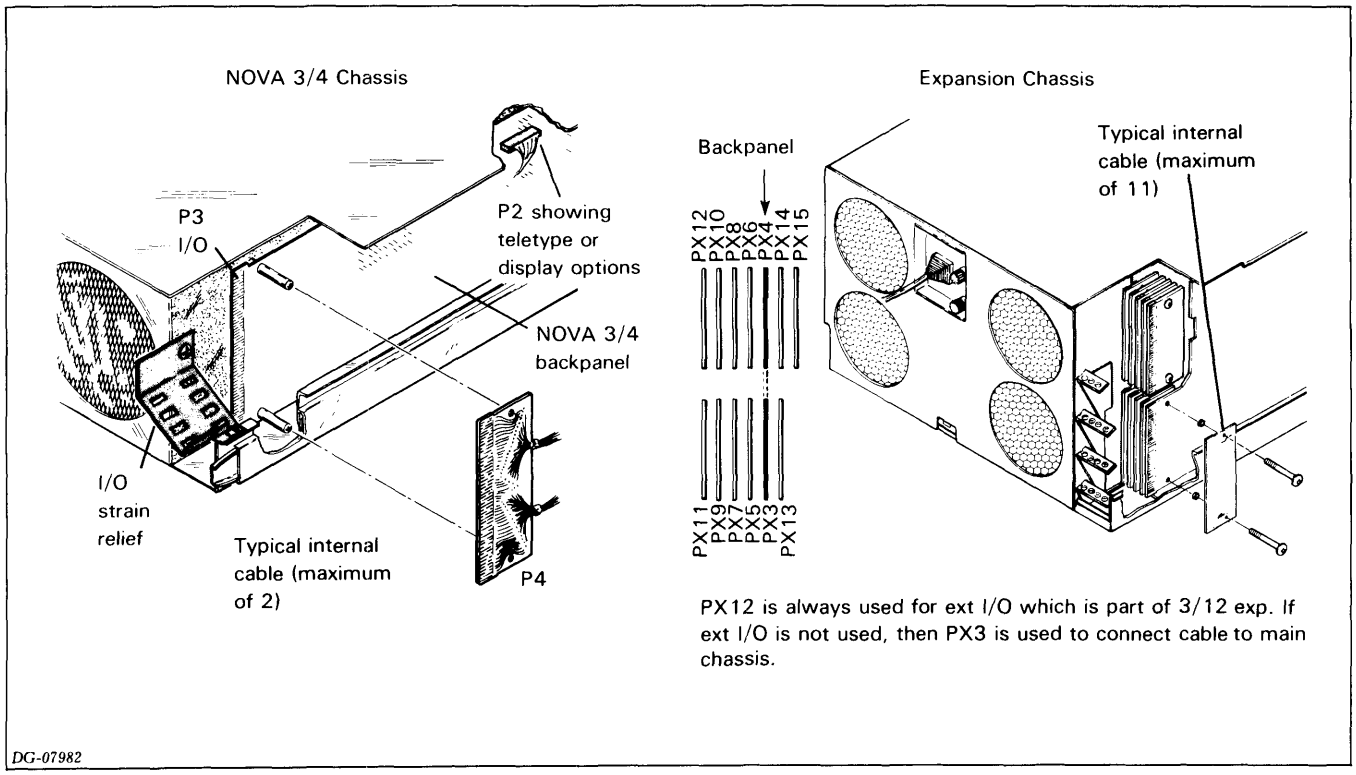


Figure 5.21 NOVA 3/4 and expansion connectors

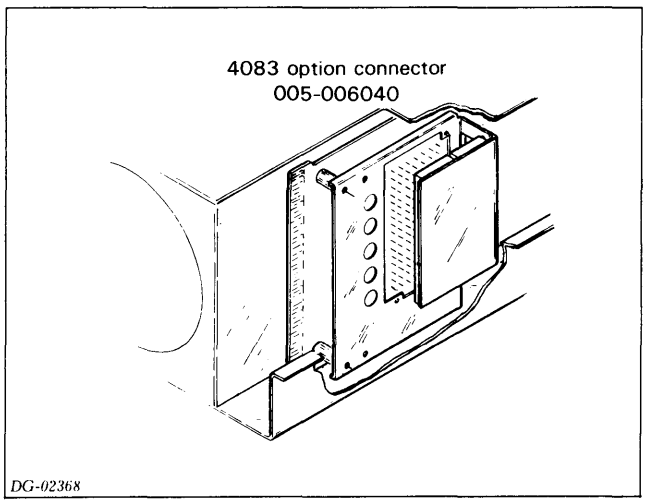


Figure 5.23 4083 option connector

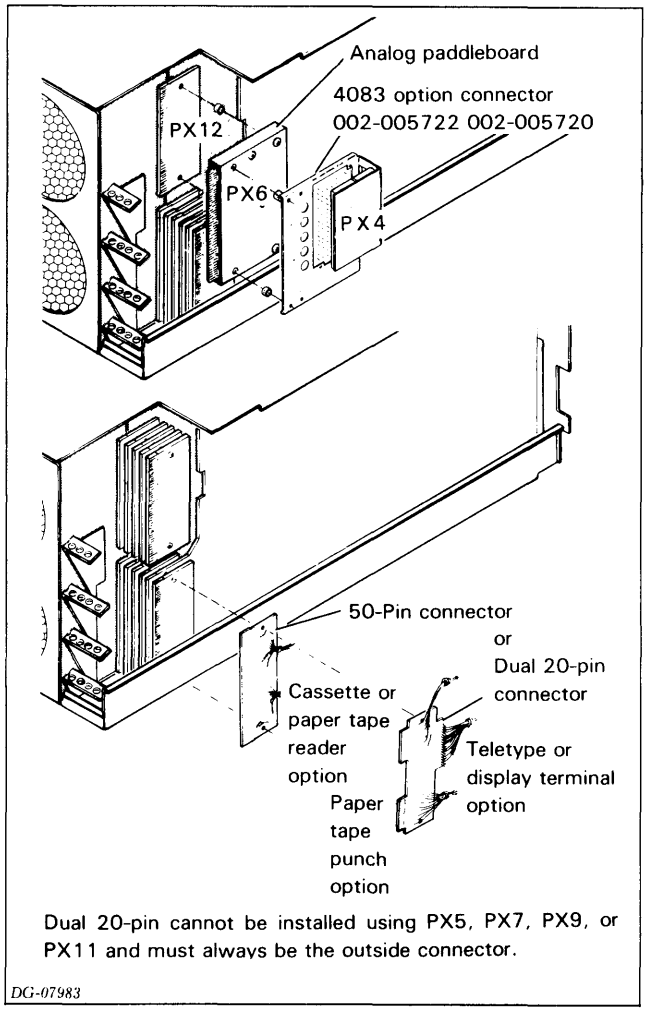


Figure 5.24 Expansion chassis

NOVA 3/12 Computers

The TTY cable is plugged onto the backpanel at P2 on slot 4. Edge P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. P4 is a fifty-pair edge connector whose fingers are permanently connected to backpanel pins of slot 10. I/O bus repeaters are installed as shown in Figure 5.26.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. The 005-003860 connector should only be used as an outside connector. The expansion chassis for the NOVA 3/12 is shown in Figure 5.24.

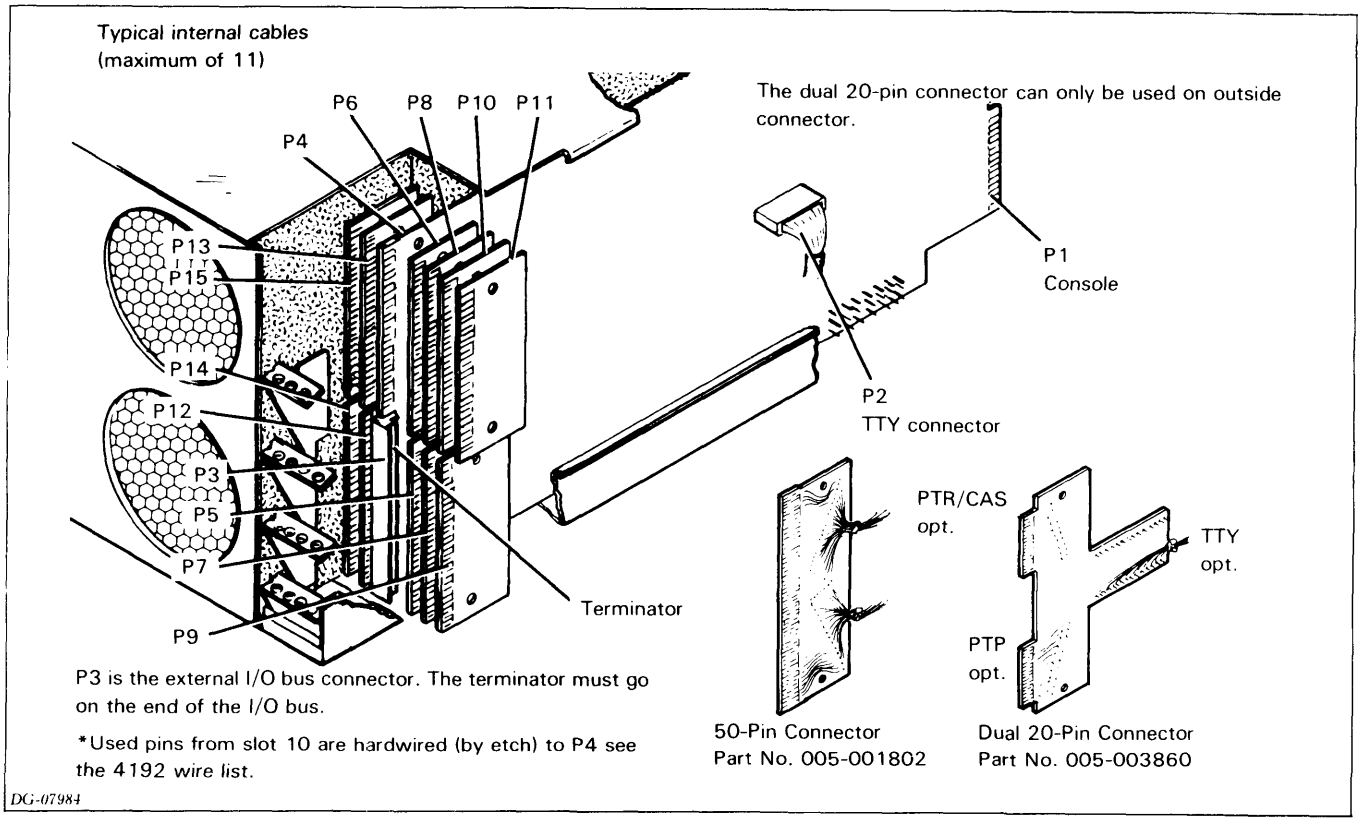


Figure 5.25 NOVA 3/12 connectors

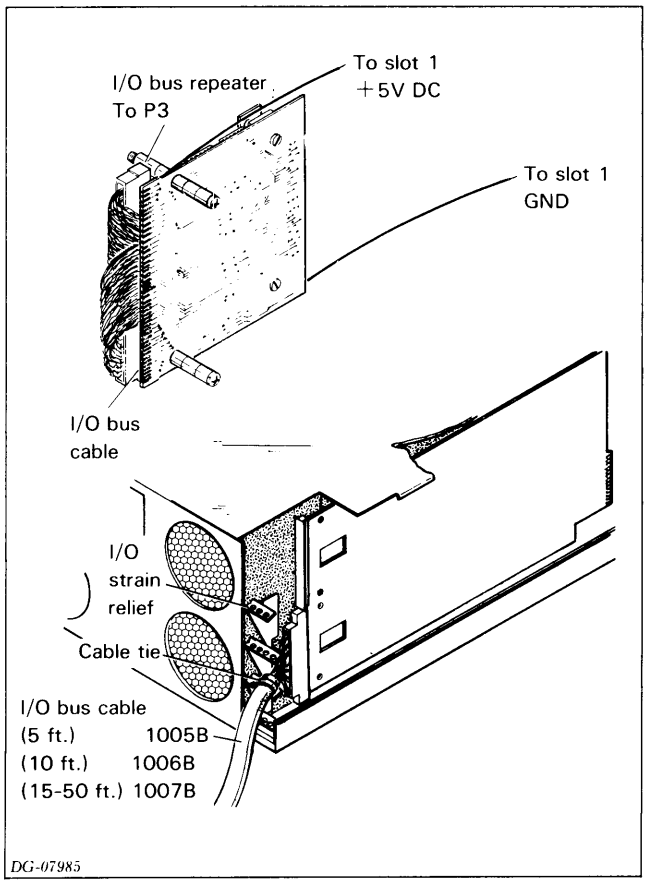


Figure 5.26 I/O bus repeater

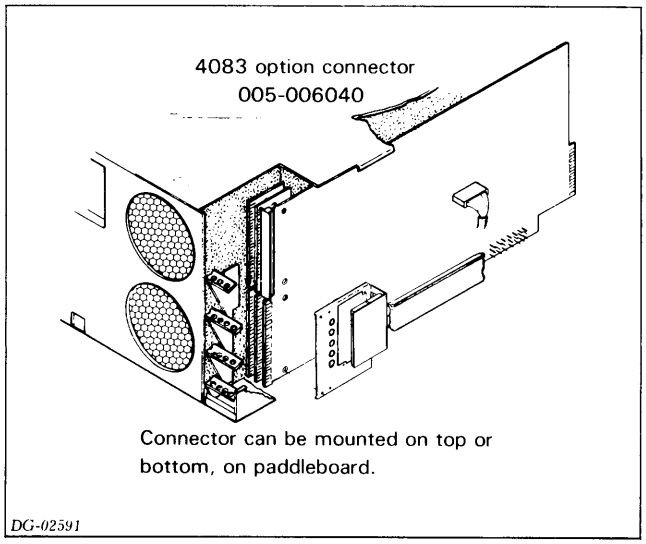


Figure 5.27 4083 option connector, 005-006040

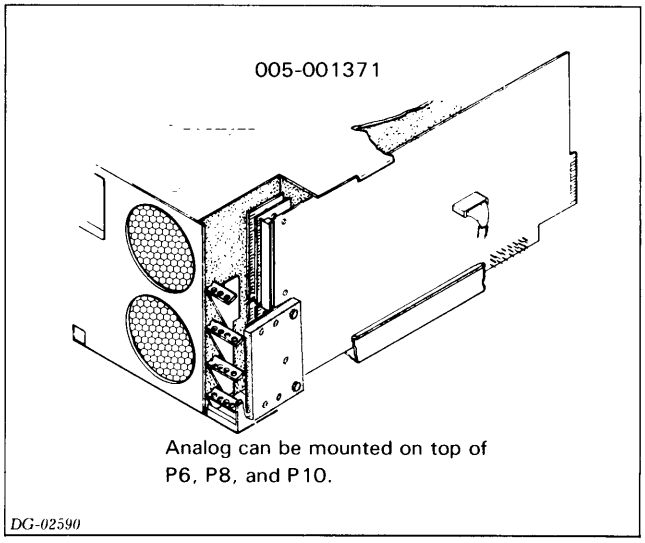


Figure 5.28 Analog paddleboard, 005-001371

NOVA 4/5 Computers

The TTY cable is plugged onto the backpanel at the location shown for slot 1. Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel. The connectors are plugged onto the proper backpanel pins.

Assembly No.	Type
005-012472	General purpose I/O
005-012751	External I/O bus**
005-012765	Universal line MUX (SYNC) model 4241, 4241A, 4242, 4243***
005-012476	I/O bus repeater model 8315
005-012590	DCU-50 models 4250, 4254
005-012473*	Asynchronous interface models 4007, 4010, 4023, 4075, 4077, 4078
005-012585	MCA model 4206***

Table 5.3 NOVA 4 I/O edge connectors

**This edge connector must be placed in the outside position: i.e., the furthest away from the edge connector support plate.*

***External I/O bus must be terminated at the end away from the computer by terminator DGC No. 005-009067, or equivalent.*

****Requires two edge connector locations.*

The I/O bus may be extended to an expansion chassis (via two internal cables similar to device cables). One of these cables connects to the backpanel of the computer, and the other to the backpanel of the expansion chassis. A daisy chain cable then connects the two internal cables.

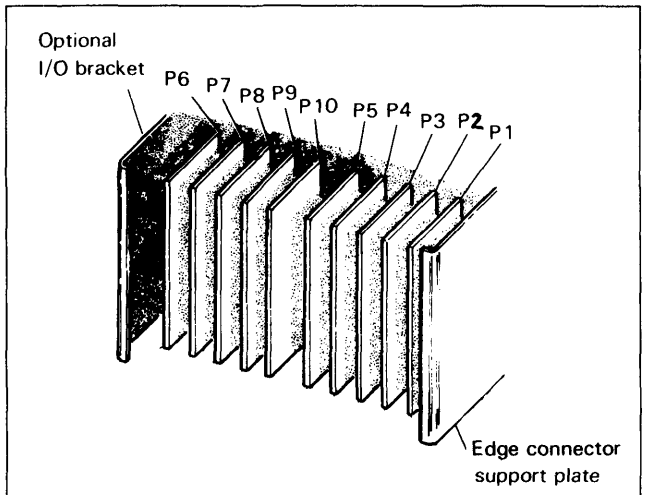


Figure 5.30 Optional I/O bracket

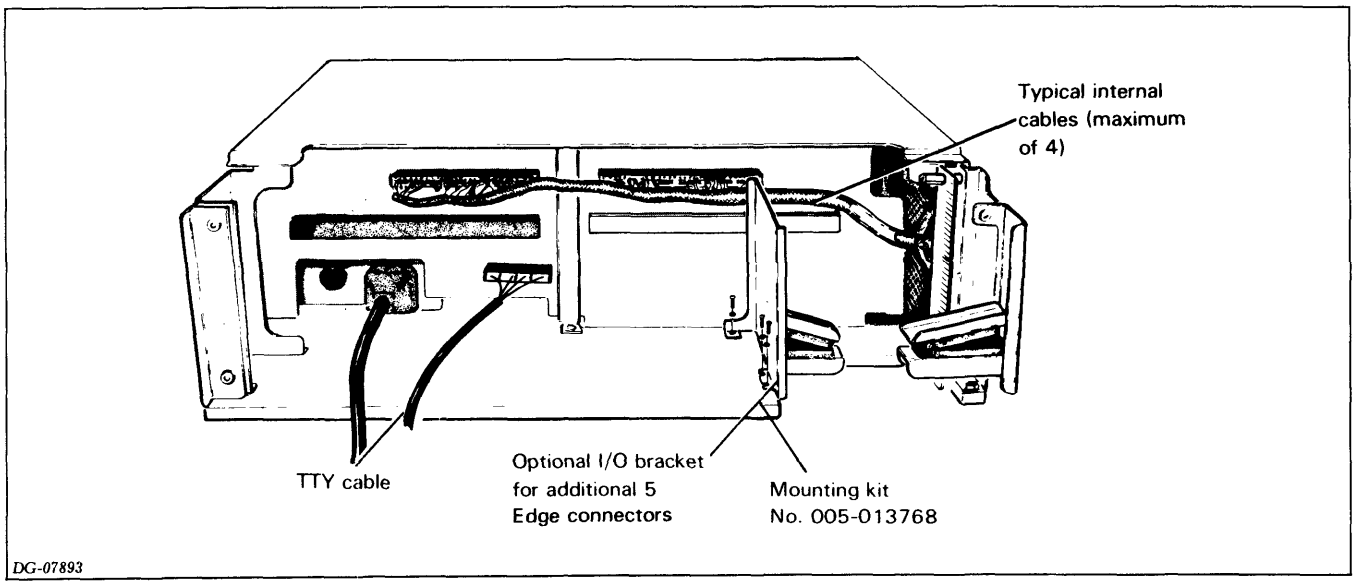


Figure 5.29 NOVA 4/5 connectors

NOVA 4/16, ECLIPSE S/140 Computers and Expansion Chassis

The TTY cable is plugged onto the backpanel at the location shown for slot 1. Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and plugged onto the proper backpanel pins.

The I/O bus may be extended to an expansion chassis (via two internal cables) similar to device cables. One of these cables connects to the backpanel of the computer, and the other to the backpanel of the expansion chassis. A daisy chain cable then connects the two internal cables.

Assembly No.	Type
005-012472	General purpose I/O
005-012751	External I/O bus**
005-012765	Universal line MUX (SYNC) model 4241, 4241A, 4242, 4243***
005-012476	I/O bus repeater model 8315
005-012590	DCU-50 models 4250, 4254
005-012473*	Asynchronous interface models 4007, 4010, 4023, 4075, 4077, 4078
005-012585	MCA model 4206***

Table 5.3 NOVA 4 I/O edge connectors

**This edge connector must be placed in the outside position: i.e., the furthest away from the edge connector support plate.*

***External I/O bus must be terminated at the end away from the computer by terminator DGC No. 005-009067, or equivalent.*

****Requires two edge connector locations.*

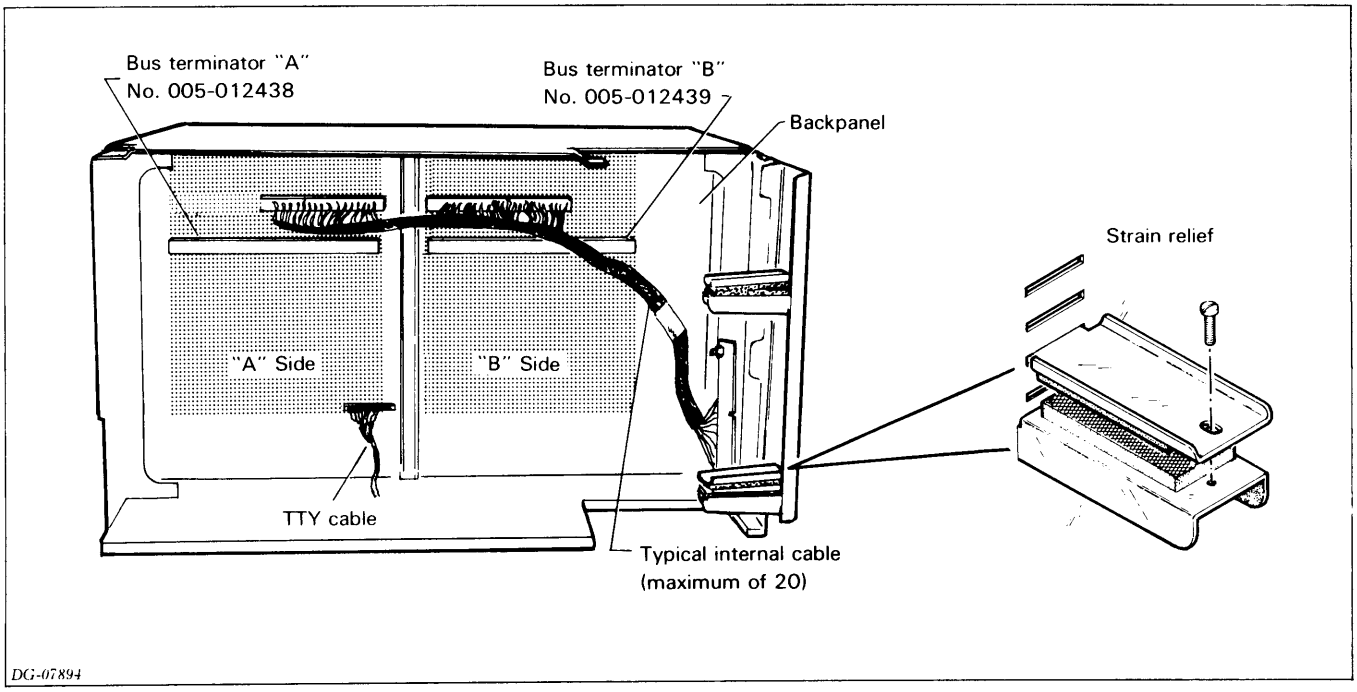


Figure 5.31 NOVA 4/16, ECLIPSE S/140 and expansion chassis connectors

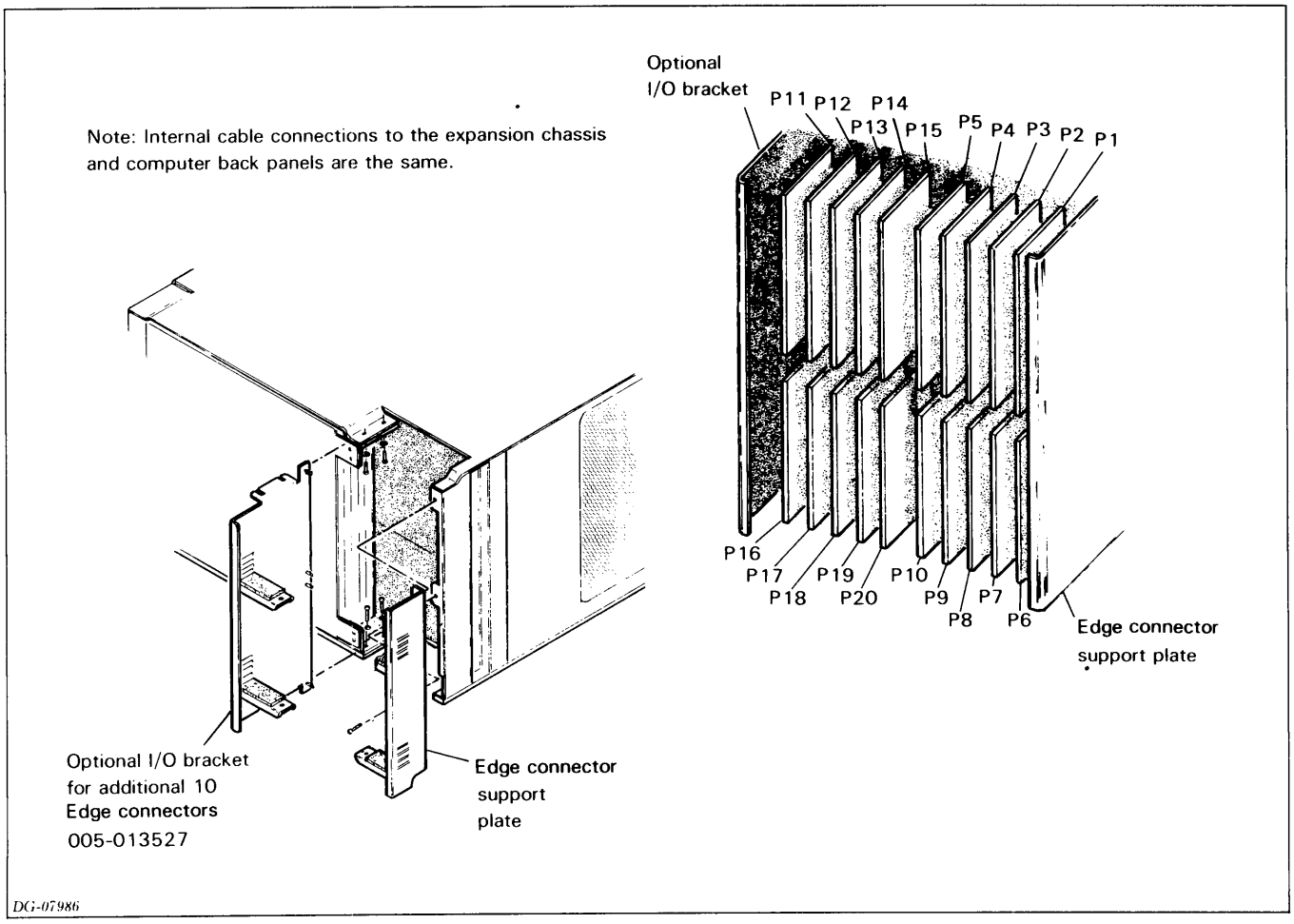


Figure 5.32 Expansion chassis for NOVA 4/16 and ECLIPSE S/140 computers

ECLIPSE S/100 Computers

The TTY cable is plugged onto the backpanel at P2 on slot 5. P3 and P4 ten-pair connectors, are permanently connected to the backpanel for use when the DGC Paper Tape Reader or Punch, or DGC cassette interfaces are installed in slot 5. The 4083 option and analog paddleboard are mounted as shown below. The necessary connectors are furnished when DGC standard interfaces are purchased.

