

**DataGeneral**

**How To Use The  
Nova  
Computers**

015-000009-09

# How To Use The Nova Computers

015-000009-09

A System Reference Manual for the computers

Nova

Supernova

Nova 1200 Series

Nova 800 Series

Supernova SC

PRICE \$10.00

DIRECT COMMENTS CONCERNING THIS MANUAL TO

DATA GENERAL CORPORATION • SOUTHBORO, MASSACHUSETTS

Ordering No. 015-000009  
©Data General Corporation 1970, 1971, 1972, 1974  
All Rights Reserved.  
Printed in the United States of America  
Rev. 09, October 1974

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel, Licensee's, 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.

NOVA, SUPERNOVA and NOVADISC are registered trademarks of Data General Corporation, Southboro, Mass.

ECLIPSE is a trademark of Data General Corporation, Southboro, Mass.

Written for Data General Corporation by William English

Printed in the United States of America

# Preface

This manual explains the programming and operation of the Nova line computers: the Nova 1200 series, the Nova 800 series, the Supernova computer, and the Nova computer. Chapter I discusses the general characteristics of these computers, the formats of the instructions they use, and the peripherals and software available with them. Chapter II describes the central processor in detail, and the remaining chapters describe the various types of peripheral in-out equipment. The appendices provide information on interfacing and installation and contain a number of reference tables.

Basically this reference manual defines in detail how the central processor or a peripheral device functions, exactly what its instructions do, how it handles data, and what its control and status information means. The programming is given in machine language. Although the basic instruction and device mnemonics defined by the assembler are used, the manual is completely self-contained—no knowledge of the assembler or any other Data General Corporation software is required. The first three sections of Chapter II, which treat the three classes of Nova line instructions, also give the conventions needed to program the computers and understand the examples given in the text.

