User's Manual

Interface Designer's Reference

NOVA® AND ECLIPSE® LINE COMPUTERS

015-000031-05

NOTICE

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

Users are cautioned that DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including, but not limited to typographical, arithmetic, or listing errors.

DATA GENERAL USER'S MANUAL

INTERFACE DESIGNER'S REFERENCE

NOVA AND ECLIPSE LINE COMPUTERS

Ordering No. 015-000031

Printed in the United States of America
Rev. 05, May 1978

TABLE OF CONTENTS

SECTION I

INTRODUCTION

SECTION II

INPUT/OUTPUT PROGRAMMING

TABLE OF CONTENTS (Continued)

SECTION II (Continued)

INPUT/OUTPUT PROGRAMMING

SECTION III

I/O BUS

TABLE OF CONTENTS (Continued)

SECTION III (Continued)

I/O BUS

SECTION IV

I/O BUS SPECIFICATIONS

TABLE OF CONTENTS (Continued)

SECTION V

CONNECTIONS, CONNECTORS AND TERMINATORS

SECTION VI

INTERFACE BOARDS

TABLE OF CONTENTS (Continued)

APPENDICES

This page intentionally left blank

SECTION I
INTRODUCTION

INTERFACING

One of the most valuable aspects of the modern digital computer is the variety of custom 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™ lines of computers are similar and boards are physically interchangeable, so that it is possible to build an interface to operate with both computer lines.

It is particularly easy to interface to Data General's computers because of a number of features. The interfaces can be built on large 15-inch square printed circuit boards, which allow 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 a custom interface assembly which can be used on NOVA and ECLIPSE lines.

While only a minimum of knowledge of the input/output architecture of the computer is needed to use a peripheral sold by Data General Corporation, the design and use of custom-built I/O equipment requires a much greater understanding of this architecture. Thus, this manual discusses the I/O bus structure, 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 sections as follows:

Section I	introduces this manual, defines terms and cites related documentation.
Section II	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.
Section III	looks at the input/output facilities from the viewpoint of the I/O bus. It is a discussion of the various I/O functions and the bus signals that are used for each.
Section IV	is a discussion of the important electrical characteristics of the I/O bus.
Section V	discusses the problems involved in packaging an interface and connecting it with the remainder of the computer system.
Section VI	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.

Although this manual could be understood by someone having little or no previous contact with Data General computers, the reader will find it helpful to consult some of the other publications listed below. These manuals will provide the designer with a discussion of the instruction sets, more extensive input/output programming procedures, machine operation, and other helpful information.

1. Programmer's Reference Manual - Peripherals - DG 015-000021.
2. Programmer's Reference Manual - NOVA line computers - DG 015-000023.
3. Programmer's Reference Manual - ECLIPSE line computers - DG 015-000024.
4. Installer's Reference Manual - ECLIPSE and NOVA line computers-DG 015-000041.

OVERVIEW OF THE COMPUTER LINES

Data General manufactures two lines of computers: 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 communications environments and, with the writeable control store feature, can include custom instructions. A full description of the instruction sets can be found in the references above.

The original computers manufactured by Data General are the NOVA and SUPERNOVA® computers. The NOVA computer has a central processor that is built on two printed circuit boards whereas the SUPERNOVA computer has a 3-board central processor.

The subsequent NOVA computers come in four series which differ from each other in the number of printed circuit boards on which the processor is built and on the cycle time of the memory with which they operate. The NOVA 1200 series has central processors built on one printed circuit board and designed to operate with memories having a cycle time of 1200 nanoseconds. The NOVA 800 series has central processors built on two printed circuit boards and is designed to operate with memories having a cycle time of 800 nanoseconds (except for the NOVA 830 which operates with memories having a cycle time of 1000 nanoseconds). The NOVA 2 computers use either 1000-nanosecond or 1200-nanosecond memories. The NOVA 3 computers use 700ns, 800ns or 1000ns memories.

The ECLIPSE computers have a central processor built on two boards. They include an asynchronous memory interface which allows them to use a variety of memories.

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. For example, a moving-arm disc controller can translate the track address received from the computer into positional commands for the disc drives 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 to complete the communications link between the device and the controller.

The communications channel through which all 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 Transfer

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. 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 an accumulator and are transferred to the controller under direct program control. The block of data is then automatically transferred directly between memory and the controller via the data channel.

Once the data channel transfer for a block of data has been set up and initiated by the program, no further action by the program is required to complete the transfer. The program 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 direct access to memory. When access is granted, the word is transferred. Because several 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.

Since the actual transfer of a word via the data channel could conflict with the program instructions being executed, the program pauses during the transfer of each word. This pause is transparent to the programmer with the exception that the time required for program execution is lengthened.

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 was dependent 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 perform decisions based on the results of the checks. However, these status checks are time-consuming, so, to avoid the necessity of continually performing such tests, all DGC computers incorporate a program interrupt facility.

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 of the processor. The processor honors such a request by interrupting the program in process, saving the address where the interruption occurred, and transferring 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 that 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 multi-level priority structure up to 16 levels deep 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 in as efficient a manner as possible. The priority interrupt structure, like the rest of the program interrupt facility, is under direct control of the program.

This page intentionally left blank

SECTION II

INPUT/OUTPUT PROGRAMMING

I/O INSTRUCTION SET

Information transfers between the computer and the various peripherals are governed by the program with eight instructions which constitute the I/O instruction set. These instructions allow the program to communicate with the peripheral's controllers and to control the program interrupt facility. This section covers only those I/O instructions used for these purposes; additional I/O instructions used for special processor functions and options are fully described in the Programmer's Reference Manuals for the ECLIPSE and NOVA line Computers. The section 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.

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. The additional structure required for a peripheral using the program interrupt facility or the data channel is discussed in the sections describing those facilities. What follows is meant only to typify the workings of a controller; controllers are tailored to the specific devices they control, so that not all 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 peripherals controllers contain a data buffer. In the case of peripherals which 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 which transfers data under data channel control, the data is transferred directly between the register in the controller and memory. Data buffers in the controllers which 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, and are therefore of importance to the program which must check on the progress of the peripheral's operation. For example, a program transferring consecutive sectors of information on a disc in a single operation can read the current sector address and sector count during the operation in order to determine how far the operation is from completion. Status flags are set by the controller to indicate error conditions or to notify 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. The most common case of this occurrence is 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 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 state the interrupt facility is in, no interrupt can occur for that peripheral until its Done flag is set to 1. Therefore, the setting to 1 of the Done flag is defined to "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 and 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.

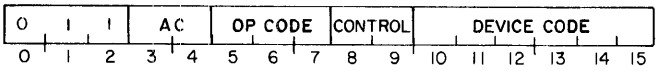
Status flags which indicate errors or malfunctions in the operation of a peripheral are termed "error flags". Two types of error flags can be characterized, according to their effect on the operation of the peripheral when they are set. The first, or passive, type is merely set by the controller in the 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 was 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 section 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 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 device codes, numbered 0-77$_8$. 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 reading the console switches and 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 as follows:

F field	Command	Mnemonic	Control Function
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.

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 as follows:

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)

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 above. 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 as follows:

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.

Accumulator Field

Bits 3 and 4 in an I/O instruction select one of the central processor's four accumulators: AC0, AC1, AC2, or 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, the accumulator field is ignored. The assembler sets bits 3 and 4 in these instructions to 0; however, any bit combination will do, and no accumulator will ever be affected by these two instructions.

Instructions

A number of abbreviations and symbols are used in this manual to aid in defining 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 symbols are not coded, rather they perform these functions:

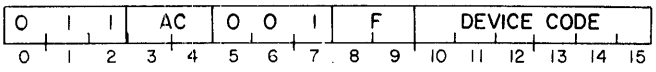
< > Indicates an optional operand. The operand enclosed in the brackets (e.g., <f>) may be coded or not, depending on whether the associated option is desired.

= Indicates a specific substitution is required. Substitute the desired number, letter or letters, or symbol from the class, as defined by the abbreviation for which the substitution is being made. For example, "ac" indicates that an accumulator specifier is required. To select AC2, code either a "2" or a symbol whose value is 2.

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. Bits in the word which are not used by the controller are shaded. Shaded bits are ignored on output and set to 0 on input.

DATA IN A

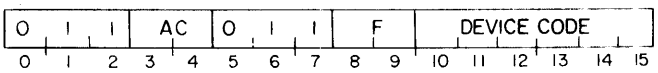
DIA<f> ac,device



The contents of the A input buffer in the specified controller are placed in the specified AC. 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

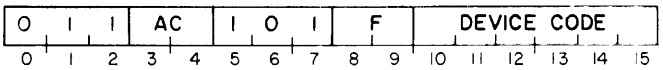
DIB<f> ac,device



The contents of the B input buffer in the specified controller are placed in the specified AC. 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

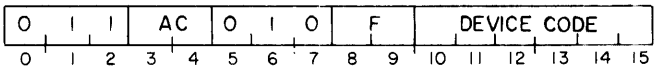
DIC<f> ac,device



The contents of the C input buffer in the specified controller are placed in the specified AC. 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

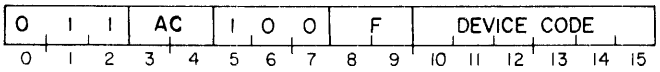
DOA<f> ac,device



The contents of the specified AC are placed into the A output buffer in the specified 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

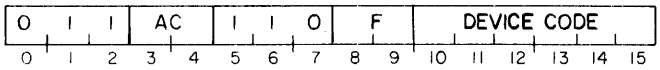
DOB<f> ac, device



The contents of the specified AC are placed into the B output buffer in the specified 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

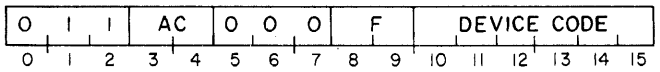
DOC<f> ac, device



The contents of the specified AC are placed into the C output buffer in the specified 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

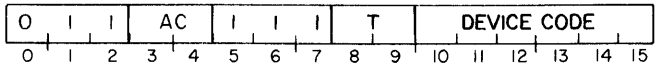
NIO<f> device



The Busy and Done flags in the controller of the specified device are set according to the function specified by F. When the assembler encounters the mnemonic NIO, it sets the AC field bits to 0. However, these bits are ignored and may have any value. The contents of all the accumulators are unchanged.

I/O SKIP

SKP t device



Skip the next sequential instruction if the test condition specified by T is true for the specified controller. When the assembler encounters the mnemonic SKP t, it sets the AC field bits to 0. However, these bits are ignored and may have any value. 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 peripheral's Done flag is set to 1. The processor can respond to, or "honor", an interrupt request by interrupting 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 section detail the instructions used to control the program interrupt facility, describe the implementation of a priority interrupt scheme, offer further suggestions for programming an interrupt handler, and explain the operation of the vector instruction, which allows the ECLIPSE computer to automatically perform much of its interrupt processing.

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 which uses 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, allowing the processor to respond to interrupt requests transmitted to it along the Interrupt Request Line. Setting ION to 0 disables the entire interrupt facility, causing 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 the RESET console switch is pressed.)

The controller for each peripheral which uses the program interrupt facility contains an Interrupt Disable flag which allows the program to disable interrupts from that peripheral. When a peripheral's Interrupt Disable flag is set to 1, the peripheral is prevented from making 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. Also, at power-up and when the RESET console switch is pressed, 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 of 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 stores the contents of the program counter (which point 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.
3. 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) should 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 and saving the carry bit if it will be used. 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 in a section of its own.

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 no more operations are to be performed at this time.

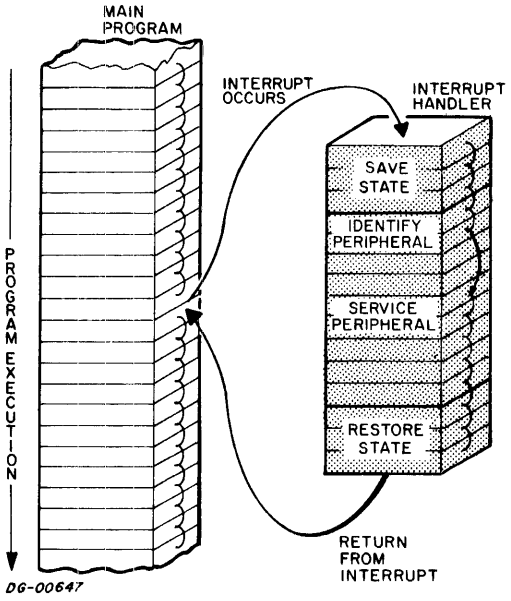
When all service for the peripheral has been completed, either the peripheral service routine or the main interrupt handler should perform the following sequence to dismiss the interrupt.

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 and the carry bit.
3. Set ION to 1 to 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 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.



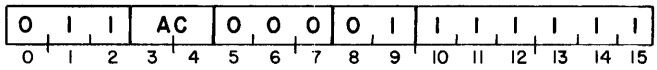
Instructions

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 PWR FF; 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

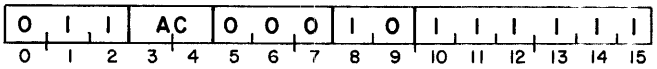


Set the Interrupt On flag to 1 to allow the processor to respond to interrupt requests. If the Interrupt On flag actually changes state (from 0 to 1), the processor will execute one more instruction before it can start an interrupt. On the ECLIPSE computer, the processor will execute one more instruction before starting an interrupt even if the Interrupt On flag was already 1. However, if that instruction is one of those that is interruptable, then 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

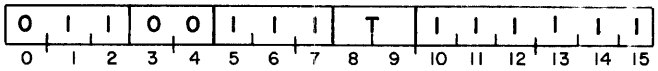
NIOC CPU



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<u>t</u> CPU



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

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.

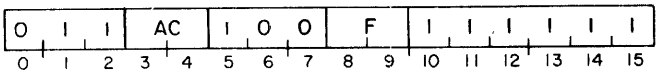
DG-01444

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 below.

MASK OUT

MSKO ac

DOB<f> ac,CPU

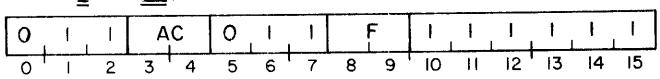


Set the Interrupt Disable flags in all peripherals according to the mask contained in the specified AC. (A 1 in a mask bit sets the flags in all peripherals assigned to that bit to 1, a 0 sets them to 0.) After the Interrupt Disable flags are set, the Interrupt On flag is set according to the function 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

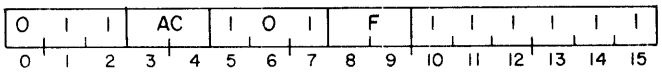


The device code of that peripheral requesting an interrupt which is closest to the processor along the I/O bus is placed in bits 10-15 of the specified AC. Bits 0-9 are set to 0. After the data transfer, the Interrupt On flag is set according to the function specified by F. If no peripheral is requesting an interrupt, the specified AC is set to 0. The assembler recognizes the instruction INTA ac as equivalent to DIB ac,CPU.

I/O RESET

IORST

DIC<f> ac,CPU



Reset all peripherals connected to the I/O bus: set their Busy, Done, and Interrupt Disable flags to 0 and, depending on the peripheral, perform 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.

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

The assembler recognizes a number of convenient mnemonics for instructions that control the program interrupt.

Mnemonic	Instruction	Mnemonic Equivalent	Octal Equivalent
INTEN	INTERRUPT ENABLE	NIOS CPU	060177
INTDS	INTERRUPT DISABLE	NIOC 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

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 give

```
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 give

```
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 Teletype® 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 Teletype 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 Teletype service routine takes 600 microseconds, since a card reader interrupt request initiated soon after Teletype 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 Teletype 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 all peripherals' Interrupt Disable flags 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. Peripherals which are not masked out assume a higher 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

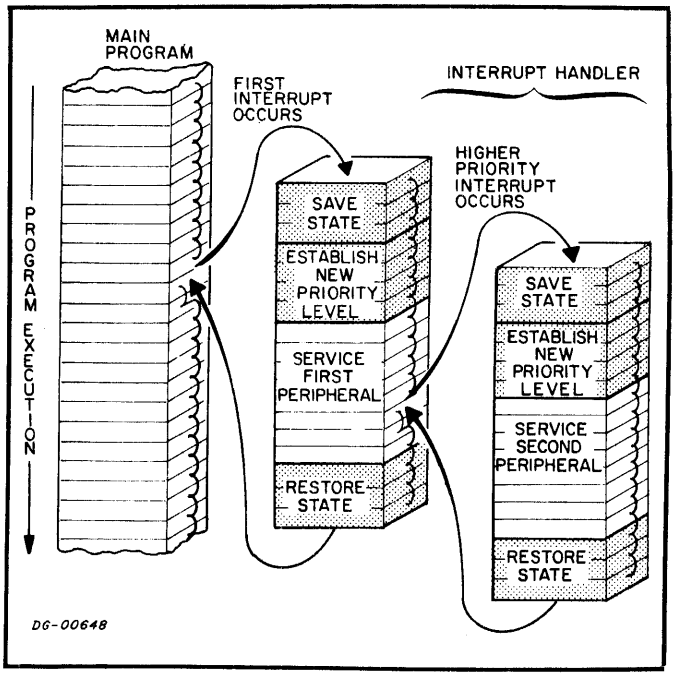
A priority interrupt handler differs from a single-level interrupt handler in several ways. The handler must be "re-entrant". This means that if a peripheral service routine is interrupted by another, higher priority peripheral, no information required by the handler to restore the state of the machine, is lost. 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.

Teletype® is a registered trademark of Teletype Corporation, Skokie, Illinois. All references to teletypes in this manual shall apply to this mark.

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, 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 for 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.

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

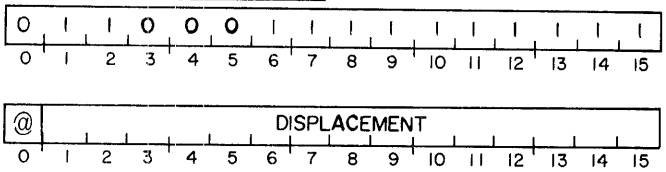


The Vector Instruction

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" and 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.

An INTERRUPT ACKNOWLEDGE instruction is performed. The device code returned is added to the address of the vector table and the vector table entry at that address is fetched. If the direct bit in the fetched vector table entry is 0, the address in bits 1-15 is taken to be the address of the device handler routine for the interrupting device and control is immediately transferred there by placing the address in the program counter.

If the direct bit is 1, the address in bits 1-15 of the vector table entry is taken to be the address of the device control table (DCT) for the interrupting device. At this point, the stack change bit is examined. If the stack change bit is 0, no stack change is performed. If the stack change bit is 1, a new stack is created by placing the contents of memory location 6 in the stack limit, and the contents of memory location 7 in the stack fault. The previous contents of memory locations 40_8-43_8 are then pushed 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.

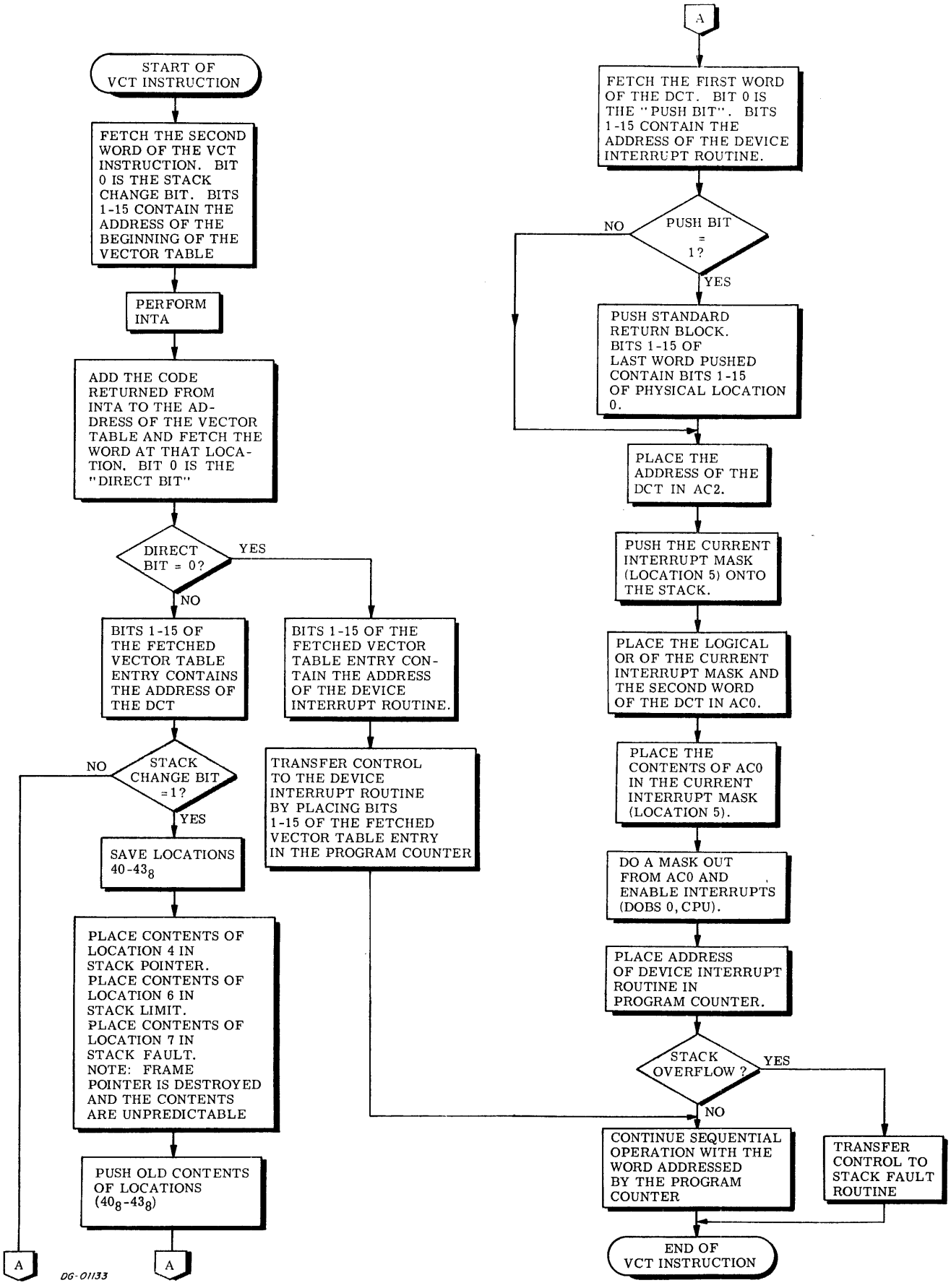
After the stack change procedure is performed, the first word of the DCT is fetched and inspected. If the push bit is 1, a standard return block is pushed onto the stack with bits 1-15 of physical location 0 placed in bits 1-15 of the last word pushed. If the push bit is 0, no return block is pushed.

Following this procedure, the address of the DCT is placed in bits 1-15 of AC2 and bit 0 of AC2 is set to 0.

Next, the current interrupt priority mask is pushed on the stack. The contents of the second word of DCT are logically OR'd with the current interrupt priority mask and the result is placed 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 DOBS 0,CPU instruction is now performed. This is a MASK OUT instruction that also enables the interrupt system.

After a new interrupt priority mask is established and the interrupt system enabled, control is transferred to the device handler by placing bits 1-15 of the first word of the DCT in the program counter.

VECTOR INSTRUCTION



Use of the Vector Instruction

The VECTOR ON INTERRUPTING DEVICE CODE 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, this instruction should be well understood before it is used.

The VECTOR instruction can operate in any one of five modes. These modes are called mode A, mode B, mode C, mode D, and mode 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 takes more time to execute.

For all modes, the VECTOR instruction uses bits 1-15 of the second word to address the vector table. An INTERRUPT ACKNOWLEDGE instruction is performed and the device code received is added to the address of the vector table and the word at that location is fetched. At this point, the mode selection process begins.

Which mode is 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 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

Note: X = Don't care

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

The uses of the five modes are described below.

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 non-buffered device with a very small latency time. Alternatively, 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, those processes that operate at base level are user programs. Those processes that operate at non-base level are the various interrupt handlers in the system.

One of the first things that the supervisor program should do when it receives an interrupt while a process is operating at base level is to change the stack environment. Two reasons lead to this conclusion. 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 the stack. 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 saves a little bit of time over modes C and E, it makes returning control to the interrupted program somewhat more complicated.

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 in 1 in their vector table entries, and control their modes of service by the setting of the push bit in their DCT's.