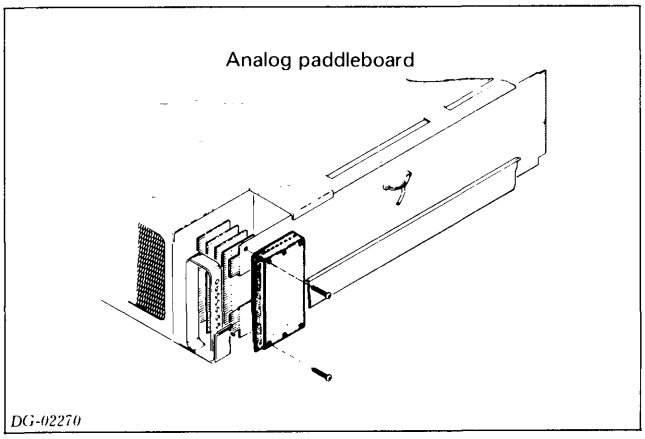


Figure 5.34 Analog paddleboard

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. The I/O bus for this machine does not have a permanently etched connector. To extend the I/O bus, an internal cable must be wire-wrapped to the proper backpanel pins.

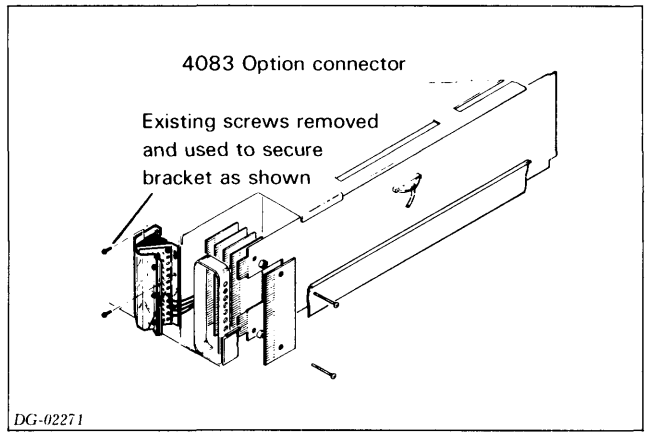


Figure 5.35 4083 option connector

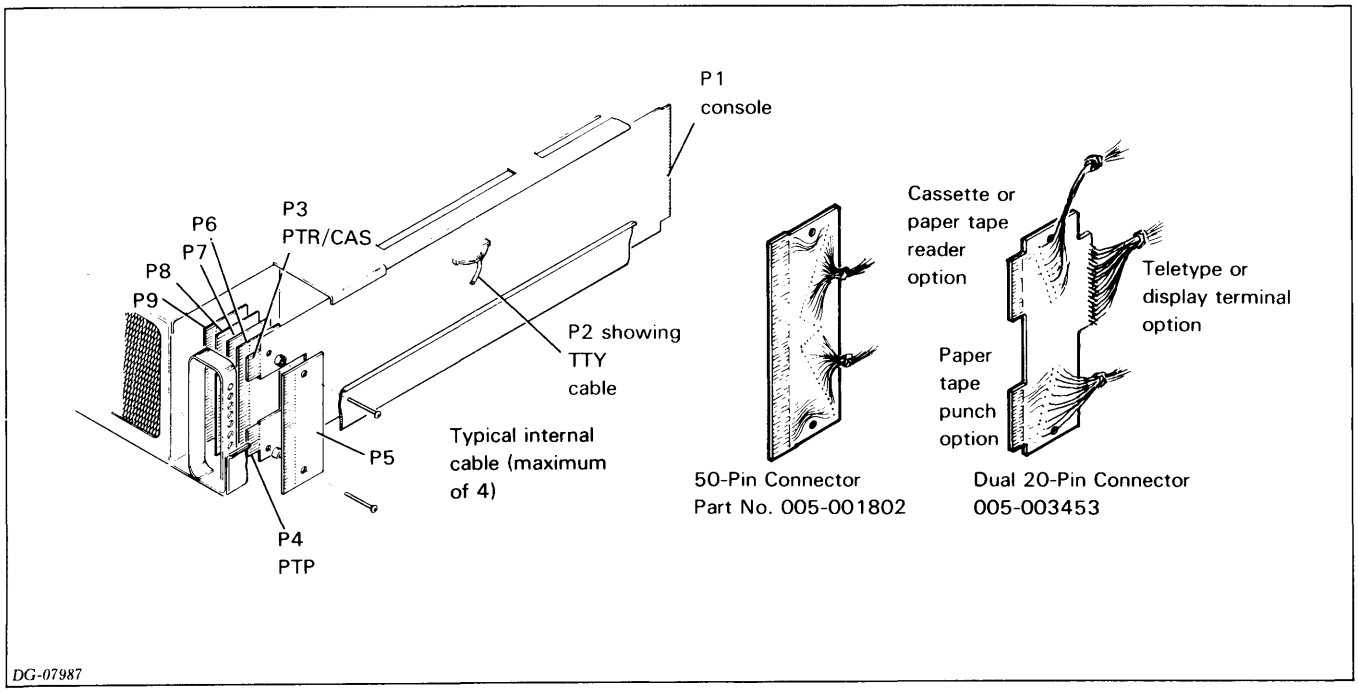


Figure 5.33 ECLIPSE S/100 computer connectors

ECLIPSE S/200, C/300 Computers and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2 on slot 6. Edge connector P5 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. Similarly, P3 and P4 (ten-pair connectors) are permanently connected for use when the DGC Paper Tape Reader or Punch, or DGC cassette interfaces are installed in slot 6.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. The 4083 option and analog paddleboard are mounted as shown below. The necessary connectors are furnished when DGC standard interfaces are purchased.

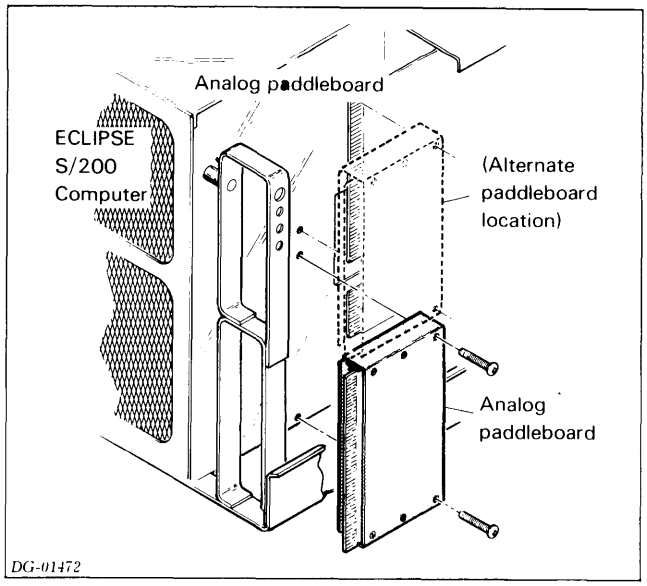


Figure 5.37 Analog paddleboard

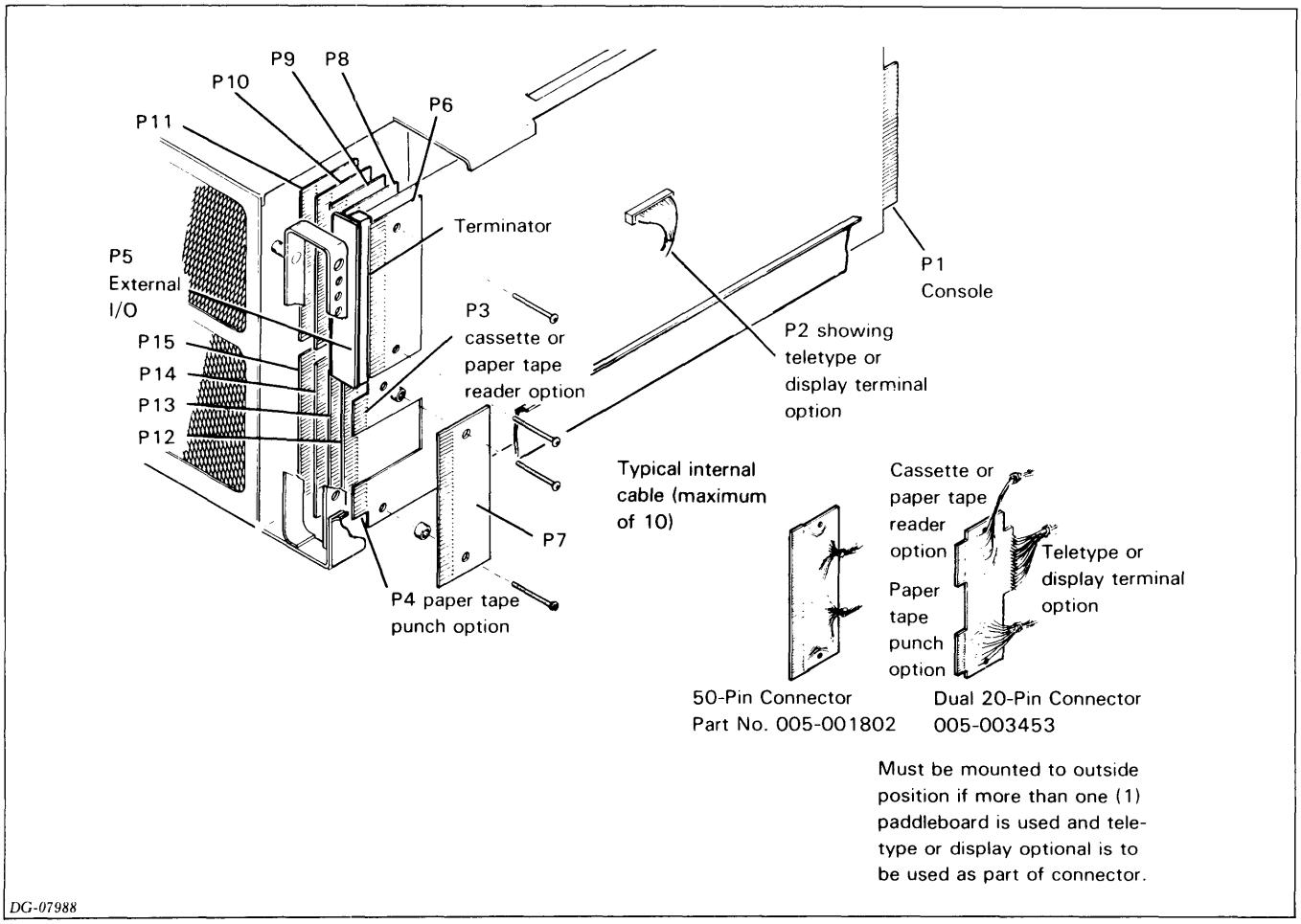


Figure 5.31 ECLIPSE S/200, C/300 computers - connectors

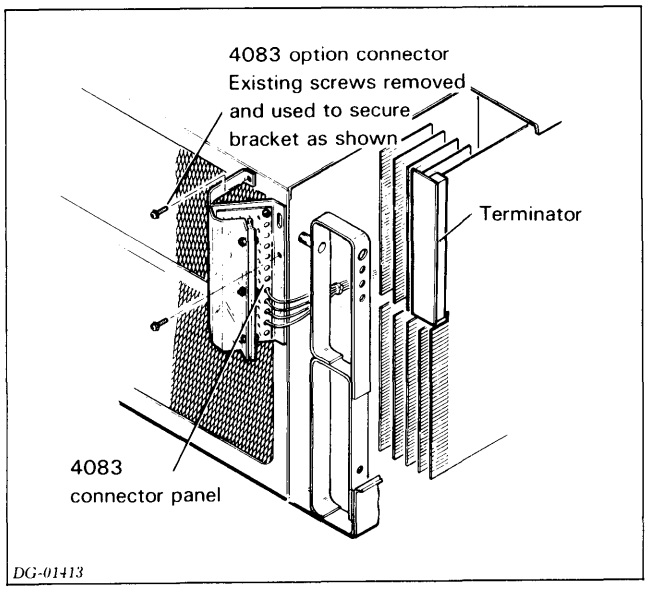
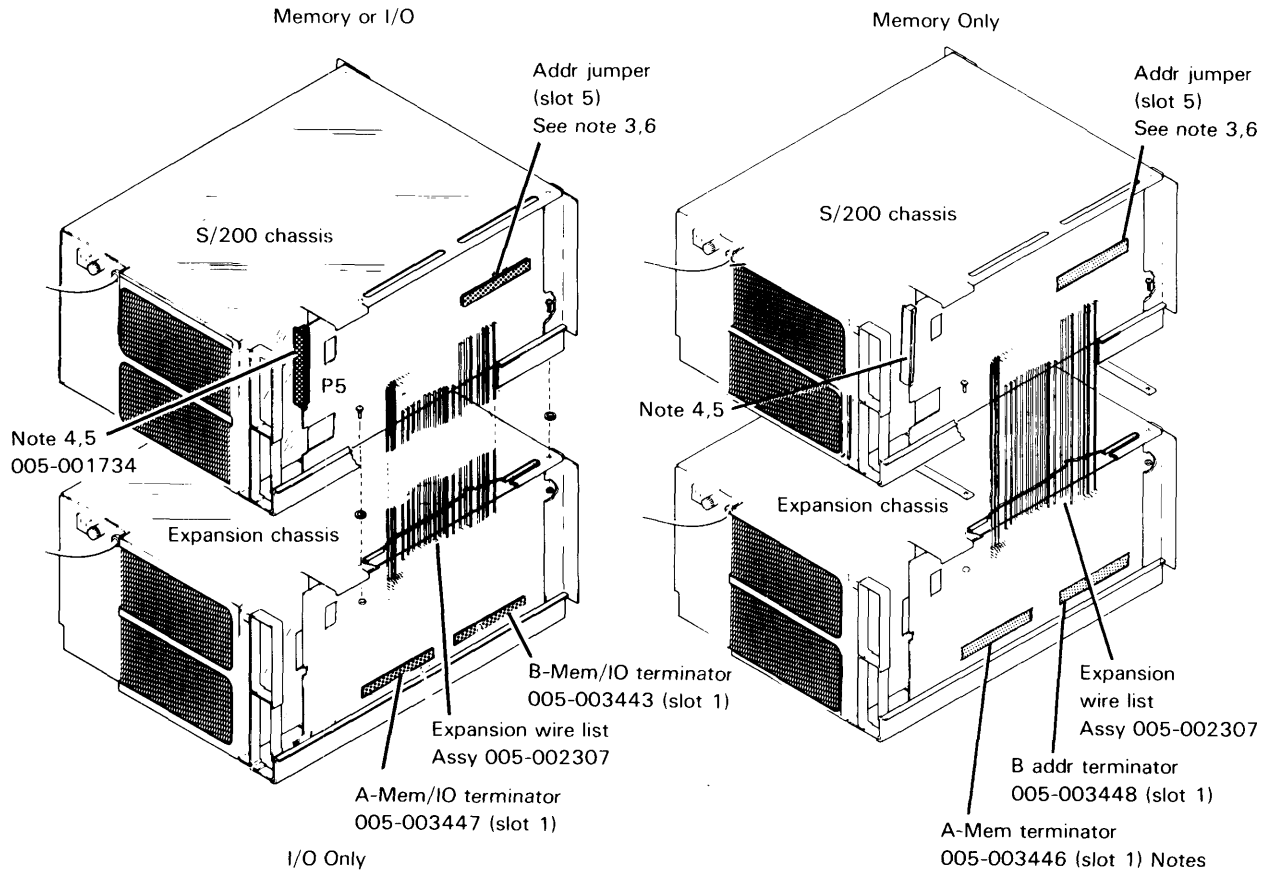


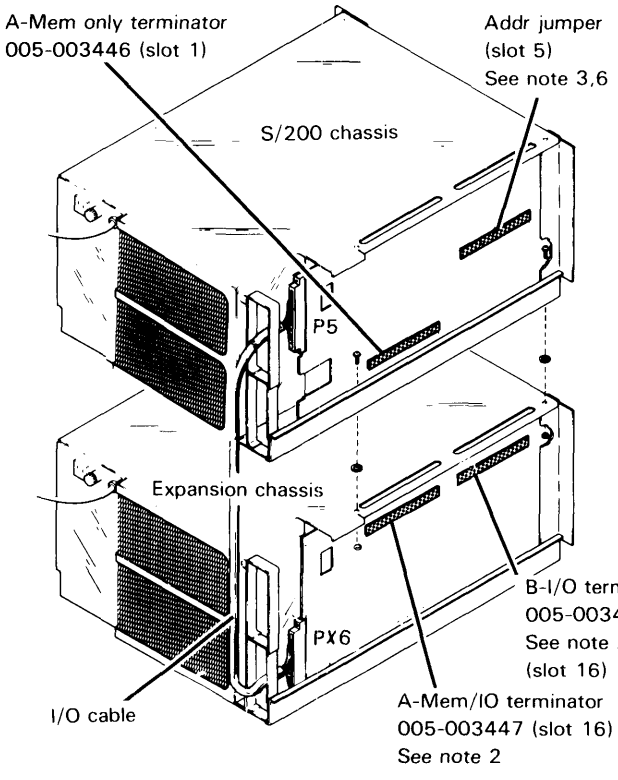
Figure 5.38 4083 option connector

Expansion Chassis

When an expansion chassis is used for I/O only, an external cable connects P5 of the main chassis to P6 of the expansion chassis. The terminator is then moved to P1 of the expansion chassis. See Figure 5.39.

When an expansion chassis is needed for memory only, or memory and I/O, the backpanels must be wire-wrapped together. The I/O terminator remains on P5 of the main chassis, and address and memory terminators are installed on slot 1 of the expansion chassis.





1. The paddleboard is wrapped to slot 1 of I/O expansion box. Wire list 008-000791 must be used for this application only.
2. If external I/O bus is connected, then terminators A-MEM/IO and B-I/O on slots 16 of I/O expansion, must be removed. The external I/O bus must always be terminated using DGC 005-001734 terminator board. To connect external I/O bus use wire list 008-000891.
3. Addr jumper used only when map board not installed in system (slot 5, main chassis).
4. The external I/O bus must always be terminated.
5. External I/O bus must always be terminated with DGC 005-001734 terminator board at P5 unless I/O is connected.
6. When map is installed, also install jumpers off wire list 008-000654.

DG-07989

Figure 5.39 Expansion chassis

ECLIPSE S/230 C/330 Computers and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2 on slot 6. Edge connector P5 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. Similarly, P3 and P4 (ten-pair connectors) are permanently connected for use when the DGC Paper Tape Reader or Punch, or DGC cassette interfaces are installed in slot 6.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped to the proper backpanel pins. The 4083 option and analog paddleboard are mounted as shown in Figures 5.41 and 5.42. The necessary connectors are furnished when DGC standard interfaces are purchased.

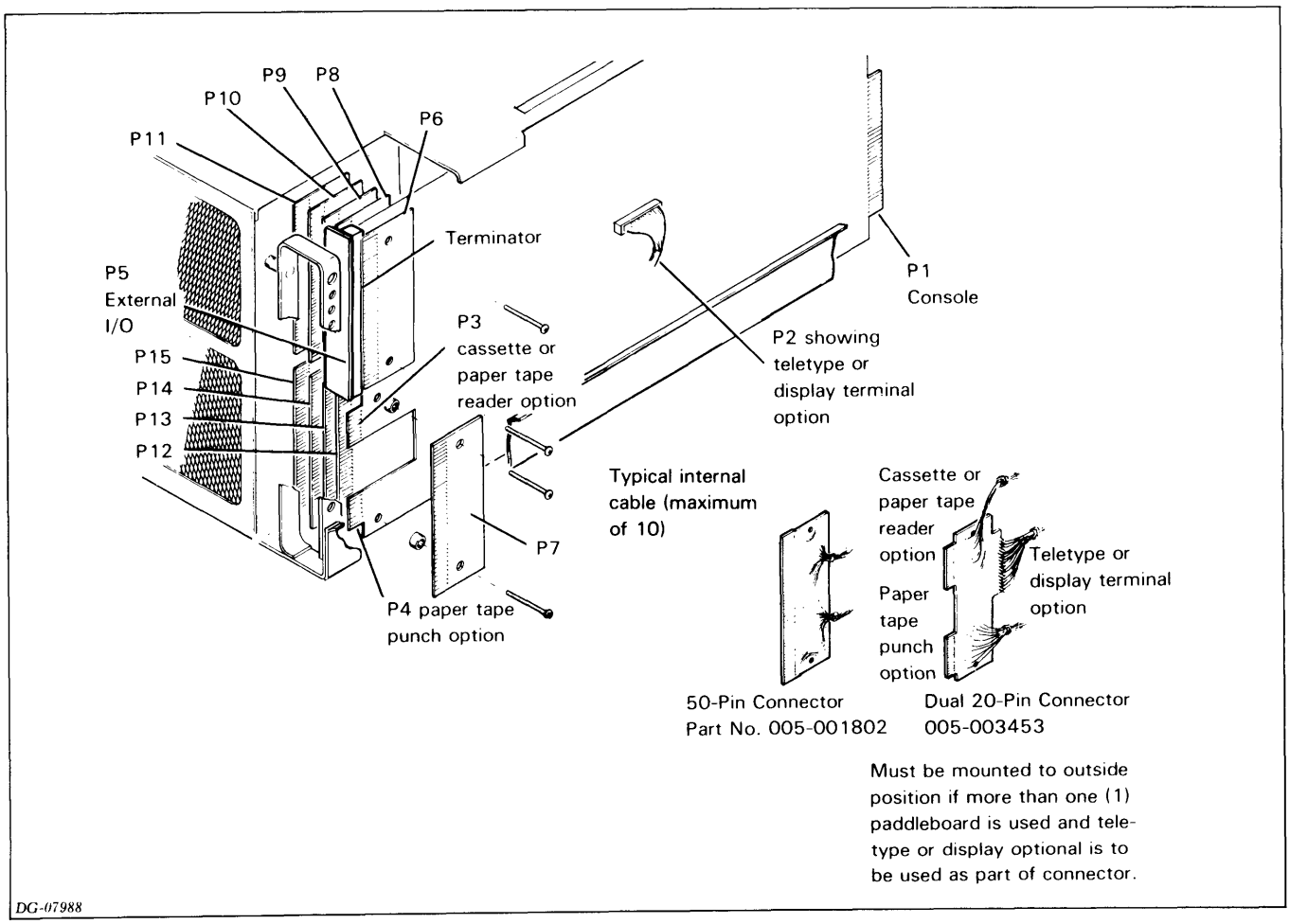


Figure 5.40 ECLIPSE S/200, C/300 computers - connectors

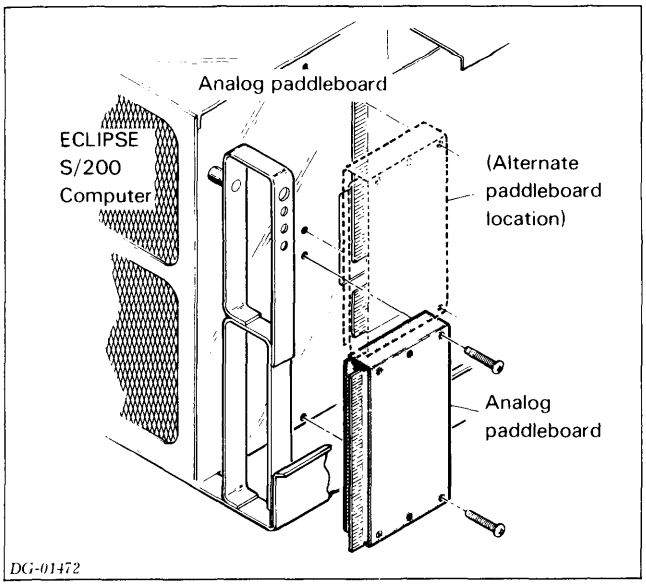


Figure 5.41 Analog paddleboard

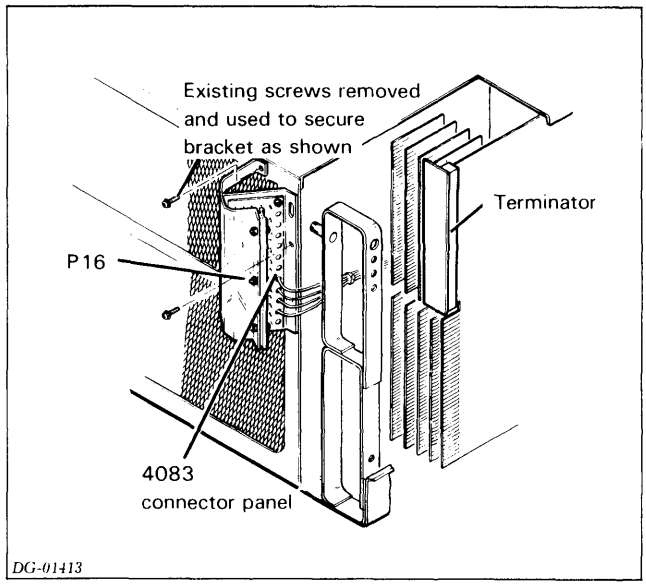


Figure 5.42 4083 option connector

ECLIPSE S/230, C/330 Expansion Chassis

When an expansion chassis is used for I/O only, an external cable connects P5 of the main chassis to P6 of the expansion chassis. The terminator, originally on P5 of the main chassis is replaced by two terminators on the backpanel of the expansion chassis.