# Contents

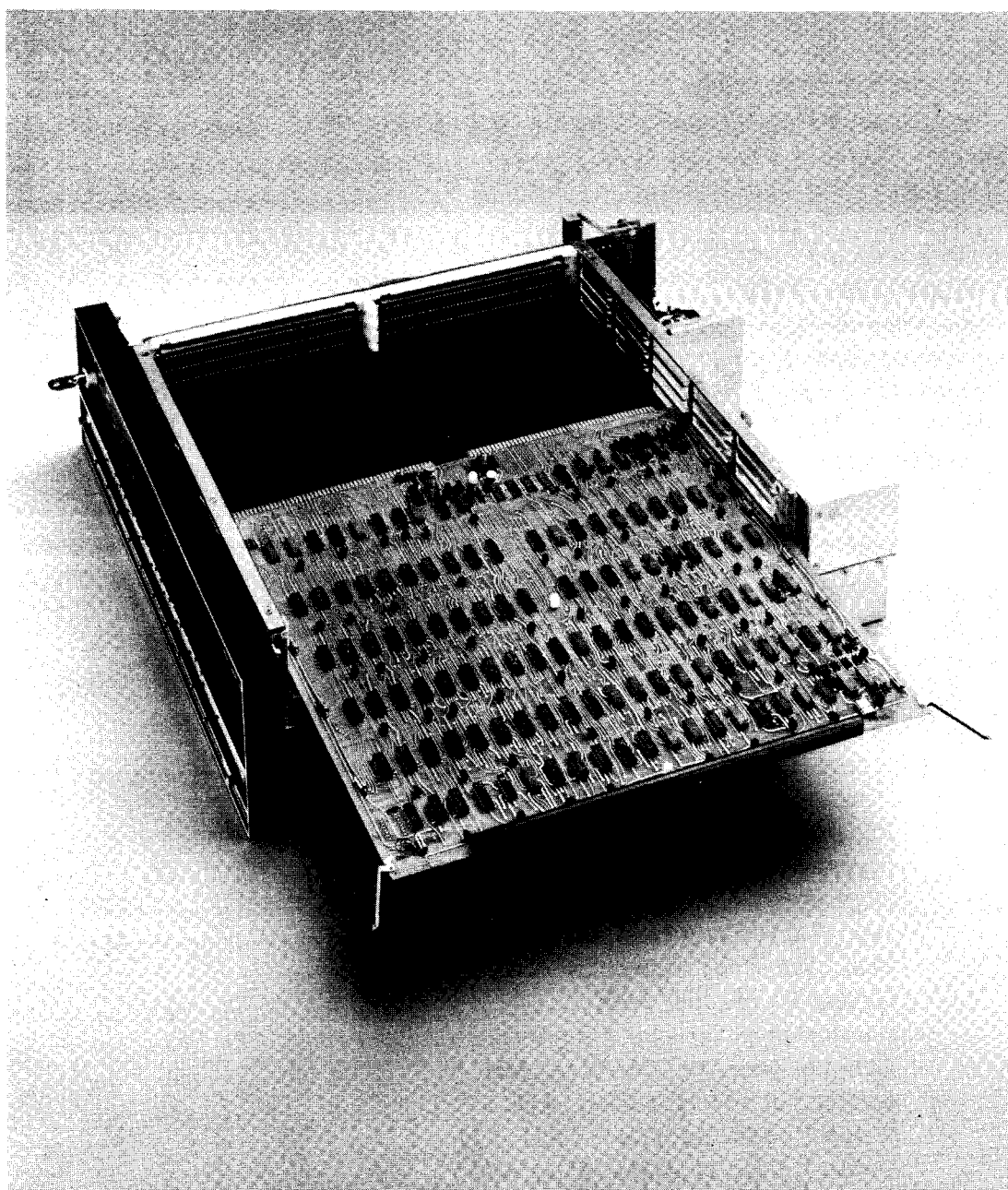
1	INTRODUCTION .....	1-1
1.1	The Nova line Computers .....	1-4
1.2	Peripherals .....	1-5
1.3	Instructions .....	1-6
	Instruction format 1-7	
1.4	Memory .....	1-9
1.5	Software .....	1-9
2	CENTRAL PROCESSOR .....	2-1
2.1	Memory Reference Instructions .....	2-2
	Move data instructions 2-5	
	Modify memory instructions 2-6	
	Jump instructions 2-7	
2.2	Arithmetic and Logical Instructions .....	2-10
	Carry, shift, and skip functions 2-12	
	Arithmetic and logical functions 2-14	
	Programming examples 2-17	
2.3	Input-Output .....	
	Special code-77 functions 2-26	
2.4	Programming Interrupt .....	2-30
2.5	Data Channel .....	2-37
2.6	Processor Options .....	2-39
	Real time clock 2-39	
	Power monitor and autorestart 2-40	
	Multiply-divide 2-40	
	Nova multiply-divide 2-41	
	Memory allocation and protection 2-42	
2.7	Operation .....	2-48
2.8	Program Loading .....	2-52
	Automatic loading 2-52	
	Manual program loading 2-55	
	Binary loader 2-56	
3	BASIC I/O EQUIPMENT .....	3-1
3.1	Teletypewriter .....	3-1
	Teletype output 3-2	
	Teletype input 3-3	
	Programming examples 3-3	
	Operation 3-5	
3.2	Paper Tape Reader .....	3-6