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 means that the amount of program overhead in the form of executing I/O instructions and loading or storing data is greatly reduced. The time required for program execution is lengthened however, since the processor pauses, as soon as it is able, each time a word is to be transferred; the transfer then occurs and the processor continues. The program need only set up the peripheral for the transfer and can then perform other, unrelated tasks.

The data channel facilities in the original NOVA, NOVA 1200 series, and the ECLIPSE line of computers all provide a single speed for data channel operation. The SUPERNOVA series, NOVA 800 series, and NOVA 2 series computers all can operate the data channel at two different speeds: standard and high speed. In addition to merely transferring data, certain arithmetic operations can be performed by the data channel in some computers. All the NOVA line computers can have the contents of any memory location incremented by 1 each time a controller requests that operation. The NOVA and SUPERNOVA computers also allow a controller to add a word to the previous contents of any memory location.

In both types of arithmetic operation, the computer sends the results back to the controller and, if the operation increased the contents of the memory location to more than $2^{16}-1$, it sends an overflow signal.

The data channel allows many peripherals to be active at the same time, providing access to memory to 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 given to that controller which is closest to the processor on the I/O bus.

A table in Appendix B includes the maximum transfer rates for all combinations of channel speed and type of transfer.

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.

Two registers usually added are a Word Counter and a Memory Address Counter. 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 Word Counter is loaded by the program 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. Typical sizes of the Word Counter are 12 and 16 bits, allowing for up to 4096-word blocks and 65,536-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--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. The table below further illustrates the correspondence between the desired word count and the value which must be loaded into a 12-bit or 16-bit Word Counter.

(negative) word count (decimal)	16-bit value (octal)	12-bit value (octal)
-1	177777	7777
-2	177776	7776
-8	177770	7770
-100	177634	7634
-2047	174001	4001
-2048	174000	4000
-2049	173777	3777
-4095	170001	0001
-4096	170000	0000
-4097	167777	
-8192	160000	
-32768	100000	
-65535	000001	
-65536	000000	

Memory Address Counter

The Memory Address Counter always contains the address in memory which is to be used by the controller for the next data transfer. It is loaded, by the program, 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 pauses as soon as it is able, and begins the data channel cycle by acknowledging the peripheral's data channel request. The acknowledgment signal dismisses the peripheral's data channel request and causes the peripheral to send back to the processor the address of the memory location involved in the transfer. Following the receipt of the address, the data itself is transferred in the appropriate direction.

At the completion of each data transfer 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 processor starts another data channel transfer if any data channel requests are pending, starts a program interrupt if one is being requested and there are no data channel requests, or resumes program execution.

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

The processor can pause for a data channel transfer only at certain, well-defined times, depending on the model of processor and, in the SUPERNOVA computer, on the channel speed used by the peripheral requesting the transfer. For the NOVA, NOVA 1200 series, and the standard channel on SUPERNOVA computers, data channel transfers, like program interrupts, can only occur between instructions. High-speed data channel requests on the SUPERNOVA computer, and all requests on the NOVA 800 series, NOVA 2 series, and ECLIPSE line computers pause for 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, program execution has priority over the data channel except at certain points in the processor's operation, at which times the data channel has absolute priority (over not only normal program execution but also over any pending program interrupt requests). At these certain 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, and program execution will stop until all the data channel transfers are made. However, when the data channel is being used at less than its maximum rate, 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 pauses to honor 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. Since all peripherals operating with the high-speed data channel must be grouped together at the beginning of the bus, all requests from high-speed controllers will be honored before any from those which operate at standard speed. To use the high-speed data channel, the controller for a peripheral must be mounted inside the mainframe of the computer and must be designed to operate within the high-speed data channel time constraints. A computer that has the two-speed capability is shipped with the high speed enabled for all controllers mounted inside the mainframe. (Controllers in an expansion chassis are constrained to operate at standard speed.)

Programming

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

1. The peripheral's status is checked, 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, 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 data block in the device is located. This usually involves giving a peripheral "address" by specifying a unit number, channel number, sector number, or the like.
3. The data block in memory is located by loading the Memory Address Counter with the address of the first word of the block.
4. The size of the data blocks is specified by loading the proper value into the Word Counter.
5. The type of transfer is specified and the operation is initiated. 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.

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 peripheral 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 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 components:

5. The time when CPU operation is suspended because data channel transfers are in progress (see following section).
6. The time during which 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. The time the 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 the ECLIPSE S series of computers it can be considerably longer due to the presence of the WCS feature which allows the programmer to code long instructions which 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 the data channel devices operating 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", DG 015-000021.

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), and
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, the CPU 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, and
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 the NOVA, NOVA 1200 series, and SUPERNOVA (standard-speed) 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 the NOVA 800 series, NOVA 2 series, ECLIPSE line and SUPERNOVA (high-speed) computers, data channel breaks may be enabled during most instructions (but not during I/O instructions), so that data channel latency is reduced.

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, the data 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 maximum 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 usually should 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 in the NOVA and 1200 series computers, and less frequent use of I/O instructions in the SUPERNOVA, 800 series, and ECLIPSE line of computers. In addition, there is an upper limit on the number of data channel transfers/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.

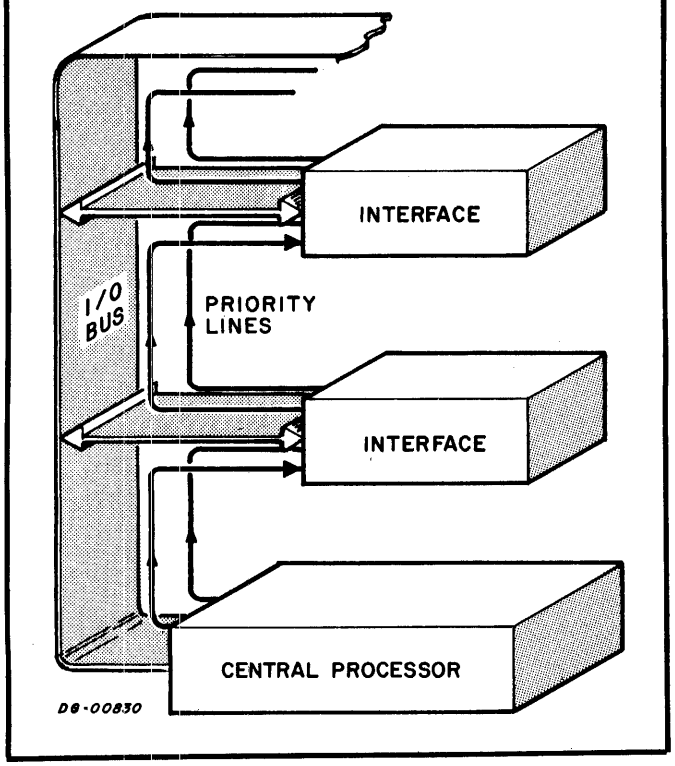
A final consideration is that high data channel use reduces the speed of program execution since the processor pauses for each transfer. This may adversely affect the CPU's capacity to respond to interrupts and service those peripherals operating under direct program control.

SECTION III
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 Data General machines. This structure embodies a single 48-line bus connecting the central processor to all interfaces. 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.

I/O BUS STRUCTURE



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 "$\overline{\text{WRITE}}$", 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 Section IV. Note that, with the exception of the two priority lines, all I/O bus lines are run in parallel to all interfaces.

Data

$\overline{\text{DATA}<0\text{-}15>}$ 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

$\overline{\text{DS}<0\text{-}5>}$ Device Select. These lines carry the low-order six bits of the instruction currently being executed; that is, the device code when the instruction is an I/O instruction. 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 the 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 and 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 and DOC instructions until after 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 (1) control function is specified). Not asserted in DIA, DIB, DIC, DOA, DOB and DOC instructions until after the data transfer has occurred. Usually used to initiate special peripheral operations.

$\overline{\text{SELB}}$ — Selected Busy. Asserted by the interface selected by the device select lines if its Busy flag is set to one.

$\overline{\text{SELD}}$ — Selected Done. Asserted by the interface selected by the device select lines if its Done flag is set to one.

Program Interrupt

$\overline{\text{INTR}}$ — Interrupt Request. Asserted by an interface to request program interrupt service.

$\overline{\text{MSKO}}$ — Mask Out. Asserted by the processor during the execution of the MSKO instruction, after the contents of the designated accumulator have been placed on the DATA lines of the I/O bus. Used to load the contents of the DATA lines into the interrupt disable flip-flops of all interfaces using the interrupt system.

$\overline{\text{INTP}}$ — 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 $\overline{\text{INTP OUT}}$ unless it is receiving an asserted $\overline{\text{INTP IN}}$ 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 $\overline{\text{INTP IN}}$ asserted and while it is requesting interrupt service, it should place its device code on the low-order DATA lines.

Data Channel

$\overline{\text{DCHR}}$ — Data Channel Request. Asserted by a device when it requires data channel service.

$\overline{\text{DCHP}}$ — 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 $\overline{\text{DCHP OUT}}$ unless it is receiving an asserted $\overline{\text{DCHP IN}}$ and it is not requesting data channel service. Also used by some processors to determine data channel speed.

$\overline{\text{DCHA}}$ — Data Channel Acknowledge. Asserted by the processor at the beginning of each data channel cycle. Should cause the interface that is receiving an asserted $\overline{\text{DCHP IN}}$ 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 the data lines and the mode on the data channel mode lines of the I/O bus.

$\overline{\text{DCHM0}}$ / $\overline{\text{DCHM1}}$** — Data Channel Mode. Asserted 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	DCHM1**	Function
0	0	Output
0	1	Increment Memory*
1	0	Input
1	1	Add to Memory

*not available on all processors.

$\overline{\text{DCHI}}$ — 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 the data lines of the I/O bus.

$\overline{\text{DCHO}}$ — Data Channel Output. Asserted by the processor for data channel output (DCHM$<0,1>$ = 10), after the data word has been placed on the DATA lines of the I/O bus. Should cause the priority-selected interface to load the data from the DATA lines.

**Not available on Eclipse line computers.

OVFLO	Overflow. Asserted by the processor during a data channel cycle that increments or adds to memory (DCHM1=1), when the result exceeds $2^{16}-1$. (Not available on Eclipse line computers)

System Control

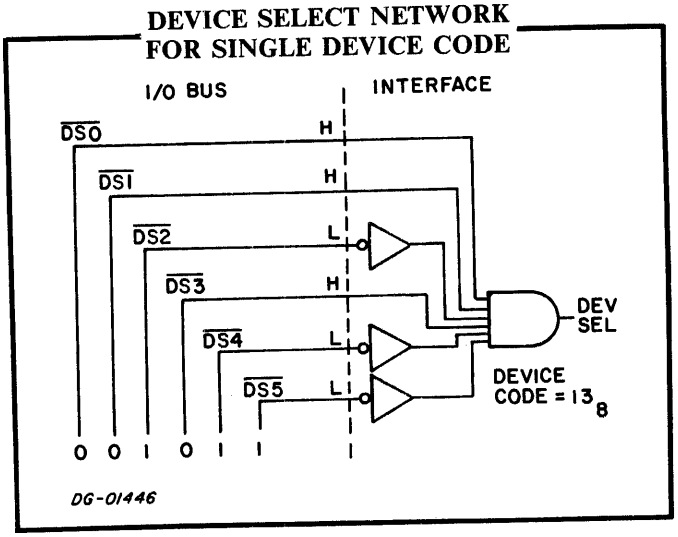
IORST	I/O Reset. Asserted by the processor during the IORST instruction or when the console RESET switch is activated. IORST is also issued prior to processor operation at power turn-on and when power is removed. This signal should be used to initialize the machine state of all interfaces in the system.
PWR ON	Power On. This 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.
$\overline{\text{RQENB}}$	Request Enable. Asserted by the processor to synchronize program interrupt and data channel requests from all interfaces. In any interface, $\overline{\text{INTR}}$ and $\overline{\text{DCHR}}$ should be clocked only on the leading edge of $\overline{\text{RQENB}}$.

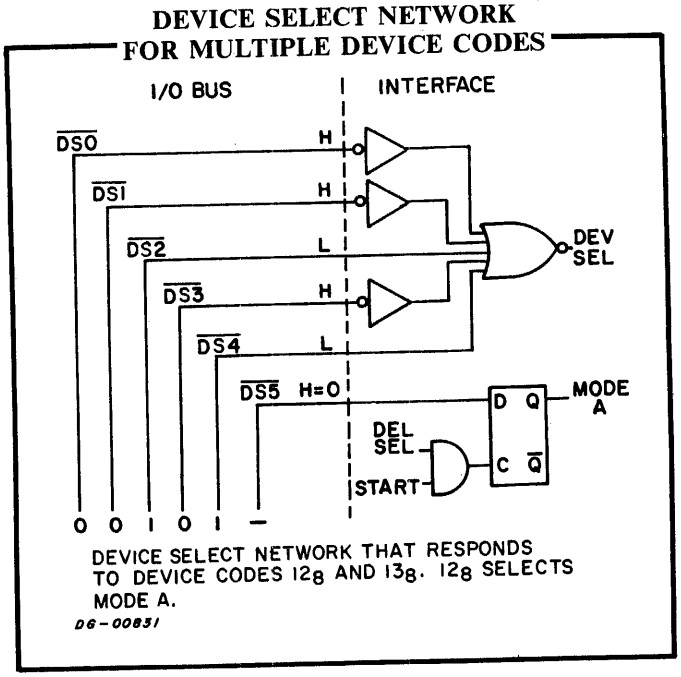
PROGRAMMED I/O PROTOCOL

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. At other times the device select lines will carry meaningless data. This random data, however, will look at times like some interface's device code. Because of this false indication, the interface should not assert the $\overline{\text{DATA}}$ 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 below for device code 13_8. 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 is 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, for example.





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 MSKO and INTA, and hence can not be assigned to an interface. Similarly, a number of programming considerations restrict the use of device code 0. The remaining sixty-two codes can be assigned to interfaces. Appendix A lists the device codes as assigned by Data General and used in all Data General software. 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.

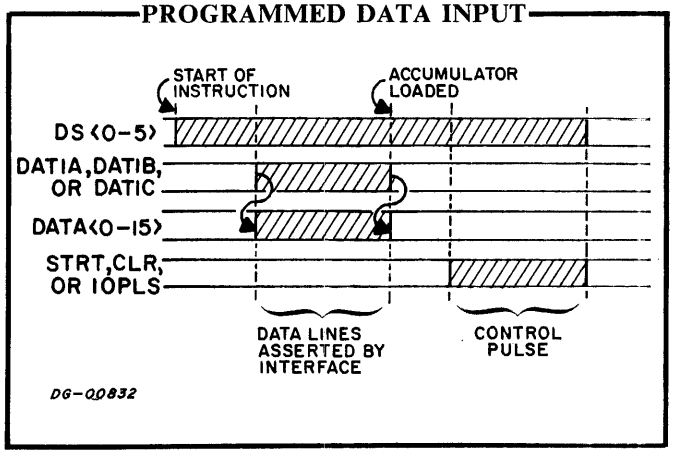
Data Transfer Signals

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 $\overline{\text{DATA}}$ lines of the bus. After a time delay, the processor will load the information on the $\overline{\text{DATA}}$ 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 $\overline{\text{DATA}}$ 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. The response of the interface to each of these signals will be discussed later.

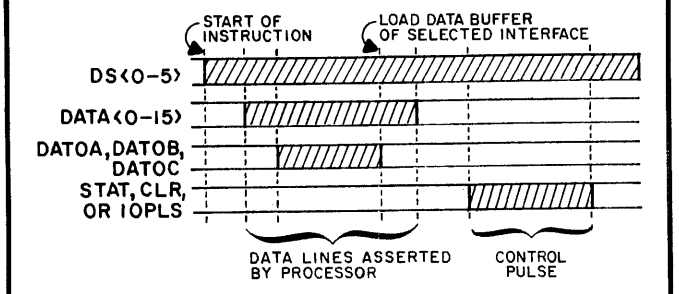
Data Output

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 $\overline{\text{DATA}}$ 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 $\overline{\text{DATA}}$ lines to the proper destination and to load a register. After dropping the transfer control signal, the processor will drop the data from the $\overline{\text{DATA}}$ 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 $\overline{\text{DATA}}$ lines during the data output instruction.

PROGRAMMED DATA OUTPUT



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. The response of the interface to each of these signals will be discussed later.

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, $\overline{\text{SELB}}$ and $\overline{\text{SELD}}$. Whenever an interface recognizes its device code on the device select lines, as discussed above, it should assert the $\overline{\text{SELB}}$ and/or $\overline{\text{SELD}}$ lines depending on the internal state of the interface. (Ordinarily, $\overline{\text{SELB}}$ will be asserted if the Busy flag is set and $\overline{\text{SELD}}$ 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 follows:

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

Start, Clear, I/O Pulse

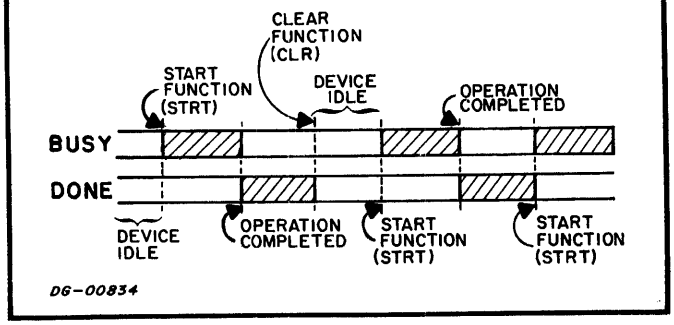
During the second portion of the execution of any I/O instruction except I/O Skip, the processor can issue one of three control signals - STRT, CLR, or IOPLS - as coded in the instruction. Though a convention is followed in the use of these signals in Data General interfaces, as explained below, the 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.

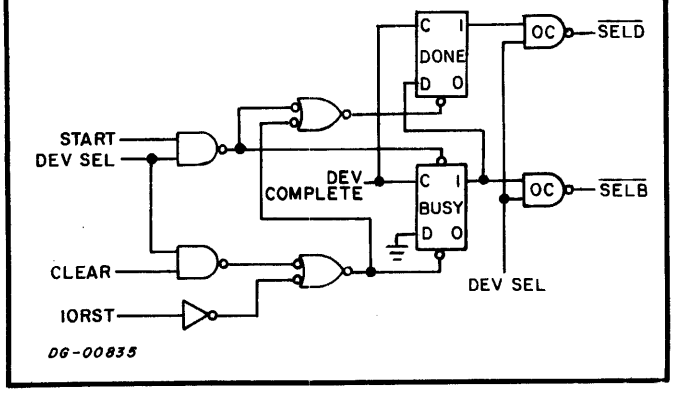
TYPICAL BUSY/DONE OPERATION SEQUENCE



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 STRT 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 CLR pulse is issued by the processor, both flags should be cleared.

The following illustration 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.

BUSY/DONE NETWORK



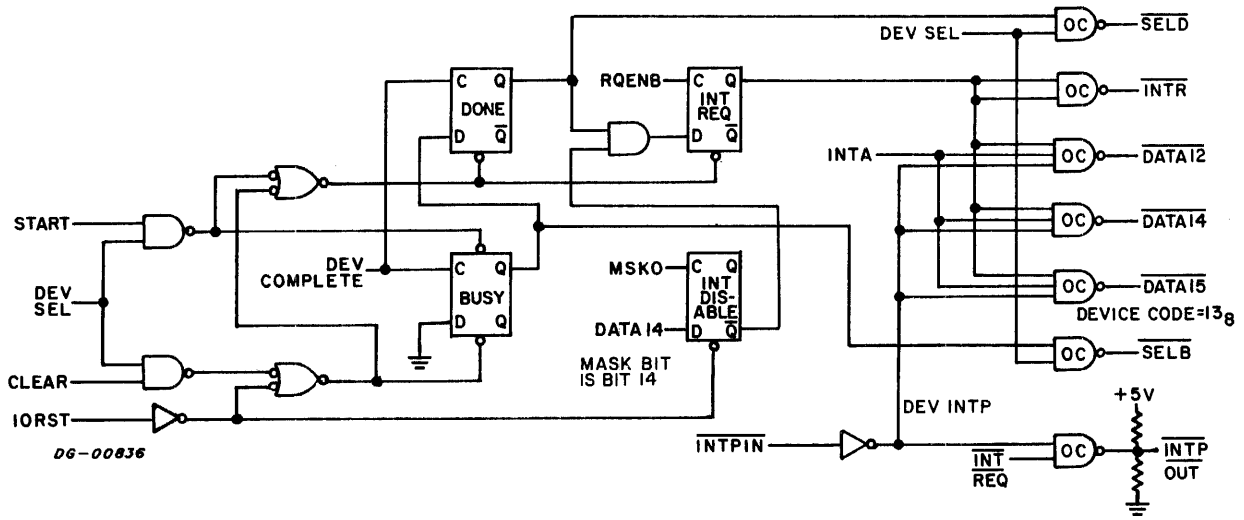
PROGRAM INTERRUPT SYSTEM

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.

The central processor generates a signal on the I/O bus, $\overline{\text{RQENB}}$, which toggles up and down as the processor is running. The timing pulses that this signal provides are important to the proper operation of the interrupt system. The interrupt request should be issued only on the leading edge of $\overline{\text{RQENB}}$, for reasons explained in Appendix D. The usual convention is to use RQENB to clock the state of the DONE flag into an INT REQ flip-flop, which in turn drives the INTR line, as shown below.

BUSY/DONE/PROGRAM INTERRUPT NETWORK



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).

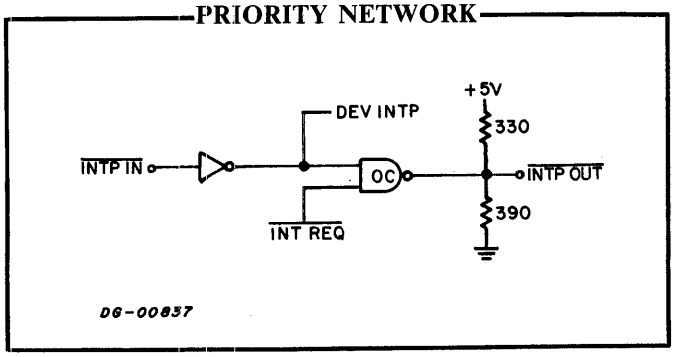
Conventionally, each interface using the interrupt system is assigned a hardware priority level corresponding to one of the sixteen bits of a data word. When the MSKO signal is received by the interface (regardless of device code), an interrupt disable flag (INT DISABLE) should be loaded from the $\overline{\text{DATA}}$ line corresponding to the priority assignment of that interface, as shown above. 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 $\overline{\text{RQENB}}$ pulse.

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 $\overline{\text{INTP IN}}$ as it enters each interface and $\overline{\text{INTP OUT}}$ as it leaves. $\overline{\text{INTP IN}}$ starts on the computer back panel, immediately above 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 $\overline{\text{INTP IN}}$ signal should pass a high $\overline{\text{INTP OUT}}$ to the following interfaces and on down the bus.

The circuit below 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 $\overline{\text{INTP OUT}}$ signal of one feeding $\overline{\text{INTP IN}}$ of the next.

Note that the terminating resistors shown are to be used only on the signals that enter or leave the board. Similarly, the open-collector NAND gate is used only for the last element on that board, while standard-output type NAND gates are used for the earlier stages.



Timing on this interrupt priority chain can be critical and becomes especially so for large systems with many interfaces. As explained in Appendix D, the time required for a change in the INTP level to propagate from the first to the last interface on the I/O bus must not exceed 300 nanoseconds. This includes both propagation time on the I/O bus cable (if used) and the delays encountered in the logic components. 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 INTP signal on to the next board. A separate chain determines the priority of the various interfaces on the board.

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 (INTA) instruction. This instruction is equivalent to a Data Input B 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 porition 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 INTA. At the end of the INTA period, the processor loads the selected accumulator with the contents of the DATA lines.