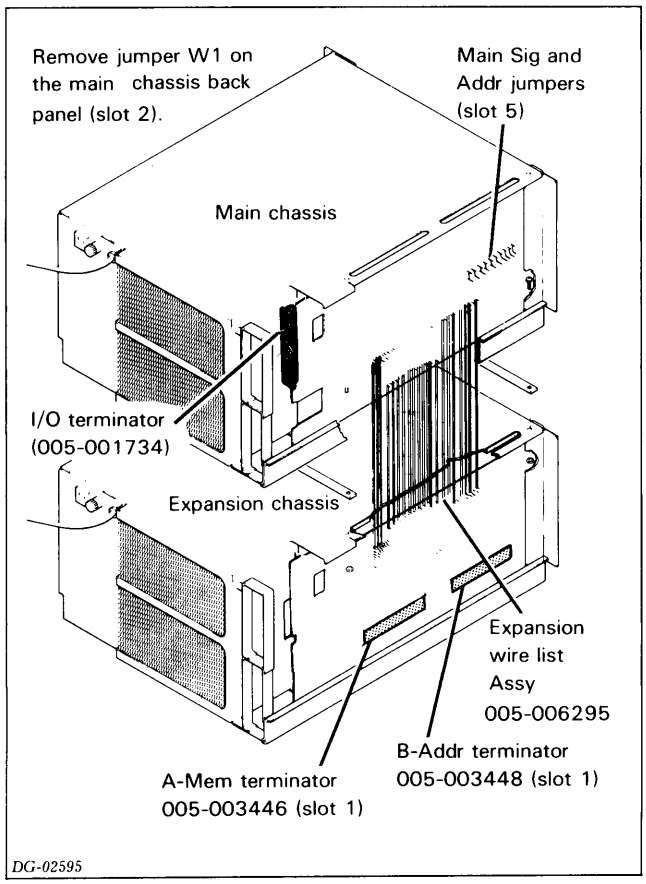


Figure 5.43 Main chassis with memory only - expansion chassis (model 8414-A)

When an expansion chassis is needed for memory only, or memory and I/O, the backpanels must be wire-wrapped together. The I/O terminator remains on P5 of the main chassis, and address and memory terminators are installed on slot 1 of the expansion chassis.

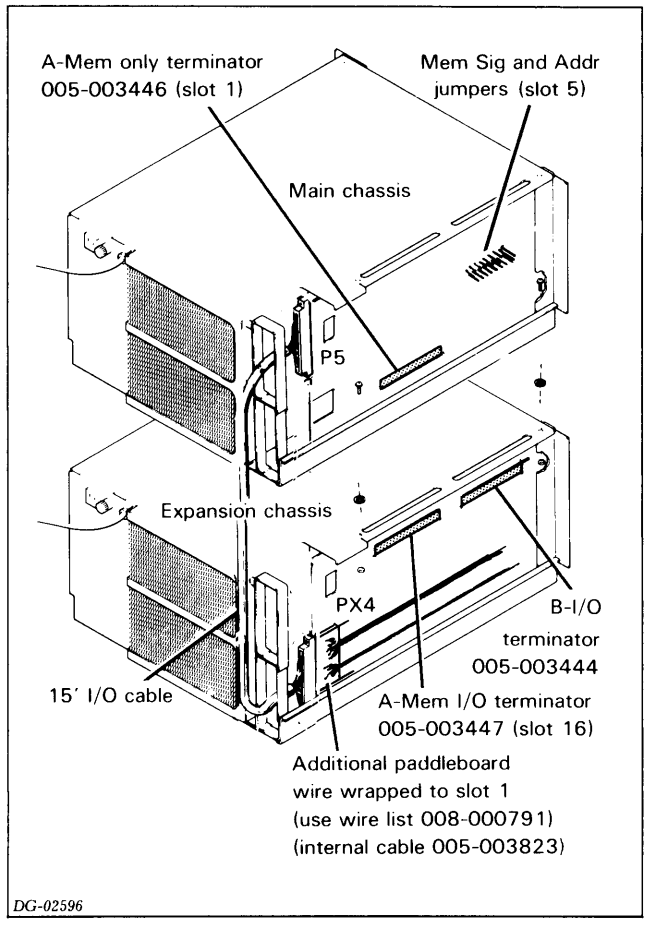
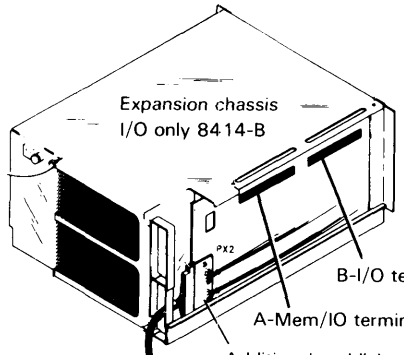


Figure 5.44 Main chassis with I/O only - expansion chassis (model 8414-B)



To connect an I/O cable from an additional paddleboard of 8414-B to an I/O device, e.g., communication chassis, use wire list 008-000791 and internal cable 005-003823. Also remove A-Mem I/O terminator (005-003447) and B-I/O terminator (005-003444) from slot 16 of 8414-B.

B-I/O terminator (005-003444) for slot 16 of 8414-B

A-Mem/IO terminator (005-003447) for slot 16 of 8414-B

Additional paddleboard
wire wrapped to slot 1
(use wire list 008-000791)
(internal cable 005-003823)

15' I/O cable

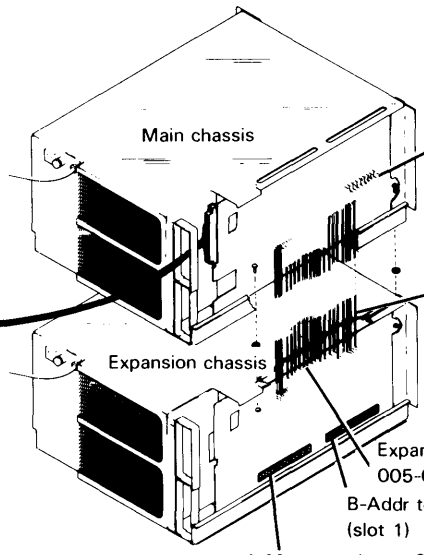
Terminators Needed:

A-Mem I/O terminator (005-003447) for slot 16 of 8414-B
B-I/O terminator (005-003444) for slot 16 of 8414-B
A-Mem terminator (005-003446) for slot 1 of 8414-A
B-Addr terminator (005-003448) for slot 1 of 8414-A

Notes:

If MMPU1 board is not present in slot 5, use wire list 008-000655 to jumper memory control signals and address lines.

If MMPU1 board is present in slot 5, use wire list 008-000654 to connect control signals to MMPU1 back panel slot.



Mem Sig and Addr jumpers
(slot 5)

Chassis are bolted together. Use wire list 005-006295 to connect Mem Bus, Addr Bus and necessary control signals between main chassis and memory only expansion chassis 8414-A.

Expansion wire list Assy
005-006295

B-Addr terminator 005-003448
(slot 1)

A-Mem terminator 005-003446
(slot 1)

DG-07899

Figure 5.45 Main chassis with I/O only (model 8414-B) and memory only (8414-A) expansion chassis

ECLIPSE S/130, AP/130 and C/150 Computers and Expansion Chassis

The TTY cable is plugged onto the backpanel at P2 on slot 7. Edge connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. An additional connector may be mounted at any edge connector location other than P5, 7, 9, or 11, and must always be the outside connector. Such a connector is needed when an interface for a paper tape reader, punch, cassette or secondary TTY is used.

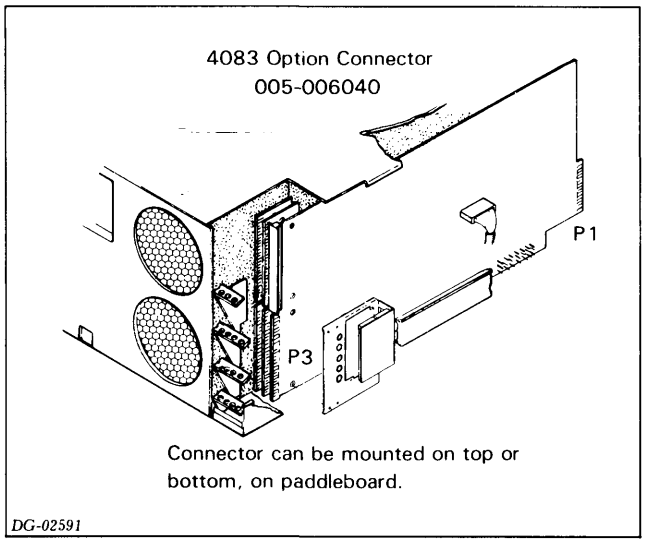


Figure 5.47 4083 option connector, 005-006040

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel, and wire-wrapped (via internal cables) to the proper backpanel pins. The 4083 option and analog paddleboard are mounted as shown in Figures 5.47 and 5.48. The necessary connectors are furnished when DGC standard interfaces are purchased. The I/O bus may be extended to an expansion chassis, via a daisy chain cable from P3 of the main chassis to PX3 of the expansion chassis. The terminator is then moved to the expansion chassis.

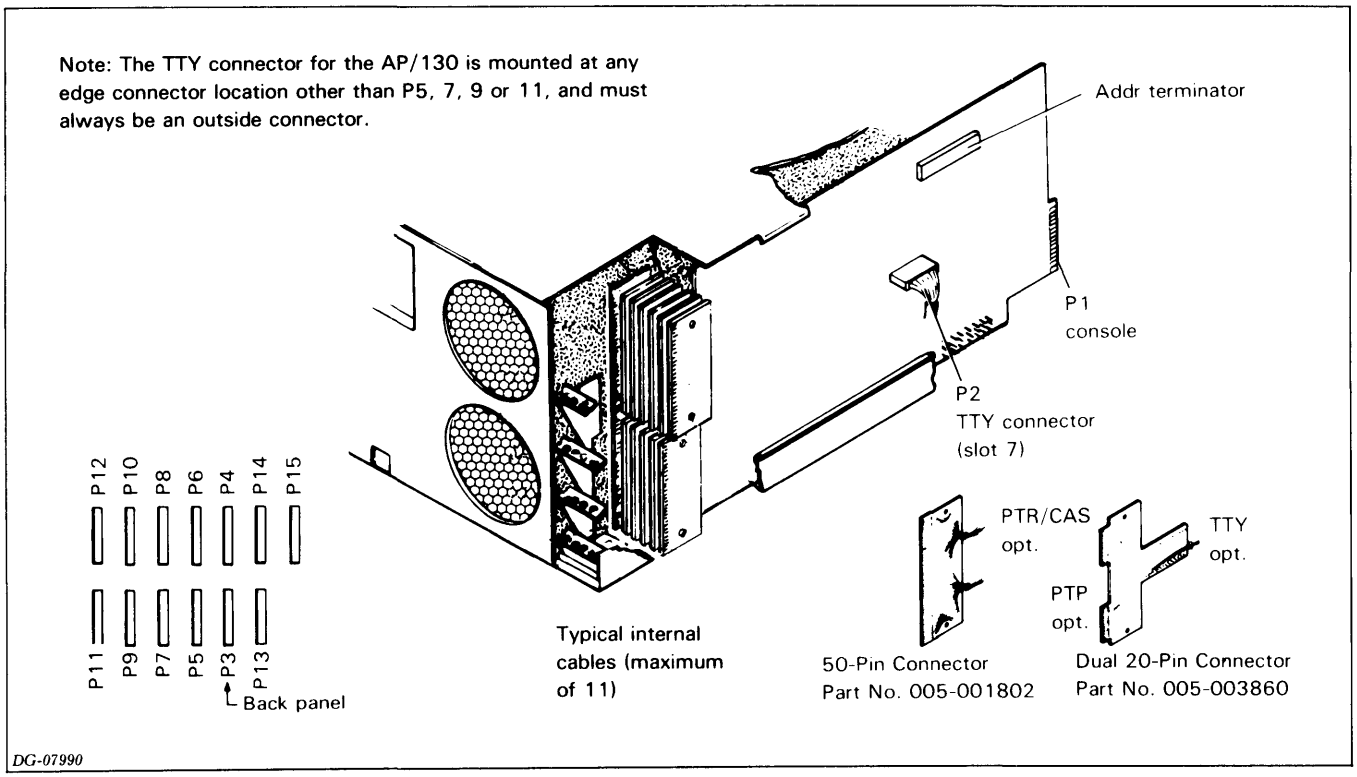


Figure 5.46 Connectors for S/130, AP/130 and C/150 computers

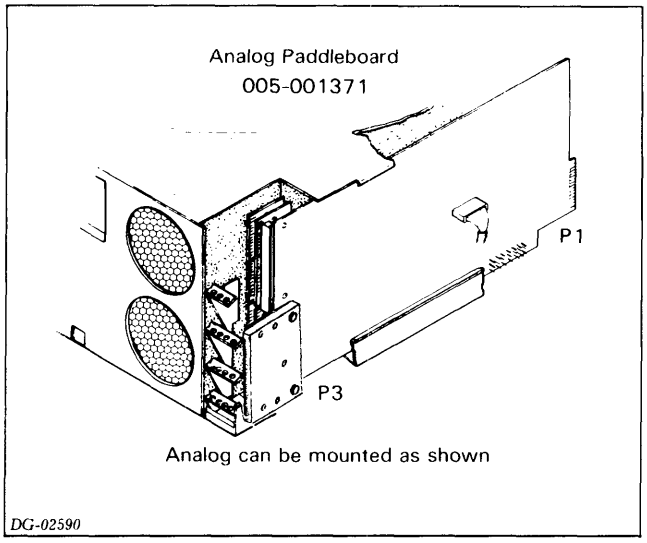
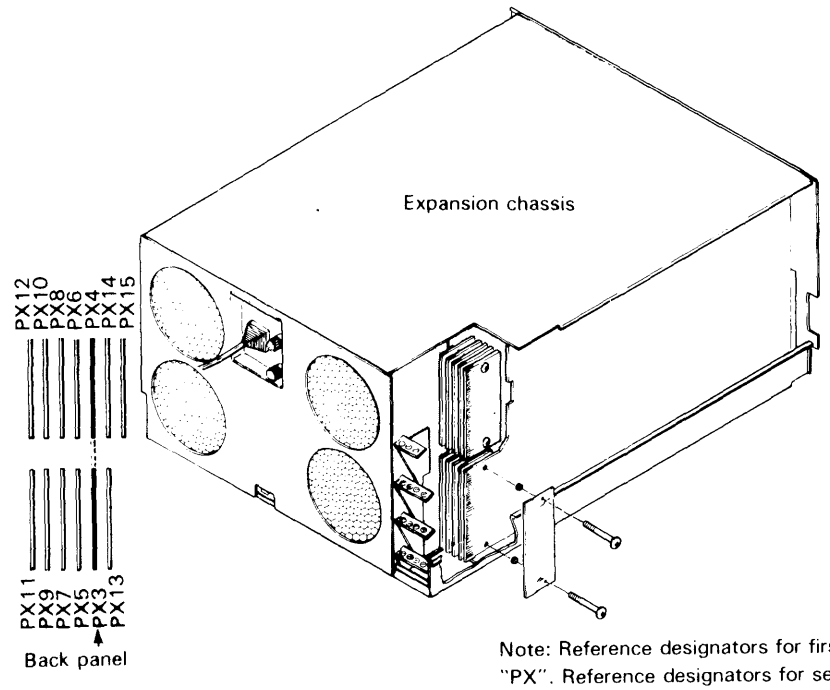
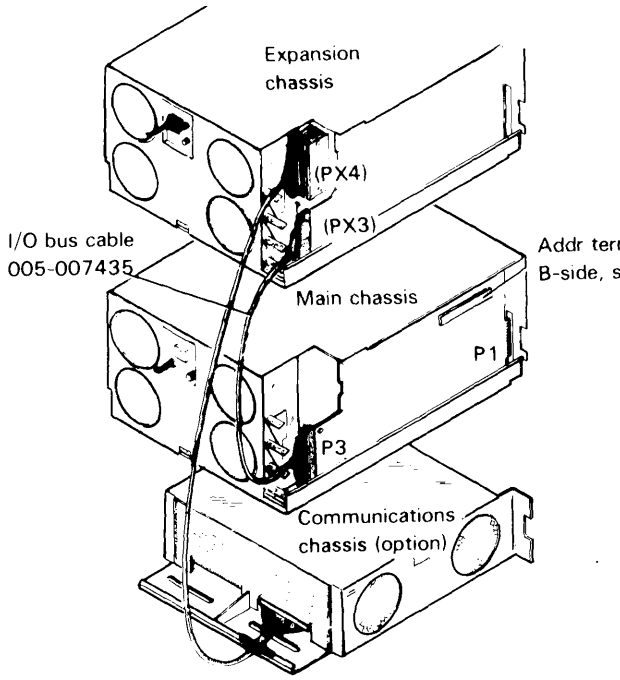


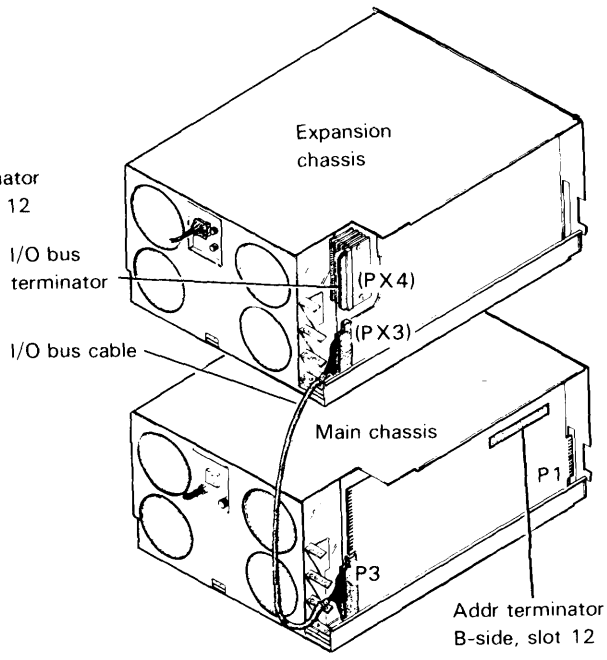
Figure 5.48 Analog paddleboard, 005-001371



Note: Reference designators for first expansion chassis are "PX". Reference designators for second expansion chassis are "PXX".



This configuration allowed only if cabinet has sufficient power.



Note: External I/O connections which are normally available on the main chassis (P3), become available on the expansion chassis (on PX4) for a one-expansion chassis subsystem.

DG-07991

Figure 5.49 Expansion chassis, S130, AP/130 and C/150

ECLIPSE S/250, C/350 Computers and System Modules

The TTY cable is plugged onto the backpanel at the location shown on slot 32. Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel.

System modules may be added to expand the physical size of the system. I/O device cables, placement of backpanel terminators, and system modules are shown on the following pages.

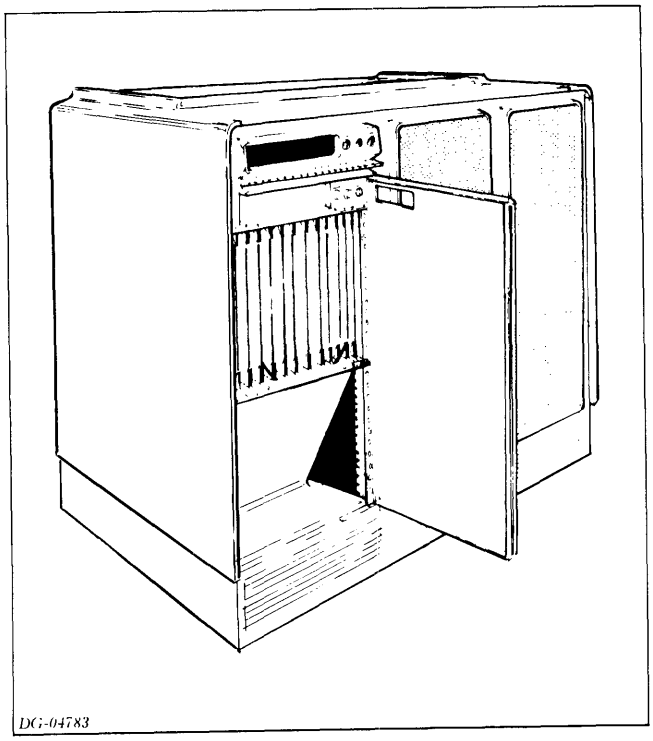


Figure 5.50 S/250, C/350 computer

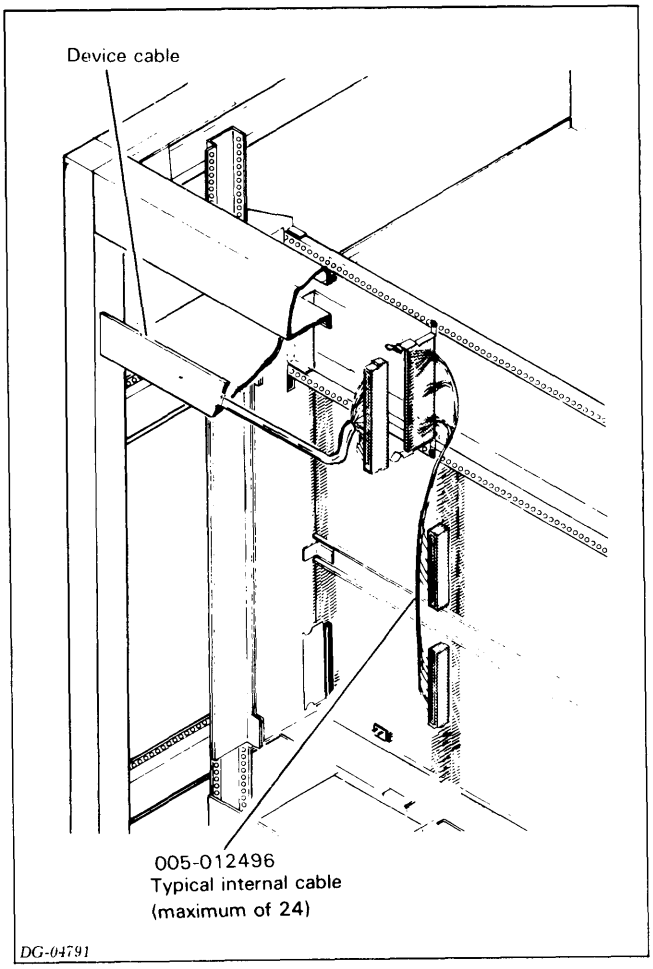


Figure 5.51 I/O cable connection

CONNECTIONS, CONNECTORS AND TERMINATORS

Internal Cables			External Cables	
From	To	Assy	From	Assy
4010	EIA Connector Panel	005-010706*	UMCS, Modem	005-010711
UMCS, ALM-16	EIA Connector Panel	005-010708	Serial I/O for 6052/3 display and serial printer	005-010707
UMCS, ALM-8	EIA Connector Panel	005-010710		
UMCS, SLM	EIA Connector Panel	005-012804		
ULM/5 SYNC Line	EIA Connector Panel	005-010709		
DCU/50 or DCU/200	EIA Connector Panel	005-012590		
MCA	EIA Connector Panel	005-012585		
TTY- 4007 Type	EIA Connector Panel	005-012473		
ULM/5	EIA Connector Panel	005-012765		
	Device			

Table 5.4 Cables for communications equipment

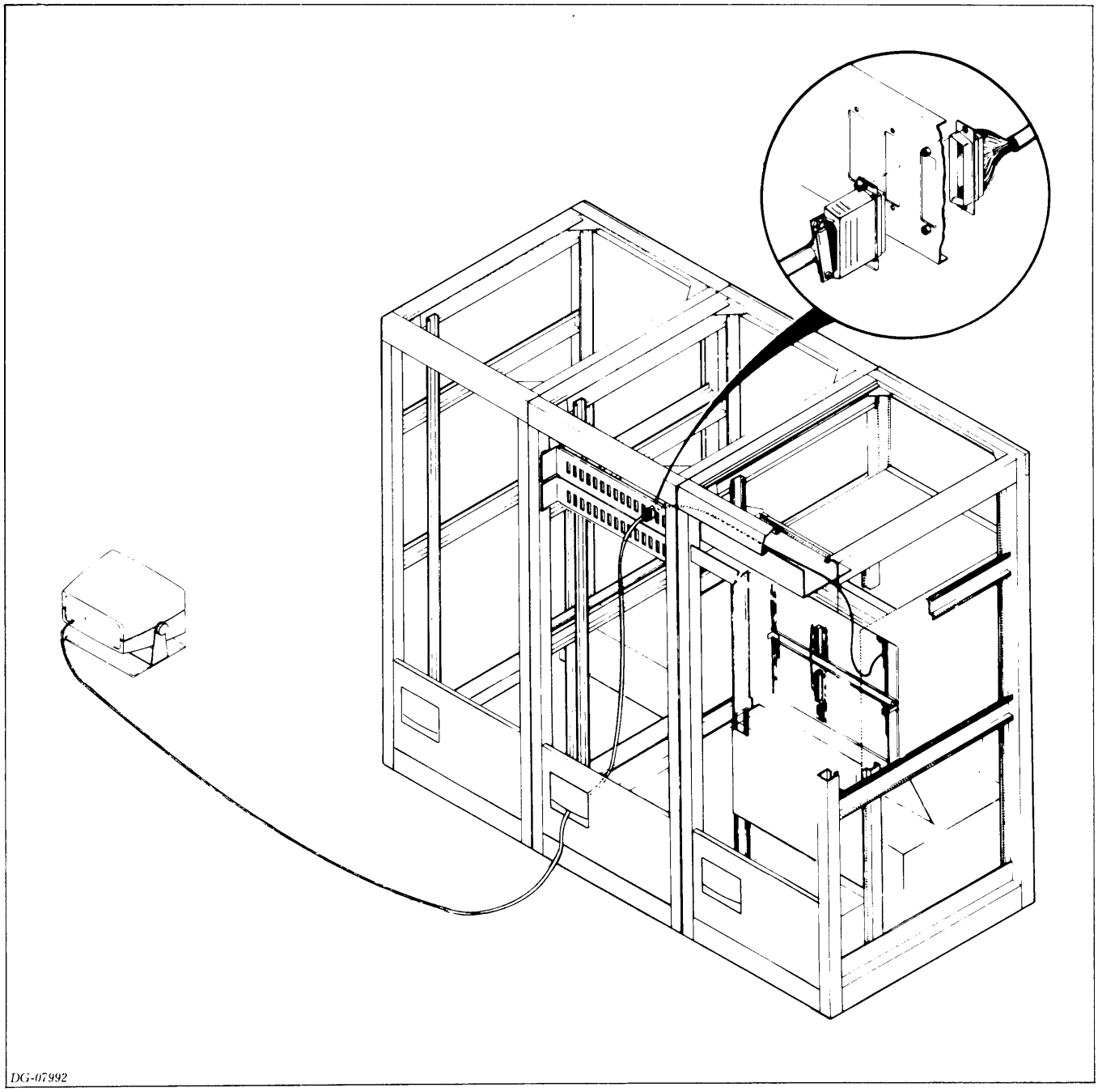


Figure 5.52 Socket connections for communications equipment (e.g. terminals)