3.3	Paper Tape Punch.....	3-7
3.4	Line Printer.....	3-8
3.5	Plotter .....	3-10
3.6	Card Reader .....	3-12
3.7	DGC Cassette .....	3-14
4	MAGNETIC TAPE .....	4-1
4.1	Tape Format.....	4-1
4.2	Instructions.....	4-3
4.3	Tape Commands.....	4-6
4.4	Tape Transports - Models 4030A through 4030H .....	4-11
	transports 4-11, 4030A, 4030B, 4030C and 4030D	
	transports 4-13, 4030G, 4030H	
	transports 4-14, 4030E, 4030F	
4.5	Tape Transport - Models 4030I and 4030J .....	4-16
5	DISCS .....	5-1
5.1	Fixed Head Disc System.....	5-2
	Data format 5-2	
	Programming considerations 5-5	
5.2	Disc Cartridge System 4047 .....	5-6
	Data format 5-6	
	Instructions 5-7	
	Programming considerations 5-10	
	Operation 5-12	
5.3	Disc Pack Systems.....	5-12
	Data format 5-12	
	Instructions 5-12	
	Programming considerations 5-17	
	Operation 5-18	
6	ANALOG CONVERSION EQUIPMENT .....	6-1
6.1	A-D, D-A System Configurations.....	6-1
6.2	A-D Conversion .....	6-3
	Interface with channel scanner 6-6	
	Wiring considerations 6-8	
6.3	D-A Conversion .....	6-9
6.4	Specifications .....	6-9
	A-D operating specifications 6-10	
	D-A operating specifications 6-12	
6.5	Oscilloscope Control 4053 .....	6-13
7	DATA COMMUNICATIONS .....	7-1
7.1	Synchronous Communications Controller 4015	
	with Clock Option 4020 and Parity Option 4021.....	7-1
	Receiver 7-1	
	Transmitter 7-3	

	Automatic answering 7-5	
7.2	Asynchronous Data Communications Multiplexor 4026 .....	7-5
7.3	Modem Controls 4023 and 4029.....	7-7
7.4	Multiprocessor Communications Adapter 4038.....	7-10
	Receiver 7-11	
	Transmitter 7-13	
	Installation 7-14	
7.5	Asynchronous Data Communications Multiplexor 4060 .....	7-15
	Instructions 7-16	
	Modem control 7-18	
7.6	IBM System 360/370 Interface 4025.....	7-18
	Instructions 7-19	

APPENDICES

A	Interfacing .....	A1
I	In-out Bus.....	A1
	Bus signals A3	
	External signal connections A10	
	Bus circuits A10	
	In-out bus signal connections A11	
II	Interface Timing.....	A13
	Programmed transfers A14	
	Program interrupt A14	
	Standard data channel transfers A16	
	High speed data channel transfers A19	
III	Design of Interface Equipment.....	A22
	Basic interface networks A22	
	Design examples A24	
IV	Construction of Interface Equipment.....	A28
	General purpose interface A31	
	Socket pin configuration A38	
	Table I basic interface 4040 A39	
	Table II data register 4041 A40	
	Table III data channel logic 4042 A41	
	Connectors and connector parts A42	
B	Installation .....	B1
	Peripheral equipment B3	
C	Floating Point Arithmetic .....	C1
D	Instruction Mnemonics and Timing.....	D1
	Numeric listing D2	
	Alphabetic listing D5	
	Instruction execution times D12	
E	In-out Codes .....	E1
	In-out devices E2	
	Teletype code E4	





# Chapter I

## Introduction

The Nova line computers are general purpose computer systems with a 16-bit word length. All machines are organized around four accumulators, two of which can be used as index registers. This accumulator/index register organization provides ease in programming and great efficiency both in time and memory use. The various machines differ from one another in features and speed, but all use the same instruction set, and the programming for all is completely compatible (except of course for programs that are time dependent).