DATA CHANNEL

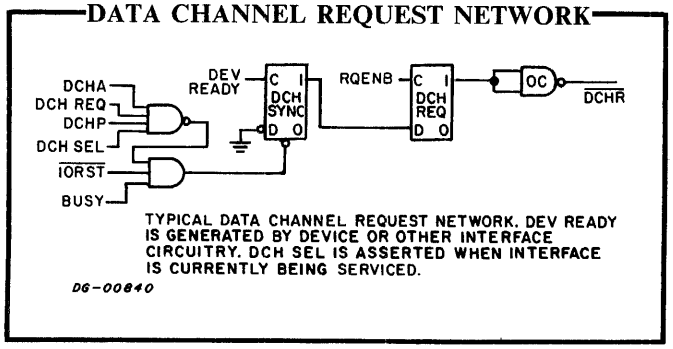
Unlike the programmed input/output transfers, which are each controlled by one unique signal on the bus, the data channel transfers are somewhat more complex. The interplay between the central processor and the interface is much more involved, due to the number of functions performed for each transfer. 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.

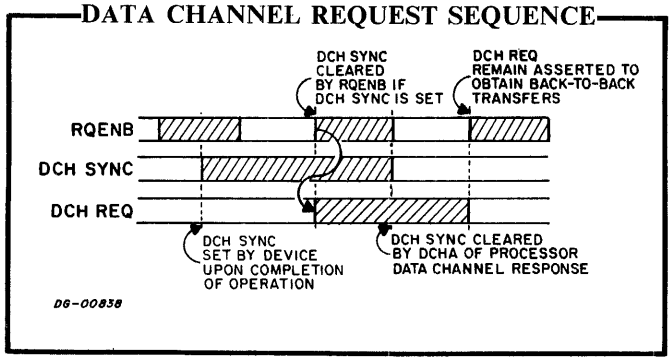
Data Channel Request

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. Normal program execution is suspended for the duration of the data channel transfer. This delay is virtually transparent to the program, however, 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 the figure below) into a data channel request (DCH REQ) flip-flop, which in turn drives the DCHR line, as shown below. It is very important that DCHR be clocked only on the leading edge of RQENB. (See Appendix D.)

When the processor sees DCHR asserted, it pauses at the next convenient point in its operation and transfers data to or from the highest priority controller on the I/O bus. 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. The processor then resumes program execution. (When more than one data channel transfer occurs during a single processor pause, or data channel break, the transfers are said to be back-to-back transfers.)



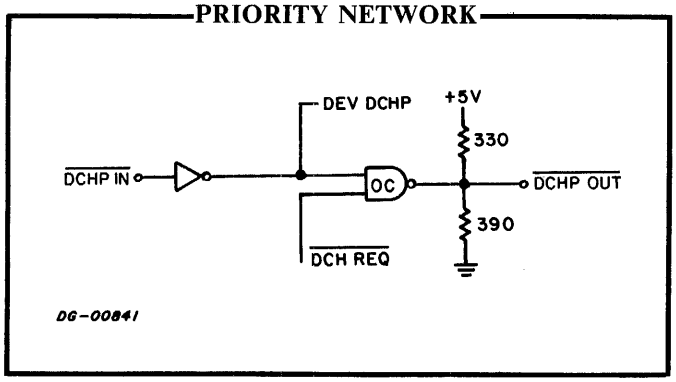


Data Channel Priority

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 priority system also serves to inform the processor whether the interface requesting service should be serviced using fast or standard data channel timing.

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 $\overline{\text{DCHP IN}}$ as it enters each interface and $\overline{\text{DCHP OUT}}$ as it leaves. $\overline{\text{DCHP IN}}$ starts on the computer back panel, immediately above 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 $\overline{\text{DCHP IN}}$ signal should pass a high $\overline{\text{DCHP OUT}}$ 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 $\overline{\text{DCHP IN}}$; that is, the interface closest to the processor that is requesting data channel service.

The circuit below shows the suggested implementation of this priority network. In many 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 $\overline{\text{DCHP OUT}}$ signal of one feeding $\overline{\text{DCHP IN}}$ of the next. Note that the terminating resistors shown are to be used only on the signals that enter or leave the board. Similarly, the open-collector NAND gate is used only for the last element on that board, while standard-output type NAND gates are used for the earlier stages.

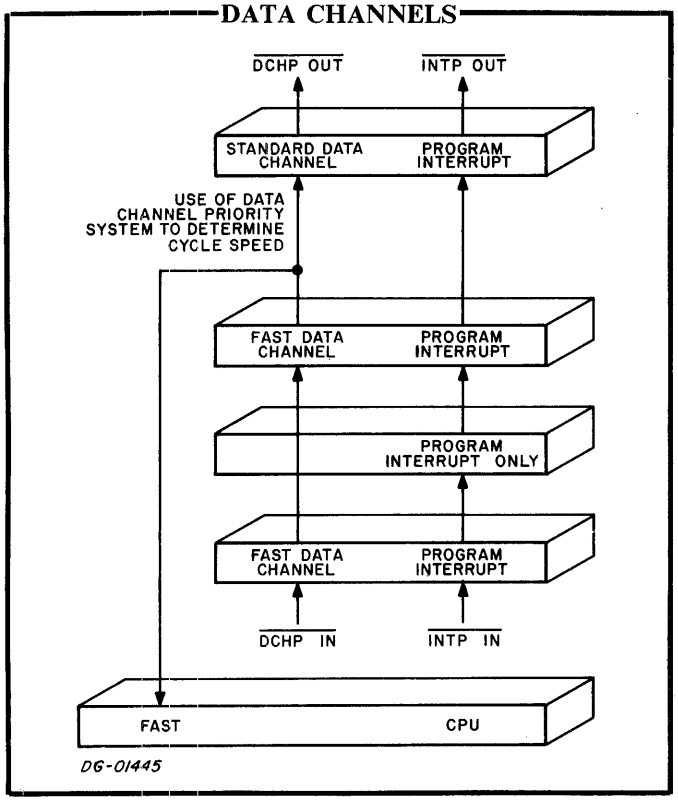


Timing on this data channel priority chain can be critical and becomes especially so for large systems with many interfaces as explained in Appendix D. The time required for a change in the $\overline{\text{DCHP}}$ level to propagate from the first to the last interface on the I/O bus must not exceed 300 nanoseconds. This includes both propagation time on the I/O bus cable (if used) and the delays encountered in the logic components. 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 $\overline{\text{DCHP}}$ signal on to the next board. A separate chain determines the priority of the various interfaces on the board.

Data Channel Speeds

The NOVA 2, 800, and SUPERNOVA computers have data channels capable of operating at a standard speed and a high speed. Except for differences in timing, the operation at both speeds is similar and the central processor services an interface at either speed based on the priority system. Due to the short pulse times of the high speed channel, however, only interfaces installed within the processor chassis are permitted to use this facility.

The processor is made aware of what type of service is being requested by testing the $\overline{\text{DCHP OUT}}$ signal of the top slot of its chassis. Whenever this signal goes high, the processor will respond by servicing the data channel request at the high speed. This signal goes high whenever an interface installed in the chassis requests service, hence, such interfaces will be serviced at the high speed. On the other hand, if an interface external to the chassis requests service, the condition of the priority line at the test point does not change and thus the interface is serviced at the standard speed.

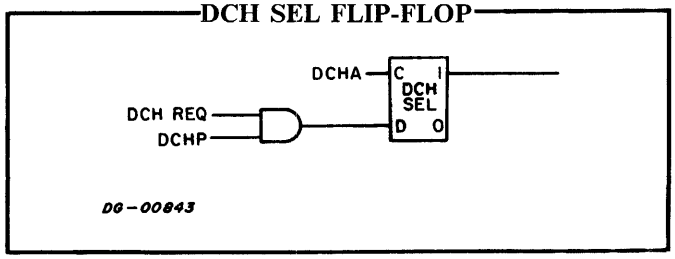


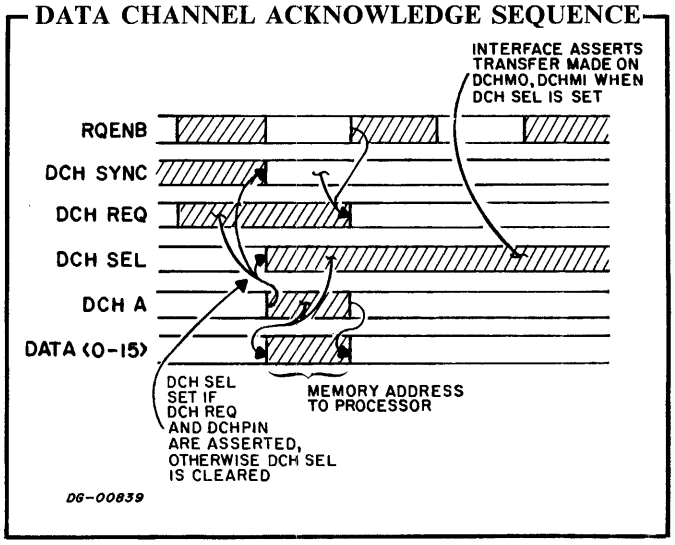
Acknowledge

As the first step in any data channel transfer the central processor will issue the data channel acknowledge signal, $\overline{\text{DCHA}}$, 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 $\overline{\text{DCHA}}$ 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 $\overline{\text{DCHP IN}}$.

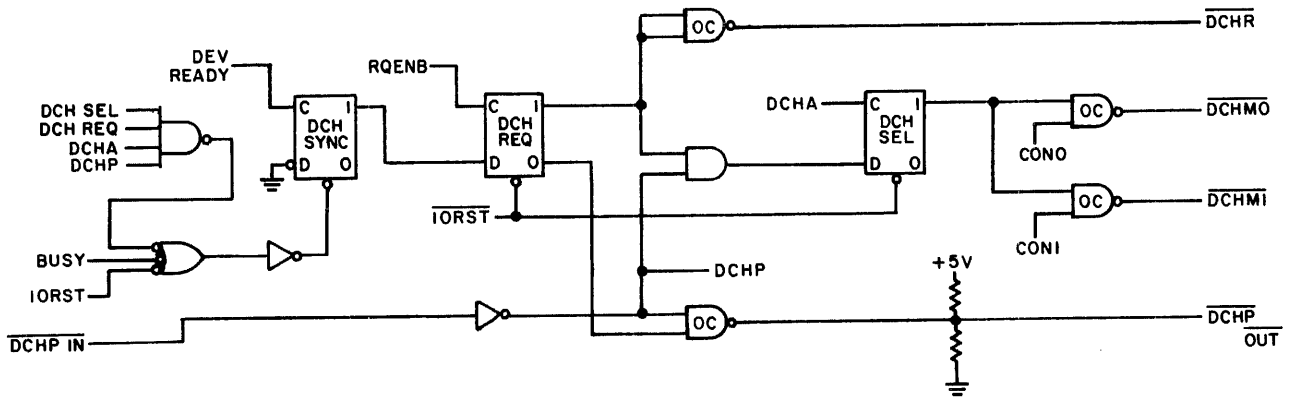
The illustration below shows the use of a flip-flop, called DCH SEL, which serves this purpose. On the leading edge of DCHA, this flip-flop will be set if DCH REQ is set and the interface is receiving an asserted $\overline{\text{DCHP IN}}$. Otherwise, it will be cleared on this signal. Thus DCH SEL 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.





While it is receiving $\overline{\text{DCHA}}$, the interface whose DCH SEL flag is set should place the memory address for this transfer on the $\overline{\text{DATA}}$ lines of the I/O bus. Finally, unless back-to-back transfers are desired, the $\overline{\text{DCHA}}$ signal should be used to clear the DCH SYNC flag, so that DCH REQ will be cleared on the next RQENB. The data channel control circuitry discussed here is shown in its entirety below.

DATA CHANNEL CONTROL NETWORK



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, i.e., the NOVA 3 and ECLIPSE series of computers, data channel map selection occurs at DCHA time. If the data channel facility is enabled at DCHA time, DATA0 selects on of two data channel maps. If DATA0 is asserted at DCHA time, DCHA MAP B is selected. If DATA0 is not asserted at DCHA time, DCH MAP A is selected.

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. Depending on the processor there are up to four possible modes. The mode is designated by a two-bit code, which sould be asserted on $\overline{\text{DCHM0}}$ and $\overline{\text{DCHM1}}$ whenever DCH SEL (or its equivalent) is set. The modes and their codes are listed below, followed by a description of the transfers in each mode.

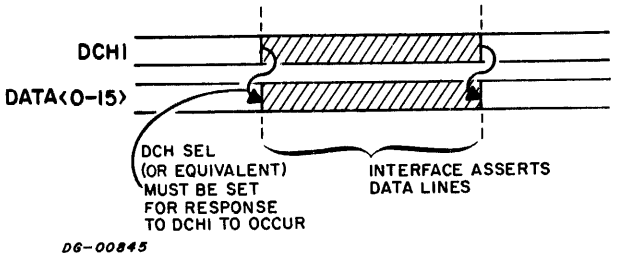
DCHM0	DCHM1	Function
0	0	Output
0	1	Increment
1	0	Input
1	1	Add to Memory

The input and output are the basic transfer modes, since the increment and add to memory modes use the same transfer signals as input and output. The only difference is in the action of the processor as it acts on the data.

Input

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

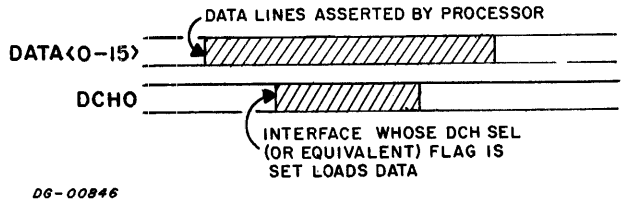
DATA CHANNEL INPUT SEQUENCE



Output

Following the end of DCHA, the central processor will assert the $\overline{\text{DATA}}$ lines of the bus with the data word read from the referenced memory location. 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 $\overline{\text{DATA}}$ lines. After the end of DCHO, the data will be dropped from the $\overline{\text{DATA}}$ lines.

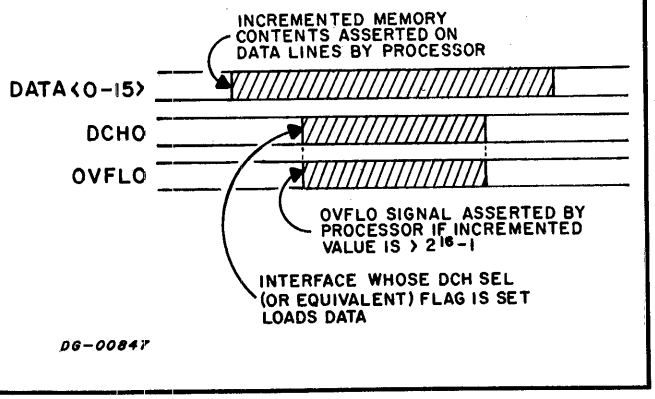
DATA CHANNEL OUTPUT SEQUENCE



Increment

The increment function (not available on the ECLIPSE line computers) looks to the interface much like the output function; however, the processor handles the data differently. Like the output function the referenced memory location is read. However, before the data is placed on the bus, it is incremented by one. This new value is placed on the bus and written into the memory location. The DCHO signal signals the interface 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.

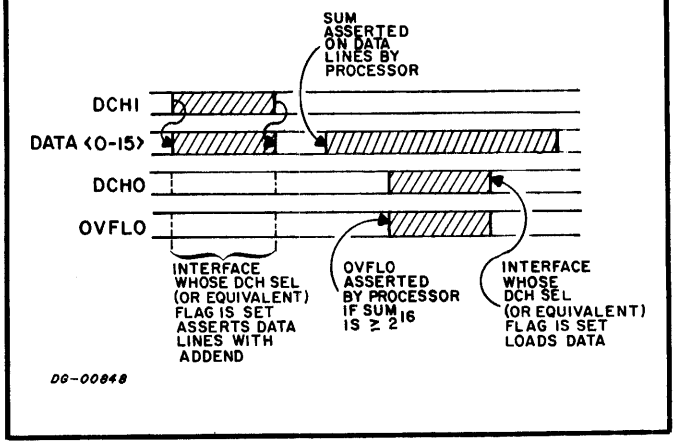
DATA CHANNEL INCREMENT SEQUENCE



Add to Memory

The Add to Memory function, available only on the NOVA and SUPERNOVA computers, 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 that obtained from the referenced memory location. The sum of these two words is written back into the memory location and transferred to the interface with a DCHO. In addition, if this sum is greater than 2^{16}-1, the OVFLO signal on the bus will be pulsed.

DATA CHANNEL ADD TO MEMORY SEQUENCE



EXAMPLES

The three 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. The Pulse Height Analyzer includes the data channel control network and uses the increment memory function of the data channel.

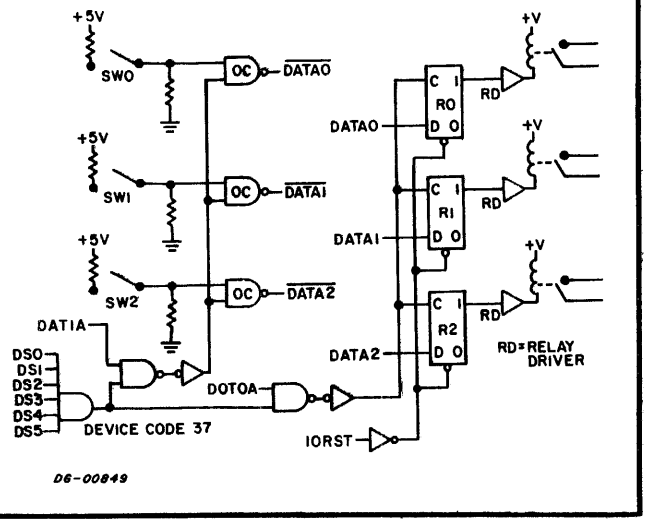
Switch Register/Relay Buffer

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 below, 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 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 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 IORST. Note that if the device comprised only the switch register or relay buffer, only a single control gate would be needed, as the transfer signal from the processor could replace one of the constant (+3V) inputs to the device selection decoder.

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.

SWITCH REGISTER RELAY BUFFER



Paper Tape Punch

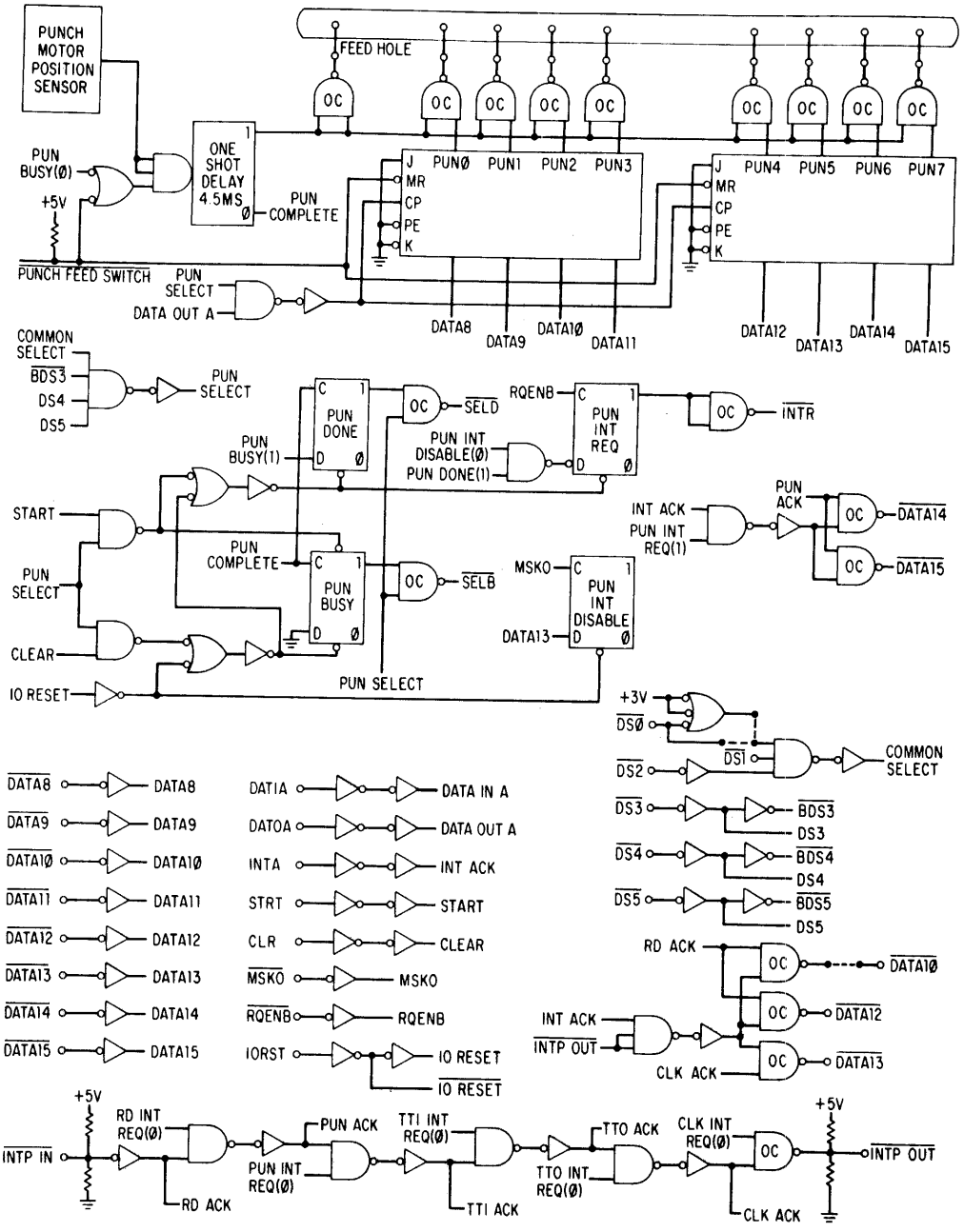
The interface for the high speed paper tape punch, which is illustrated below, shares a single board with the interfaces for the Teletype, tape reader and real time clock. The lower half of the drawing contains circuits for functions common to all of the interfaces. At the left are the receivers for the data lines and other signals. At the right are networks that generate a COMMON SELECT signal by decoding $\overline{DS<0\text{-}2>}$ and generate common device code digits for INTA. The codes are 10 through 14 so 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 through 54 instead.) Across the bottom is a chain that receives $\overline{INTP\ IN}$, 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 $\overline{DS<3\text{-}5>}$ 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 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 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 leftmost 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).

HIGH SPEED PAPER TAPE PUNCH INTERFACE



DG-00850

Pulse Height Analyzer

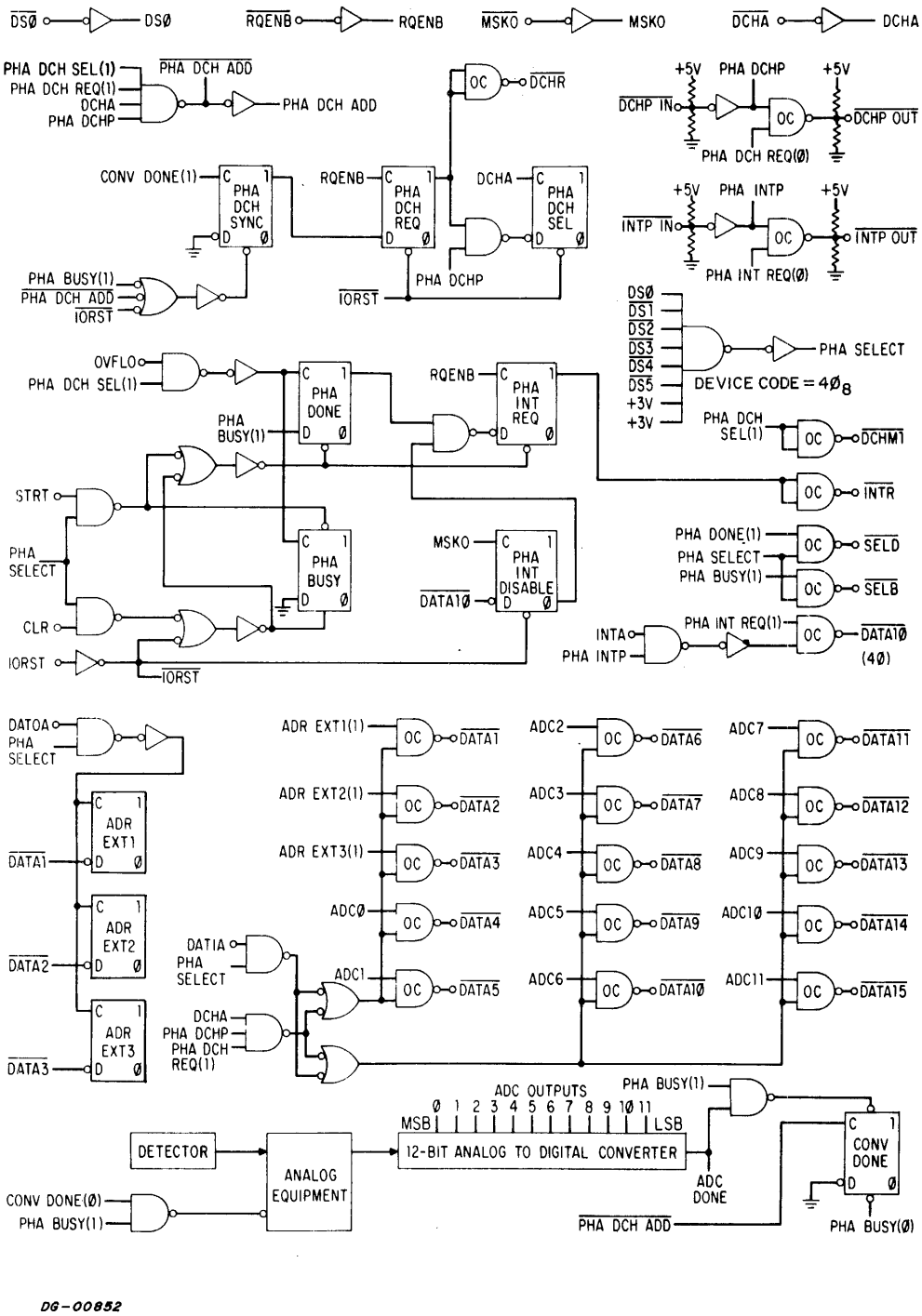
The interface shown below uses the data channel as well as programmed transfers. Its function is to increment the word in the memory location whose address is determined by the output of an analog-to-digital converter. The upper half of the drawing contains only standard circuits already described. At the right are the stages in the priority chains for the data channel and program interrupt, the device selection network, the single driver required for the data channel mode lines, and the network that supplies the device code for an interrupt acknowledgment. At the left are the state/interrupt network and the data channel request logic (INT DISABLE is controlled by mask bit 10). The gate in the upper left corner determines when the address is being sent in on the data channel; it clears DCH SYNC and is also used by the interface logic to determine when the address transfer is complete.