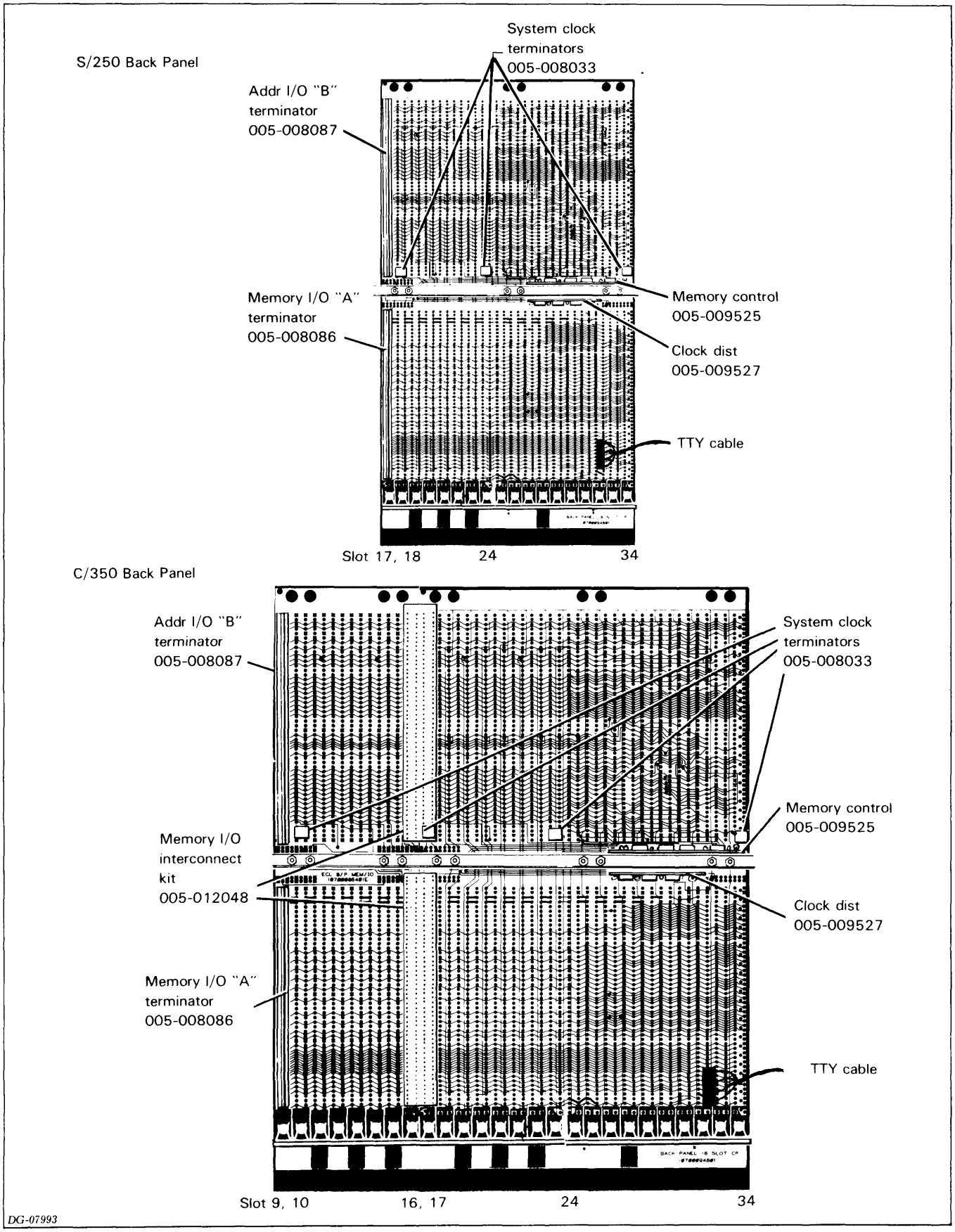


Figure 5.53 TTY cable and terminator placement

ECLIPSE M/600 Computers

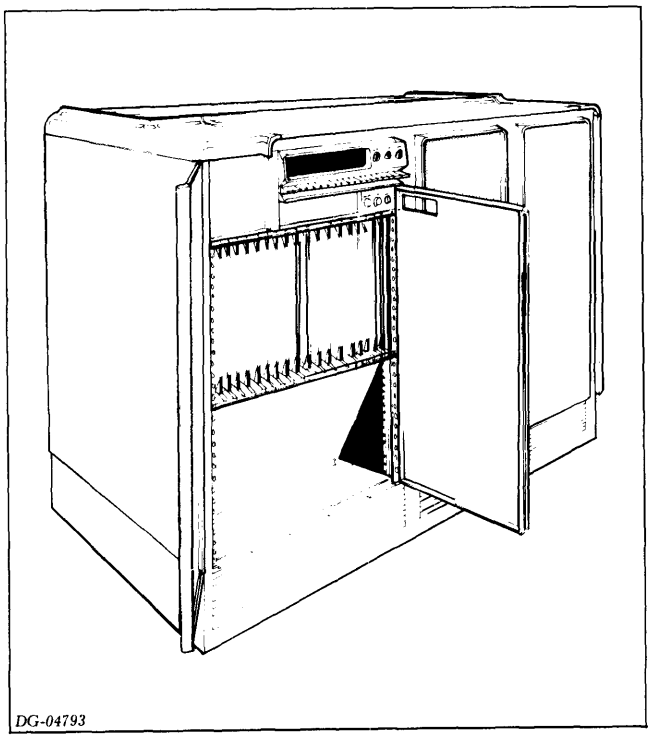


Figure 5.54 M/600 computer

The TTY cable is plugged onto the back panel at slot 22 shown in Figure 5.53. Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel.

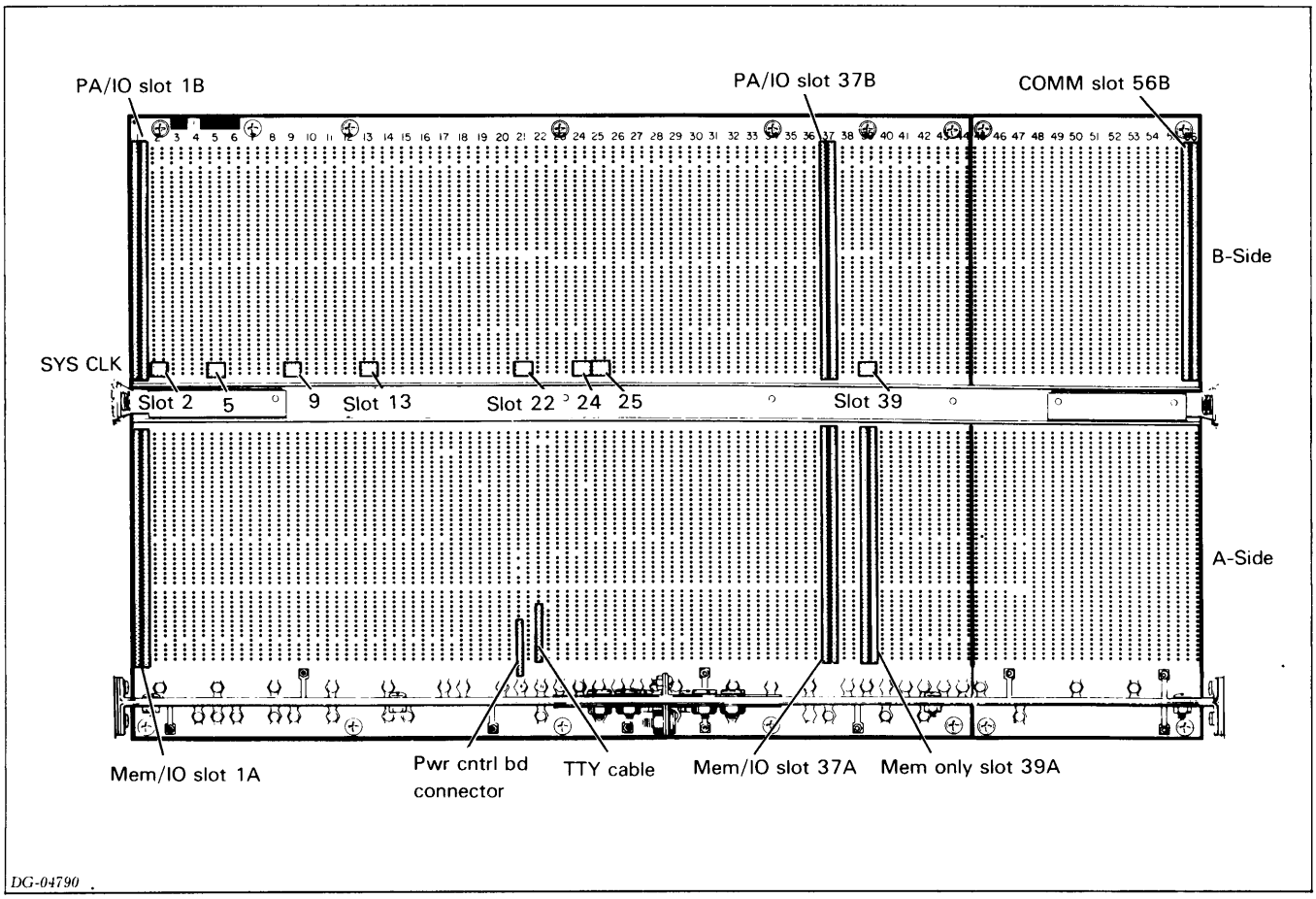


Figure 5.55 M/600 backpanel with terminators

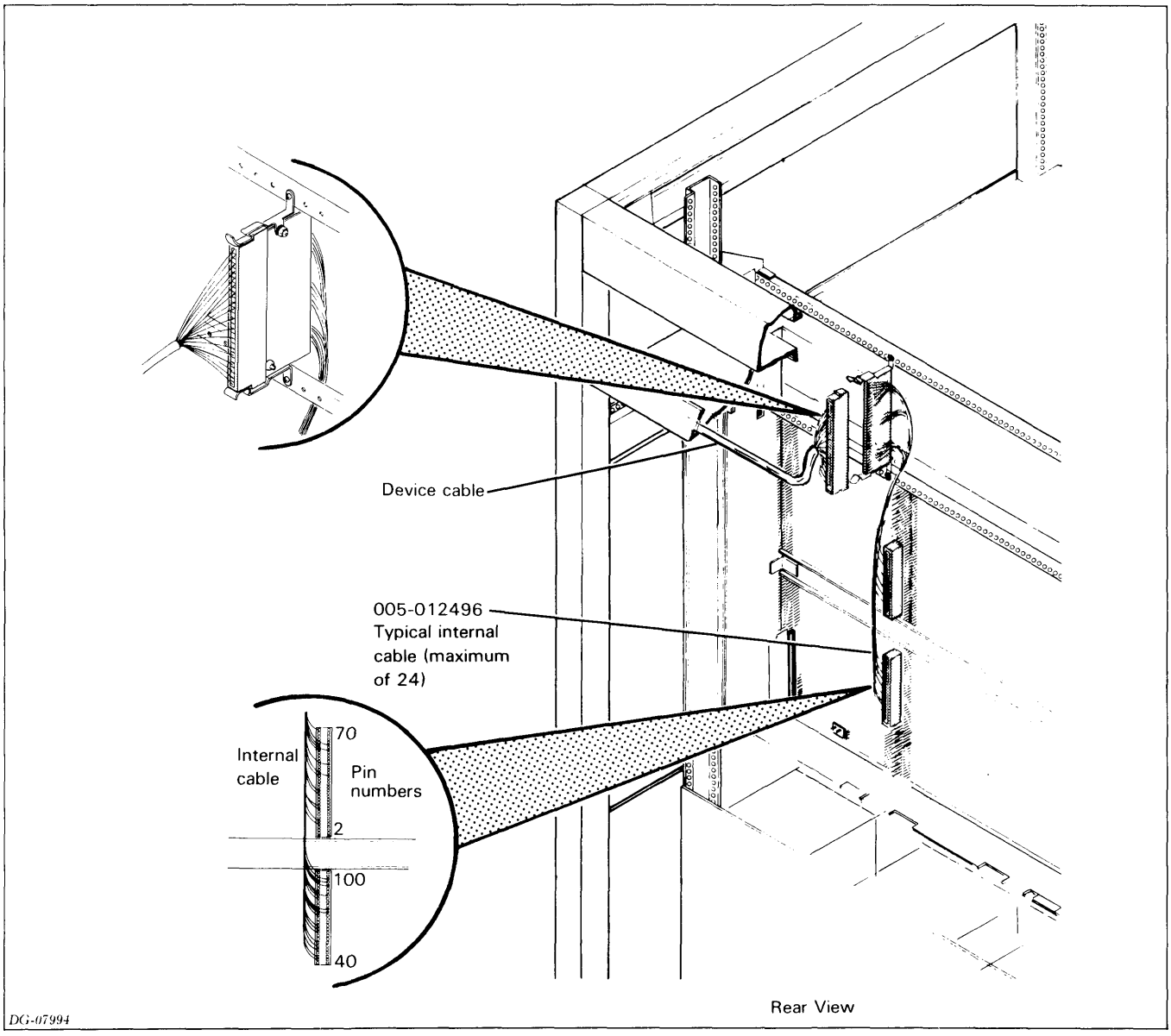


Figure 5.56 I/O cable connection

ECLIPSE MV/8000 Computers

The TTY cable is plugged onto the asynchronous port of the microNOVA board, as shown in Figure 5.57. Terminators are shown in Figure 5.58.

Other device cables are plugged onto connectors mounted, as needed, at the rear of the backpanel.

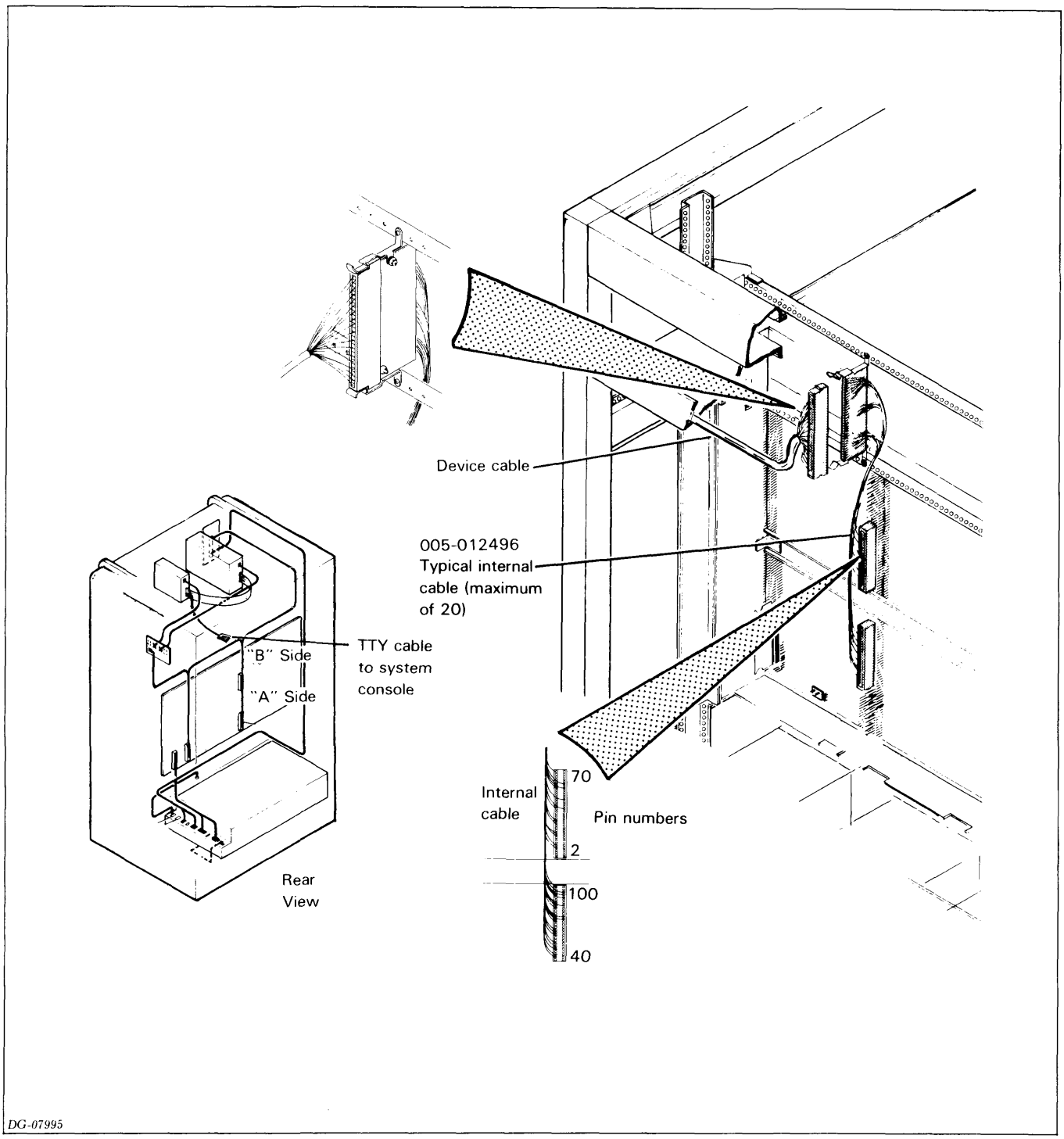
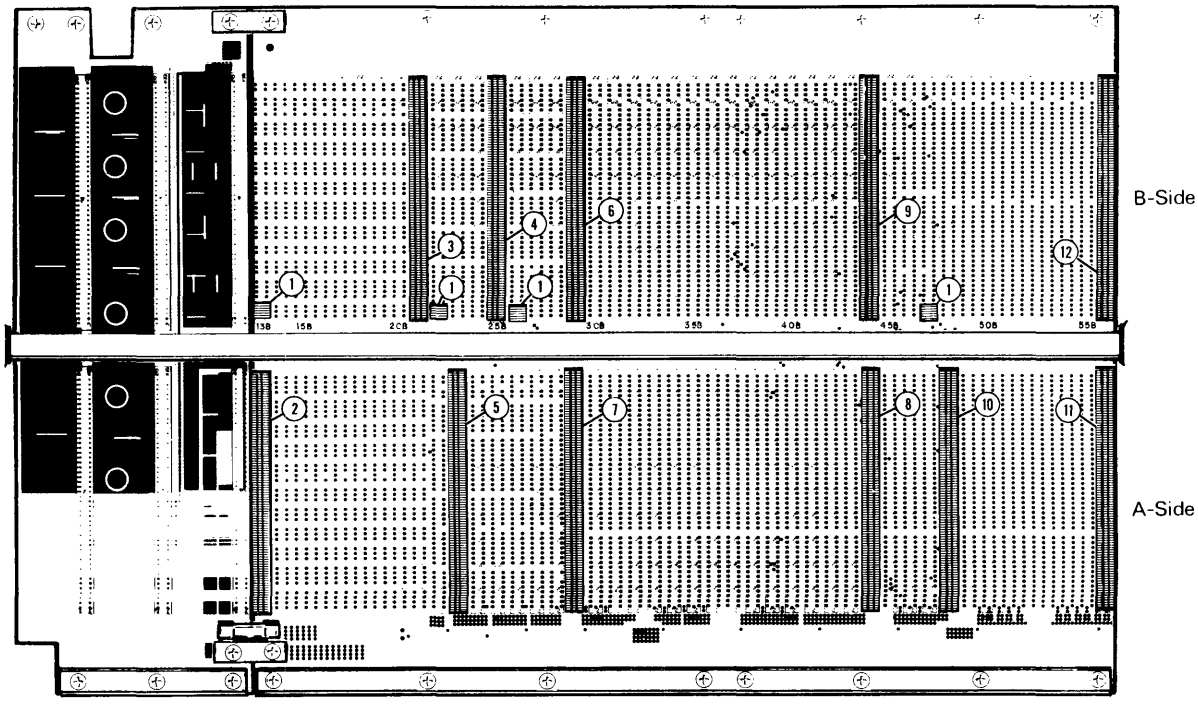


Figure 5.57 I/O cable connection



Terminator		
1	SYS CLK	005-014291
2	MEM A	005-014301
3	MEM B	005-014289
4	CPORT A	005-014293
5	CPORT B	005-014295
6	IPORT A	005-014297
7	IPORT B	005-014299
8	MEM I/O	005-008086
9	I/O only	005-015695
10	MEM only	005-008034
11	MEM I/O	005-008086
12	J/O only	005-015695

DG-07273

Figure 5.58 Terminator placement, backpanel, rear view

Connectors

When standard devices are purchased from Data General, the internal cables and connectors necessary for installation are included with the machine. When a custom interface/device installation is being planned, however, the necessary connectors should be included in the plans. The following is a list of some of the general purpose connectors available from DGC.

4192

A general purpose external device connector that includes both the internal cabling and the proper chassis-mounted cable connector; a 50-pair paddleboard connector, or a 50-pin female cannon connector. The relationship between backpanel pins and assigned connector pins is listed at the end of this section.

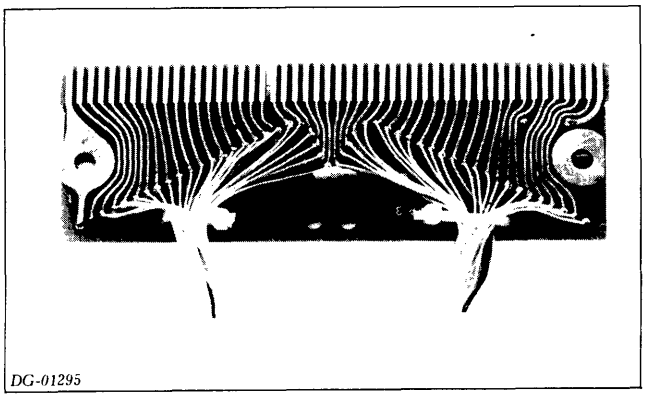


Figure 5.59 Typical wire-wrapped edge-connector

1070B

The 50-pair paddleboard connector used on the 4192.

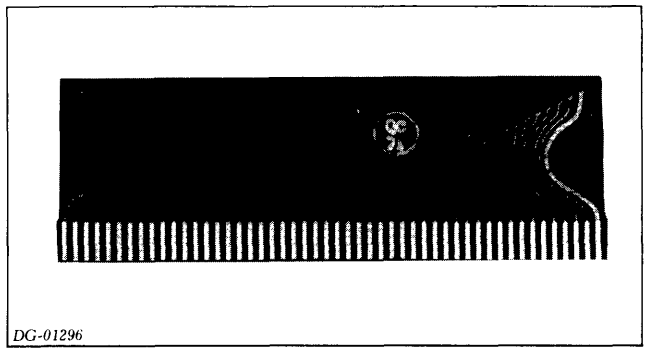


Figure 5.60 50-pair paddleboard connector

005-001858

A 50-pair, female paddleboard cable connector, to mate with the 4192, and 1070B.

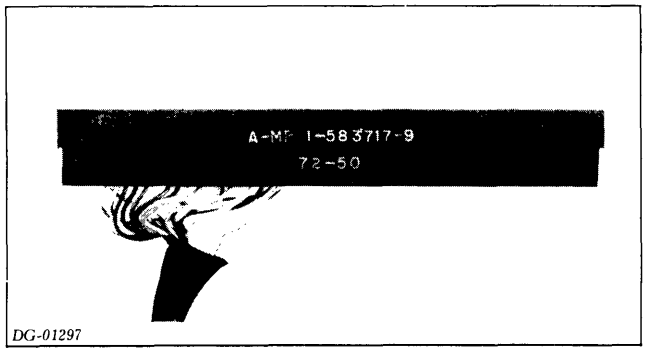


Figure 5.61 Typical edge-connector external cable

4083

A connector panel which includes sixteen 13-pin, male pin type connectors. This panel mounts on the chassis of the NOVA 2/4 or NOVA 2/10.

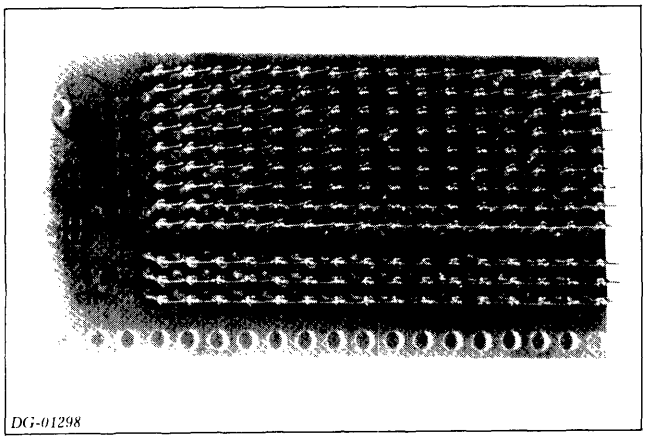


Figure 5.62 Connector panel

1051G

A 13-pin, female pin-type connector, to mate with the male connectors as used on the 4083.

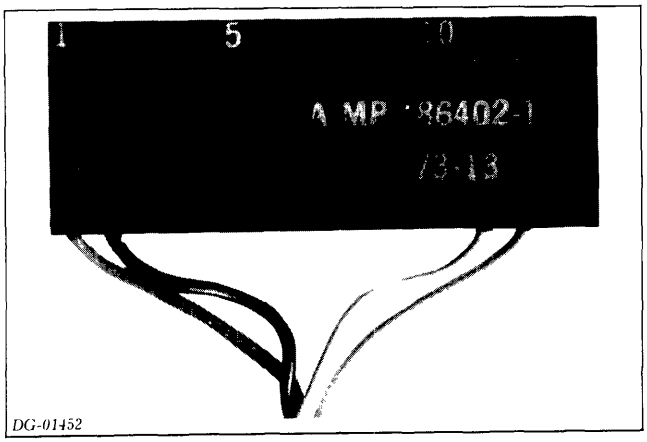


Figure 5.63 Typical pin type connector

005-012496

This is a general purpose internal cable. If the NOVA 4/5, NOVA 4/16, ECLIPSE M/600, C/350, S/250, C/150, S/140, or MV/8000 is specified, DGC No. 005-012496 internal cable is used.