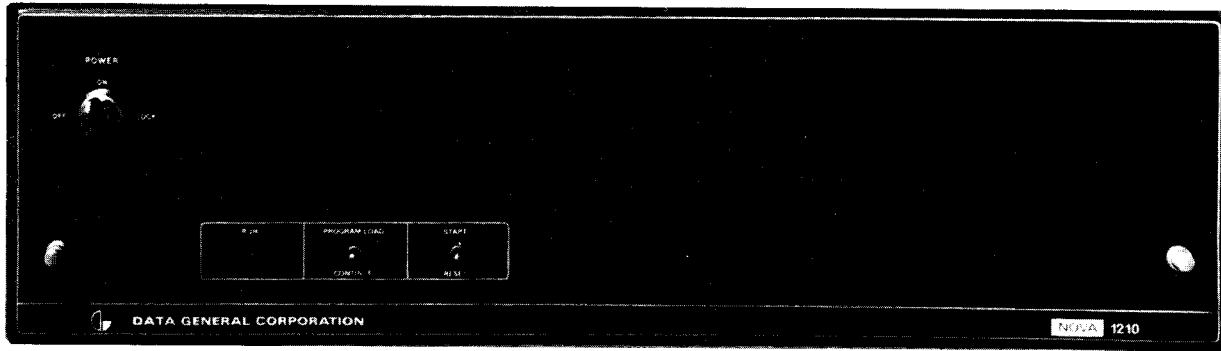
Any Nova line computer can have both alterable memory and read-only memory. Any machine may have either a programmer's console or a turnkey console that has a minimum of controls and is used in dedicated applications with well-defined, debugged software. With the console removed entirely, a system can be operated as a hardwired controller, whose functions can be altered simply by substituting different read-only memories.

Although the computer main frames vary in size, and therefore in the amount of memory and the number of IO interfaces they can contain, nonetheless they all mount in a standard 19-inch rack. Processor options include real time clock, power monitor and auto restart, multiply-divide, automatic program load, and memory allocation and protection. Available peripheral equipment includes Teletype<sup>®</sup>\*, high speed paper tape reader and punch, card reader, line printer, incremental plotter, display, magnetic tape, cassette tape, magnetic disc, A-D and D-A conversion equipment, and data communications equipment. There are also interfaces for interconnecting a number of Nova line computers and connecting a Nova line computer to an IBM 360/370 system.

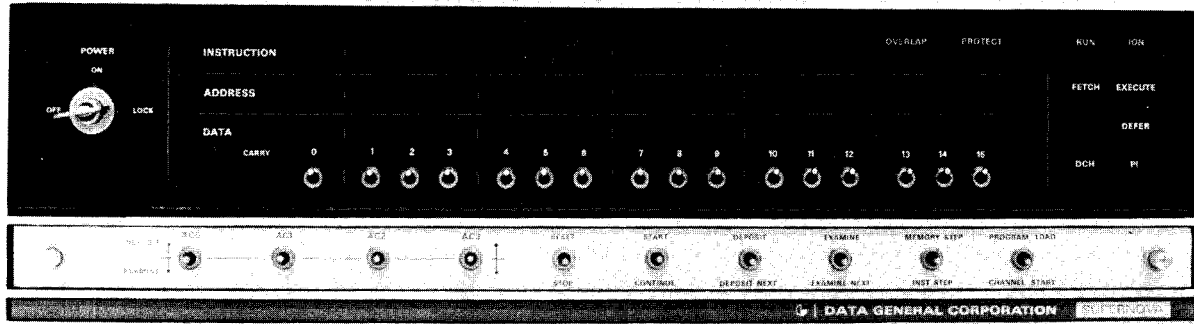
The central processor is the control unit for the entire system: it governs all peripheral in-out equipment, performs all arithmetic, logical, and data handling operations, and sequences the program. It is connected to the memory by a memory bus and to the peripheral equipment by an in-out bus. The processor handles words of sixteen bits, which are stored in a memory with a maximum capacity of 32,768 words. The bits of a word are numbered 0 to 15, left to right, as are the bits in the registers that handle the words. Registers that hold addresses are fifteen bits, numbered according to the position of the address in a word, *ie* 1 to 15. Words are used either as computer instructions in a program, as addresses, or as operands, *ie* data for the program. The program can interpret an operand as a logical word, an address, a pair of 8-bit bytes, or a 16-digit signed or unsigned binary number. The arithmetic instructions operate on fixed point binary numbers, either unsigned or the equivalent signed numbers using twos complement conventions.