In the lower half of the drawing is the logic unique to this particular interface. At the bottom is the analog equipment and digital logic to control it (this logic may vary to match a specific analog unit). Above it are the drivers and associated gating to place the address on the data lines. At the left is a 3-bit address extension register that is loaded by the program. The converter supplies only the low order twelve bits of the address; the program supplied the high order three bits and thus specifies a block of 4096 words to be used as the data area.

To place the device in operation the program issues a Data Output A with a Start function (DOAS ac, 40), which supplies the address extension and sets BUSY to enable the conversion equipment. When a pulse is detected, the converter translates it to a 12-bit number and at completion generates ADC DONE. This pulse sets CONV DONE, which disables the converter and sets DCH SYNC. The next leading edge of RQENB sets DCH REQ to generate $\overline{\text{DCHR}}$. When DCHA turns on and this device has priority, DCH SEL is set, generating $\overline{\text{DCHM1}}$ to specify an increment memory cycle, and the address from the extension register and converter is placed on the DATA lines. The processor increments this location in memory and sends the result back over the bus, but it is not used in this particular interface. The termination of DCHA turns off the logic level DCH ADD, which in turn clears CONV DONE to reenable the converter.

If a location is incremented to 2^{16}, the overflow pulse (OVFLO) sent by the processor clears BUSY and sets DONE, turning off the device and requesting an interrupt. (Clearing BUSY turns off the converter and clears both CONV DONE and DCH SYNC.) The program can give a DIA ac, 40 to read the address and hence determine which location overflowed. The program can resume conversions simply by setting BUSY (as by DIAS ac, 40 which reads and restarts), and it can stop the process at any time by giving an NIOC to clear BUSY.

PULSE HEIGHT ANALYZER INTERFACE



DG-00852

This page intentionally left blank

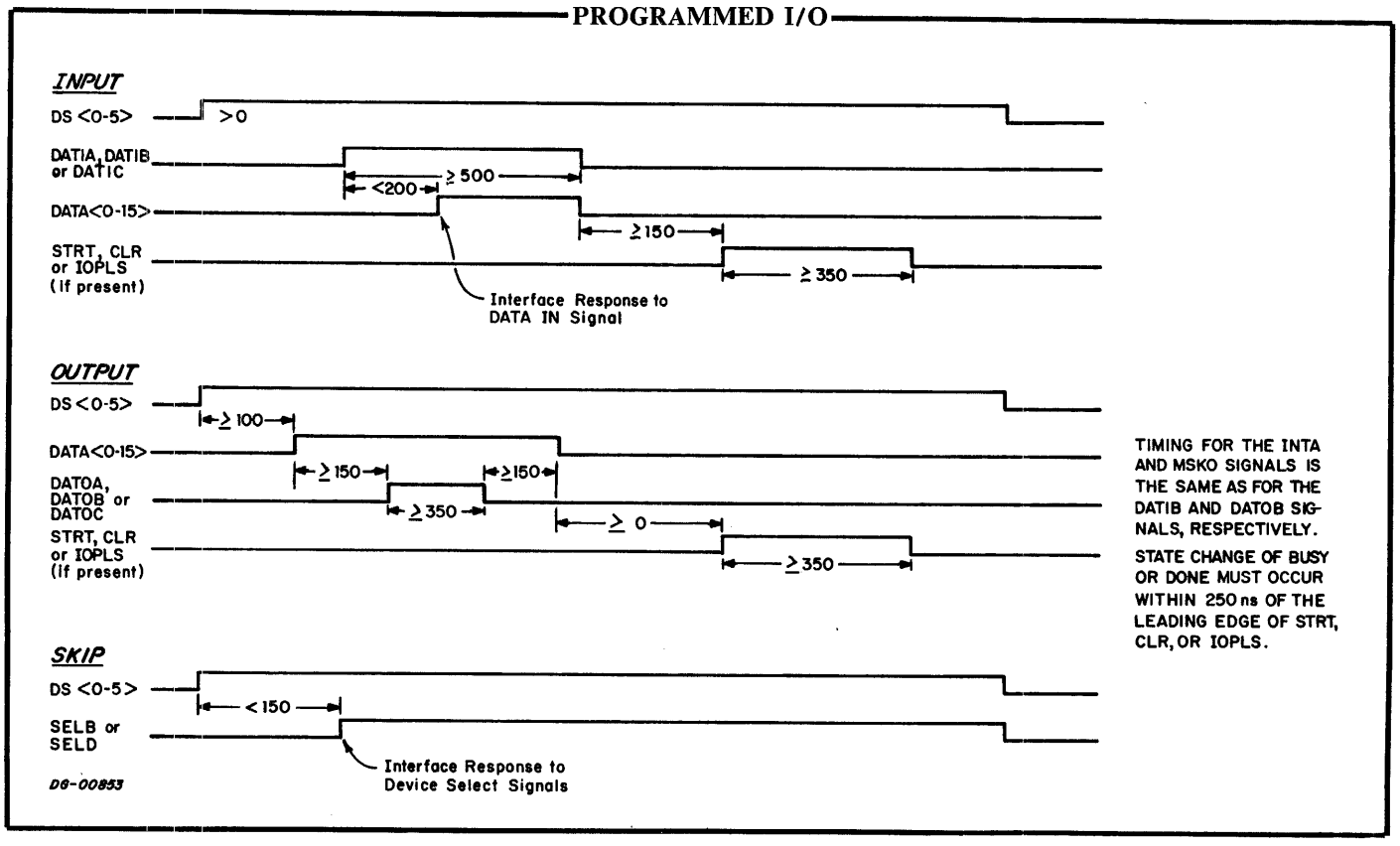
SECTION IV

I/O BUS SPECIFICATIONS

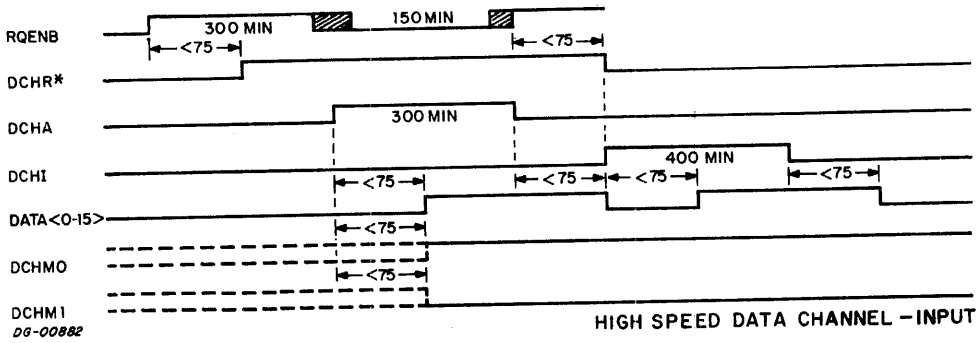
TIMING

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 Data General central processors. Timing data for a specific machine can be found in its respective Technical Manual.

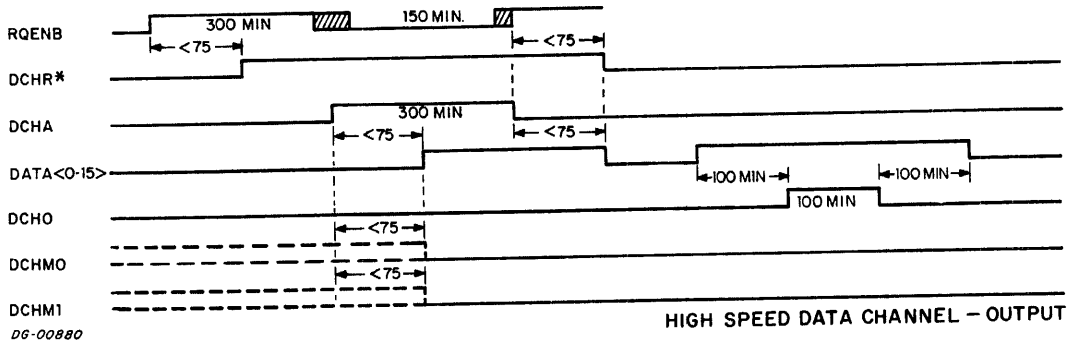
The timing diagrams show two types of timing information. The times shown for the processor originated functions (e.g., DCHI pulse) are the minimum times for which the interface must be designed. The times between processor bus functions and the interface responses on the bus to those functions are shown as the maximum allowable times. An interface designed to operate with these minimum function times and maximum response times will operate with any Data General processor. All times are in nanoseconds.



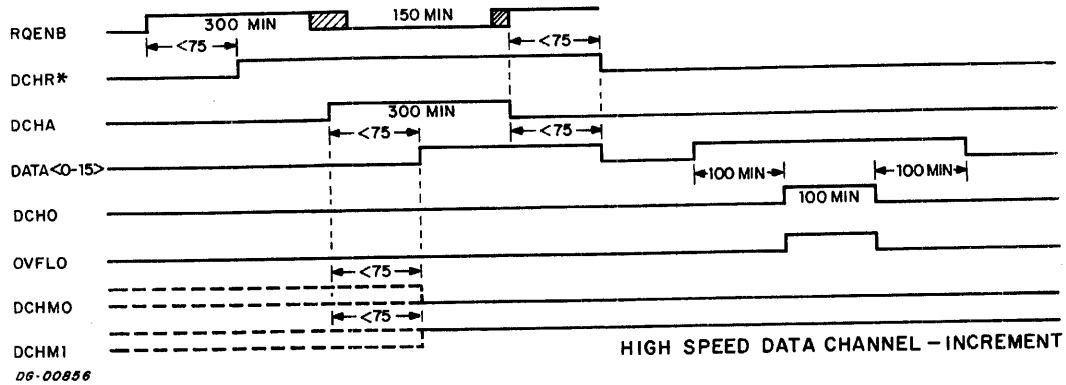
HIGH SPEED DATA CHANNEL



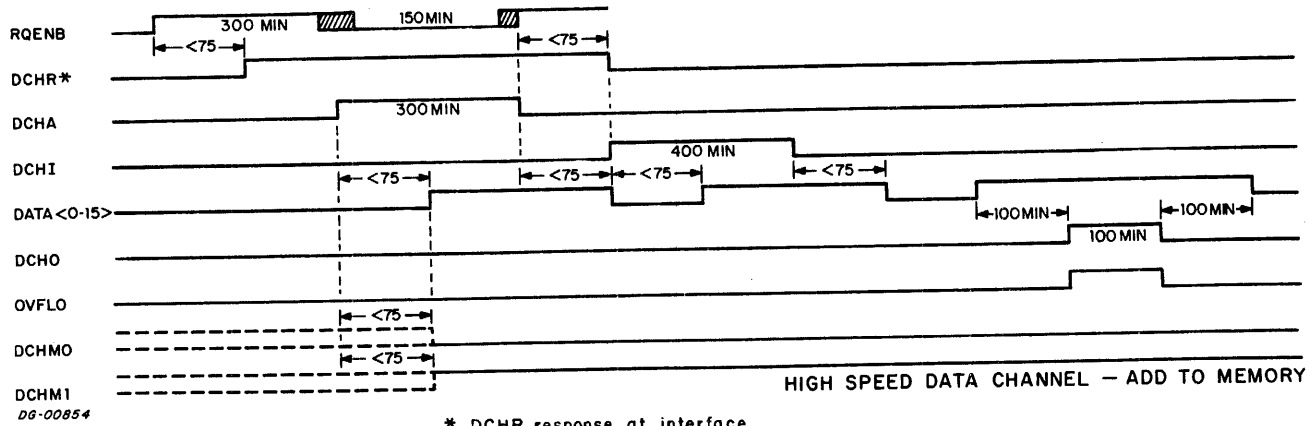
HIGH SPEED DATA CHANNEL – INPUT



HIGH SPEED DATA CHANNEL – OUTPUT



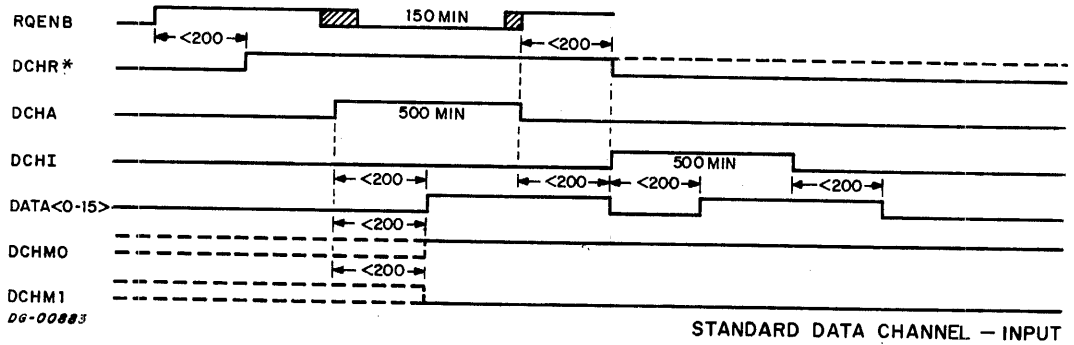
HIGH SPEED DATA CHANNEL – INCREMENT



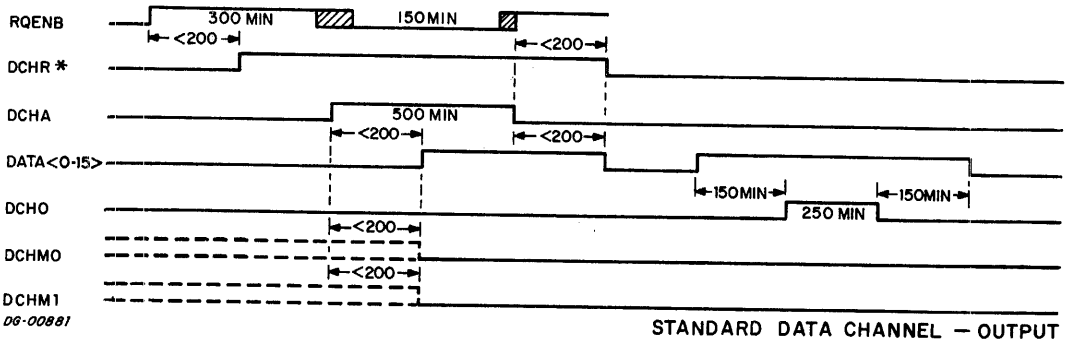
HIGH SPEED DATA CHANNEL – ADD TO MEMORY

* DCHR response at interface

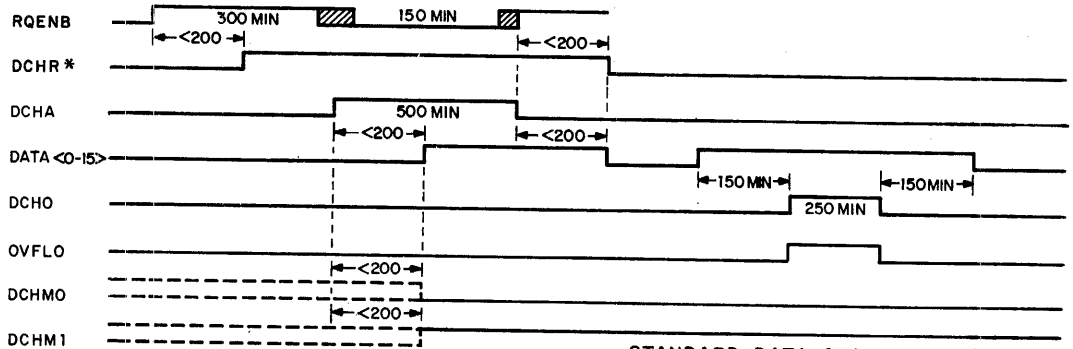
STANDARD DATA CHANNEL



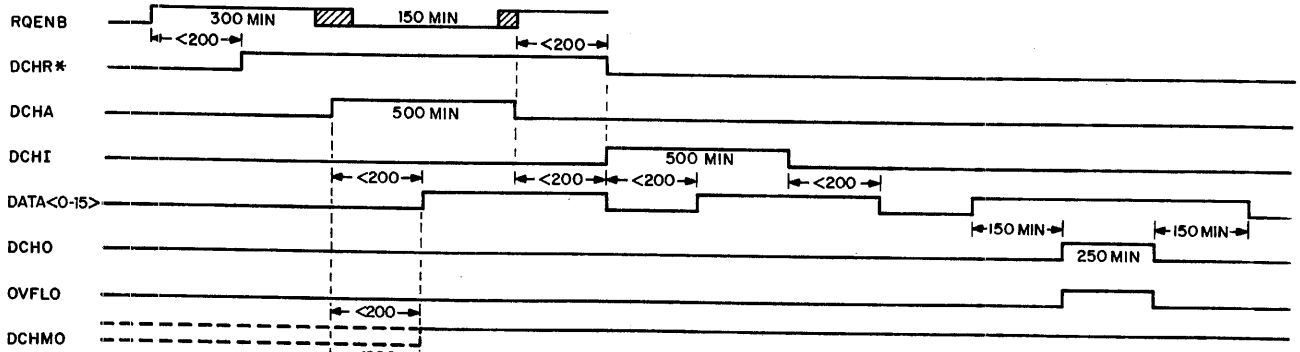
STANDARD DATA CHANNEL – INPUT



STANDARD DATA CHANNEL – OUTPUT



STANDARD DATA CHANNEL – INCREMENT

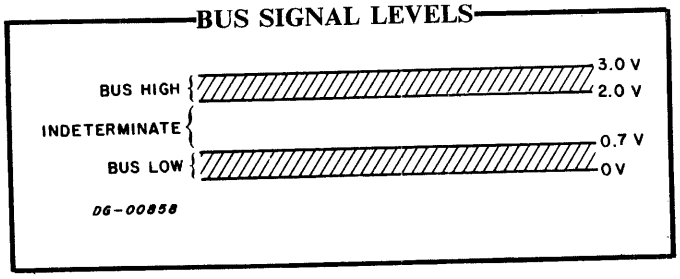


STANDARD DATA CHANNEL – ADD TO MEMORY

* DCHR response at interface

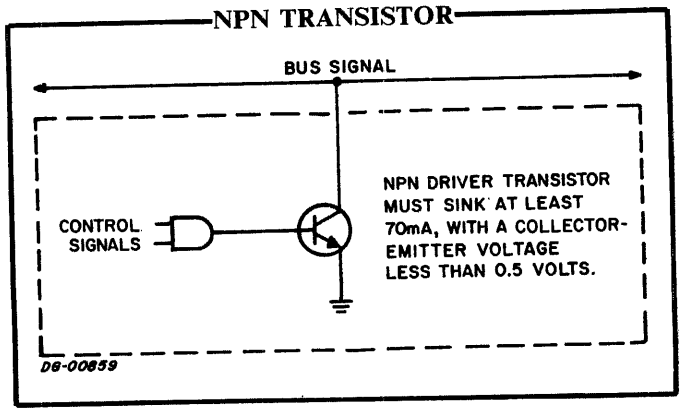
SIGNAL LEVELS

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 higher than +2.0 volts is considered high. The nominal voltages for high and low levels are 0 and +2.7 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 Section I.



DRIVERS

The $\overline{\text{DATA}}$ lines and lines from interface to processor 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 below. 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 #100000117 or #10000078 drivers are recommended for this application.



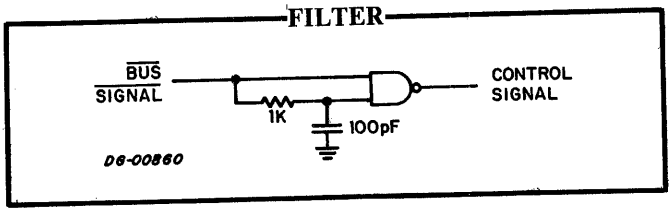
RECEIVERS

Receivers for the I/O bus signals may be either standard TTL devices or special purpose interface receivers. The net load that can be attached to the bus depends on the drivers used. The load due to all of the receivers plus the load caused by the terminations can not exceed the sink capability of any driver on the bus. Using drivers capable of sinking at least 70mA, as specified, the net load, exclusive of termination, should not exceed 16mA.

To keep the noise on the bus signals low, as few receivers as possible should be used for each signal. Thus, 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. The usual procedure is to use an inverter to buffer the I/O bus signal before it is used by any other logic.

The ac noise margin of any interface can be improved substantially by using the filter shown below for the receivers of the signals listed below:

DATIA	DATOA	DCHA	INTA	OVFLO
DATIB	DATOB	DCHI	IOPLS	RQENB
DATIC	DATOC	DCHO	IORST	STRT



SECTION V

CONNECTIONS, CONNECTORS AND TERMINATORS

INTRODUCTION

This section explains how to connect controllers in a NOVA and 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.

CONNECTIONS

A controller must be connected to the I/O bus and to the device it controls.

When the controller is mounted in a NOVA or ECLIPSE line chassis, these connections are made through the back panel. Connections to the I/O bus are made via back panel etch. Connections to the device are made in two parts: By etch or wires (called an internal cable) from the back panel to a connector mounted on the chassis and by a cable between that connector and the device.

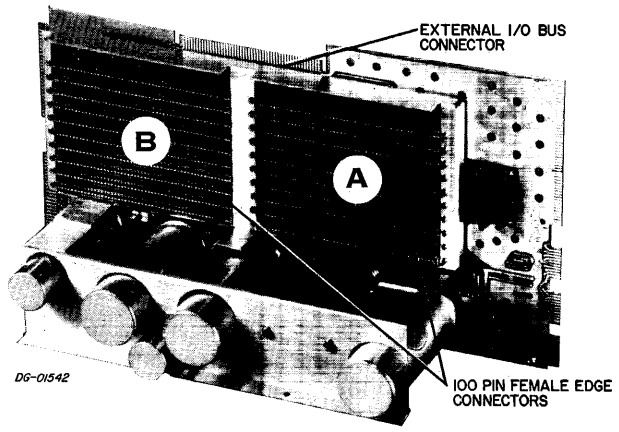
When the controller is not mounted in a NOVA or ECLIPSE line chassis, it must be connected to the I/O bus in such a chassis by an external I/O bus cable. The connector on one end of the cable is plugged into the external I/O bus connector on the chassis, and the other end of the cable is connected to the controller. The interface determines how the controller is connected to the device.