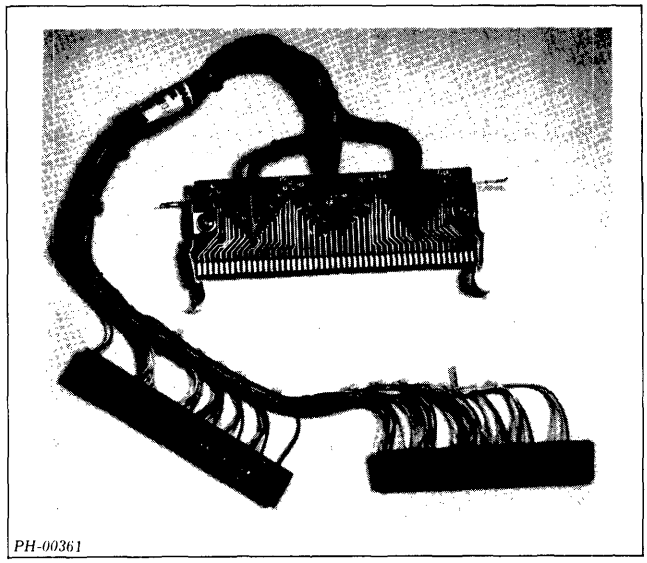


Figure 5.64 Typical push-on edge-connector

Socket Connectors

The various socket connectors and connector configurations available from Data General as standard items are listed at the end of this chapter. Note that for chassis mounting, the female connector is mounted on the chassis, while the male connector is attached to the cable.

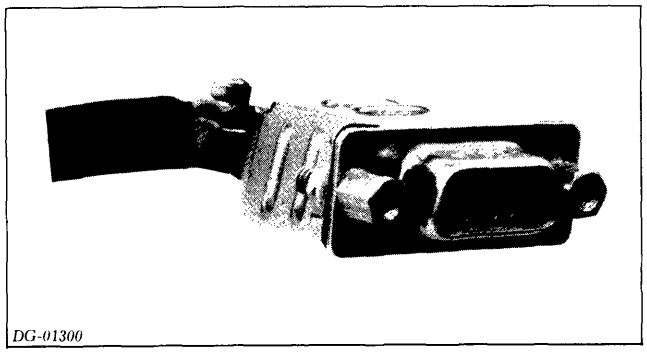
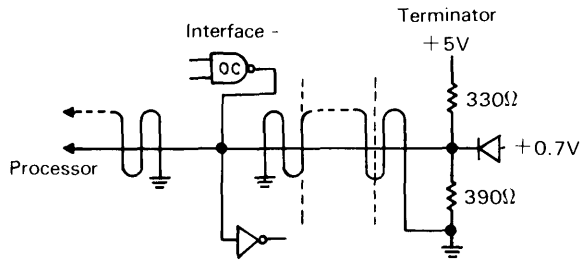


Figure 5.65 Typical socket connector

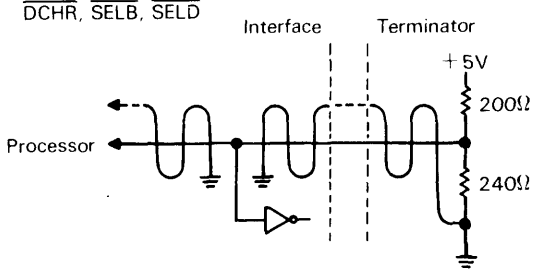
Terminators

Because the I/O bus cable is being used as a high-frequency transmission line, it is very important that the cable be terminated at or near its characteristic impedance. Any mismatch between the terminator and the cable will cause electrical reflections within the cable, which manifest themselves as a damped oscillation, or "ringing". This ringing appears not only at the end of the line, but at all points on the line. Such ringing creates several problems, chief of which is an increase in the time delay before a signal has settled down sufficiently so that it can be considered reliable. This problem will become especially evident when using the high-speed data channel, despite the fact that the interface is mounted within the chassis.

Figure 5.66 shows schematics of the recommended termination of I/O bus signals originated by the central processor and by the interface. This is the scheme used in the terminators shown in Figures 5.67 and 5.68.



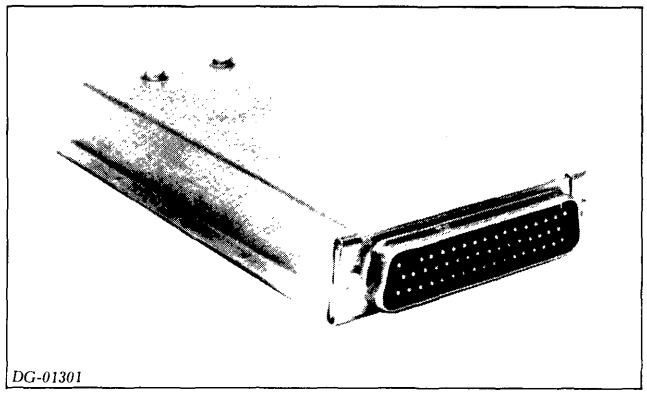
Recommended termination for bi-directional lines and signals from interface to processor: $\overline{\text{DATA}}$ <0-15>, $\overline{\text{DCHM}}$<0-1>, $\overline{\text{DCHR}}$, $\overline{\text{SELB}}$, $\overline{\text{SELD}}$



Recommended termination for processor originated I/O bus signals CLR, DATIA, DATIB, DATIC, DATOA, DATOB, DATOC, $\overline{\text{DCHA}}$, DCHI, DCHO, $\overline{\text{DS}}$<0-5>, INTA, IORST, IOPLS, $\overline{\text{MSKO}}$, OVFLO, $\overline{\text{RQENB}}$, STRT

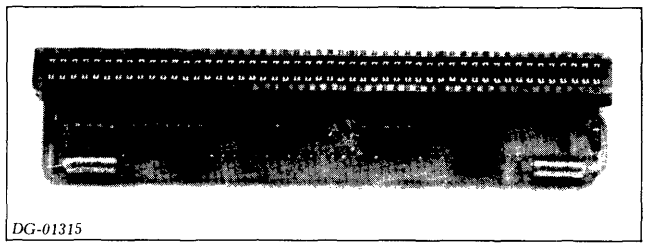
DG-07996

Figure 5.66 Recommended termination



DG-01301

Figure 5.67 Socket terminator



DG-01315

Figure 5.68 Paddleboard-type terminator

A terminator should always be used when the I/O bus extends beyond the computer chassis.

Computer Model	Type of Connector	Terminator Part Number
NOVA Computer	Socket	005-000116
SUPERNOVA Computer	Socket	005-000116
1200, Jumbo 1200	Socket	005-000116
1220	Paddleboard	005-001219
800, Jumbo 800, 830, 840	Socket	005-000116
820	Paddleboard	005-001219
*NOVA 2	Paddleboard	005-001734
**		

Table 5.5 Extended I/O bus terminators

**Terminators are used on the computer chassis of the NOVA 2/10*

***For later Data General computers see the Installation and Packaging for Data General Products DGC No. 014-000605.*

Terminators can be mounted on either paddleboard or socket connectors. Table 5.5 lists terminator part numbers and shows the type of connector used. These terminators are plugged directly onto the mating connector, which would otherwise be used to extend the I/O bus.

Type Number	Connector
005-002244	100-pin female connector; consists of 111-12 and 111-22
005-002245	100-pin male connector; consists of 111-11, 111-21, and 111-25
005-002246	52-pin female connector; consists of 111-10 and 111-22
005-002247	52-pin male connector; consists of 111-9, 111-20, and 111-24
005-002248	25-pin female connector; consists of 111-4 and 111-22
005-002249	25-pin male connector; consists of 111-3, 111-20, and 111-24
005-002250	19-pin female connector; consists of 111-8 and 111-22
005-002251	19-pin male connector; consists of 111-7, 111-19 and 111-23
005-002252	9-pin female connector; consists of 111-2 and 111-22
005-002253	9-pin male connector; consists of 111-1, 111-19 and 111-23
005-002254	50-pin I/O female connector; consists of 111-6 and 111-22
005-002255	50-pin I/O male connector; consists of 111-5, 111-21 and 111-25
111-000001	9-pin male connector
111-000002	9-pin female connector
111-000003	25-pin male connector
111-000004	25-pin female connector
111-000005	50-pin male connector
111-000006	50-pin female connector
111-000007	19-pin male connector
111-000008	19-pin female connector
111-000009	52-pin male connector
111-000010	52-pin female connector
111-000011	100-pin male connector
111-000012	100-pin female connector
111-000019	9/19 pin junction shell
111-000020	25/52 pin junction shell
111-000021	50/100 pin junction shell
111-000022	Screw lock assembly, female
111-000023	9/19 pin screw lock assembly, male
111-000024	25/52 screw lock assembly, male
111-000025	50/100 pin screw lock assembly, male

Table 5.6 Socket connectors

Chapter 6
INTERFACE BOARDS

Introduction

This chapter gives the specifications that printed circuit interface boards must meet in order to be installed in NOVA and ECLIPSE line chassis and describes prefabricated interface boards available from Data General.

Printed Circuit Board Specifications

NOVA and ECLIPSE line chassis impose certain restrictions on the dimensions, vertical clearances, power consumption and heat dissipation of the printed circuit boards which can be installed in them.

Dimensions

Figures 6.1 and 6.2 show the dimensions of printed circuit boards to be installed into NOVA and ECLIPSE line chassis.

Vertical Clearances

The clearance between boards in a computer chassis or between a board and the top of the chassis is 3/8 inch. To maintain installation compatibility with other boards in the chassis, components should be mounted only on the top surface of the board. Components must not project more than 0.312 inches above the board, and no protrusion below the board may be greater than 0.062 inches. In general, a board should be constructed with as low a profile as possible, so that air flow is restricted as little as possible.

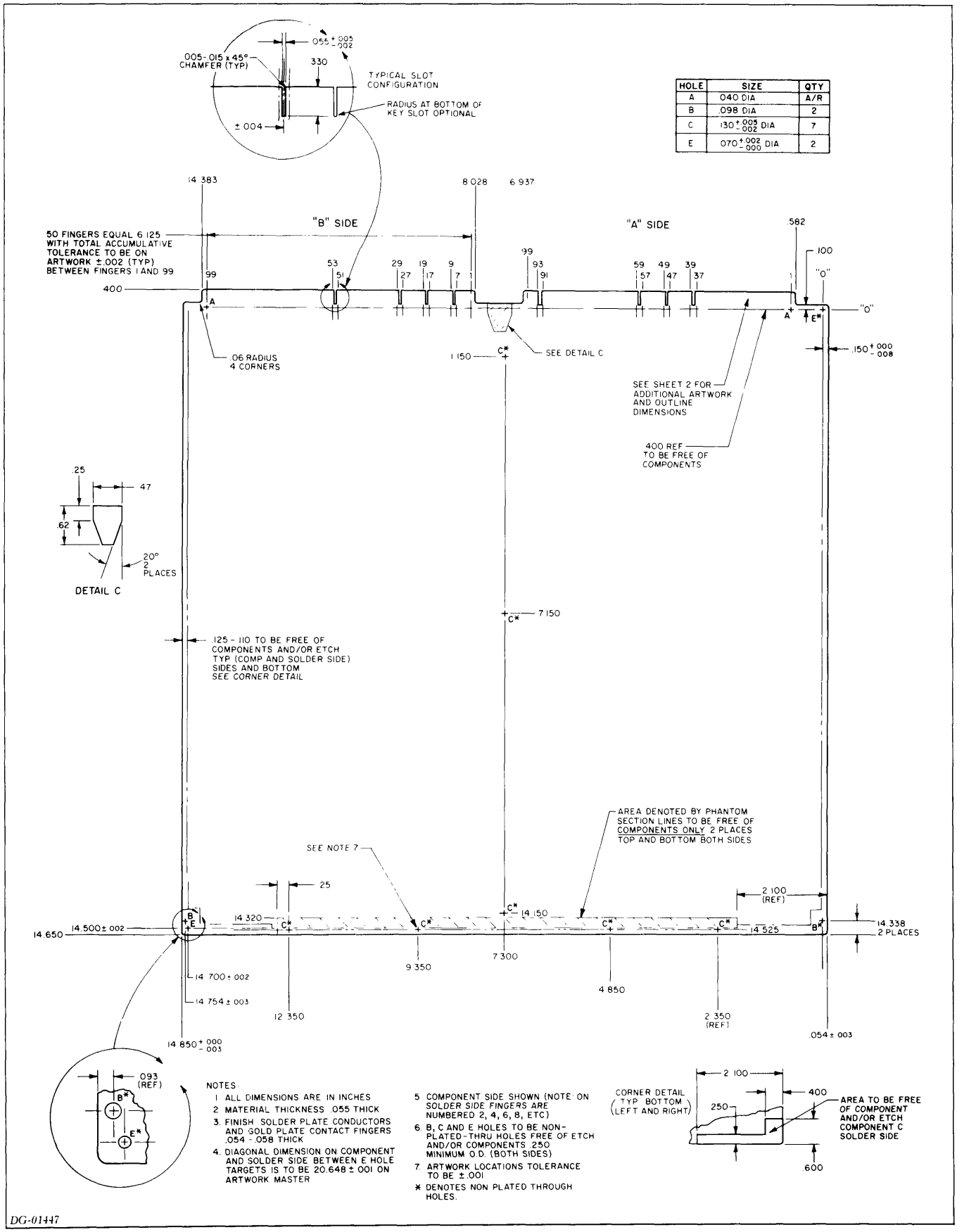
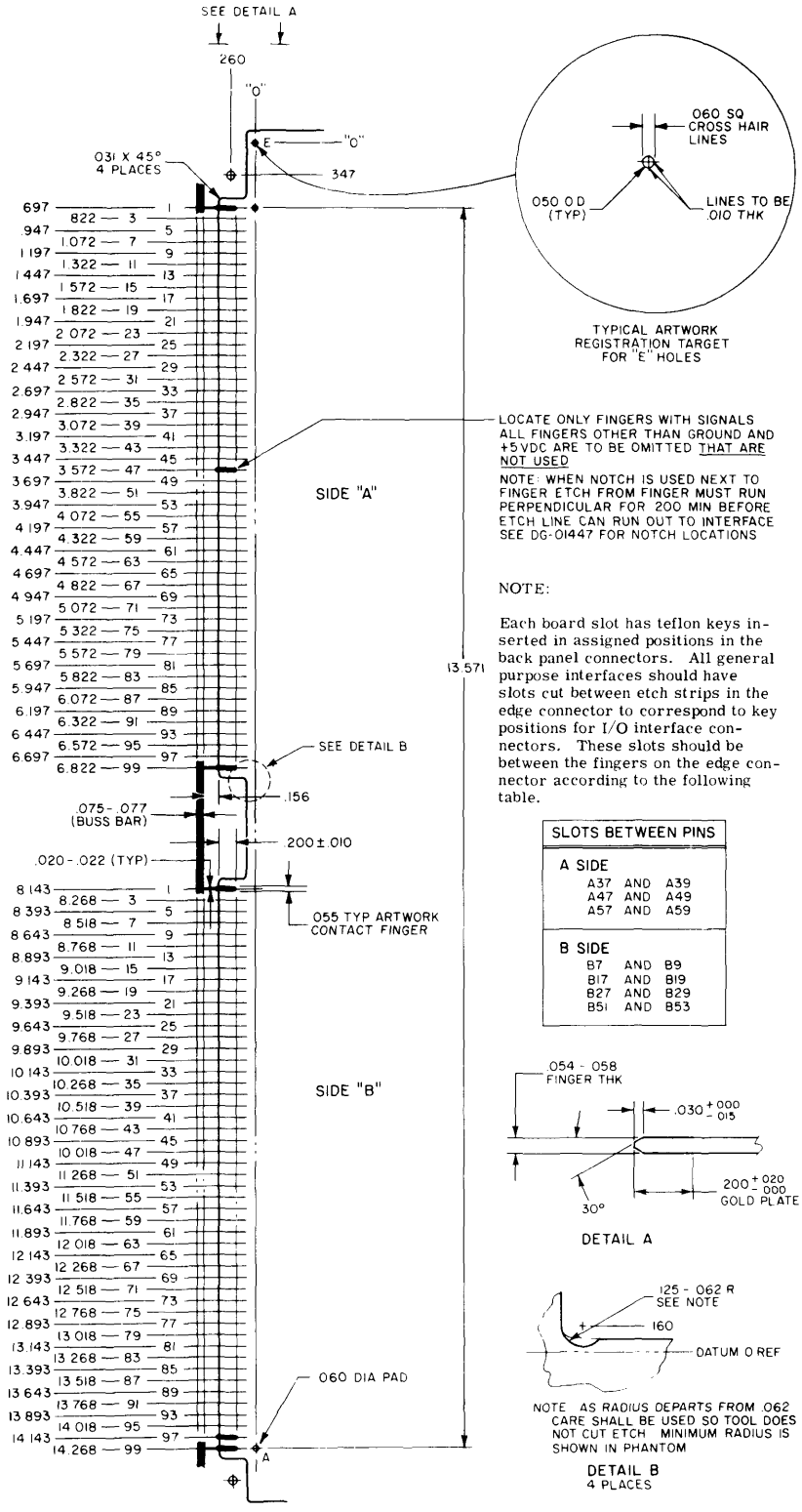


Figure 6.1 Mechanical dimensions



NOTE:

Each board slot has teflon keys inserted in assigned positions in the back panel connectors. All general purpose interfaces should have slots cut between etch strips in the edge connector to correspond to key positions for I/O interface connectors. These slots should be between the fingers on the edge connector according to the following table.

SLOTS BETWEEN PINS	
A SIDE	
A37 AND A39	
A47 AND A49	
A57 AND A59	
B SIDE	
B7 AND B9	
B17 AND B19	
B27 AND B29	
B51 AND B53	

DETAIL A

NOTE AS RADIUS DEPARTS FROM .062 CARE SHALL BE USED SO TOOL DOES NOT CUT ETCH MINIMUM RADIUS IS SHOWN IN PHANTOM

DETAIL B
4 PLACES

DG-01448

Figure 6.2 Connector specifications

DC Power Requirements

Interfaces installed in a computer chassis or expansion chassis receive power (+5Vd.c. and ground) through the back panel from the chassis power supply. This immediately presents two considerations for the designer. The first is the capacity of the power supply and the second is the quality of the voltages supplied. This chapter provides a table of power supply and back panel print numbers for each chassis.

It is important that an interface added to a computer or expansion chassis not overload the power supply of that chassis. Thus, the designer should know what the capacity of his machine is and design accordingly. Note that the capacity here is the total capacity of the power supply minus the load from the boards already installed in the chassis.

All chassis produce +5Vd.c., +15Vd.c. or +12Vd.c., and -5Vd.c.. Many computer chassis also produce -15Vd.c. for customer use. The price list and installation material contains the +5Vd.c. capacities of the computers as well as the current draw on this supply by various circuit boards. The designer should use this information to calculate the actual capacity of his computer system.

If it is found that there is insufficient capacity to drive the planned interface, there are a number of possible courses of action. The designer may wish to configure his system somewhat differently, perhaps using an expansion chassis for this or other interfaces. Another possibility, especially if the planned interface is fairly large, is to build most (or all) of the interface in its own chassis, with its own power supply.

In addition to power supply capacity, the interface designer may have to concern himself with the quality of the voltage available. A characteristic of TTL logic is the tendency to superimpose switching transients and other noise on the power supply line. This noise may be particularly troublesome to analog circuitry, especially where, for instance, high-gain amplifiers are being used to deal with low-level signals. If power supply noise is a problem, it may be desirable to isolate the interface circuitry from the supply either by incorporating additional regulation on the interface assembly, or by using a DC/DC converter.

Heat Dissipation of Interface Boards

The problem of heat dissipation really has two facets. The first, a local effect, is the consideration of how the heat produced by hot components may affect nearby components; the second is the degree to which the heat produced by an interface assembly will raise the ambient temperature of the adjacent boards in the chassis. Both of these questions involve analyses that will be discussed only in the most basic terms here; but they are mentioned in the hope that the designer will remain aware of the possible restrictions.

The principal problem involved with localized heating is caused by high dissipation devices concentrated together creating hot spots. Every computer and expansion chassis includes fans for forced air movement over the boards. Sufficient air moves past the heat-producing components to keep the air temperature rise within reason, thereby partially limiting the temperature level. Spreading the heat source over a wider area will improve the heat transfer across the board. Whenever possible, heat sensitive circuits and components should be mounted as far as possible from high-temperature components.

Effects on internal temperature rise can be minimized principally by limiting the total heat dissipation of the interface board. If excessive heat is dissipated on the board, the result will be a rise in the ambient temperature for nearby boards. Care should be taken that the profile of the board is low enough so as not to interfere with the air flow across the board. In general, if the total dissipation of an interface board is less than fifteen watts, there will be no problem of overheating other boards.

Prefabricated Interface Boards

To aid the designer in the construction of custom interface assemblies, Data General makes available two series of interface circuit boards. These are particularly useful for building limited quantities of a special interface; that is, where it would be economically unsuitable to lay out and etch a printed circuit board. The 1000 and 1020 series general purpose wiring boards are blank component boards that allow the designer a high level of flexibility in the configuration of his design.

1000 Series General Purpose Wiring Boards

The 1000 wiring board, shown in Figure 6.3, is a system consisting of 6 1/2 x 3 1/4 inch module boards mounted on a 15-inch square frame (type 1001). This frame has two 100-pin edge connectors with solder pads, which are compatible with the female back panel edge connectors. There is space on the frame for up to eight wiring modules.

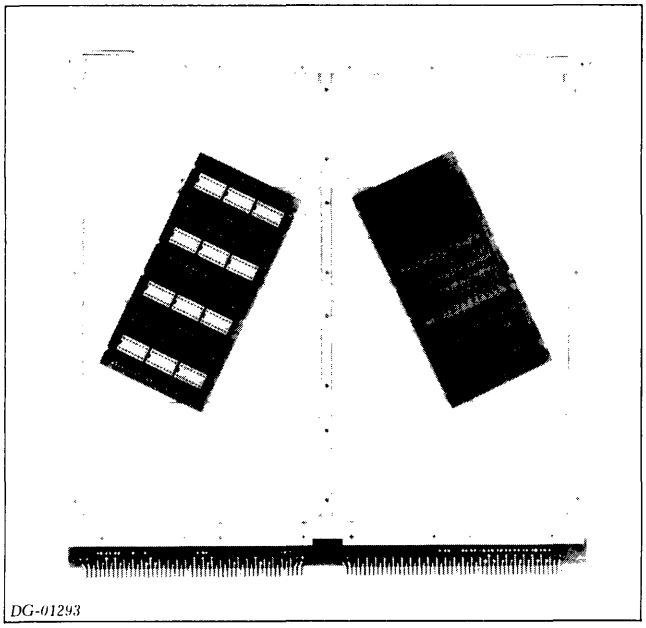


Figure 6.3 1000 series wiring boards

The modules themselves are available in three varieties. The 1002 is a basic board with hole patterns for twelve 14- or 16-pin integrated circuits. Discrete components can also be mounted in these holes. Solder pads are provided for the larger 24- and 36-pin chips, but the holes are not predrilled through the board and the solder pads. Interconnections on the 1002 are made by soldering interconnecting wires of #30 AWG teflon-coated solid wire to solder pads provided for each integrated circuit terminal. The 1003 board provides wire-wrap pins for these connections, as shown in Figure 6.4. For added convenience in mounting the chips, the 1004 board provides, in addition to the wire-wrap pins, twelve 16-pin low-profile sockets for dual in-line chips.

Finally, there is a protective cover type 1014 which fits over the 1001 wiring frame.

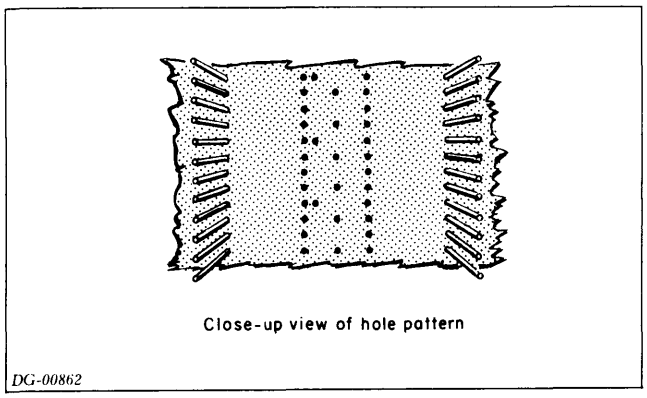


Figure 6.4 1000 series hole pattern

1020 Series General Purpose Wiring Boards

The 1021 wiring board, shown in Figure 6.5, is a 15-inch square printed circuit board with a hole pattern for 14-, 16-, 24-, and 36-pin integrated circuit chips, as well as discrete components. The board has two 100-pin edge connectors for the processor back panel, and includes power supply and ground buses throughout the board, including decoupling capacitors. The board can hold 155 14- or 16-pin integrated circuit packages. A pair of 24-pin packages replaces three 14- or 16-pin packages, while each 36-pin package replaces two 14- or 16-pin packages.

Interconnections on the 1021 board are made by soldering interconnecting wires of #30 AWG teflon-coated solid wire to solder pads provided for each integrated circuit terminal. The 1022 board provides wire-wrap pins for these interconnections. For added convenience in mounting the smaller integrated circuit packages, the 1023 board provides, in addition to the wire-wrap pins, 155 16-pin low-profile sockets for dual in-line integrated circuit packages. The protective cover used with this series is the 1024.