The processor performs a program by executing instructions retrieved from consecutive memory locations as counted by the 15-bit program counter PC. At the end of each instruction PC is incremented by one so that the next instruction is normally taken from the next consecutive location. Sequential program flow is altered by changing the contents of PC, either by incrementing it an extra time in a test skip instruction or by replacing its contents with the value specified by a jump instruction. The other internal registers of importance to the programmer are four 16-bit accumulators, AC0 to AC3. Data can be moved in either direction between any memory location and any accumulator. Although a word in memory can be incremented or decremented, all other arithmetic and logical operations are performed on operands in the accumulators, with the result appearing in an accumulator. Associated with the accumulators is the Carry flag, which indicates when a carry occurs out of bit 0 in an arithmetic instruction. The left and right halves of any accumulator can be swapped, the contents of any accumulator can be tested for a skip, and the 17-bit word contained in any accumulator combined with

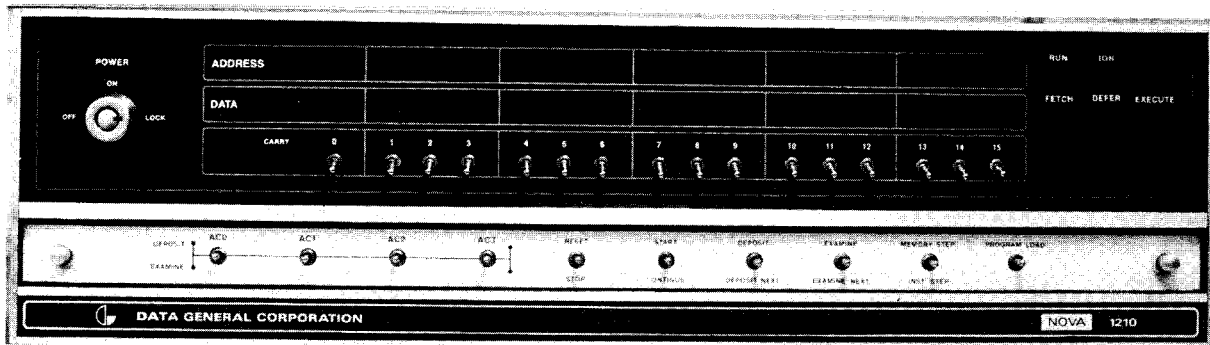
\*Teletype<sup>®</sup> is a registered trademark of Teletype Corporation, Skokie, Ill. All references to teletypes in this manual shall apply to this mark.



Turnkey Operator's Console



Supernova Operator's Console



Nova 800 Series and Nova 1200 Series Operator's Console

Carry can be rotated right or left. An instruction that references memory can address AC2 or AC3 as an index register, and transfers to and from peripheral devices are also made through the accumulators.

On the programmer's console is a set of data switches through which the operator can supply words and addresses to the program. The console also has control switches that allow the operator to start and stop the program, to deposit the contents of the data switches in any memory location or accumulator, and to display the contents of any location or accumulator in the data lights. An optional console feature on the Nova 800 series and 1200 series provides automatic loading when there is no program in memory (this feature is standard on the Supernova computer). The address lights display the contents of PC; the remaining lights display the Carry flag and a number of internal control conditions that are useful in program debugging.

Any instruction that references memory may address AC2 or AC3 as an index register. Instructions that move data to and from memory or the peripherals address a single accumulator as a source or destination of data

while addressing a memory location or an in-out device. But the arithmetic and logical instructions do not have to reference memory; they simply address two accumulators, either or both of which may supply operands, and one of which may receive the result. Thus memory is used for storage of the program and permanent data, but all calculations are carried out in the accumulators and intermediate results are held right in them. This reduces considerably the amount of data movement as compared with a single accumulator system, and thus saves instructions. For example, in as trivial an operation as exchanging the contents of two memory locations A and B, the multi-accumulator organization reduces the time by one third.