Back Panel Connections

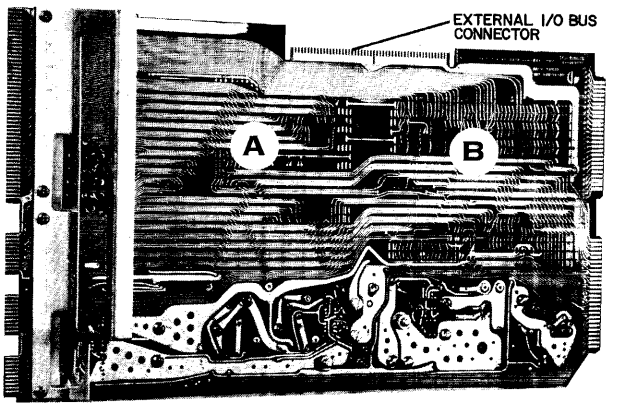
Connections are made to a printed circuit board in a NOVA or ECLIPSE line chassis via pins on the back panel. These pins are extensions of the contacts in the two 100-pin female edge connectors into which the board is inserted. The pins in each connector are arranged in two rows of fifty pins, the upper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

The back panel 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 female edge connector toward the rear of the chassis is designated connector "A"; the other female edge connector is designated connector "B". In each connector, pins are numbered from 1 to 100 starting with the pin closest to the rear of the chassis. The upper pins have odd numbers, and the lower pins have even numbers.

BACK PANEL (CONNECTOR SIDE)



BACK PANEL (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 back panel; 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 section 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 back panel. 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 Section 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. One end of the I/O bus cable is connected to the controller, and the other end is connected to an edge or socket connector on the computer chassis, called 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 back panel. Appendix C shows the assignments of I/O bus signals to pins on the back panel and to pins in the external I/O bus connector.

Cabling to an Adapter or Device

An interface 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 back panel 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 back panel pins are used to connect to the I/O bus, and which pins are available to connect to external devices or adapters.

When an internal cable must be installed for a user-designed interface, it is recommended that DGC type 4192 be used because it is a general purpose internal cable compatible with many standard Data General interfaces. This cable is supplied with the proper connector for the machine specified. The cable comprises fifty wires, each of which is tagged with a back panel pin number. When each wire is connected to the pin designated by the attached tag, this slot/connector combination will be compatible with the slot/connector pin correspondence of slot 9 of the NOVA 820, 1200 and 2/10 back panels. The correspondence between back panel pins and connector pins for this cable is shown below in the table for the General Purpose Internal Cable.

General Purpose Internal Cable

4192

Paddleboard Connector	Socket Connector	Back Panel 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

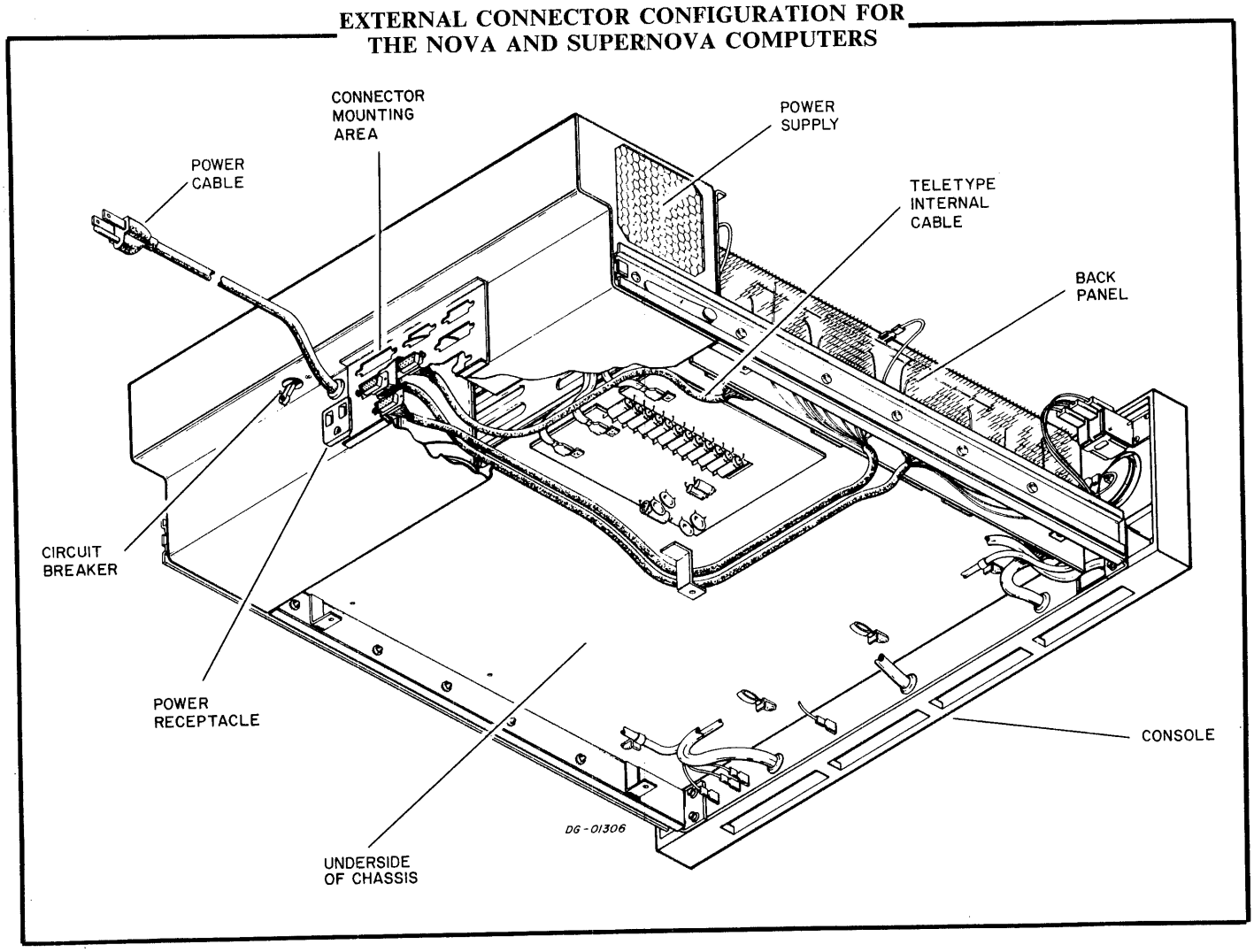
DG-00887

The following diagrams illustrate the different internal cables used in the earlier Data General computers. Paddleboard and pin connectors are used on the NOVA 820, 1210, 1220, 2/4, and 2/10. Socket connectors are used on the NOVA and SUPERNOVA computers, NOVA 800, 830, 840, 1200 Jumbo and 800 Jumbo computers. On all paddleboard connector machines, the first Teletype cable is plugged directly onto the back panel through a pin connector similar to DGC model 1051G. On all socket connector machines, the signals for the Teletype are carried through back panel etch to a socket connector. All other device cables plug onto connectors mounted at the rear of the back panel. These connectors, in turn, are wired to the appropriate back panel pins, either through internal cables or back panel etch.

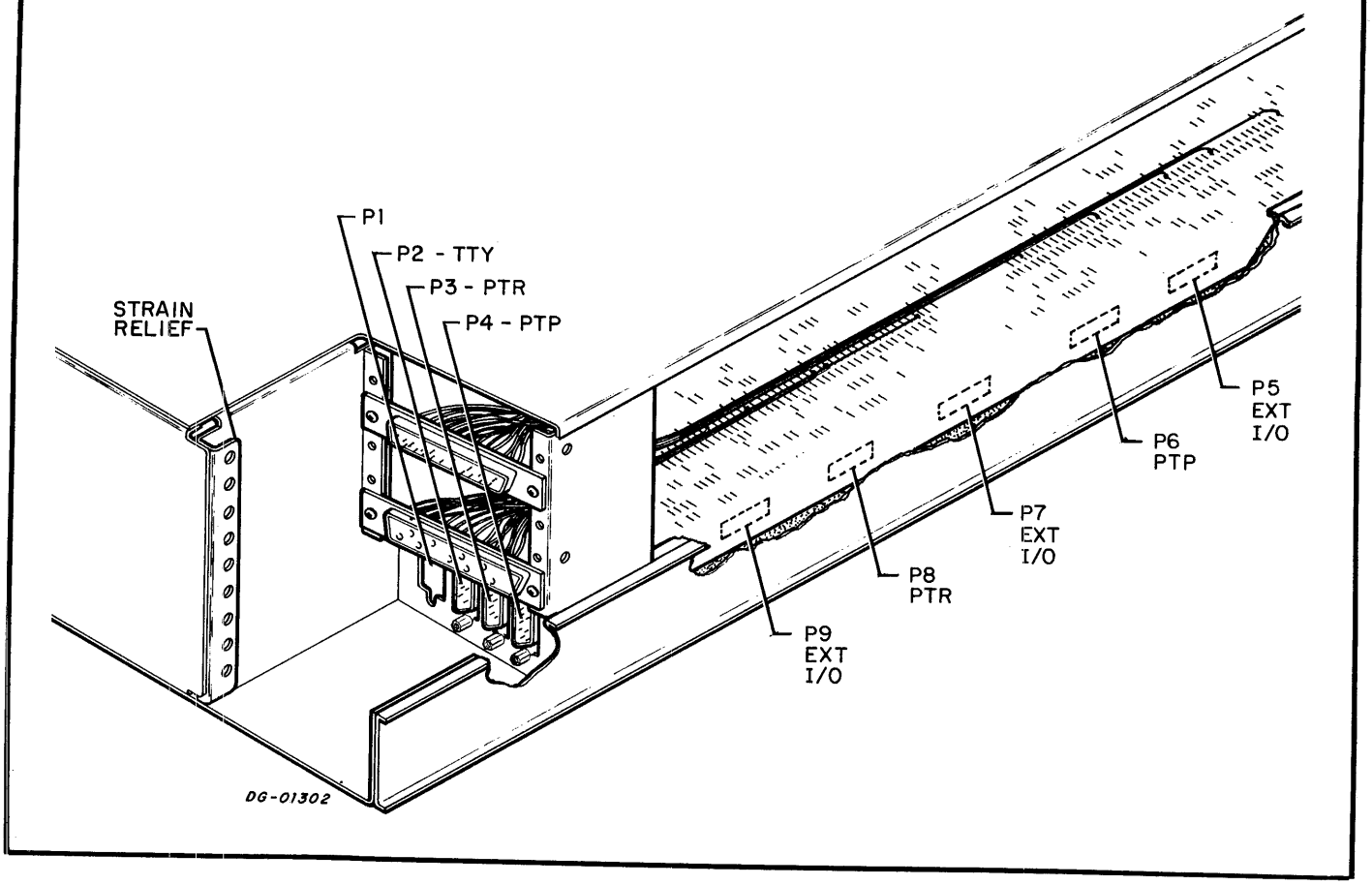
Internal cables of the later Data General computers are described in the Installer's Reference Manual DG 015-000041.

The drawing below shows the internal cable configuration for the NOVA computer; cabling for SUPERNOVA computers is quite similar. The internal cables are wire-wrapped to pins on the back panel. 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 cannon connectors for their device cables.

EXTERNAL CONNECTOR CONFIGURATION FOR THE NOVA AND SUPERNOVA COMPUTERS



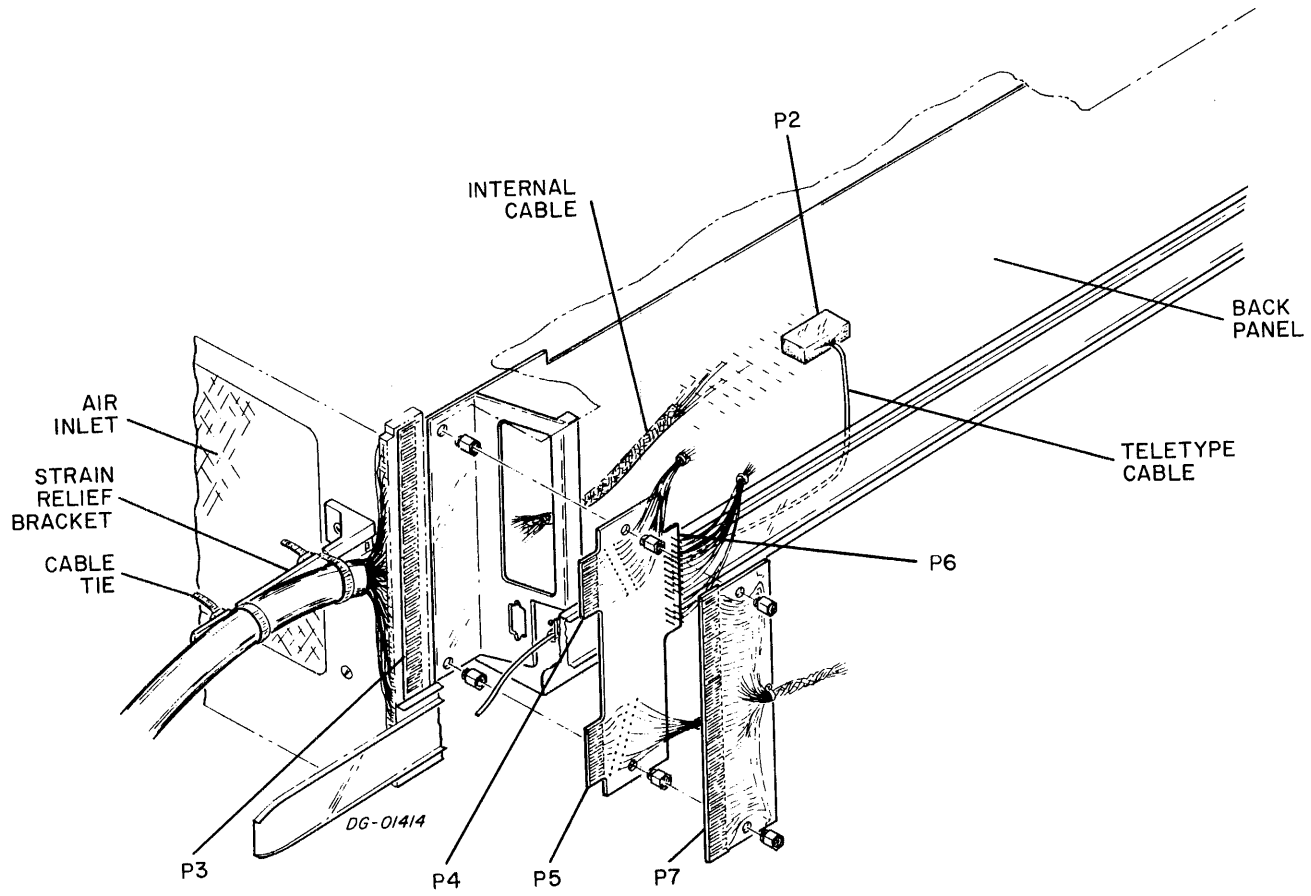
EXTERNAL CONNECTOR CONFIGURATION FOR NOVA 800, 1200, 1200 JUMBO, 830, 840 COMPUTERS AND THEIR EXPANSION CHASSIS



Socket connectors are used exclusively on these machines. Of the four holes at the bottom of the connector bracket, P2, P3, and P4 are assigned respectively to the terminal, paper tape punch, and paper tape reader. Socket P2 is connected by etch on the back panel 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 back panel. P6 and P8 are also connected by etch on the back panel 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 back panel, with an internal cable. To connect other device cables, an internal cable should be run from an external cable connector to the appropriate back panel pins.

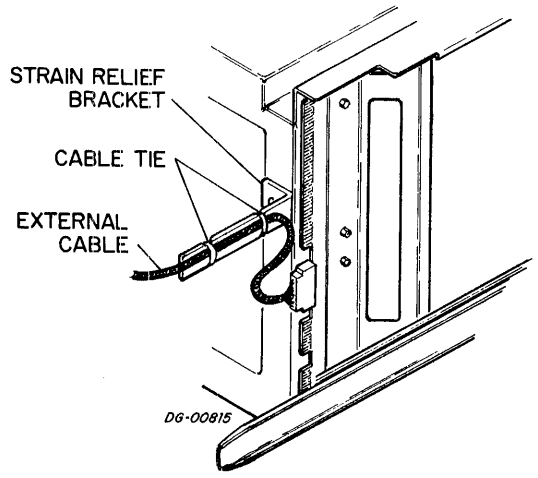
NOTE These machines use the 4050 or 4051 junction box instead of the 4083 connector panel used on paddleboard machines.

EXTERNAL CONNECTOR CONFIGURATION FOR NOVA 1210 COMPUTER

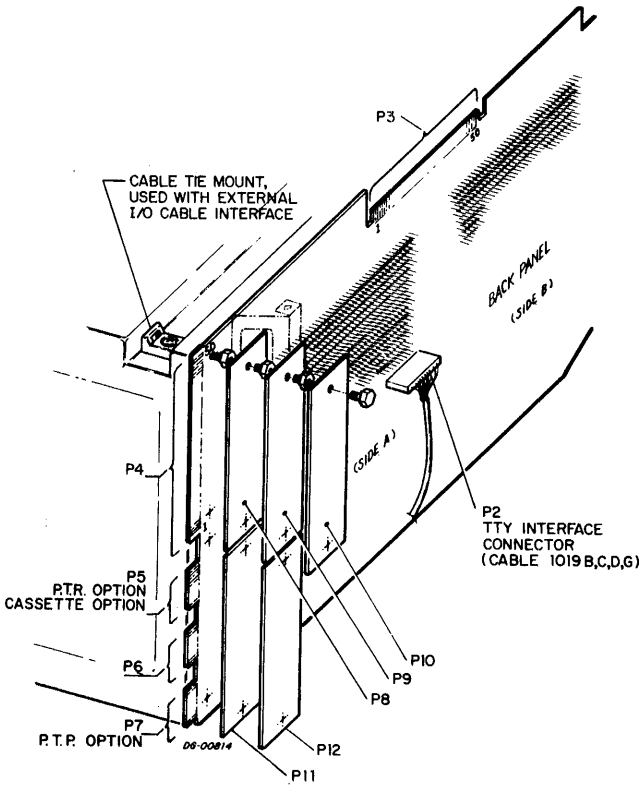


The TTY cable is plugged onto the back panel at P2, on slot 3. Paddleboard connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers on this fifty-pair connector. Device cables other than that of the TTY are plugged onto connectors mounted, as needed, at the rear of the back panel and wire-wrapped (via internal cables) to the proper back panel pins. P4 and P5, each a ten-pair paddleboard 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 paddleboard connector, is mounted on standoffs beside P4-6 and wire-wrapped to back panel pins when it is needed. The necessary connectors are furnished when DGC standard interfaces are purchased.

EXTERNAL CONNECTOR CONFIGURATION FOR NOVA 820, 1220 COMPUTERS AND EXPANSION CHASSIS



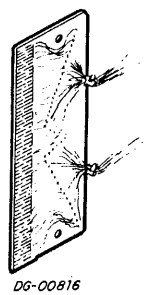
SUGGESTED STRAIN RELIEF OF EXTERNAL CABLE, AS REQUIRED



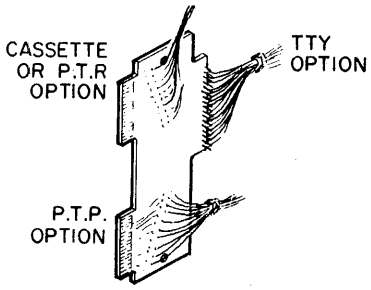
The TTY cable is plugged onto the back panel at P2, on slot 3. Paddleboard connector P3 is the external I/O bus connector. The I/O bus signals are permanently etched to the fingers of this connector. P4 is a fifty-pair paddleboard connector whose fingers are permanently connected to back panel pins of 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 back panel, and wire-wrapped (via internal cables) to the proper back panel pins. P8 through P12 may be either connector shown to the left below. If required, socket-type connectors can be installed in the spaces provided in the I/O bracket, shown to the right below. The necessary connectors are furnished when DGC standard interfaces are purchased.

ADDITIONAL PADDLEBOARDS

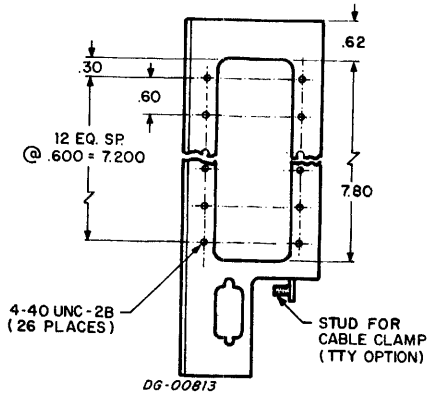


4192/CPU BOARD WITH INTERNAL CABLE 1046B-CONNECTOR ALONE

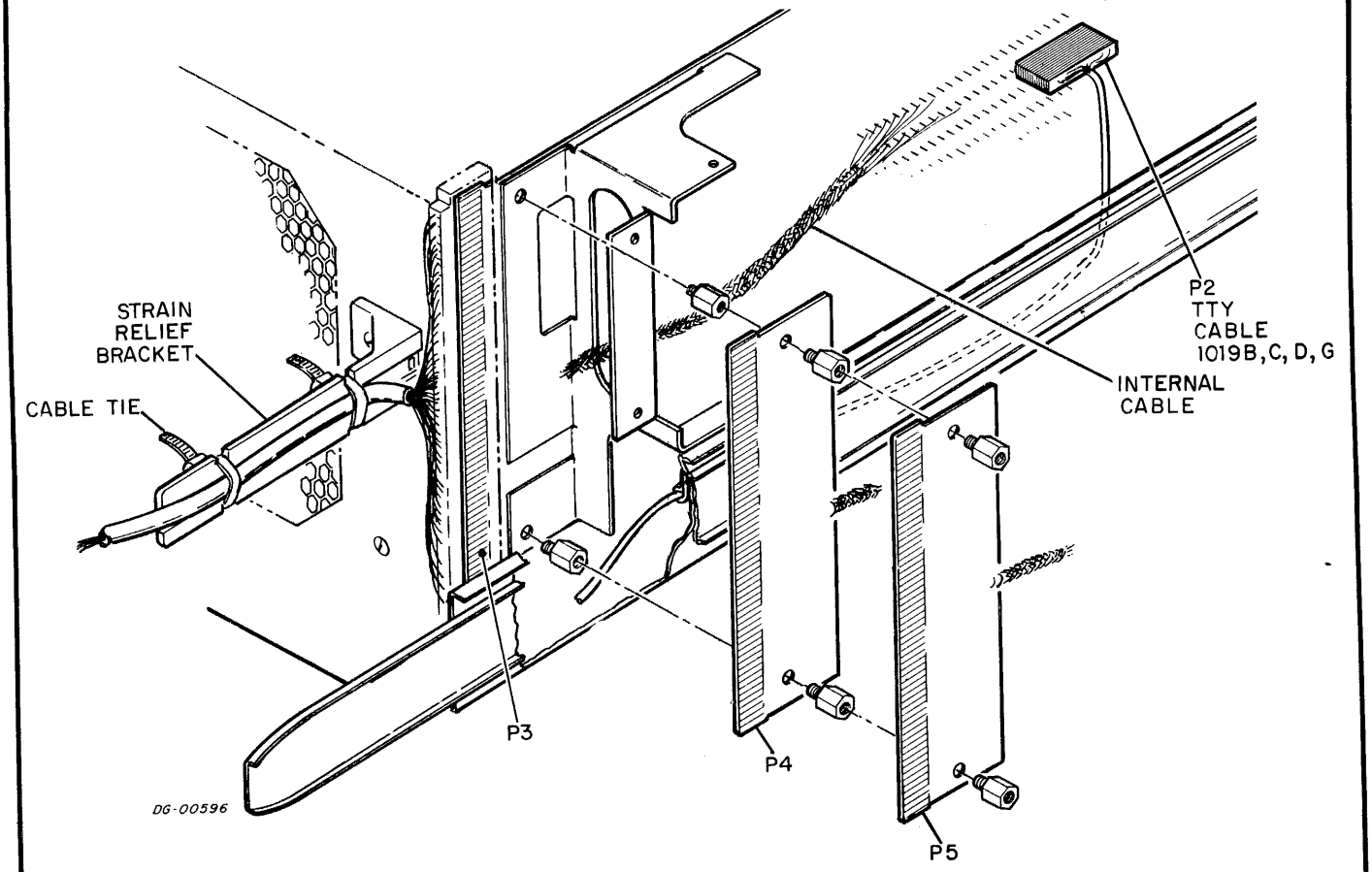


DUAL 20-PIN CONNECTOR

I/O BRACKET

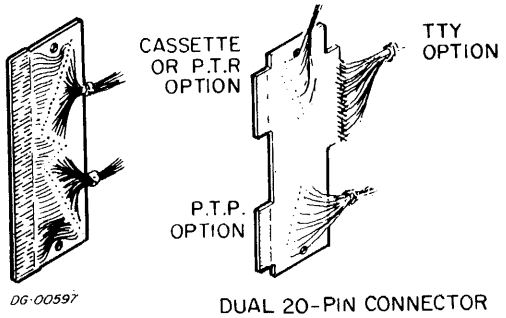


EXTERNAL CONNECTOR CONFIGURATION FOR NOVA 2/4 COMPUTERS



The TTY cable is plugged onto the back panel at P2, on slot 3. Paddleboard 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 back panel and wire-wrapped (via internal cables) to the proper back panel pins. P4 and P5 optional connectors may be either connector shown to the left below. Additionally, the 4083 connector panel may be mounted as shown to the right below. The necessary connectors are furnished when DGC standard interfaces are purchased.