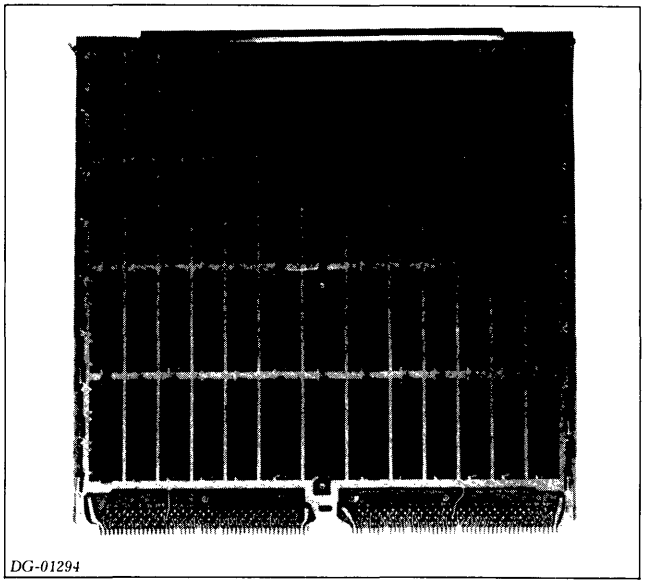
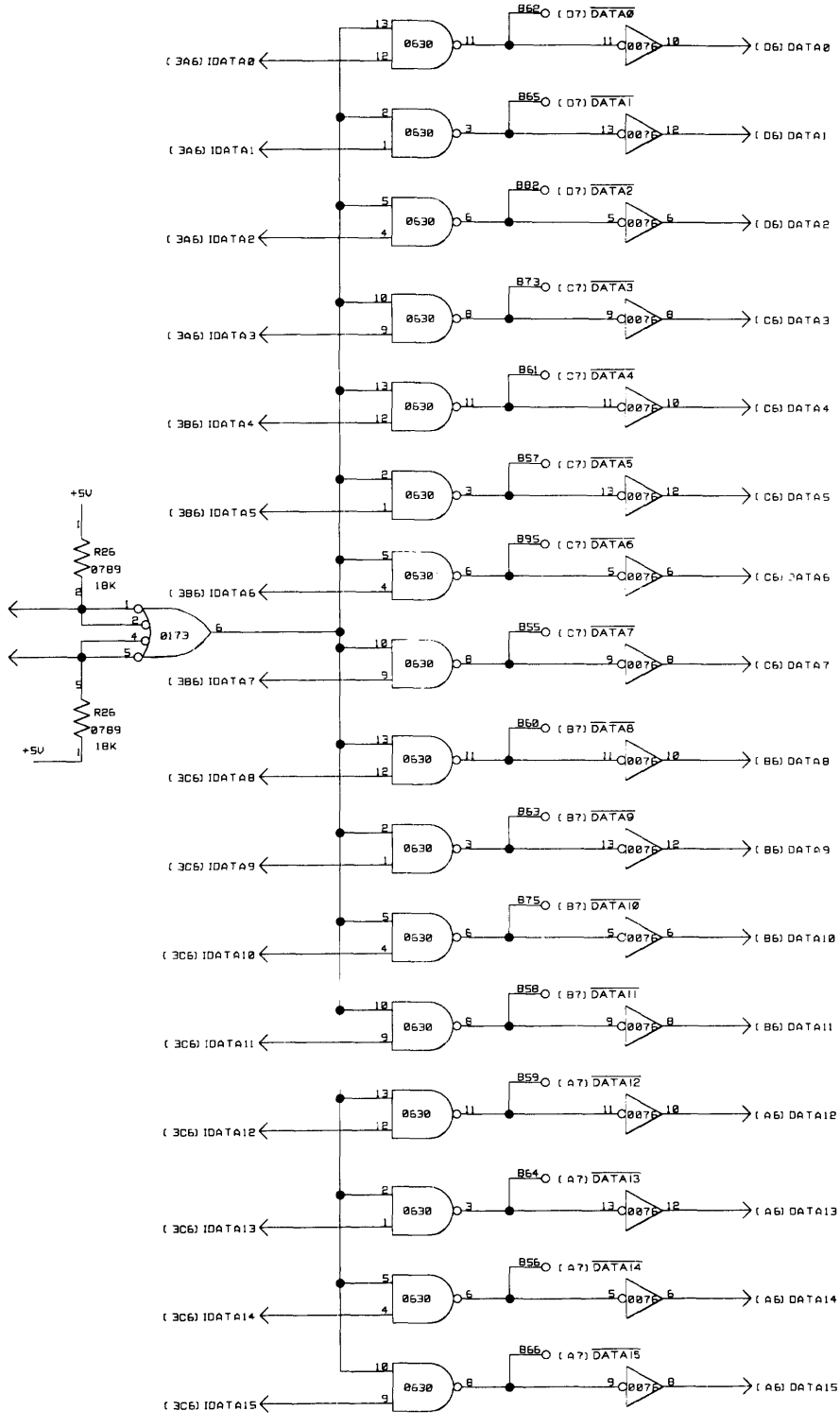


Figure 6.5 1020 series wiring board

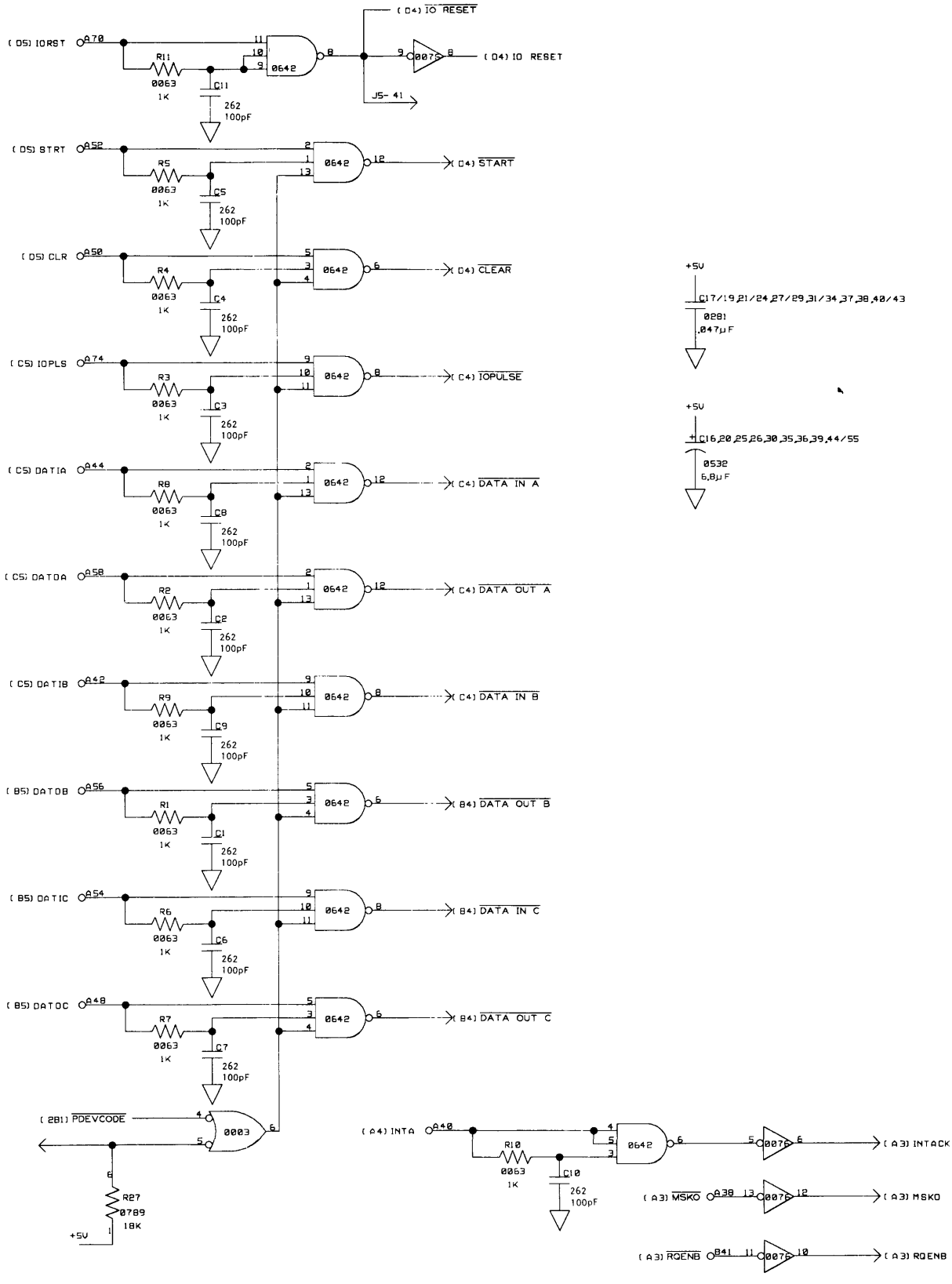
Sample Interface

Figures 6.6 to 6.10 illustrate suggested implementations for a general purpose I/O interface.



DG-08471

Figure 6.6 Programmed I/O interface receivers


(05) IORST
(05) STRT
(05) CLR
(05) IOPLS
(05) DATIA
(05) DATOA
(05) DATIB
(85) DATOB
(85) DATIC
(85) DATOC
(281) PDEVCODE
(04) IO RESET
(04) START
(04) CLEAR
(04) IOPULSE
(04) DATA IN A
(04) DATA OUT A
(04) DATA IN B
(84) DATA OUT B
(84) DATA IN C
(84) DATA OUT C
(J5-41)
0642
0007
0003
+5V
C17/19,21/24,27/29,31/34,37,38,40/43
0281
.047µF
C16,20,25,26,30,35,36,39,44/55
0532
6.8µF
(A4) INTA
(A3) INTACK
(A3) MSKO
(A3) MSKO
(A3) RQENB
(A3) RQENB

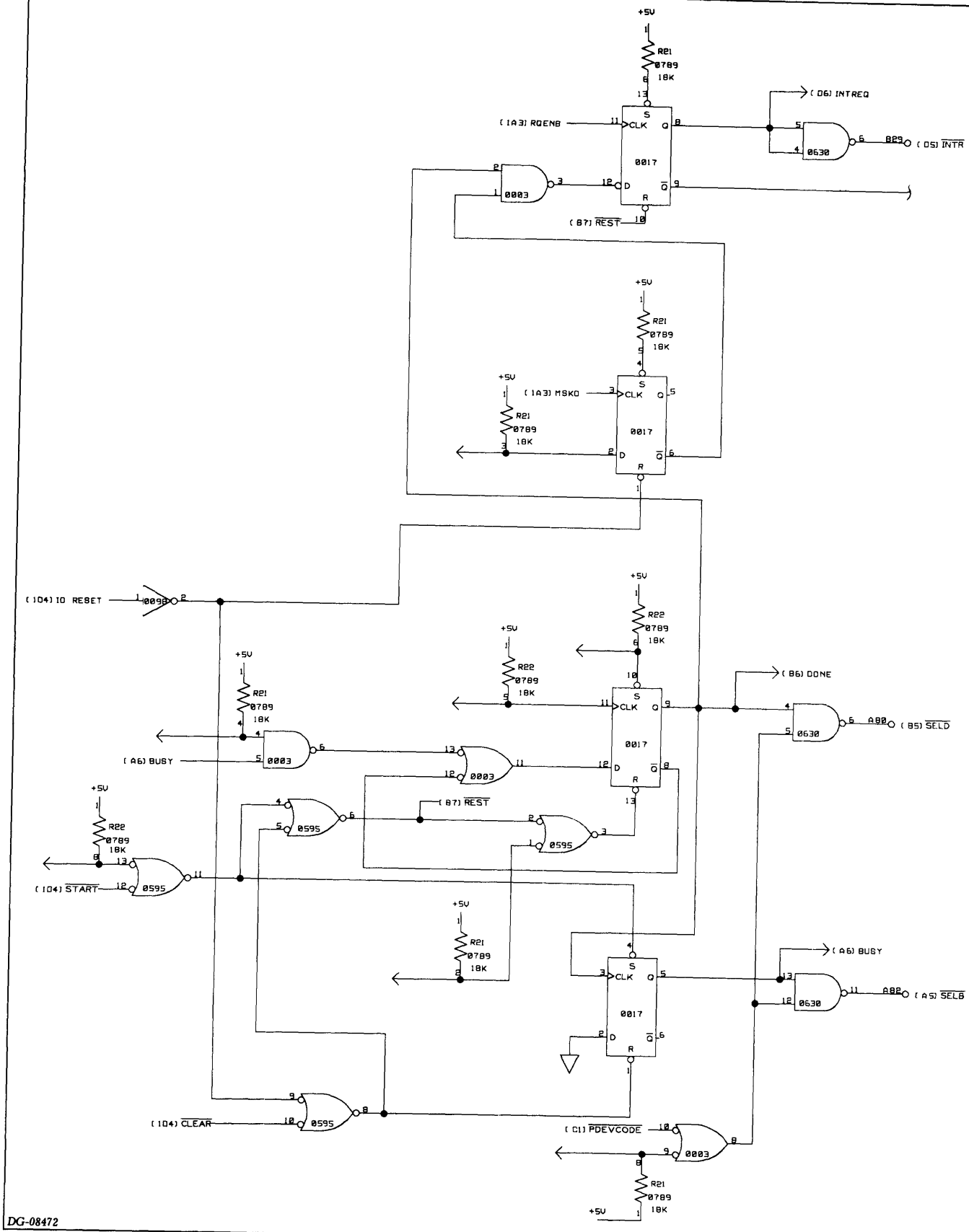
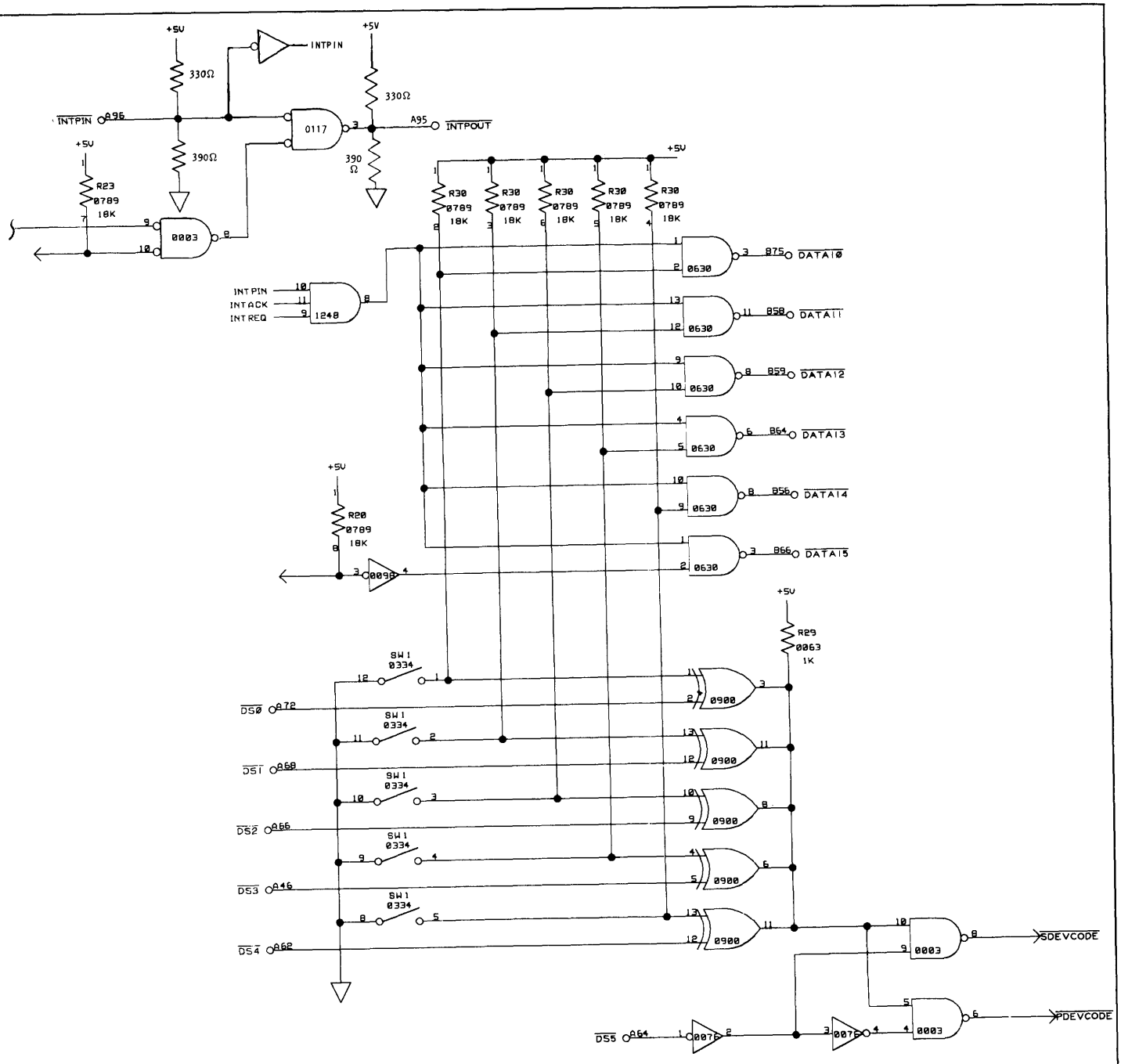
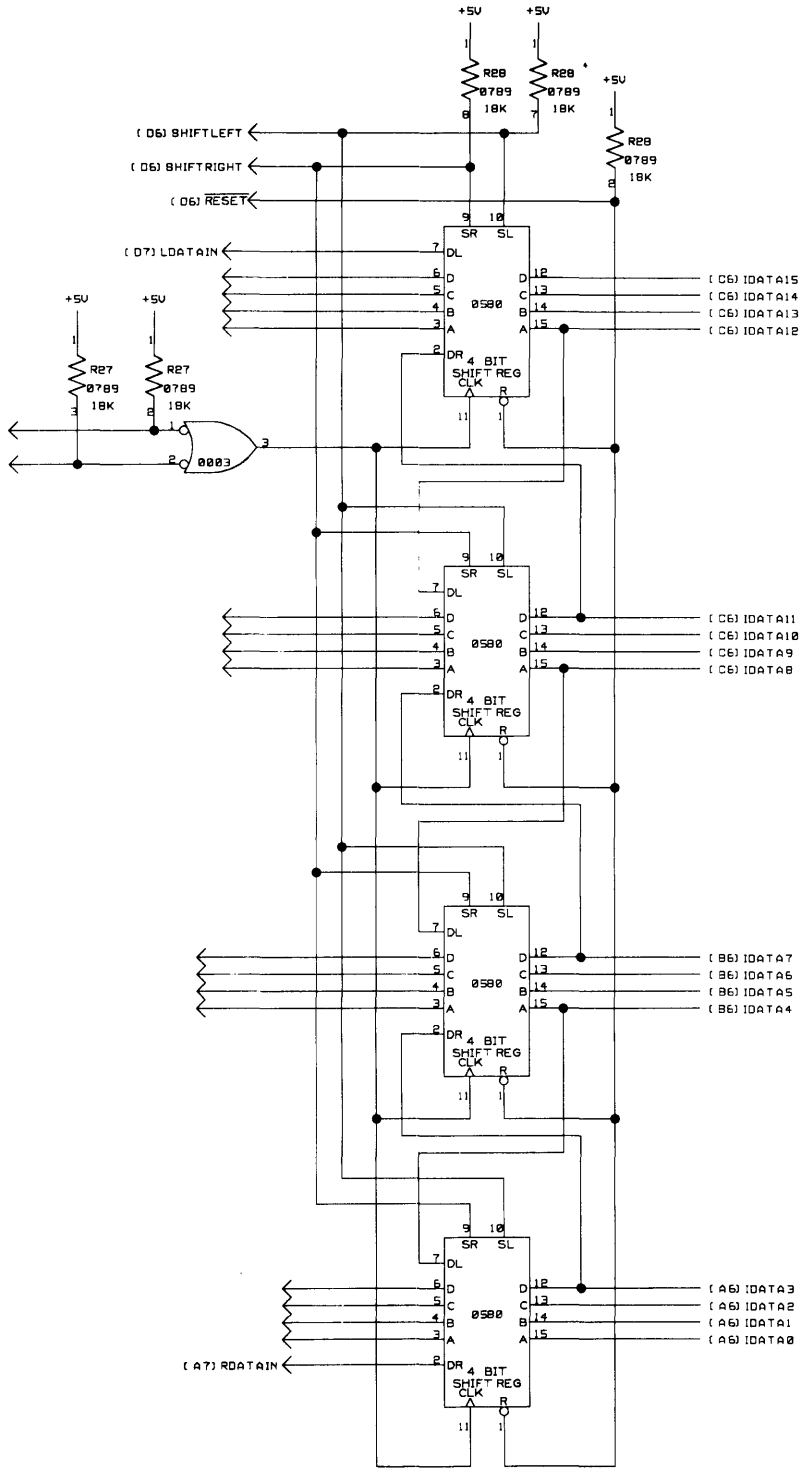


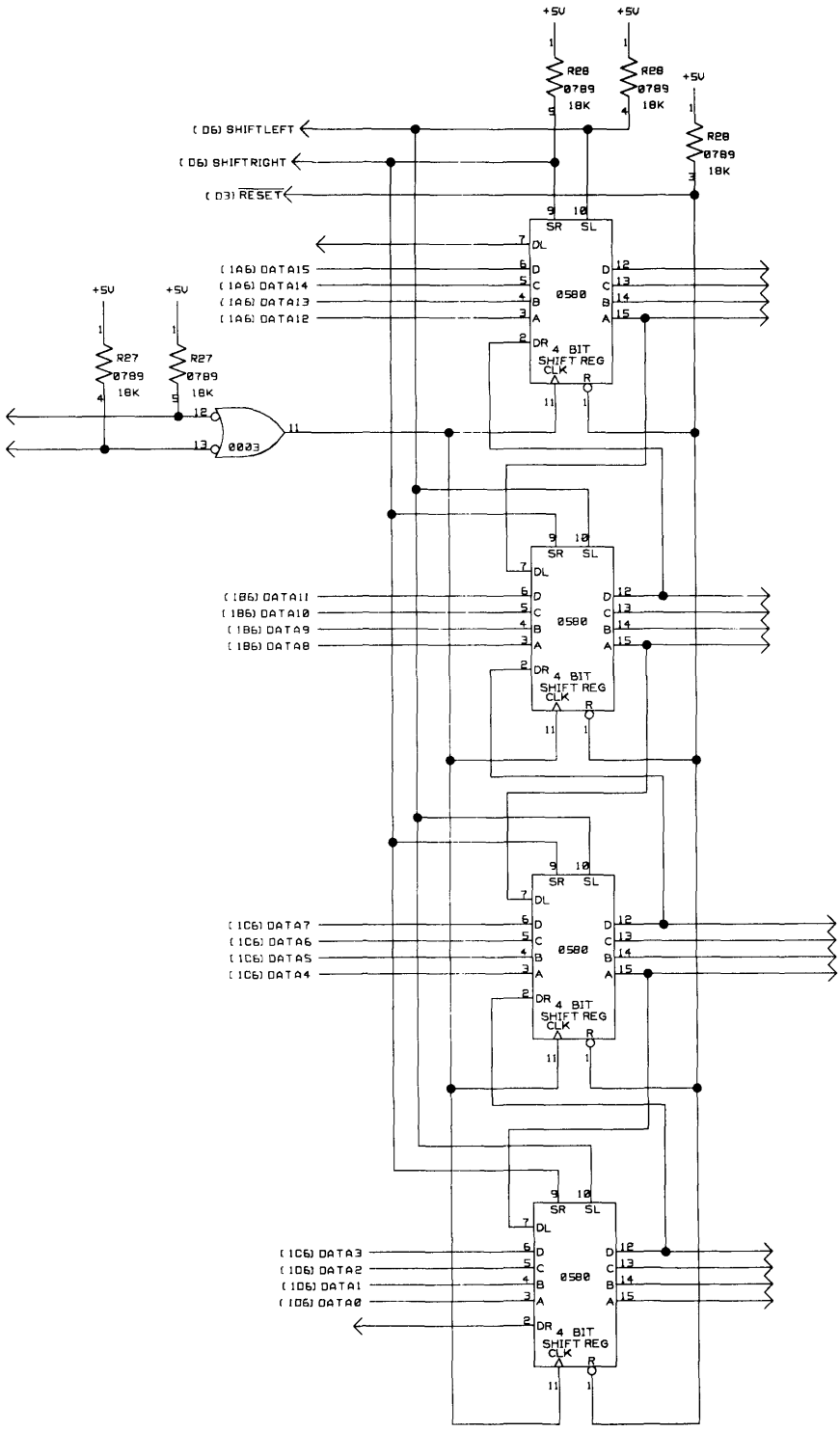
Figure 6.7 Programmed I/O control circuits

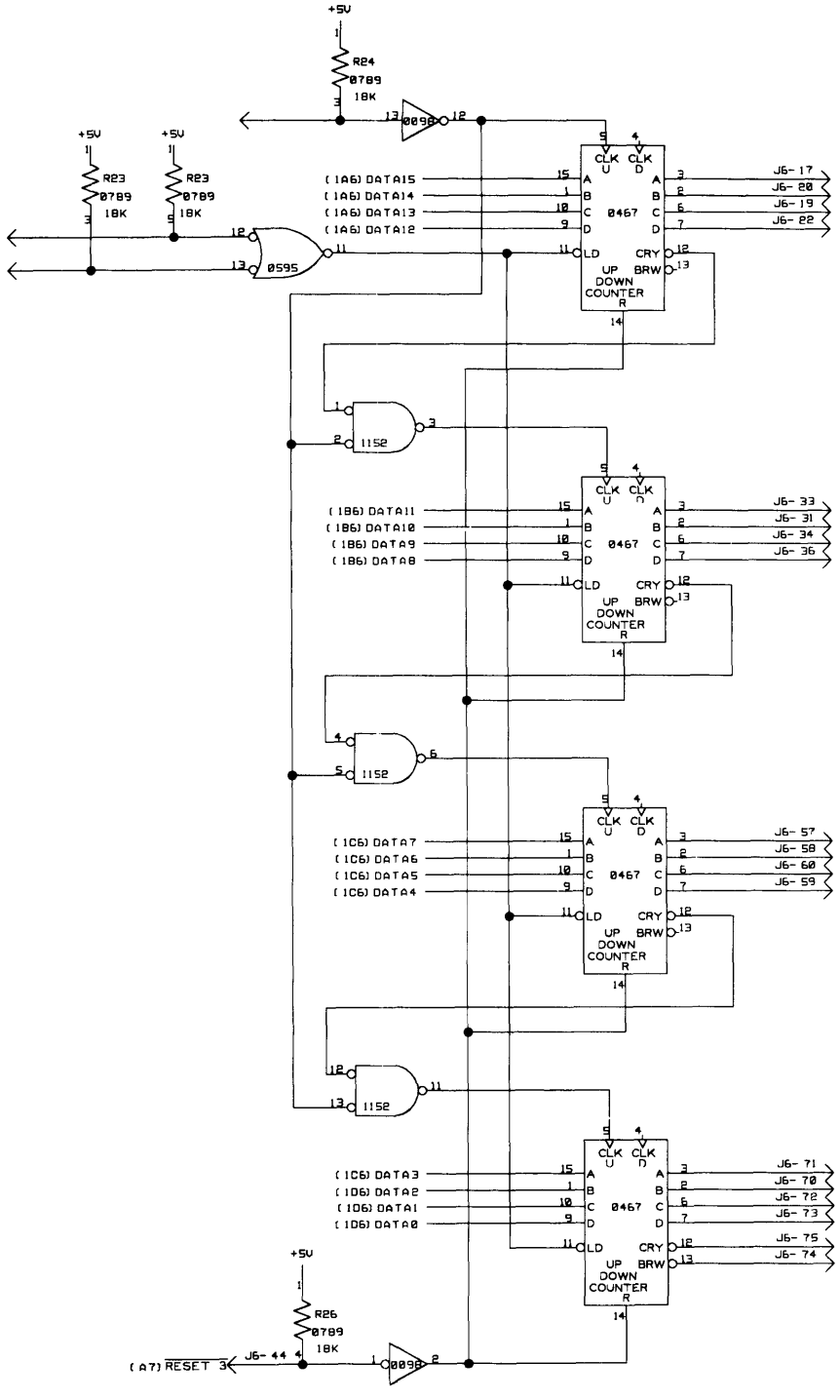

+5V
330Ω
INTPIN
INTPIN A96
0117
A95 INTPOUT
390Ω
390 Ω
R23 0789 18K
0003
R30 0789 18K
INTPIN
INTACK
INTREQ
1248
0630
B75 DATA10
B58 DATA11
B59 DATA12
B64 DATA13
B56 DATA14
B66 DATA15
R20 0789 18K
0090
R29 0063 1K
SW1 0334
DS0 A72
DS1 A68
DS2 A66
DS3 A46
DS4 A62
0900
SDEVCODE
PDEVCODE
DS5 A64
0076
NOTE: PDEVCODE IS EVEN
NOTE: AS SWITCH- OPEN=1