*Exchange with  
one accumulator*

A → AC  
AC → TEMP  
B → AC  
AC → A  
TEMP → AC  
AC → B

*Exchange with  
two accumulators*

A → AC1  
B → AC2  
AC1 → B  
AC2 → A

Since an arithmetic or logical instruction does not contain a memory address, there are many bits that can be used for functions other than specifying the basic operation and the operands: the same instruction that adds or subtracts can also shift the result or swap its halves, test the result and/or carry for a skip, and specify whether or not the result shall actually be retained. Hence the percentage of time saved increases with the complexity of the program.

And there are advantages other than speed. The system is much more convenient to use, programming is much easier because the data being processed is much handier. The accumulators and their associated logic are essentially like the pad one uses at one's desk, whereas the memory fulfills the function of a set of reference books and a notebook kept on one's side. The results of address calculations are immediately available for index purposes to the memory reference instructions. One accumulator can be used for in-out data transmission without disturbing others being used continually for computations. Complex software routines such as multiplication, division and floating point can be performed without constantly referencing memory.

The input-output hardware allows the program to address up to sixty-two devices. A single instruction can transfer a word between an accumulator and a device and at the same time control the device operation. Included in the in-out system are facilities for program interrupts and high speed data transfers. The interrupt system facilitates processor control of the peripheral equipment by allowing any device to interrupt the normal program flow on a priority basis. The processor acknowledges an interrupt request by storing PC in location 0 and executing the instruction addressed by the contents of location 1. A high speed device, such as magnetic tape or disc, can gain direct access to memory through a data channel without requiring the execution of any instructions; the program simply pauses while access is made. The data channel logic allows the transfer of data to or from memory, incrementing of a memory word, and (in some machines) adding external data to a word already in memory. The latter two features allow such functions as pulse height analysis and signal averaging.

Processor options available on all Nova line computers are the power monitor and auto restart, hardware multiply-divide, and real time clock. The first of these has a flag that can interrupt when power failure is imminent, so the program can provide for an orderly shutdown; the program restarts automatically when power is restored. The other two options are handled by the program as though they were in-out interfaces. The multiply-divide

tures, but are equal in size and slot capacity to the 800 and 800 Jumbo. However, the 1200 series computers require only one slot for the central processor, so the 1200 and 1200 Jumbo respectively have six and sixteen slots available for memory and IO interfaces.

The core memory of the Supernova computer has an 800 nanosecond cycle, but this computer can also operate with semiconductor memory, which has a 300 nanosecond cycle (the two memory types can be mixed). With semiconductor memory, overlapping allows the Supernova computer to execute arithmetic and logical instructions in a single 300 nanosecond memory cycle. Automatic program load is standard equipment on the Supernova computer; memory allocation and protection is available as an option, and the hardware multiply-divide can be mounted on one of the central processor boards. The main frame is 5¼ inches high and has seven slots, of which three are for the central processor. An expansion chassis of the same size and capacity and containing its own power supply is available for this computer.

The basic processor cycle time of the Nova computer is 2.6 microseconds with core memory, 2.4 microseconds with read-only memory. The Nova computer chassis is 5¼ inches high with seven slots, of which two are for the central processor. An expansion chassis can be added for additional memory and I/O devices. The hardware multiple-divide option for this computer functions in the manner of an in-out interface and must be mounted on a separate board.

## 1.2 PERIPHERALS

A variety of peripheral equipment is available with all of the Nova line computers. The basic in-out devices are the teletypewriter and high speed paper tape reader and punch. Teletype models available are the ASR33, KSR33 and KSR35, all of which operate at ten characters per second, and the KSR37 and ASR37, which operate at fifteen characters per second. The reader handles 8-channel, fanfold perforated paper or Mylar tape photoelectrically at 400 characters per second. The punch can output 63.3 characters per second on 8-channel fanfold paper tape.