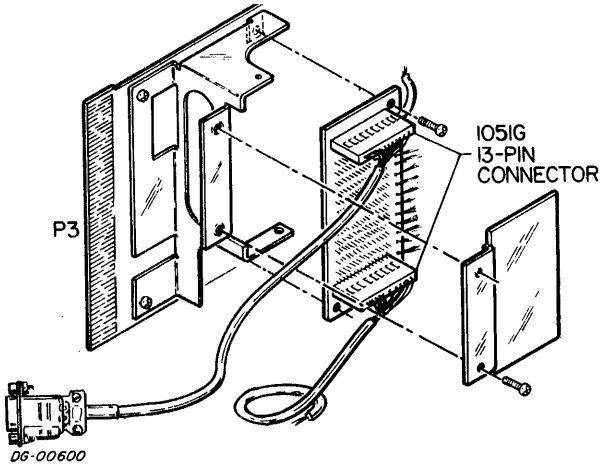
ADDITIONAL PADDLEBOARDS



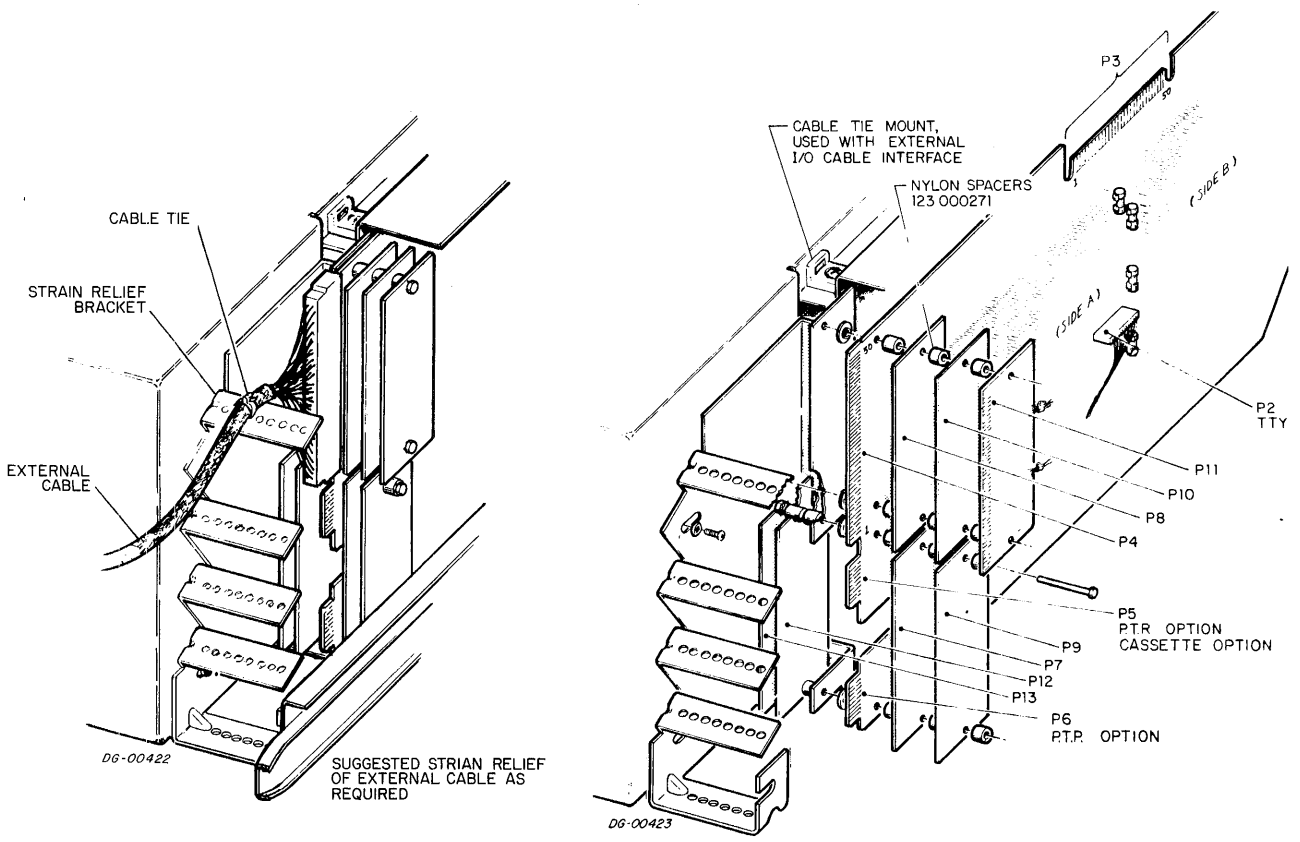
DUAL 20-PIN CONNECTOR

4192/CPU-
BOARD WITH INTERNAL
CABLE
1070B-
CONNECTOR ALONE

4083 CONNECTOR PANEL



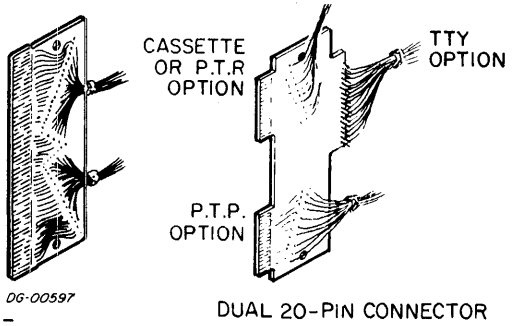
EXTERNAL CONNECTOR CONFIGURATION FOR NOVA 2/10 COMPUTER AND EXPANSION CHASSIS



The TTY cable is plugged onto the back panel at P2, on slot 3. Paddleboard 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 paddleboard connector whose fingers are permanently connected to back panel 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 back panel, and wire-wrapped (via internal cables) to the proper back panel pins. P7 through P11 may be either connector shown to the left below, while P12 and P13 must be the 4192 (fifty-pair) type. The 4083 connector panel may be mounted as shown to the right below. The necessary connectors are furnished when DGC standard interfaces are purchased.

ADDITIONAL PADDLEBOARDS

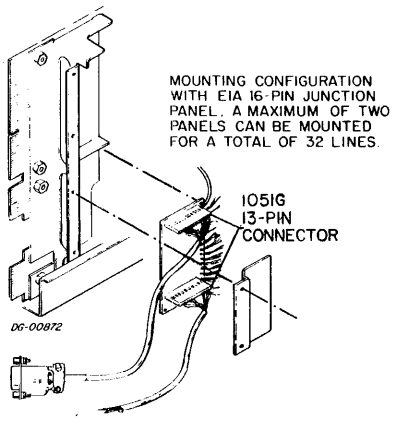


4192/CPU-BOARD WITH INTERNAL CABLE 1070B-CONNECTOR ALONE

DUAL 20-PIN CONNECTOR

NOTE: EARLIER MACHINES MAY HAVE SLIGHTLY DIFFERENT PADDLE BOARDS

4083 CONNECTOR PANEL

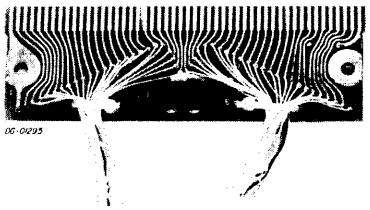


CONNECTORS

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.

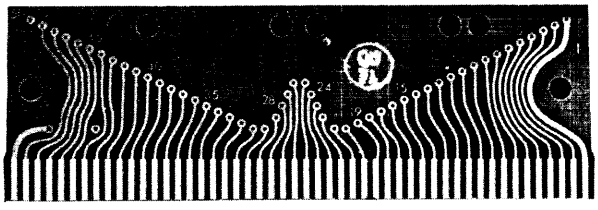
4192

This is a general purpose external device connector and includes both the internal cabling and the proper chassis-mounted cable connector. If the NOVA 1210, NOVA 820, NOVA 1220, NOVA 2/4, NOVA 2/10, or ECLIPSE line is specified, the connector is supplied with a 50-pair paddleboard connector. When any other machine is specified, the 4192 includes a 50-pin female cannon connector. The relationship between back panel pins and assigned connector pins is listed at the end of this section.



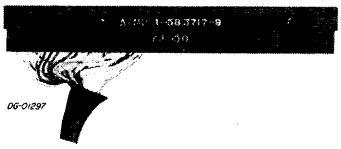
1070B

This is the 50-pair paddleboard connector used on the 4192.



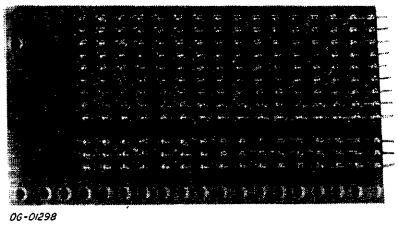
005-001858

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



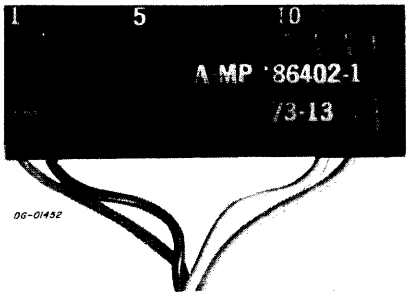
4083

This is 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.



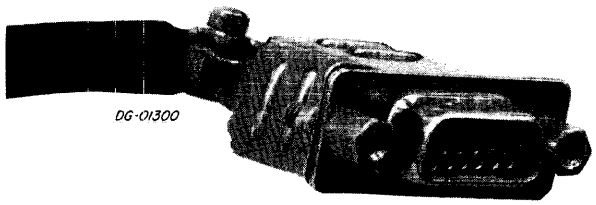
1051G

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



Socket Connectors

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

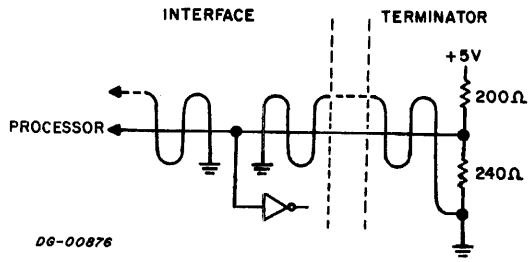


TERMINATORS

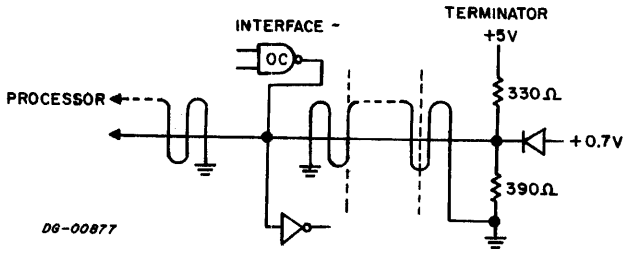
Because the I/O bus cable is being used as a high-frequency transmission line, it is very important that the cable be terminated at or near its characteristic impedance. Any mismatch between the terminator and the cable will cause electrical reflections within the cable, which manifest themselves as 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.

The diagrams below show 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 the picutres below.

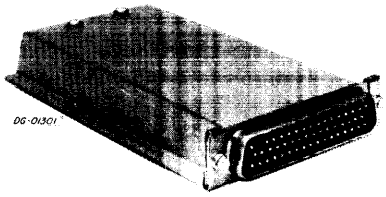
RECOMMENDED TERMINATION



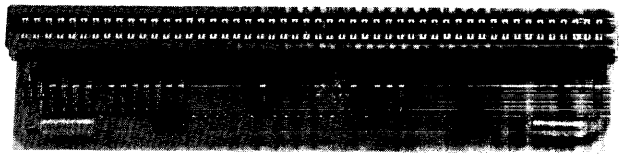
RECOMMENDED TERMINATION FOR PROCESSOR ORIGINATED I/O BUS SIGNALS: CLR, DATIA, DATIB, DATIC, DATOA, DATOB, DATOC, DCHA, DCHI, DCHO, DS<0-5>, INTA, IORST, IOPLS, MSKO, OVFLO, RQENB, STRT



RECOMMENDED TERMINATION FOR BI-DIRECTIONAL LINES AND SIGNALS FROM INTERFACE TO PROCESSOR:
DATA<0-15>, DCHM<0,1>, DCHR, SELB, SELD



SOCKET TERMINATOR



PADDLEBOARD-TYPE TERMINATOR

A terminator should always be used when the I/O bus extends beyond the computer chassis. The same terminator should also be used in a NOVA 2/10 or the ECLIPSE S/200, C/300 lines even if the I/O bus does not extend beyond the computer chassis.

Terminators can be mounted on either paddleboard or socket connectors. The table below 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.

Extended I/O Bus Terminators

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
**		

DG-01292

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

**For later Data General computers see the Installer's Reference Manual DG 015-000041.

Socket Connectors

Type Number	Connector
005-002244	100-pin female connector; consists of 111-12 and 111-22
2245	100-pin male connector; consists of 111-11, 111-21, and 111-25
2246	52-pin female connector; consists of 111-10 and 111-22
2247	52-pin male connector; consists of 111-9, 111-20, and 111-24
2248	25-pin female connector; consists of 111-4 and 111-22
2249	25-pin male connector; consists of 111-3, 111-20, and 111-24
2250	19-pin female connector; consists of 111-8 and 111-22
2251	19-pin male connector; consists of 111-7, 111-19, and 111-23
2252	9-pin female connector; consists of 111-2 and 111-22
2253	9-pin male connector; consists of 111-1, 111-19, and 111-23
2254	50-pin I/O female connector; consists of 111-6 and 111-22
2255	50-pin I/O male connector; consists of 111-5, 111-21, and 111-25
111-000001	9-pin male connector
0002	9-pin female connector
0003	25-pin male connector
0004	25-pin female connector
0005	50-pin male connector
0006	50-pin female connector
0007	19-pin male connector
0008	19-pin female connector
0009	52-pin male connector
0010	52-pin female connector
0011	100-pin male connector
0012	100-pin female connector
0019	9/19 pin junction shell
0020	25/52 pin junction shell
0021	50/100 pin junction shell
0022	Screw lock assembly, female
0023	9/19 pin screw lock assembly, male
0024	25/52 pin screw lock assembly, male
0025	50/100 pin screw lock assembly, male

DG-00888

This page intentionally left blank.

SECTION VI

INTERFACE BOARDS

INTRODUCTION

This section 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

The illustrations on the following pages 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.

DC Power Requirements

Interfaces installed in a computer chassis or expansion chassis receive power (+5Vdc 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 section 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 +5Vdc, +15Vdc and -5Vdc. Many computer chassis also produce -15Vdc for customer use. The +5Vdc is used for the logic components; the +15Vdc is used for core memory; the -5Vdc is used for the Teletype interface. The price list contains the +5Vdc 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. In general, no more than 2A should be drawn off the +15Vdc and -5Vdc supplies by the customer.

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 (e.g., DGC #116-000003).

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 here in the hope that the designer will remain aware of the possible restrictions.

The principle problem involved with localized heating is caused by high dissipation devices concentrated together creating hot spots. Ever computer and expansion chassis includes fans for forced air movement over the boards. Sufficient air moves over the heat-producing components to keep the air temperature rise within reason, thereby limiting part temperature. 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, particularly that directly above the interface in question. Care should be taken that the profile of the board is low enough so as not to interfere with the air flow over 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.

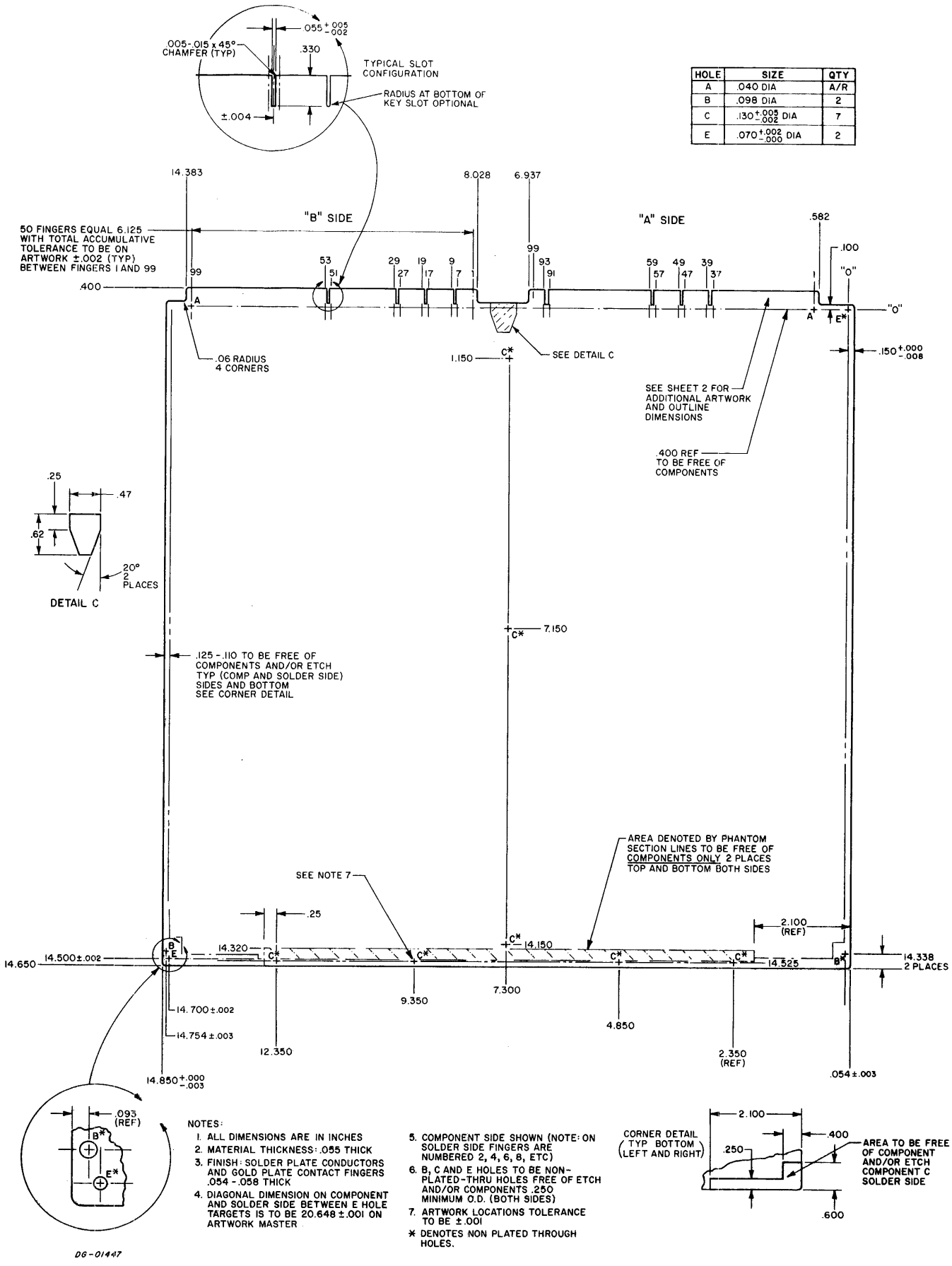
Power Supply and Back Panel Engineering Drawings

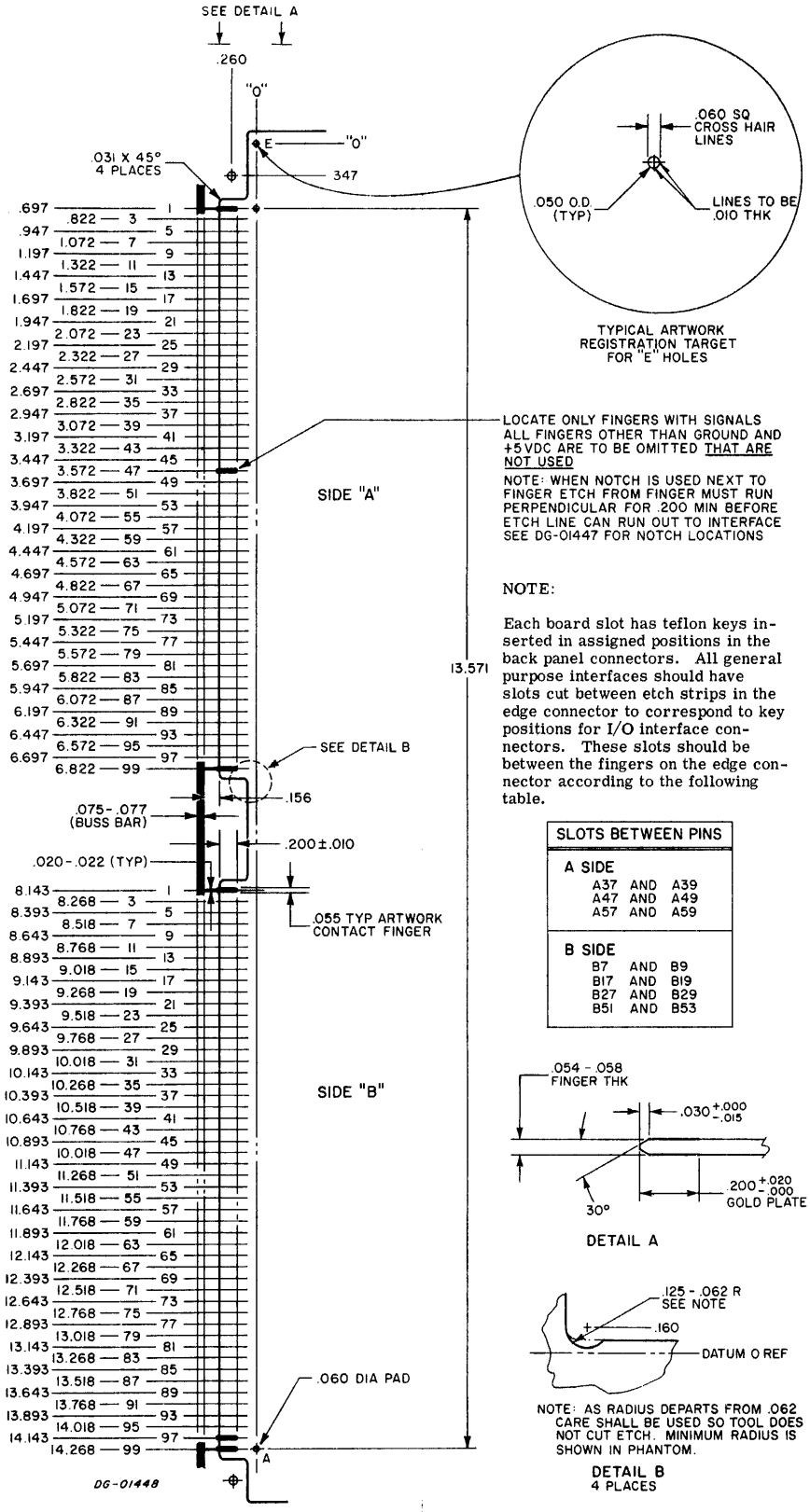
COMPUTER CHASSIS	Engineering Drawings	
	Power Supply	Back Panel
NOVA Computer	001-000063	001-000024
SUPERNOVA Computer	001-000063	001-000046
1200 Series:		
1200	001-000551	001-000090
1200 Jumbo	001-000551, 553	001-000169
1210	001-000660	001-000207
1220	001-000173	001-000208
800 Series:		
800	001-000551	001-000094
800 Jumbo	001-000551, 553	001-000169
820	001-000172	001-000209
830	001-000551, 553	001-000729
840	001-000551, 553	001-000538
2 Series:		
2/4	001-000530	001-000560, 645
2/10	001-000473	001-000566
3 Series		
3/4	001-000839	001-000839
3D and 3/12	001-000959	001-000853
S/230 and C330	001-000669	001-000679
S/130	001-001073	001-001062
ECLIPSE Computer Series:		
S/100	001-000617	001-000670
S/200 and C/300	001-000669	001-000679

EXPANSION CHASSIS		Engineering Drawings	
Size	Series	Power Supply	Back Panel
	NOVA Computer	001-000145	-
	SUPERNOVA Computer	001-000145	-
	SC Memory	001-000205	-
7-slot	1200, 800 and 830	001-000149	-
10-slot	1220, 820	001-000215	001-000230
17-slot	840 only	001-000553	001-000554
10-slot	2/10	001-000719	001-000718
12-slot	3/12 3D S/130	001-001150	001-001058
16-slot	S/230 or C/330 Memory Only	001-000680	001-000710
16-slot	S/200,S/230, C300 Series I/O Only	001-000680	001-000901

DG-01289

PRINTED CIRCUIT BOARD 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.