DG-08473

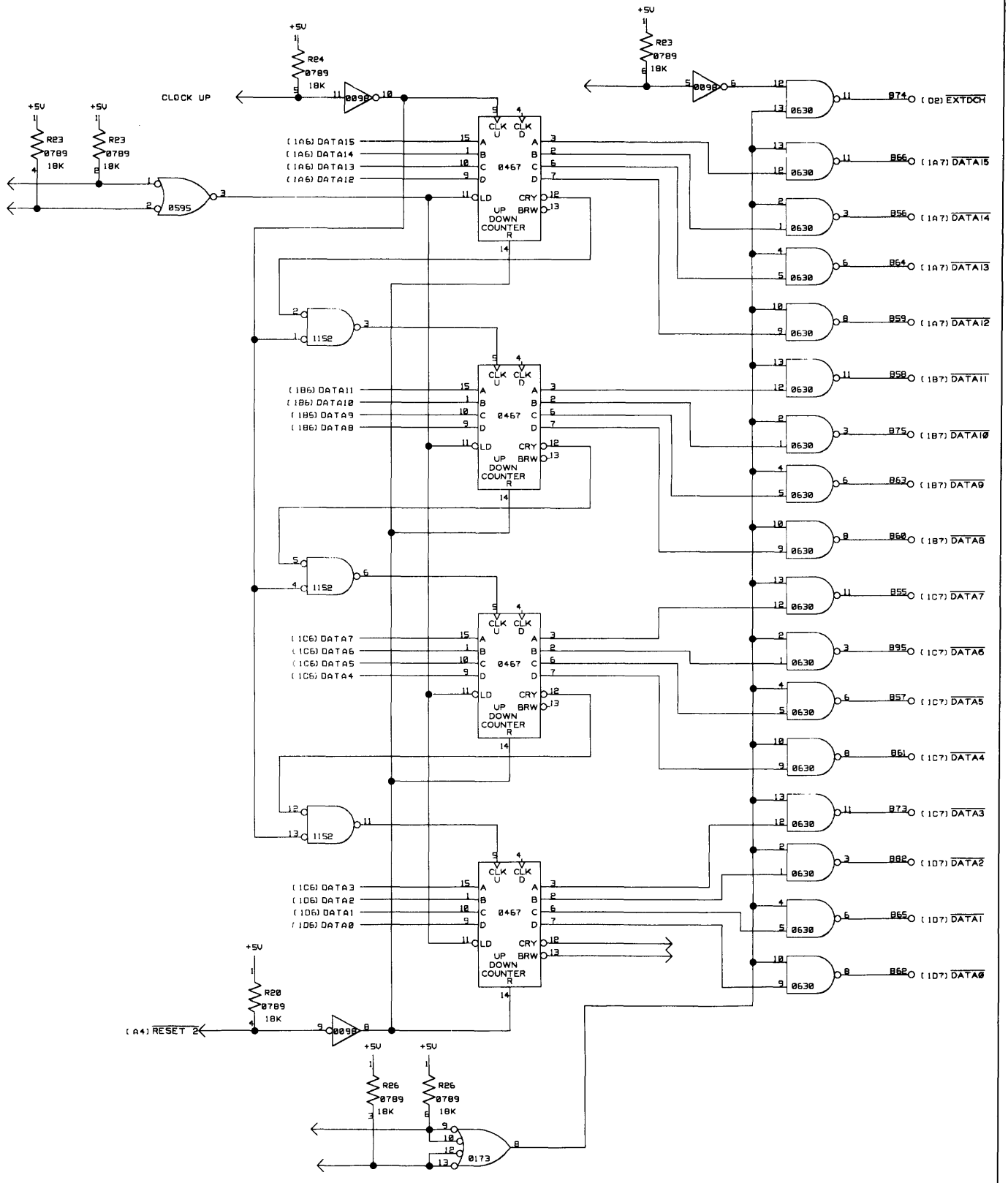
Figure 6.8 Data registers


+5V
+5V
+5V
R28
0789
18K
R28
0789
18K
R28
0789
18K
(D6) SHIFTLEFT
(D6) SHIFTRIGHT
(D3) RESET
SR
SL
DL
(1A6) DATA15
(1A6) DATA14
(1A6) DATA13
(1A6) DATA12
D
C
B
A
0580
DR
4 BIT
SHIFT REG
CLK
R
+5V
+5V
R27
0789
18K
R27
0789
18K
0003
(1B6) DATA11
(1B6) DATA10
(1B6) DATA9
(1B6) DATA8
(1C6) DATA7
(1C6) DATA6
(1C6) DATA5
(1C6) DATA4
(1C6) DATA3
(1D6) DATA2
(1D6) DATA1
(1D6) DATA0



DG-08475

Figure 6.9 Counters and data drivers


+5V
R24
0789
18K
CLOCK UP
0098
R23
0789
18K
0595
1152
0467
UP DOWN COUNTER
CLK U
CLK D
LD
CRY
BRW
(1A6) DATA15
(1A6) DATA14
(1A6) DATA13
(1A6) DATA12
(1B6) DATA11
(1B6) DATA10
(1B5) DATA9
(1B6) DATA8
(1C6) DATA7
(1C6) DATA6
(1C6) DATA5
(1C6) DATA4
(1C6) DATA3
(1D6) DATA2
(1D6) DATA1
(1D6) DATA0
R20
(A4) RESET 2
R26
0173
0630
B74 (0B) EXTDCH
B66 (1A7) DATA15
B56 (1A7) DATA14
B64 (1A7) DATA13
B59 (1A7) DATA12
B58 (1B7) DATA11
B75 (1B7) DATA10
B63 (1B7) DATA9
B60 (1B7) DATA8
B55 (1C7) DATA7
B95 (1C7) DATA6
B57 (1C7) DATA5
B61 (1C7) DATA4
B73 (1C7) DATA3
B82 (1D7) DATA2
B65 (1D7) DATA1
B62 (1D7) DATA0

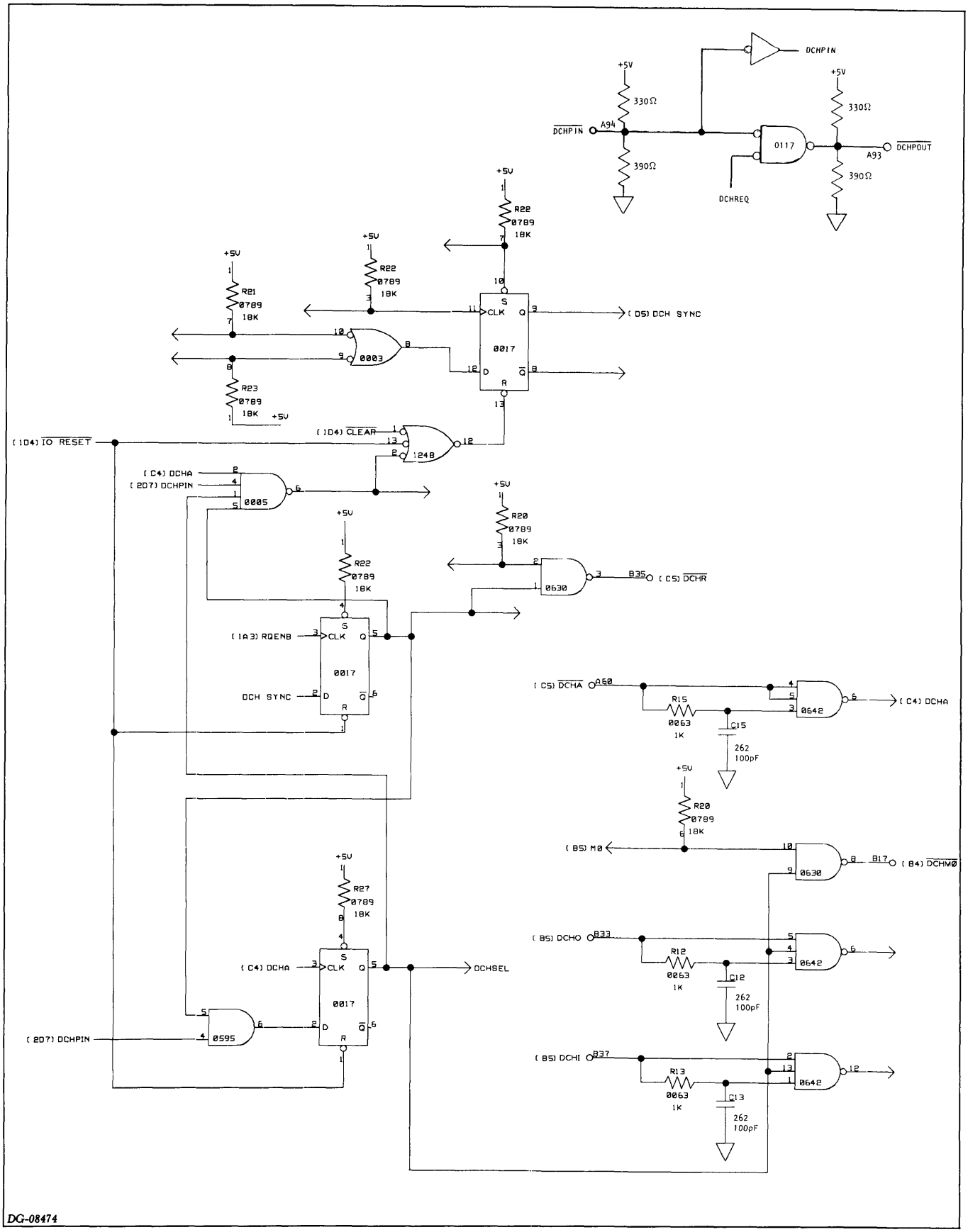


Figure 6.10 Data channel control circuits

Appendix A
STANDARD I/O DEVICE CODES

Table A.1 shows the device codes used in all Data General computers. Additional device codes may be used in specific computers; consult the programmer's reference manual for your computer.

Octal Device Codes	Mnemonic	Priority Mask Bit	Device Name
00	—	—	Unused
01	—	—	Unused
02			
03	MAP	—	Memory allocation and protection
04	HOST	2	unit
			IOP MAP
05	BMC	—	
06	MCAT	12	Multiprocessor adapter transmitter
07	MCAR	12	Multiprocessor adapter receiver
10	TTI	14	TTY input
11	TTO	15	TTY output
12			
13			
14	RTC	13	Real-time clock
15	PLT	12	Incremental plotter
16	CDR	10	Card reader
17	LPT	12	Line printer
20			
21			
22	MTB	10	Magnetic tape
23			
24			
25			
26	DKB	9	Fixed-head DG/Disc
27	DPF	7	DG/Disc storage subsystem
30			
31			
32			
33	DKP	7	Moving head disc
34[1]	DCU[2]	4	Data control unit
35			
36			
37			
40			
41			
42			
43	—	—	Programmable Interval Timer
	PIT	6	
44			
45	SCP	14	Second multiprocessor transmitter
46	MCAT1	12	Second multiprocessor receiver
47	MCAR1	12	
50			
51			
52			
53			
54	RTC1	13	Second real-time clock
55	PLT1	12	Second incremental plotter
56	CDR1	10	Second card reader
57	LPT1	12	Second line printer
60			
61			
62	MTB1	10	Second magnetic tape
63			
64			
65	IOP	5[3]	Host to IOP interface
66	DKB1	9	Second fixed-head DG/Disc
67	DPF1	7	Second DG/Disc storage subsystem
70			
71[1]			
72			
73	DKP1	7	Second moving head disc
74[1]			
75			
76	DPU	4	DCU To Host interface
77	CPU	--	CPU and console functions

Table A.1 Standard I/O device codes

1. Code returned by INTA and used by VCT.
2. Can be set to any unused device code between 1 and 76.
3. Micro interrupts are not maskable.

Appendix B
TIMING

This appendix presents the maximum latency times and maximum data channel transfer rates for Data General computers. Several computers have two different times or rates, depending on how far the controller is from the CPU (high-speed or standard). Some computers also have separate speeds, depending on whether the computer has the hardware multiply and divide option.

Computer Model	High-Speed Data Channel		Standard Data Channel		Interrupt	
	With MUL/DIV	Without MUL/DIV	With MUL/DIV	Without MUL/DIV	With MUL/DIV	Without MUL/DIV
NOVA Computer	N/A	N/A	17.3	17.3	12.0	12.0
SUPERNOVA Computer	5.7	3.7	11.8	7.8	9.0	5.0
SUPERNOVA SC	5.7	3.7	11.8	7.8	9.0	5.0
NOVA 1200 Series	N/A	N/A	9.4	9.4	7.0	7.0
NOVA 800, 820, 840	4.8	3.2	5.8	5.8	10.6	4.6
NOVA 830	5.4	3.6	6.4	6.4	12.0	6.0
NOVA 2, 8K	4.3	4.3	5.2	5.2	5.8	1.9
NOVA 2, 16K	4.4	4.4	5.3	5.3	5.9	2.3
NOVA 3	5.7	5.7	5.7	5.7	11.7	11.3
NOVA 4	7.6	7.6	7.6	7.6		
ECLIPSE (Except S/140 & MV8000)	6.8	6.8	NA	NA	36.2	36.2
S/140	17.6	17.6	17.6	17.6		
MV8000	9.02	9.02	NA	NA		

Table B.1 Maximum* latency time (microseconds)

**The maximum latency times are considered to be for the peripheral with the highest priority.*

Computer Model	High Speed		Standard		Increment Memory		Add to Memory		Pause During Instruction	Processor Pauses For Data Channel
	In	Out	In	Out	High Speed	Standard	High Speed	Standard		
NOVA Computer	—	—	285,500	227,500	—	227,500	—	187,500	No	Yes
SUPERNOVA Computer	1,250,000	1,000,000	434,700	357,100	833,333	357,000	833,333	357,100	No	Yes
SUPERNOVA SC	1,250,000	1,000,000	434,700	357,100	833,333	357,000	833,333	357,000	No	Yes
NOVA 1200 Series	—	—	833,333	555,555	—	416,666	—	—	No	Yes
NOVA 800,820,840	1,250,000	1,000,000	500,000	500,000	833,333	454,545	—	—	Yes	Yes
NOVA 830	1,000,000	833,333	454,545	454,545	833,333	416,000	—	—	Yes	Yes
NOVA 2, 8K	1,250,000	833,333	500,000	475,000	770,000	454,545	—	—	Yes	Yes
NOVA 2, 16K	1,110,000	770,000	475,000	454,545	715,000	435,000	—	—	Yes	Yes
NOVA3	1,400,000	1,000,000	700,000	500,000	—	—	—	—	Yes	Yes
NOVA4	1,000,000	700,000	700,000	550,000	—	—	—	—	Yes	Yes
ECLIPSE (except S/140, MV8000)	1,250,000	714,000	—	—	—	—	—	—	Yes	Yes
S/140	1,000,000	700,000	700,000	550,000	—	—	—	—	Yes	Yes
MV8000	1,250,000	714,000	—	—	—	—	—	—	N/A	No

Table B.2 Maximum data channel transfer rates (words/second)

Appendix C
EXTERNAL I/O BUS CONNECTOR WIRE LIST

Signal	Source	Panel Pin	External Connector Pin @1 ***	External Connector Pin @2 ****	Function
$\overline{\text{DATA0}}$	B	B62	w	3	These sixteen lines carry sixteen bits of data between the processor and interface for all data transfers.
$\overline{\text{DATA1}}$	B	B65	z	4	
$\overline{\text{DATA2}}$	B	B82	AD	5	
$\overline{\text{DATA3}}$	B	B73	AB	6	
$\overline{\text{DATA4}}$	B	B61	v	7	
$\overline{\text{DATA5}}$	B	B57	r	8	
$\overline{\text{DATA6}}$	B	B95	AE	9	
$\overline{\text{DATA7}}$	B	B55	n	10	
$\overline{\text{DATA8}}$	B	B60	u	11	
$\overline{\text{DATA9}}$	B	B63	x	12	
$\overline{\text{DATA10}}$	B	B75	AC	13	
$\overline{\text{DATA11}}$	B	B58	s	14	
$\overline{\text{DATA12}}$	B	B59	t	15	
$\overline{\text{DATA13}}$	B	B64	y	16	
$\overline{\text{DATA14}}$	B	B56	p	17	
$\overline{\text{DATA15}}$	B	B66	AA	18	
$\overline{\text{DS0}}$	P	A72	X	32	These six lines carry the six-bit device code during I/O instruction execution.
$\overline{\text{DS1}}$	P	A68	V	33	
$\overline{\text{DS2}}$	P	A66	U	34	
$\overline{\text{DS3}}$	P	A46	H	35	
$\overline{\text{DS4}}$	P	A62	S	36	
$\overline{\text{DS5}}$	P	A64	T	37	
DATIA	P**	A44	F	19	Each signals the interface that it should place data on the DATA lines.
DATIB	P**	A42	E	20	
DATIC	P**	A54	M	21	
DATOA	P**	A58	P	22	Each signals the interface that data is on the DATA lines.
DATOB	P**	A56	N	23	
DATOC	P**	A48	J	24	
$\overline{\text{SELB}}$	D	A82	a	46	Carry the states of the Busy and Done flags of the selected interface.
$\overline{\text{SELD}}$	D	A80	Z	47	

Signal	Source	Panel Pin	External Connector Pin @1 ***	External Connector Pin @2 ****	Function
CLR	P**	A50	K	2	Interface status control signals.
IOPLS	P**	A74	Y	41	
STRT	P**	A52	L	48	
$\overline{\text{RQENB}}$	P**	B41	m	45	Synchronizes $\overline{\text{INTR}}$ and $\overline{\text{DCHR}}$.
$\overline{\text{MSKO}}$	P**	A38	C	43	Signals interface to load Interrupt Disable flag.
$\overline{\text{INTR}}$	D	B29	f	40	Interrupt Request.
$\overline{\text{INTP IN}}$	*	A96			Interrupt Priority
$\overline{\text{INTP OUT}}$	*	A95	c	39	Chain.
INTA	P**	A40	D	38	Interrupt Acknowledge.
$\overline{\text{DCHR}}$	D	B35	j	31	Data Channel Request.
$\overline{\text{DCHP IN}}$	*	A94			Data Channel
$\overline{\text{DCHP OUT}}$	*	A93	b	30	Priority Chain
$\overline{\text{DCHA}}$	P**	A60	R	25	Data Channel Acknowledge.
$\overline{\text{DCHM0}}$	D	B17	d	27	Data Channel Transfer Mode.
DCHI	P**	B37	k	26	Signals interface that data is on DATA lines.
DCHO	P**	B33	h	29	Signals interface to place data on DATA lines.
IORST	P	A70	W	42	System reset.
EXT MAP SEL	D	B74			Extended Map Select.

**For the two pairs of priority-determining signals, the IN signal comes from the processor or the preceding device, the OUT signal goes to the next device. If the computer is operated with an interface board removed (or a slot is not used), jumper pin A93 to A94 and A95 to A96 to maintain bus continuity.*

***Use filters described in text.*

****@1 Paddleboard connector.*

*****@2 Socket connector.*

Appendix D
I/O BUS FACILITIES ON EARLIER MACHINES

Several additional facilities of the I/O bus are no longer supported by Data General. The facilities were designed into previous Data General computers; no future Data General computers will have them. These facilities are: two additional data channel modes - memory increment, and add to memory; and a power on signal. The following Data General computers contain some or all of these facilities: NOVA, SUPERNOVA, NOVA 800 series, NOVA 1200 series, and NOVA 2 series. The information about these facilities is included in this appendix.

The following discussion assumes you have read and understood all the material on the I/O bus in the remainder of this manual.

Power On Signal

The power on signal, **PWR ON**, is a +5 volt signal that is asserted whenever the power supply of the computer is powered-up. It is to be used to drive relays in peripheral equipment for power turn-on. It cannot be used to power any peripheral equipment directly.

Data Channel Facilities

The data channel of these machines is capable of two additional types of transfer (in addition to the input and output modes). These transfers are: increment, and add to memory. The two modes are selected by an additional data channel mode select line, **DCHM1**, which is on pin B21 of the backpanel of these machines. Note that pin B21 of other Data General computers may be used for other purposes.

The mode select lines, **DCHM0** and **DCHM1**, select the data channel mode as shown in Table D.1.

DCHM0	DCHM1	Function
0=H	0=H	Output
0=H	1=L	Increment memory
1=L	0=H	Input
1=L	1=L	Add to memory

Table D.1 Data channel modes

Increment Memory mode

The increment mode looks to the controller much like the output mode; however, the processor handles the data differently. Like the output mode, the referenced memory location is read, but before the data is placed on the bus, it is incremented by one. This new value is placed on the I/O bus and written into the memory location as shown in Figure D.1. The **DCH0** signal signals the controller to load the data, as usual. Also, if the increment caused the sum to exceed 2^{16}-1; that is, if the 16-bit sum is all zeros, the **OVFLO** signal on the bus will be pulsed. The **OVFLO** signal is on pin B39 of the backpanel of these machines. Note that pin B39 on other Data General computers may be used for other purposes.

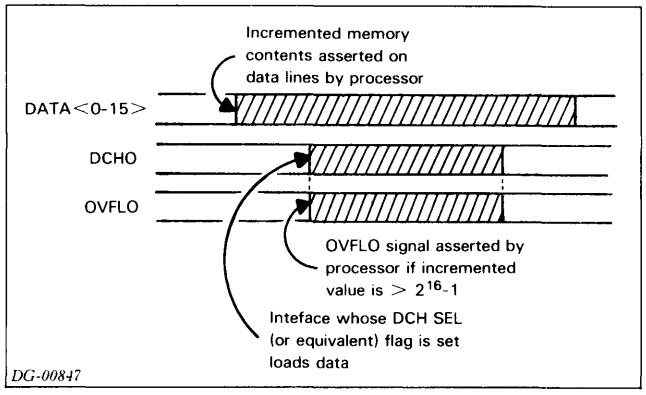


Figure D.1 Data channel increment sequence

Add to Memory mode

The add to memory mode involves both an input and output transfer. The **DCHI** signal transfers a data word from the interface to the processor, where this word is added to the one obtained from the referenced memory location, as shown in Figure D.2. The sum of these two words is written back into the memory location and transferred to the interface with a **DCH0**. In addition, if this sum is greater than 2^{16}-1, the **OVFLO** signal on the bus will be pulsed.

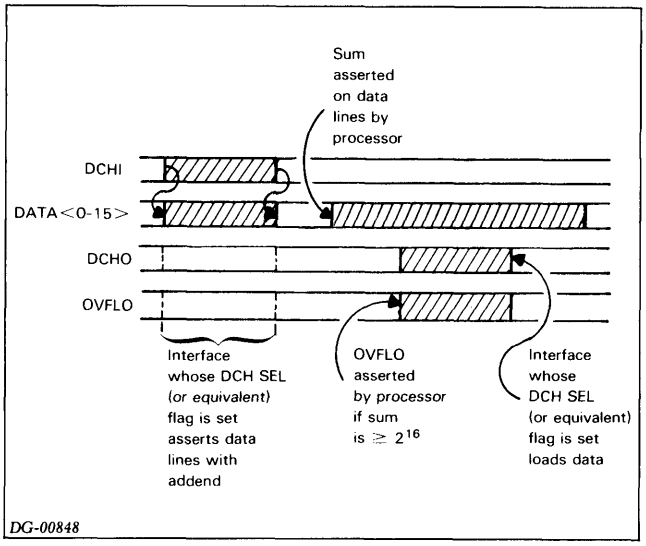


Figure D.2 Data channel add to memory sequence

The following diagrams, Figure D.3 through Figure D.6, show both standard and high speed versions of the two additional data channel modes. High speed implementations are used in the main chassis of the computer; standard speed is used in the expansion chassis. Controllers designed for high speed will work in all cases.

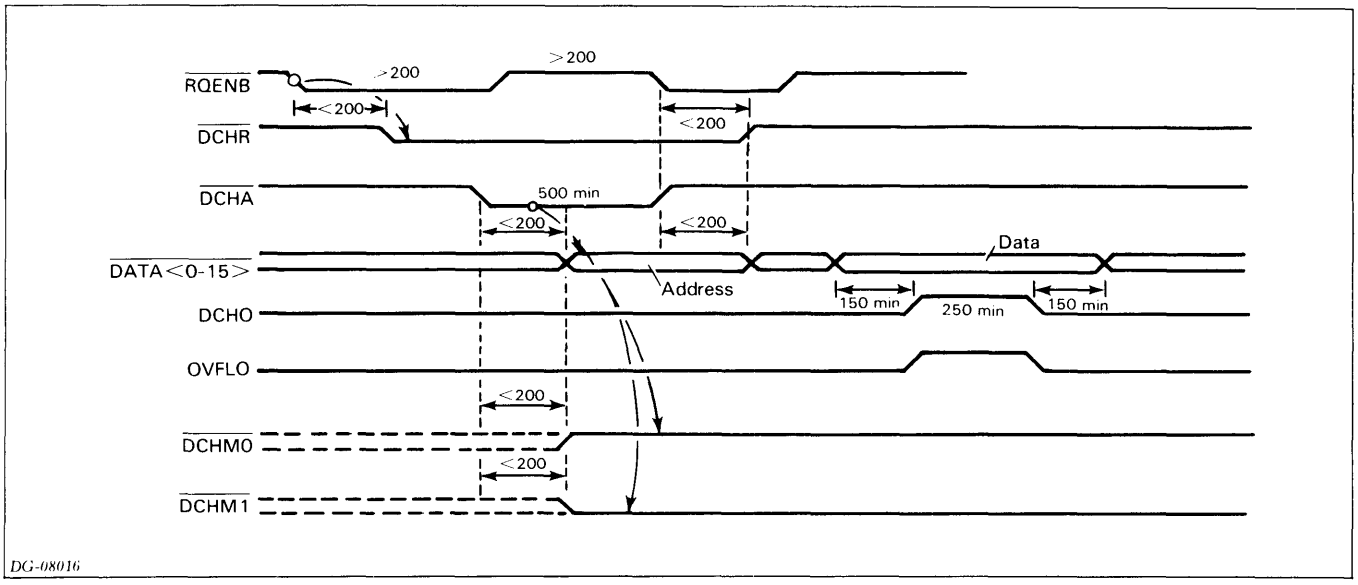


Figure D.3 Standard data channel--increment

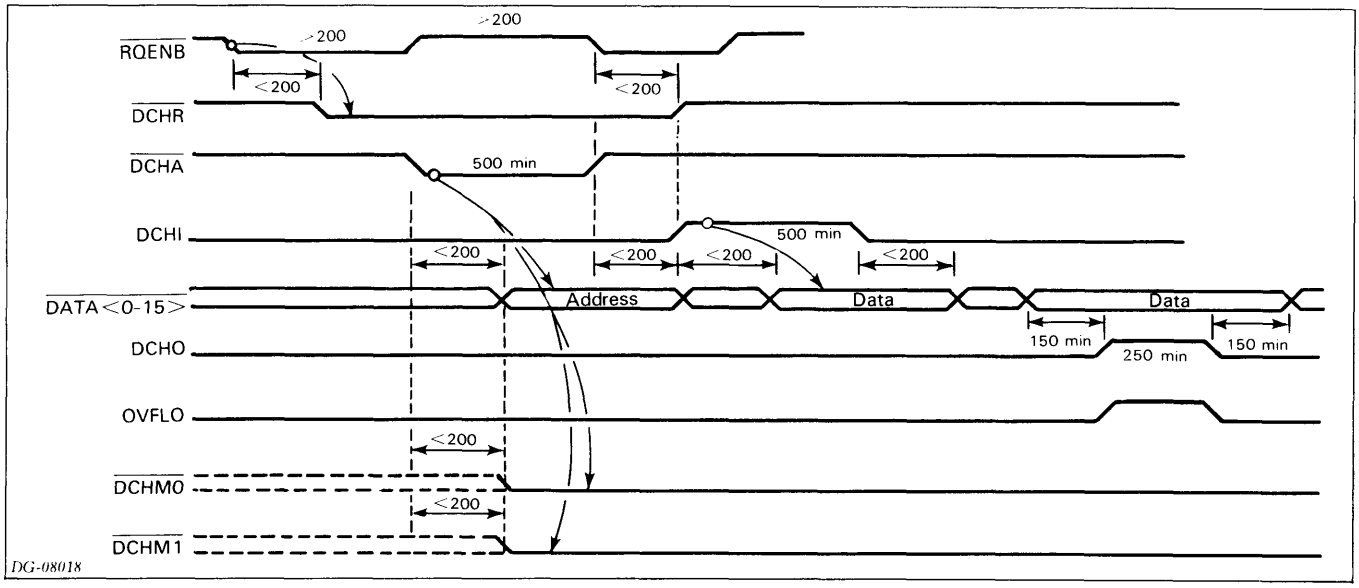


Figure D.4 Standard data channel--add to memory

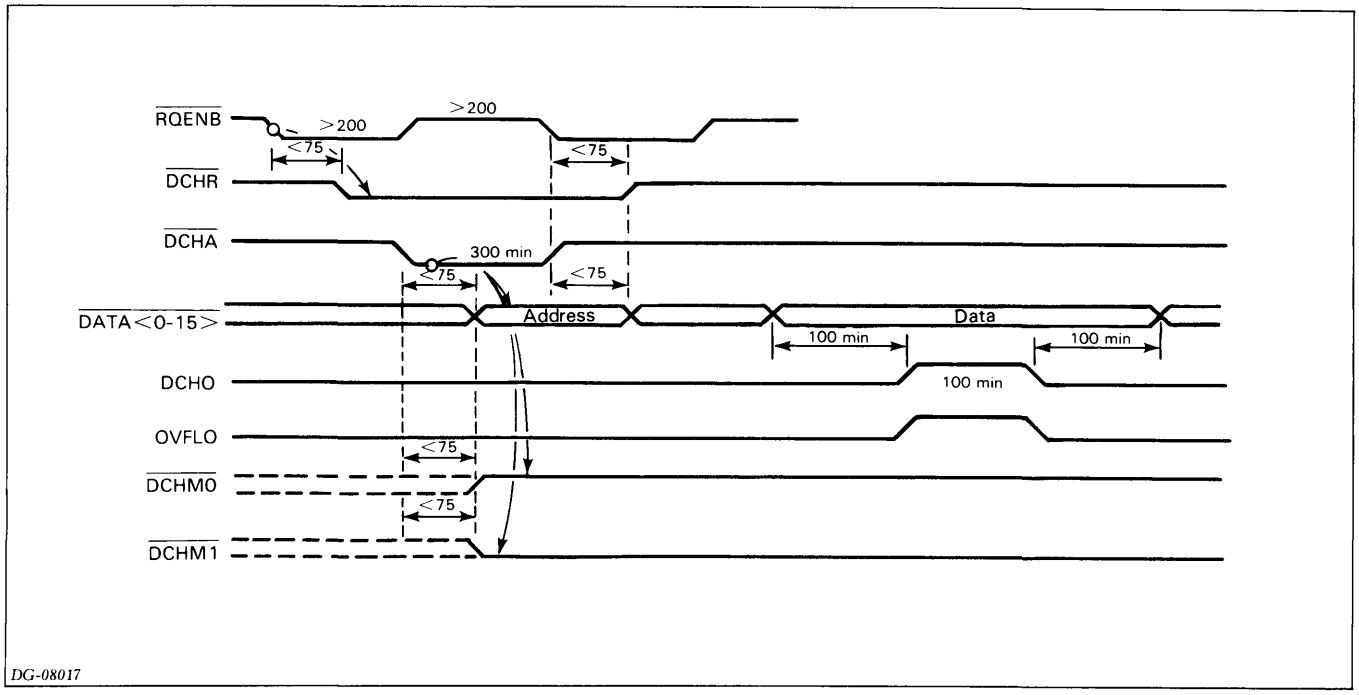


Figure D.5 High-speed data channel--increment

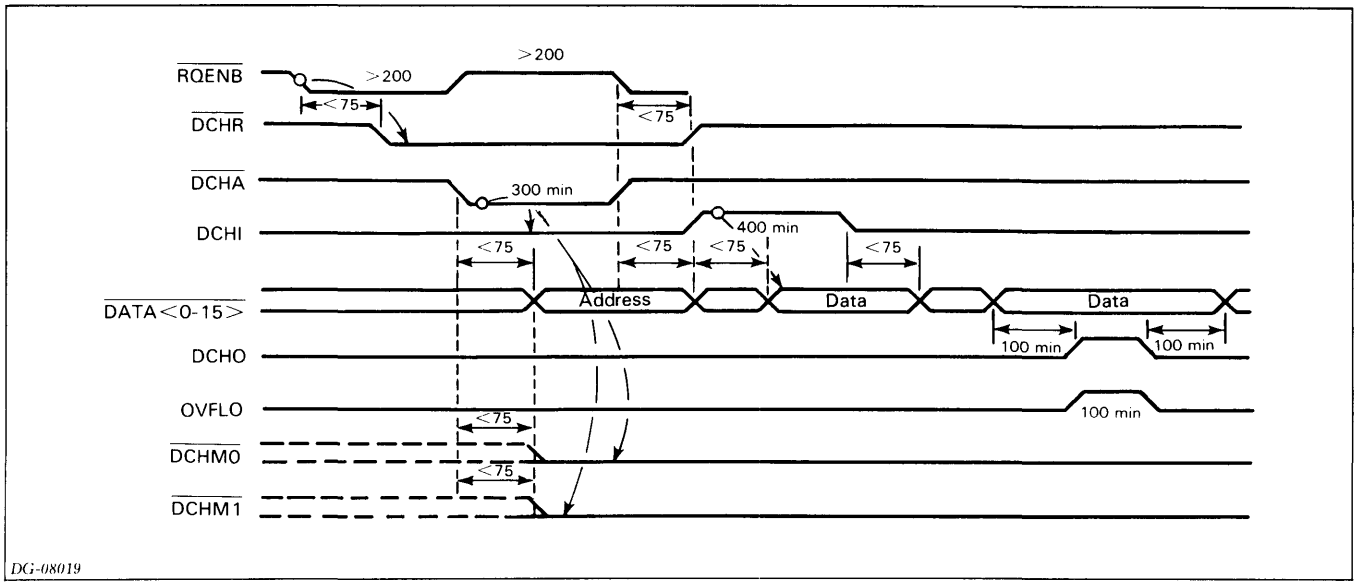


Figure D.6 High-speed data channel--add to memory

Appendix E
BACKPANEL CONNECTOR LAYOUT

Figure E.1 shows the location of the various I/O bus signals and the power and ground lines on the back panel. Two types of slots exist; an I/O only slot where only I/O controllers may reside and a Memory or I/O slot where memory modules or I/O controllers may reside. (Consult the documentation on your machine for which slots in your machine are of which type.) Either of these two types of slots are available on all slots not reserved for processor boards.

The locations labeled C are available to the controller for signals to external devices, via an internal cable. The locations labeled M are used for memory signals in those slots designed for memory modules. In the slots designed for I/O interfaces only, those locations are also available to the interface. Any interface using locations labeled M, however, is incompatible with the standard back panel configuration and cannot be plugged into one of the slots in which those locations are not free. The engineering drawing of the back panel for a particular channel should always be consulted to verify pin assignments.

Notes to Backpanel Connector Layout

The following notes refer to the numbers appearing in certain pin locations in Figure E.1.

1) Reserved for future use.
2) Used on some machines —see the specific schematic.
3) **$\overline{\text{DCHM1}}$** on some machines; **PWRFAIL** on some machines — see the schematic.
4) **OVFLO** on some machines; **PWROK** on some machines — see the schematic.
5) Memory pins on original NOVA computer; otherwise ground.
6) Memory pins on some machines; +12V on some machines — see schematic.
7) Ground on some machines — see schematic.
8) S bus: first used on ECLIPSE MV/8000; pin 7 — **S BUS CLK**; pin 8 — **S BUS DATA**; pin 9 — **S BUS HARD**; pin 10 — **S BUS SOFT**.
9) Used on some machines — see the specific schematic.
10) Communications priority signals on some machines — see the schematic.
11) Memory pins on some machines; **EXT MAP SEL** (or **EXDCH**) on some machines — see the schematic.
12) The signals **$\overline{\text{DCHP IN}}$**, **$\overline{\text{DCHP OUT}}$**, **$\overline{\text{INTP IN}}$**, and **$\overline{\text{INTP OUT}}$** are connected so that pin A93 connects to pin A94 on the slot above it or to the left of it and pin A95 connects to pin A96 on the slot above it or to the left of it.
13) **VINH** — +15V on some machines; otherwise labeled C.
14) **SYSRST** on ECLIPSE line computers.
15) Terminator power pins on some machines — see the schematic.
16) Ground on some machines — see the schematic.

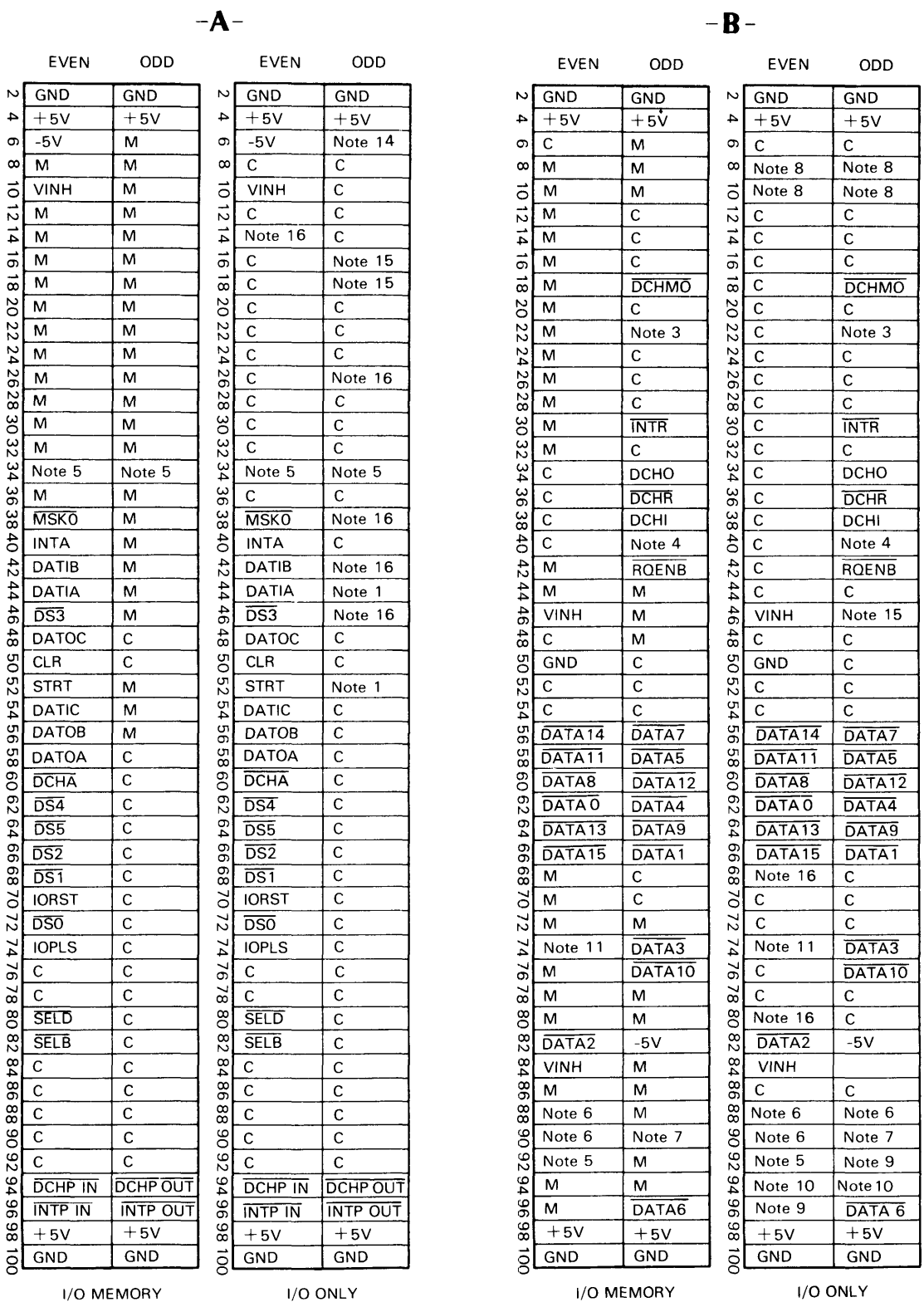


Figure E.1 Backpanel connector layout

Notes to Backpanel Connector Layout

Appendix F
I/O BUS SUMMARY

Signal Mnem.	Signal Name	Descript.	B/P pin	Src.	Dest.	Responds to	Initiates	Pulse Duration	Response Times	Distrib. Type	Function	Comments
$\overline{DATA}$ <0-15>	Data	16-bit data, or address	B62 B65 B82 B73 B61 B57 B95 B55 B60 B63 B75 B58 B59 B64 B56 B66	CPU or CNTL.	CNTL. or CPU	program instruct.; DCHA; DCHI; INTA.				Parallel	All data and addresses, for both data channel and programmed I/O, are transferred between the processor and the controller on these 16 bi-directional lines. The interrupt disable mask and interrupt acknowledge information are also carried on these lines.	
$\overline{DS}$ <0-5>	Device Select	6-bit device code	A72 A68 A66 A46 A62 A64	CPU	CNTL.	program instruct.	SELB/SELD			Parallel	These lines carry the low-order six bits of each I/O instruction; that is, the device code.	These lines may contain random information during non-I/O instructions.
DATIA	Data In A	DIA data enable	A44	CPU	CNTL.	program instruct.		>500ns		Parallel	Asserted by the processor during the execution of a DIA instruction.	The controller should place the contents of its register A on the Data lines during this signal.
DATIB	Data In B	DIB data enable	A42	CPU	CNTL.	program instruct.		>500ns		Parallel	Asserted by the processor during the execution of a DIB instruction.	The controller should place the contents of its register B on the Data lines during this signal.
DATIC	Data In C	DIC data enable	A54	CPU	CNTL.	program instruct.		>500ns		Parallel	Asserted by the processor during the execution of a DIC instruction.	The controller should place the contents of its register C on the Data lines during this signal.
DATOA	Data Out A	DOA data strobe	A58	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during the execution of a DOA instruction.	The controller should store the contents of the Data lines in its register A during this signal.
DATOB	Data Out B	DOB data strobe	A56	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during the execution of a DOB instruction.	The controller should store the contents of the Data lines in its register B during this signal.
DATOC	Data Out C	DOC data strobe	A48	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during the execution of a DOC instruction.	The controller should store the contents of the Data lines in its register C during this signal.
STRT	Start	S flag command	A52	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during any I/O instruction which specifies S.	This signal is typically used to start a controller operation.
CLR	Clear	C flag command	A50	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during any I/O instruction which specifies C.	This signal is typically used to clear the controller of any interrupt requests or other conditions.
IOPLS	I/O Pulse	P flag command	A74	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted by the processor during any I/O instruction which specifies P.	This signal can be used by the controller for anything.
$\overline{SELB}$	Selected Busy	Busy test	A82	CNTL.	CPU	DS <0-5>			0-150ns	Parallel	Asserted low by the controller selected by the device code whose Busy flag is 1.	This signal is tested by the CPU to check if the controller is busy.
$\overline{SELD}$	Selected Done	Done test	A80	CNTL.	CPU	DS <0-5>			0-150ns	Parallel	Asserted low by the controller selected by the device code whose Done flag is 1.	This signal is tested by the CPU to check if the controller is done.
$\overline{INTR}$	Interrupt Request	Service request	B29	CNTL.	CPU	INTP & INT REQ	INTA		0-100ns	Parallel	Asserted low by any controller to request program interrupt service.	The controller should assert this signal (if not masked out) whenever it completes an ioperation or experiences an error.
$\overline{MSKO}$	Mask Out	Interrupt mask strobe	A38	CPU	CNTL.	program instruct.		>350ns		Parallel	Asserted low by the processor during the execution of the MSKO instruction.	The controller should store its assigned Data line(s) (mask bit(s)) in its interrupt disable flag.
INTP Note 1	Interrupt Priority	Serial priority chain	A96 A95	PREV. BOARD	NEXT BOARD	INTP of prev. board	INTR		0-30ns (300ns total)	Serial	Seen asserted by the first controller on the I/O bus using the program interrupt facility, and transmitted serially through each controller in order.	Each controller must use this line to determine if it is making the highest priority interrupt request. If it is, it outputs to all lower priority controllers an inactive INTP which inhibits them from responding to interrupt acknowledge (INTA).
INTA	Interrupt Acknow.	Service granted	A40	CPU	CNTL.	INTR		>500ns		Parallel	Asserted by the processor during the execution of the INTA instruction.	The controller with the highest priority request must send to the CPU via the data lines, its device code during this signal.
$\overline{DCHR}$	Data Channel Request	Transfer request	B35	CNTL.	CPU	DCHP & DCH REQ & RQENB	DCHA		0-75ns	Parallel	Asserted low by any controller which requires data channel service.	The controller should assert this signal whenever it is ready to transfer information to or from the CPU via the data channel.
DCHP Note 2	Data Channel Priority	Serial priority chain	A94 A93	PREV. BOARD	NEXT BOARD	DCHP of prev. board	DCHR		0-30ns	Serial	Seen asserted by the first data channel controller on the I/O bus, and transmitted serially through each controller in order.	Each controller must use this line to determine if it is making the highest priority data ichannel request. If it is, it outputs to all lower priority controllers an inactive DCHP which inhibits them from responding to data channel acknowledge (DCHA).
$\overline{DCHA}$	Data Channel Acknow.	Transfer granted	A60	CPU	CNTL.	DCHR	DCHI or DCHO	>300ns		Parallel	Asserted low by the processor at the beginning of each data channel cycle.	The controller with the highest priority request must send to the CPU via the data lines and DCHM0, the address and mode of the transfer it requires.
$\overline{DCHM0}$	Data Channel Mode	Select transfer type	B17	CNTL.	CPU	DCHA			0-75ns	Parallel	Asserted low by the controller during data channel acknowledge to select mode.	Modes are as follows (DCHM0 - function): 0 - Output 1 - Input
DCHI	Data Channel Input	Input transfer strobe	B37	CPU	CNTL.	DCHM0	Data	>350ns		Parallel	Asserted by the processor for data channel input.	During this signal, the controller must send the data to the CPU via the data lines.
DCHO	Data Channel Output	Output transfer strobe	B33	CPU	CNTL.	DCHM0		>100ns		Parallel	Asserted by the processor for data channel output.	During this signal, the controller must store the data on the data lines that it requested.
IORST	I/O Reset	Reset all I/O	A70	CPU	CNTL.	program instruct.		>500ns		Parallel	Asserted by the processor during a IORST instruction or when RESET occurs.	All controllers should reset all operations and logic with this signal.
$\overline{RQENB}$	Request Enable	Syncs. requests	B41	CPU	CNTL.		INTR or DCHR	5(>75ns <5MHz		Parallel	Asserted low by the processor to enable program interrupt & data channel requests	This signal synchronizes both INTR and DCHR.

Note 1: Pin A96 brings $\overline{INTP\ IN}$ from the previous board; pin A95 takes $\overline{INTP\ OUT}$ to the next board.

Note 2: Pin A94 brings $\overline{DCHP\ IN}$ from the previous board; pin A93 takes $\overline{DCHP\ OUT}$ to the next board.

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

DG-04976

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Australia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogato
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

Engineering Publications Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: _____

Document No. 014-000629-01

Yes	No			
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.	○ Other:
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ As a reference ○ As an introduction to the product	○ To instruct a class. ○ Other:
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ Other:	
☐	☐	Does the manual tell you all you need to know? What additional information would you like?		
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)		

Name: _____ Title: _____

Company: _____ Division: _____

Address: _____ City: _____

State: _____ Zip: _____ Telephone: _____ Date: _____

DG-06895

Data General

Data General Corporation, Westboro, Massachusetts 01580

CUT ALONG DOTTED LINE

FOLD FOLD

TAPE TAPE

FOLD FOLD

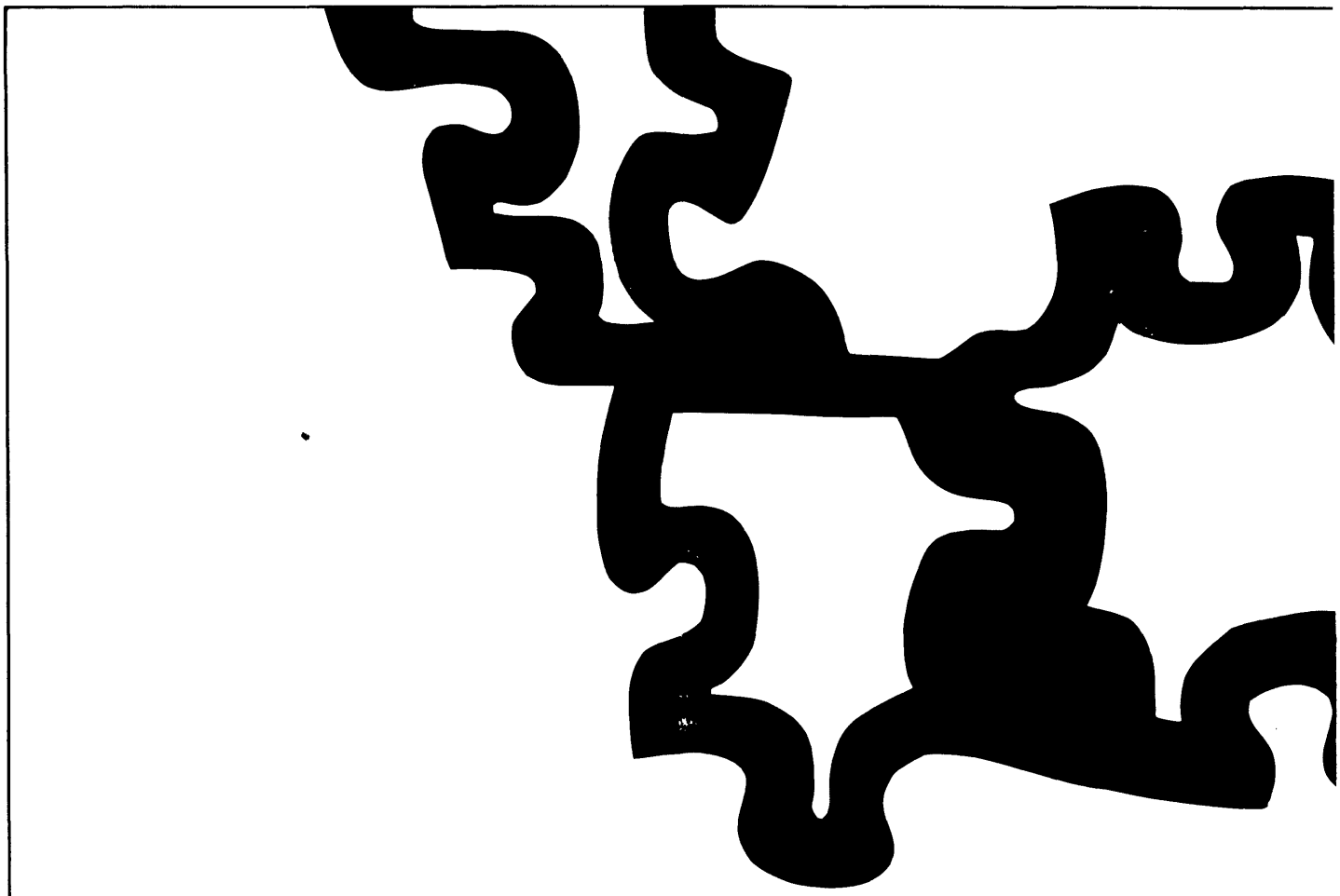
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Engineering Publications (C-138)
4400 Computer Drive
Westboro, MA 01581



Data General

Data General Corporation, Westboro, Massachusetts 01580