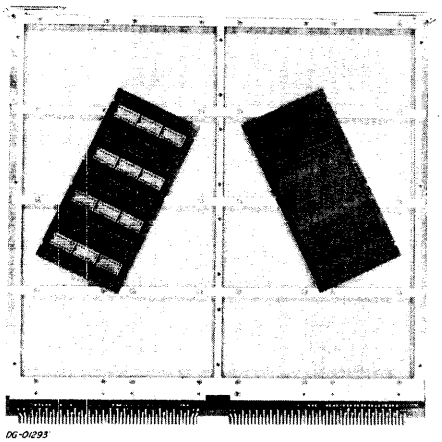
DETAIL B
4 PLACES

PREFABRICATED INTERFACE BOARDS

To aid the designer in the construction of custom interface assemblies, Data General makes available three 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. To simplify the job somewhat, the 4040 series general purpose interface boards include most of the fundamental interface logic discussed earlier in this manual. However, ample room is still available for custom circuitry, making these boards particularly useful for breadboard and prototype applications.

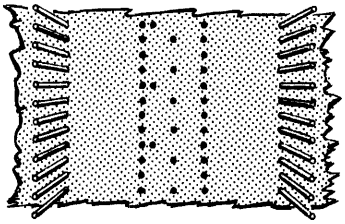
1000 Series General Purpose Wiring Boards

The 1000 wiring board 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.



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 the illustration. 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.

1003 WIRING BOARD



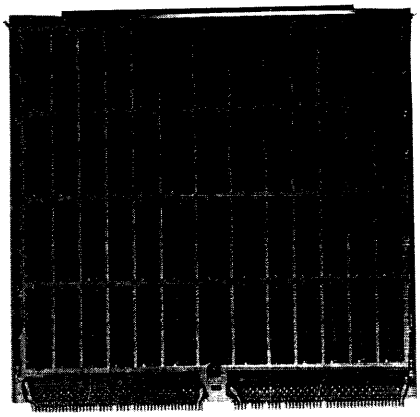
Close-up view of hole pattern

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

1020 Series General Purpose Wiring Boards

The 1021 wiring board 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.



DG-01294

4040 Series General Purpose Interfaces

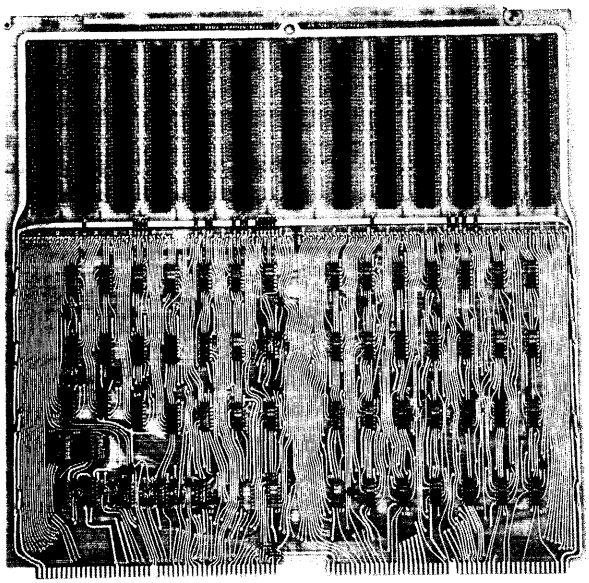
Since the I/O bus is the same on all Data General computers, there must be a certain amount of circuitry common to all of the interfaces. This, as described earlier, consists of the logic that actually functions with the I/O bus signals. The general purpose interface boards (4040 series) include this basic circuitry as well as space for extensive custom logic. Thus these boards eliminate a good deal of the design and construction effort necessary to implement a custom interface.

4040 General Purpose Interface Board

The 4040 general purpose interface board is a standard 15-inch square printed circuit board with two 100-pin edge connectors along one side, making the board compatible with the NOVA line back panel. The board includes the basic interface control logic necessary for programmed I/O operation. This includes the device selection network, busy/done logic, bus receivers and drivers, and the interrupt logic.

For custom logic, low-profile sockets are provided for up to sixty-five 14- or 16-pin integrated circuits. Printed circuit etch connects each socket terminal to a wire-wrap post for interconnecting the integrated circuit chips. A row of 201 wire-wrap pins physically divides the custom logic from the standard interface logic. The pins are connected by etch to various signals in the interface logic and to uncommitted back panel edge connections. Connections between the integrated circuit chips and between these chips and the standard interface logic are made by wire-wrapping, using #30 AWG teflon-coated solid wire, to the appropriate posts.

The drawings on the following two pages show the logic included on this board. Back panel connections are shown by circles and wire-wrap pins by squares. The accompanying table lists the signals available to the custom logic and the characteristics of these signals.



DG-01314

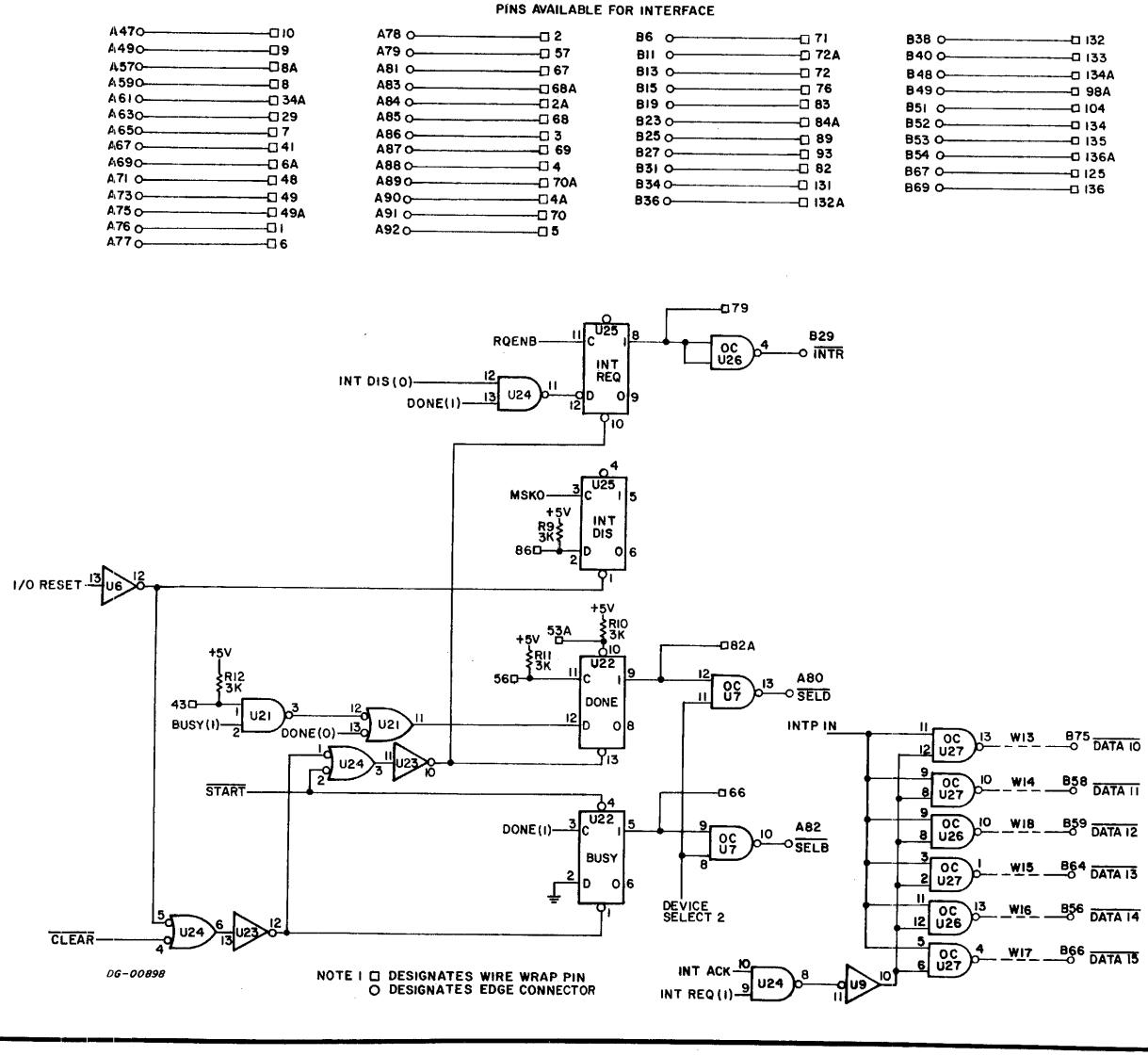
4041 Data Register Option

The 4041 Data Register Option is additional logic which can be included on the 4040. It consists, as shown in the appropriate drawings, of two 16-bit registers used as data buffers between the interface and the I/O bus. The input (to the processor) register receives parallel or serial data from the custom logic and transfers data in parallel on the I/O bus. The output register receives parallel data from the I/O bus and transfers data to the custom logic either in parallel or serially. These registers can be used for either programmed I/O or data channel transfers. The signals available to the custom logic from these registers are listed in the table following the drawings of this circuitry.

4042 Data Channel Connection Option

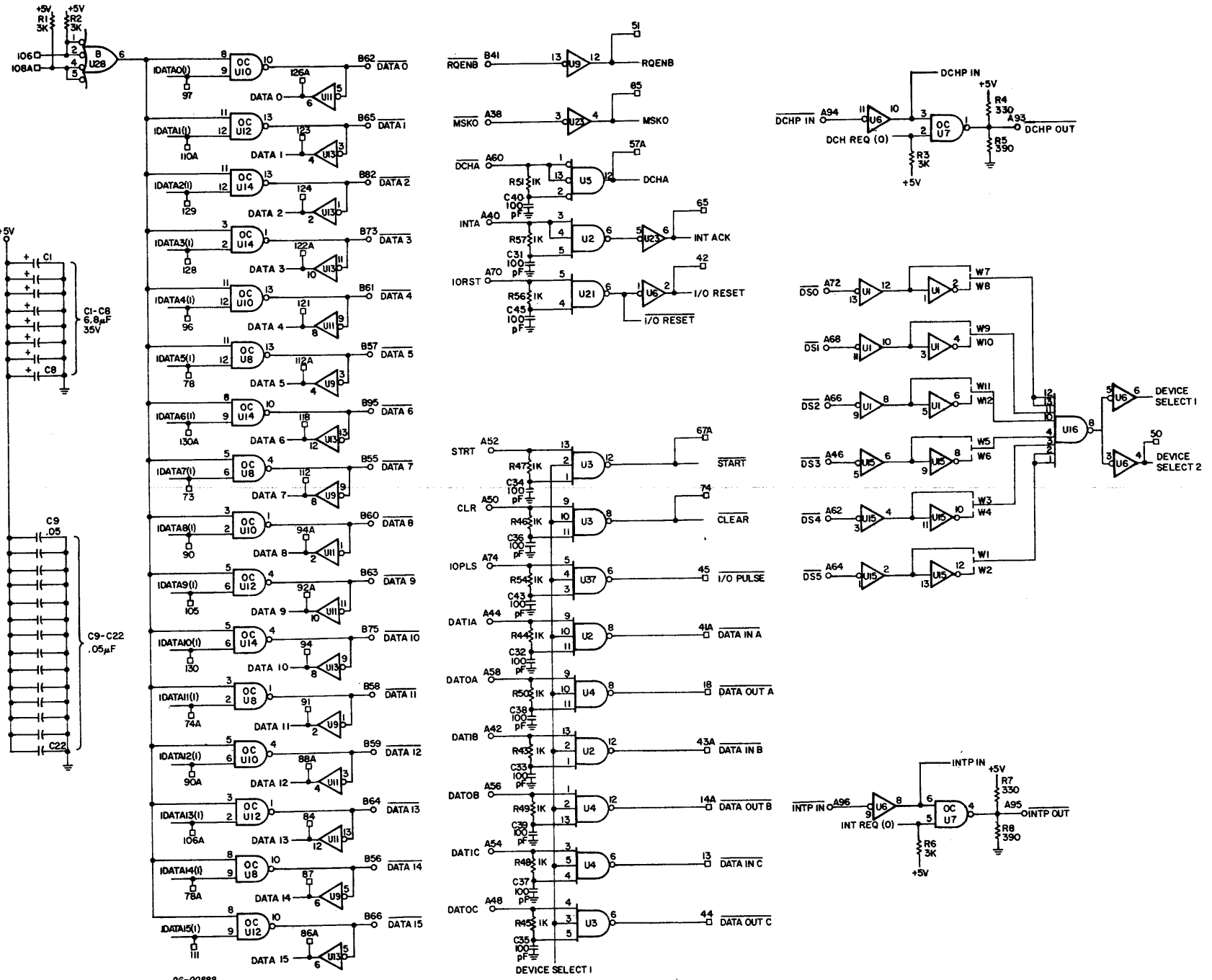
When included on the 4040, the data channel connection logic provides the basic control for all data transfers via the data channel. This logic includes the control logic discussed earlier as well as two 16-bit counters. One counter maintains the word count, or the number of words still to be processed, and the other counter maintains the current address, which is the memory location for the next word transferred. This logic is shown on the appropriate drawings, while the accompanying table lists the signals available to the custom logic.

GENERAL PURPOSE INTERFACE - 4040 BUSY/DONE NETWORK INTERRUPT CONTROL

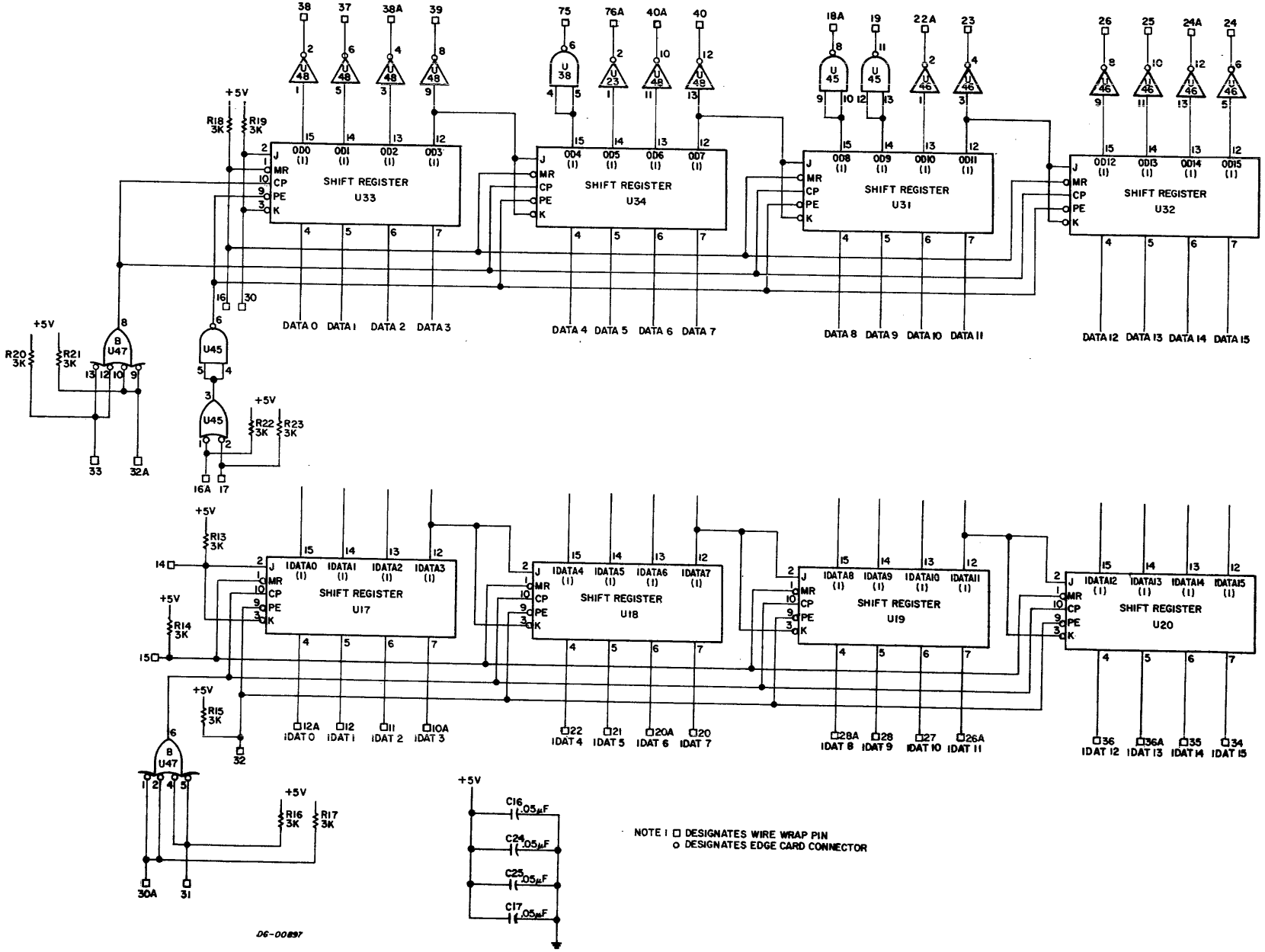


4043 and 4044 Options

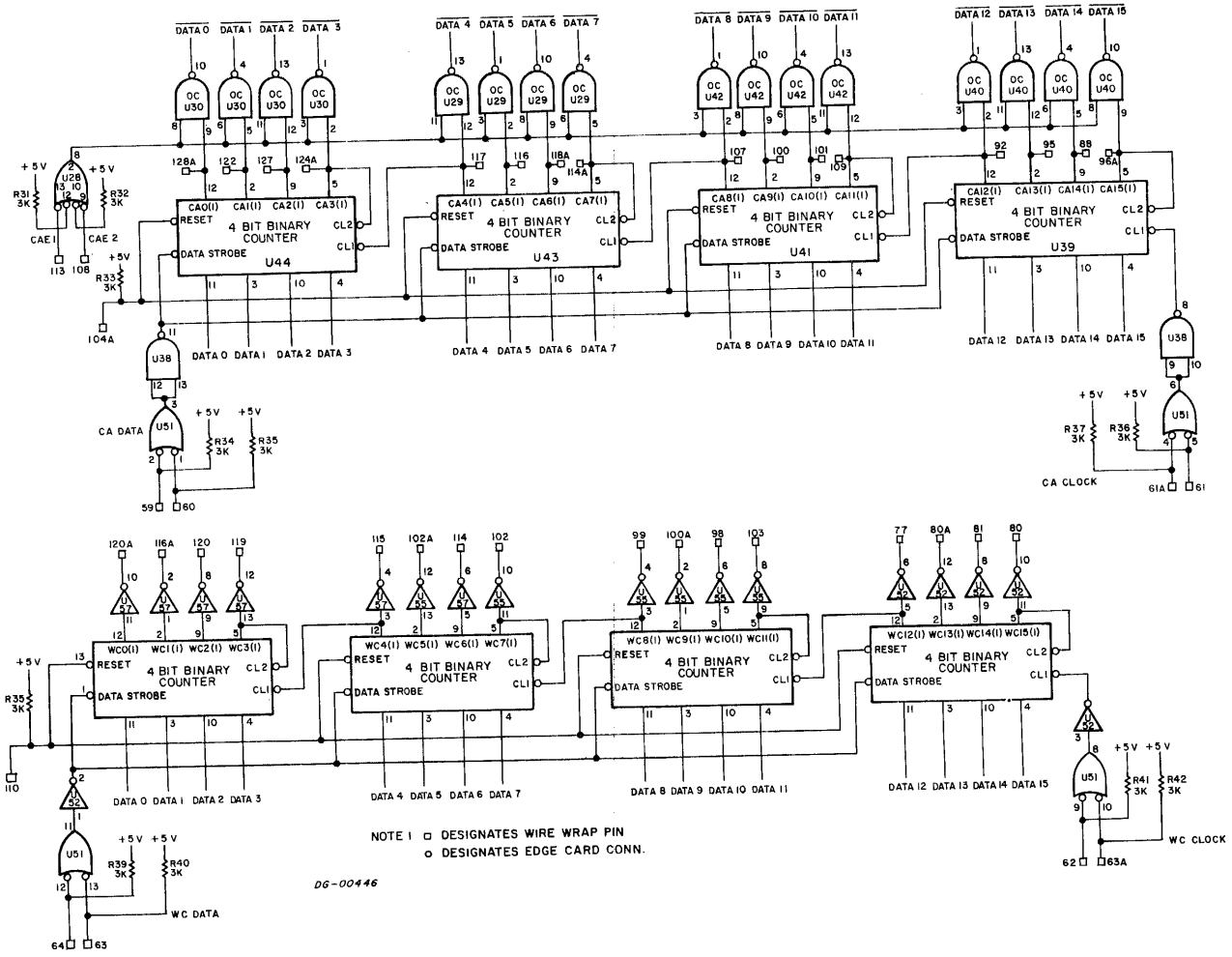
The 4043 adds wire-wrap pins to each logic interconnection pad in the customer logic section of the 4040, and the 4044 adds 65-16 pin low profile sockets for dual in-line IC's, to the 4043.


+5V
R1 3K
R2 3K
106
108A
B U28
IDATA0(I)
97
OC U10
126A
DATA 0
B62 DATA 0
IDATA1(I)
110A
OC U12
123
DATA 1
B65 DATA 1
IDATA2(I)
129
OC U14
124
DATA 2
B82 DATA 2
IDATA3(I)
128
OC U14
122A
DATA 3
B73 DATA 3
IDATA4(I)
96
OC U10
121
DATA 4
B61 DATA 4
IDATA5(I)
78
OC U8
112A
DATA 5
B57 DATA 5
IDATA6(I)
130A
OC U14
118
DATA 6
B95 DATA 6
IDATA7(I)
73
OC U8
112
DATA 7
B55 DATA 7
IDATA8(I)
90
OC U10
94A
DATA 8
B60 DATA 8
IDATA9(I)
105
OC U12
92A
DATA 9
B63 DATA 9
IDATA10(I)
130
OC U14
94
DATA 10
B75 DATA 10
IDATA11(I)
74A
OC U8
91
DATA 11
B58 DATA 11
IDATA12(I)
90A
OC U10
88A
DATA 12
B59 DATA 12
IDATA13(I)
106A
OC U12
84
DATA 13
B64 DATA 13
IDATA14(I)
78A
OC U8
87
DATA 14
B56 DATA 14
IDATA15(I)
111
OC U12
86A
DATA 15
B66 DATA 15
DG-00899
+5V
C1
C1-C8 6.8µF 35V
C8
C9 .05
C9-C22 .05µF
C22
RQENB B41
U9
51
RQENB
MSKO A38
U23
85
MSKO
DCHA A60
R51 1K
C40 100 pF
U5
57A
DCHA
INTA A40
R57 1K
C31 100 pF
U2
U23
65
INT ACK
IORST A70
R56 1K
C45 100 pF
U21
U6
42
I/O RESET
I/O RESET
STRT A52
R47 1K
C34 100 pF
U3
67A
START
CLR A50
R46 1K
C36 100 pF
74
CLEAR
IOPLS A74
R54 1K
C43 100 pF
U37
45
I/O PULSE
DATIA A44
R44 1K
C32 100 pF
U2
41A
DATA IN A
DATOA A58
R50 1K
C38 100 pF
U4
18
DATA OUT A
DATIB A42
R43 1K
C33 100 pF
U2
43A
DATA IN B
DATOB A56
R49 1K
C39 100 pF
U4
14A
DATA OUT B
DATIC A54
R48 1K
C37 100 pF
U4
13
DATA IN C
DATOC A48
R45 1K
C35 100 pF
U3
44
DATA OUT C
DEVICE SELECT I
DCHP IN
+5V
DCHP IN A94
U6
OC U7
R4 330
A93
DCHP OUT
DCH REQ (O)
R3 3K
R5 390
+5V
DS0 A72
DSI A68
DS2 A66
DS3 A46
DS4 A62
DS5 A64
U1
U15
W7
W8
W9
W10
W11
W12
W5
W6
W3
W4
W1
W2
U16
U6
DEVICE SELECT 1
50
DEVICE SELECT 2
INTP IN
+5V
INTP IN A96
U6
OC U7
R7 330
A95
INTP OUT
INT REQ (O)
R6 3K
R8 390
+5V

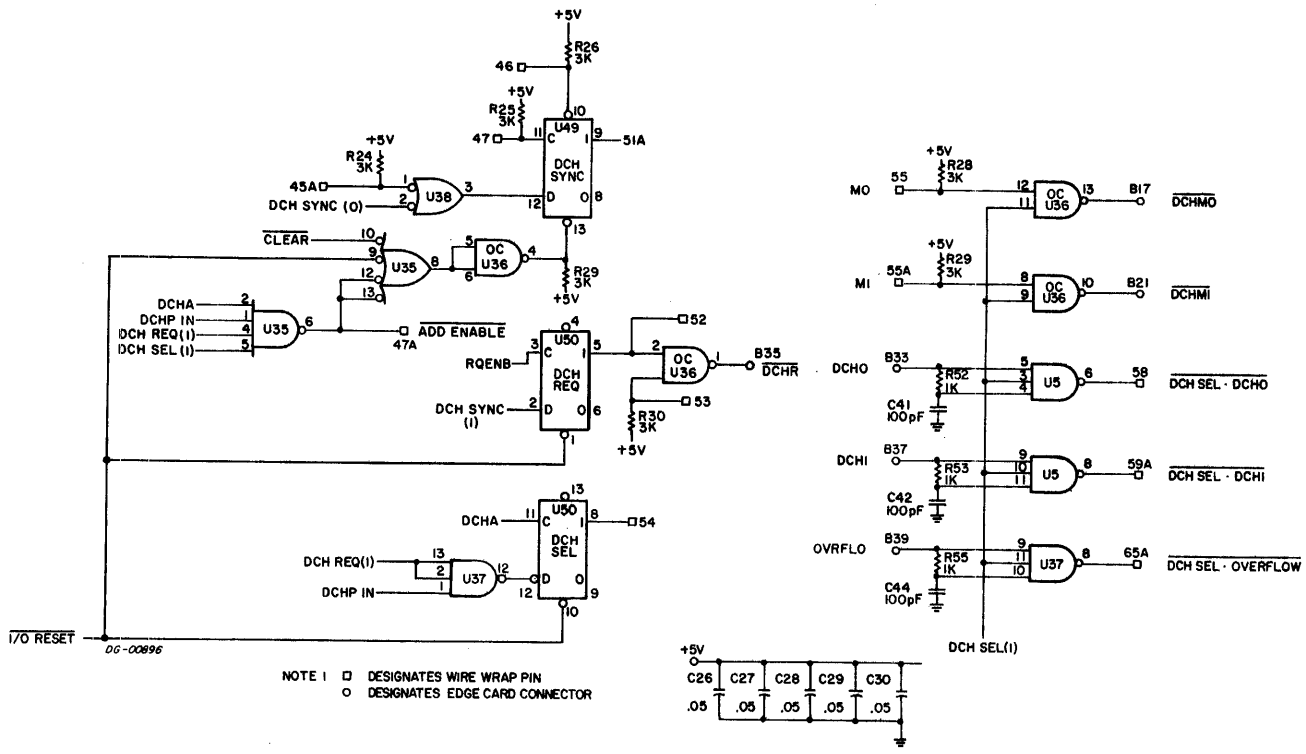
GENERAL PURPOSE INTERFACE - 4041 INPUT/OUTPUT DATA REGISTERS



GENERAL PURPOSE INTERFACE
4042 DATA CHANNEL OPTION ADDRESS REGISTER AND WORD COUNTER



GENERAL PURPOSE INTERFACE - 4042 DATA CHANNEL CONTROL



SIGNAL NAME - PIN ASSIGNMENTS ON THE 4040, 4041 AND 4042 GENERAL PURPOSE I/O INTERFACES

BASIC INTERFACE 4040

Signal	Pin	Fanout	Load
DATA0-DATA15	See below	10*	
IDATA0(1)-IDATA15(1)	See below		1
OB1	106		4
OB2	108A		4
RQENB	51	6	
MSKO	85	8	
DCHA	57A	7	
INT ACK	65	9	
IO RESET	42	5	
$\overline{\text{START}}$	67A	7	
$\overline{\text{CLEAR}}$	74	8	
$\overline{\text{IO PULSE}}$	45	10	
$\overline{\text{DATA IN A}}$	41A	10	
$\overline{\text{DATA OUT A}}$	18	10	
$\overline{\text{DATA IN B}}$	43A	10	
$\overline{\text{DATA OUT B}}$	14A	10	
$\overline{\text{DATA IN C}}$	13	10	
$\overline{\text{DATA OUT C}}$	44	10	
DEVICE SELECT-2	50	8	
INT REQ(1)	79	8	
INT DIS, D terminal	86		1
DONE, set terminal	53A		2
DONE, clock terminal (C)	56		2
DONE(1)	82A	9	
DONE, D terminal gated internally	43		1
BUSY(1)	66	9	

DATA0-DATA15 Pins

DATA-	Pin	DATA-	Pin
0	126A	8	94A
1	123	9	92A
2	124	10	94
3	122A	11	91
4	121	12	88A
5	112A	13	84
6	118	14	87
7	113	15	86A

IDATA0(1)-IDATA15(1) Pins

IDATA-	Pin	IDATA-	Pin
0	97	8	90
1	110A	9	105
2	129	10	130
3	128	11	74A
4	96	12	90A
5	78	13	106A
6	130A	14	78A
7	73	15	111

*Without 4041 option (see Data Registers 4041).

DG-00877

DATA REGISTERS 4041

Signal	Pin	Fanout	Load
Input register			
ICP1	30A		4
ICP2	31		4
IPE	32		12
IMR	15		4
IJ-K	14		2
IDAT0-IDAT15	See below		1
IDATA0(1)-IDATA15(1)	See below	5	
Output register			
OCP1	32A		4
OCP2	33		4
OPE1	16A		1
OPE2	17		1
OMR	16		4
OJ-K	30		2
$\overline{\text{OD0(1)-OD15(1)}}$	See below	10	
DATA0-DATA15	See below	9*	

IDAT0-IDAT15 Pins

IDAT-	Pin	IDAT-	Pin
0	12A	8	28A
1	12	9	28
2	11	10	27
3	10A	11	26A
4	22	12	36
5	21	13	36A
6	20A	14	35
7	20	15	34

IDATA0(1)-IDATA15(1) Pins

IDATA-	Pin	IDATA-	Pin
0	97	8	90
1	110A	9	105
2	129	10	130
3	128	11	74A
4	96	12	90A
5	78	13	106A
6	130A	14	78A
7	73	15	111

OD0(1)-OD15(1) Pins

OD-	Pin	OD-	Pin
0	38	8	18A
1	37	9	19
2	38A	10	22A
3	39	11	23
4	75	12	26
5	76A	13	25
6	40A	14	24A
7	40	15	24

DATA0-DATA15 Pins

DATA-	Pin	DATA-	Pin
0	126A	8	94A
1	123	9	92A
2	124	10	94
3	122A	11	91
4	121	12	88A
5	112A	13	84
6	118	14	87
7	113	15	86A

*Reflects additional load on data line receivers due to register.

DG-00878

SIGNAL NAME - PIN ASSIGNMENTS ON THE 4040, 4041 AND 4042 GENERAL PURPOSE I/O INTERFACES

DATA CHANNEL LOGIC 4042

Signal	Pin	Fanout	Load
DCH SYNC(1)	51A	10	
DCH SYNC, set terminal	46		2
DCH SYNC, C terminal	47		2
DCH SYNC, D terminal gated internally	45A		1
$\overline{\text{ADD ENABLE}}$	47A	9	
DCH REQ(1)	52	9	
DCH SEL(1)	64	5	
$\overline{\text{DCHO}}$	58	10	
$\overline{\text{DCHI}}$	59A	10	
$\overline{\text{OVERFLOW}}$	65A	10	
CAE1	113		4
CAE2	108		4
CA RESET	104A		4
CA DATA 1	59		1
CA DATA 2	60		1
CA CLOCK 1	61A		1
CA CLOCK 2	61		1
CA0(1)-CA15(1)	See below	3	
WC RESET	110		4
WC DATA 1	63		1
WC DATA 2	64		1
WC CLOCK 1	62		1
WC CLOCK 2	63A		1
WC0(1)-WC15(1)	See below	3	

CA0(1)-CA15(1) Pins

CA-	Pin	CA-	Pin
0	128A	8	107
1	122	9	100
2	127	10	101
3	124A	11	109
4	117	12	92
5	116	13	95
6	118A	14	88
7	114A	15	96A

WC0(1)-WC15(1) Pins

WC-	Pin	WC-	Pin
0	120A	8	99
1	116A	9	100A
2	120	10	98
3	119	11	103
4	115	12	77
5	102A	13	80A
6	114	14	81
7	102	15	80

DG-00879

REFERENCE PRINTS

001-000051 001-000052 001-000053	Logic Diagrams
016-000170 016-000171 016-000172 016-000173	IPL's

APPENDIX A

I/O DEVICE CODES AND DATA GENERAL MNEMONICS

Device Code (Octal)	Mnemonic	Priority Mask Bit	Device
00	--	--	Power fail
01●	WCS	--	Writeable control store
02●	ERCC	--	Error checking and correction
03●	MAP	--	Memory Allocation and Protection
01○	MDV	--	Multiply/Divide
02○	MMPU	--	Memory Management and Protection Unit
02*○	MAP0	--	Memory Allocation and Protection
03○	MAP1	--	
04○	MAP2	--	
05			
06	MCAT	12	Multiprocessor adapter transmitter
07	MCAR	12	Multiprocessor adapter receiver
10	TTI	14	Teletype input
11	TTO	15	Teletype output
12	PTR	11	Paper tape reader
13	PTP	13	Paper tape punch
14	RTC	13	Real time clock option
15	PLT	12	Incremental plotter
16	CDR	10	Card reader
17	LPT	12	Line printer
20	DSK	9	Fixed head disc
21	ADCV	8	A/D converter
22	MTA	10	Magnetic tape
23	DACV	--	D/A converter
24	DCM	0	Data communications multiplexor
25			
26			
27			
30	QTY	14	Asynchronous hardware multiplexor
31*	IBM1	13	IBM 360/370 interface
32	IBM2		
33	DKP	7	Moving head disc
34	CAS	10	Cassette tape
34*	MX1	11	Multiline asynchronous controller
35	MX2		
36	IPB	6	Interprocessor bus--half-duplex
37	IVT	6	IPB watchdog timer
40	DPI	8	IPB full-duplex input
41	DPO	8	IPB full-duplex output
40+	SCR	8	Synchronous communication receiver
41·	SCT	8	Synchronous communication transmitter

DG-01450

● ECLIPSE computer only

○ NOVA line computers only

* code returned by INTA and used by VCT for ECLIPSE computer

+ may be set up with any unused even device code 40 or greater

· may be set up with any unused odd device code 41 or greater

APPENDIX A (Continued)

I/O DEVICE CODES AND DATA GENERAL MNEMONICS

Device Code (Octal)	Mnemonic	Priority Mask Bit	Device
42	DIO	7	Digital I/O
43	DIOT	6	Digital I/O timer
44	MXM	12	Modem control for multiline asynchronous controller
45			
46	MCAT1	12	Second multiprocessor transmitter
47	MCAR1	12	Second multiprocessor receiver
50	TTI1	14	Second teletype input
51	TTO1	15	Second teletype output
52	PTR1	11	Second paper tape reader
53	PTP1	13	Second paper tape punch
54	RTC1	13	Second real time clock option
55	PLT1	12	Second incremental plotter
56	CDR1	10	Second card reader
57	LPT1	12	Second line printer
60	DSK1	9	Second fixed head disc
61	ADCV1	8	A/D converter
62	MTA1	10	Second magnetic tape
63	DACV1	--	D/A converter
64*^{O}	FPU1		
65^{O}	FPU2	5	Alternate location for floating point
66^{O}	FPU4		
67			
70	QTY1	14	Second asynchronous hardware multiplexor
70	SLA1	14	Second synchronous line adapter
71* } 72 }		13	Second IBM 360/370 interface
73	DKP1	7	Second moving head disc
74	CAS1	10	Second cassette tape
74* } 75 }		11	Second multiline asynchronous controller
74*^{O}	FPU1 }		
75^{O}	FPU2 }	5	Floating point
76^{O}	FPU		
77	CPU	--	Central processor and console functions

DG-01450

*code returned by INTA and used by VCT for ECLIPSE computer

^{O}NOVA line computers only

APPENDIX B
MAXIMUM LATENCY TIMES *

Computer Model	Standard Data Channel		High-Speed Data Channel		Interrupt	
	With MUL/DIV	Without MUL/DIV	With MUL/DIV	Without MUL/DIV	With MUL/DIV	Without MUL/DIV
NOVA Computer	17.3	17.3	N/A	N/A	12.0	12.0
SUPERNOVA Computer	11.8	7.8	5.7	3.7	9.0	5.0
SUPERNOVA SC	11.8	7.8	5.7	3.7	9.0	5.0
NOVA 1200 Series	9.4	9.4	N/A	N/A	7.0	7.0
NOVA 800, 820, 840	5.8	5.8	4.8	3.2	10.6	4.6
NOVA 830	6.4	6.4	5.4	3.6	12.0	6.0
NOVA 2, 8K	5.2	5.2	4.3	4.3	5.8	1.9
NOVA 2, 16K	5.3	5.3	4.4	4.4	5.9	2.3
ECLIPSE Computer	4.3	4.3	N/A	N/A	36.2	36.2
	All times are in microseconds.					

DG-01304

*For highest priority peripheral.

APPENDIX B (Continued)
MAXIMUM DATA CHANNEL TRANSFER RATES

Computer Model	Standard		High Speed		Increment Memory		Add to Memory	
	In	Out	In	Out	Standard	High Speed	Standard	High Speed
NOVA Computer	285,500	227,500	-	-	227,500	-	187,500	-
SUPERNOVA Computer	434,700	357,100	1,250,000	1,000,000	357,100	833,333	357,100	833,333
SUPERNOVA SC	434,700	357,100	1,250,000	1,000,000	357,100	833,333	357,100	833,333
NOVA 1200 Series	833,333	555,555	-	-	416,666	-	-	-
NOVA 800, 820, 840	500,000	500,000	1,250,000	1,000,000	454,545	833,333	-	-
NOVA 830	454,545	454,545	1,000,000	833,333	416,666	833,333	-	-
NOVA 2, 8K	500,000	475,000	1,250,000	833,333	454,545	770,000	-	-
NOVA 2, 16K	475,000	454,545	1,110,000	770,000	435,000	715,000	-	-
ECLIPSE Computer	1,250,000	714,000	-	-	-	-	-	-
	All rates are in words/second.							

DG-01309

APPENDIX C
EXTERNAL I/O BUS CONNECTOR WIRE LIST

Signal	Source	Panel Pin	External Connector Pin		Function
			@1	@2	
$\overline{\text{DATA0}}$	B	B62	ω	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 is should place data on the $\overline{\text{DATA}}$ 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 $\overline{\text{DATA}}$ 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 selected interface.
$\overline{\text{SELD}}$	D	A80	Z	47	
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 $\overline{\text{INTR}}$, $\overline{\text{DCHR}}$
$\overline{\text{MSKO}}$	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 Chain
$\overline{\text{INTP OUT}}$	*	A95	c	39	
INTA	P**	A40	D	38	Interrupt Acknowledge
$\overline{\text{DCHR}}$	D	B35	j	31	Data Channel Request

DG-00890

Continued on following page

*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 C (Continued)
EXTERNAL I/O BUS CONNECTOR WIRE LIST

Signal	Source	Panel Pin	External Connector Pin		Function
			@1	@2	
$\overline{\text{DCHP IN}}$	*	A94			Data Channel Priority Chain
$\overline{\text{DCHP OUT}}$	*	A93	b	30	
$\overline{\text{DCHA}}$	P**	A60	R	25	Data Channel Acknowledge
$\overline{\text{DCHM0}}$	D	B17	d	27	Data Channel Transfer Mode
$\overline{\text{DCHM1}}$	D	B21	e	28	
DCHI	P**	B37	k	26	Signals interface that data is on $\overline{\text{DATA}}$ lines.
DCHO	P**	B33	h	29	Signals interface to place data on $\overline{\text{DATA}}$ lines.
OVFLO	P**	B39	ℓ	44	Increment or Add to Memory result $\leq 2^{16}$
IORST	P	A70	W	42	System reset
PWR ON	P		B	49	+5V for remote turn-on

DG-00890

*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

CRITICAL I/O BUS TIMING

TIMING CONSTRAINTS ON PRIORITY CHAINS

The architecture of the program interrupt and data channel priority systems in the NOVA and ECLIPSE line computer systems imposes two constraints on the design of I/O interfaces that operate in these systems. First, interrupt and data channel requests may be issued only on the leading edge of the request enable signal (RQENB), and second, the propagation delay through either priority chain may not exceed 300 nanoseconds.

These constraints eliminate the possibility of a race condition between the interrupt acknowledge signal (INTA) and the interrupt priority chain signals (INTP IN and INTP OUT in each controller.) The race condition may arise if there is not a sufficient delay between the time when a controller makes an interrupt request and the time when the processor asserts INTA. If the delay is not long enough, a controller making a lower priority request may spuriously place its device code on the DATA lines in response to INTA before news of the higher priority request arrives via the interrupt priority chain. Whenever any interface makes an interrupt request, there must be time for all interfaces with lower interrupt priorities to determine that a higher priority request has been made before the processor asserts INTA. Therefore, the delay between an interrupt request and the assertion of INTA must always be longer than the maximum time required for an interface to determine that a higher priority interrupt request has been made, that is, longer than the maximum time for the news of the higher priority interrupt request to ripple out the interrupt priority chain.

The NOVA and ECLIPSE line processors never assert INTA less than 300 nanoseconds after they assert RQENB. Therefore, if interfaces only make interrupt requests on the leading edge of RQENB and the maximum propagation delay in the interrupt priority chain is less than 300 nanoseconds, there will always be enough time for news of an interrupt request to ripple out the interrupt priority chain to the last controller in the chain.

These constraints also eliminate the possibility of a race condition between the data channel acknowledge signal (DCHA) and the data channel priority chain signals (DCHP IN and DCHP OUT in each controller). The race condition may arise if there is not a sufficient delay between the time when a controller makes a data channel request and the time when the processor asserts DCHA. If the delay is not long enough, a controller making a lower priority data channel request may place a memory address on the DATA lines in response to DCHA before news of the higher priority request arrives via the data channel priority chain.

In the NOVA and ECLIPSE line processors, there is a minimum delay of 300 nanoseconds between the leading edge of RQENB and the leading edge of DCHA. If interfaces only make data channel requests on the leading edge of RQENB and the maximum propagation delay in the data channel priority chain is less than 300 nanoseconds, there will always be enough time for news of a data channel request to ripple out the interrupt priority chain to the last controller in the chain.

TIMING CONSTRAINTS ON DATA CHANNEL TRANSFER MODE SIGNALS

The architecture of the data channel transfer mode circuitry in the NOVA and ECLIPSE line computer systems imposes two constraints on interfaces that operate in these systems. First, an interface may only cease to assert the data channel transfer mode signals ($\overline{DCHM<0\text{-}1>}$) on the leading edge of the data channel acknowledge signal ($\overline{DCHA}$), and second, when the interface ceases to assert $\overline{DCHM<0\text{-}1>}$, the trailing edge of those signals must arrive at the processor before the subsequent trailing edge of $\overline{DCHA}$ leaves the processor.

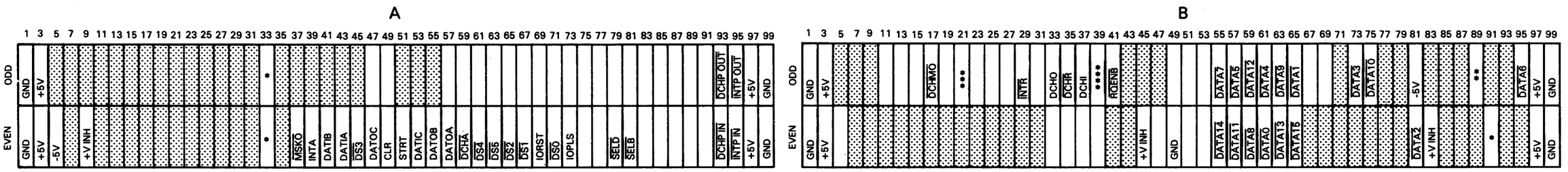
The first constraint insures that the data channel transfer mode can be sensed by the processor on $\overline{DCHM<0\text{-}1>}$ at any time during a transfer.

The second constraint eliminates the possibility of one interface asserting $\overline{DCHM<0\text{-}1>}$ while another interface is receiving data channel service.

These two constraints together imply that NOVA line computer systems must not include both interfaces that request data channel service on an extended I/O bus and interfaces in the main chassis that request high-speed data channel service. The reason is that the amount of time available for an interface to retract $\overline{DCHM<0\text{-}1>}$ depends on the length of the I/O bus between the interface and the processor. It also depends on the duration of $\overline{DCHA}$, which is determined by the interface that is receiving data channel service, not by the interface that has already been serviced and must now retract $\overline{DCHM<0\text{-}1>}$. When an interface receives high-speed data channel service, $\overline{DCHA}$ may be asserted by the processor for as little as 300 nanoseconds. If the last interface to be serviced was on the extended I/O bus and asserted $\overline{DCHM<0\text{-}1>}$, it might not be able to retract those signals soon enough to avoid interfering with the interface currently being serviced.

APPENDIX E

BACK PANEL CONNECTOR LAYOUT



* Memory pins on original NOVA computer, but ground on all subsequent computers in the NOVA line.
** Ground on NOVA 2 computer.
*** DCHM1 is not available on the ECLIPSE line.
**** OVFLO is not available on the ECLIPSE line.

DG-00869

This illustration shows the location of the various I/O bus signals and the power and ground lines on the back panel. These signals are available on all slots not reserved for central processor boards. The unshaded locations with no signal designation are available to the interface for signals to external devices, via an internal cable. The shaded locations 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 these shaded locations, 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.

CUT ALONG DOTTED LINE

READERS COMMENT FORM

DOCUMENT TITLE: .....

Your comments, accompanied by answers to the following questions, help us improve the quality and usefulness of our publications. If your answer to a question is "no" or requires qualification, please explain.

How did you use this publication?

( ) As an introduction to the subject.
( ) As an aid for advanced knowledge.
( ) For information about operating procedures.
( ) To instruct in a class.
( ) As a student in a class.
( ) As a reference manual.
( ) Other.....

Did you find the material:

- Useful.....YES ( ) NO ( )
- Complete.....YES ( ) NO ( )
- Accurate.....YES ( ) NO ( )
- Well organized.....YES ( ) NO ( )
- Well written.....YES ( ) NO ( )
- Well illustrated.....YES ( ) NO ( )
- Well indexed.....YES ( ) NO ( )
- Easy to read.....YES ( ) NO ( )
- Easy to understand.....YES ( ) NO ( )

We would appreciate any other comments; please label each comment as an addition, deletion, change, or error and reference page numbers where applicable.

COMMENTS

PAGE	COL	PARA	LINE	FROM	TO

From:

NAME..... TITLE.....
FIRM..... DIV. .....
ADDRESS.....
CITY..... STATE..... ZIP.....
TELEPHONE..... DATE.....

Data General Corporation
ENGINEERING PUBLICATIONS
COMMENT FORM
DG-00935

FOLD DOWN FIRST FOLD DOWN

FIRST CLASS
PERMIT NO. 26
SOUTHBORO
MASS. 01772

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

Postage will be paid by:

DataGeneral
Southboro, Massachusetts 01772

ATTENTION: Engineering Publications

FOLD UP SECOND FOLD UP

STAPLE