To simplify customer design of interfaces for Nova line computers, Data General has available a general purpose wiring frame with standard connector and space for mounting small wiring boards, which are available with or without wire-wrap pins and sockets; a general purpose printed circuit board with hole pattern for integrated circuits as well as discrete components (also available with wire-wrap pins or pins and sockets); and a general purpose interface that has the standard logic for several types of interfacing to the IO bus, plus room for the customer's own interface circuits.

**Card Readers.** Punched card and Mark Sense card readers are available at speeds ranging from 150 to 1,000 cards per minute.

**Incremental Plotters.** Both drum and flatbed plotters are available; model 4017E uses Z-fold paper.

**Fixed Head Disc Drives.** Head-per-track discs have storage capacities of 64K, 128K, 256K, 512K and 768K 16-bit words. Data transfer is through the data channel. Discs are rack-mountable. As many as eight logical units may use the same control.

**Cartridge Disc Drives.** Two compact cartridge disc drives are available, one with a single removable cartridge (capacity 1.247 million 16-bit words) and one with both a removable cartridge and a fixed disc (total capacity 2.494 million 16-bit words). Data is transferred at the rate of one word every 11.5 microseconds. The disc drive has two moving heads per disc. The drive mounts in a standard computer cabinet and the cartridge is IBM compatible.

**Disc Pack Drives.** The disc pack drives use IBM-compatible 6- or 11-disc packs with 10 or 20 read/write surfaces and 10 or 20 moving read/write heads. Capacity is 3.072 or 12.288 million 16-bit words. Data is transferred at the rate of one word every 12.8 microseconds.

**Magnetic Tape Transports.** Synchronous read/write 7- or 9-track, 12.5, 45 or 75 ips, industry-compatible tape transports.

**DGC Cassette.** Input/output to single channel magnetic tapes housed in portable interchangeable cassettes. Up to eight transports may use the same control.

**Line Printers.** Line printers are either 356 lpm, 80 columns, 64 characters or 245 lpm, 132 columns, 64 characters. Full ASCII interface with paper-advance characters is included.

**High Speed Communications Controller.** Used with high speed full-duplex or half-duplex synchronous data sets (Bell 201, Bell 301, or equivalent), the controller allows automatic line synchronization, word assembly, and end-of-transmission recognition.

**Data Communications Interface.** Input-output interface for Bell System 202 data set or equivalent for operation at 1200 bits/second.

**16-Line Teletype Multiplexor.** Controls up to 16 Teletypes or Bell 103 modems. Allows for programmed bit assembly/disassembly of characters.

**Asynchronous Multiplexor.** Enables any Nova line computer to communicate with and control terminal devices over a variety of communication facilities. The modularity of the hardware permits simply system expansion and the addition of new features or special purpose equipment.

**Multiprocessor Communications Adapter.** Connects up to fifteen Nova line computers into a multiprocessor system by permitting blocks of data to be transferred from one computer to another through their data channels.

**360 Interface.** A very flexible, generalized interface that makes it possible for any Nova line computer with appropriate software to emulate all standard IBM peripheral controllers. Communication takes place via the System 360 selector or multiplexor channel.

**Analog-to-Digital Converters.** Analog-to-digital converters are available having 1 to 256 channels and word length of 8 to 15 bits. The analog-to-digital interface also runs 3 1/2-digit panel meters.

**Digital-to-Analog Converters.** Digital-to-analog converters are available having 1 to 24 channels and word lengths of 8 to 14 bits.

**CRT Display Terminals.** Terminals display 20 lines, 80 characters long. Code structure and baud rate are variable.

### 1.3 INSTRUCTIONS

The types of functions performed by instructions in most computers are the following.

1. Move data between memory and the operating registers.
2. Modify memory, usually in conjunction with a test to determine whether to alter the program sequence.
3. Alter the program sequence by jumping to a new location.
4. Perform an arithmetic or logical operation.
5. Test the value of a word or flag, or one word against another, to determine whether to alter the program sequence.
6. Transfer data to or from the peripheral equipment.

In many computers the first and fourth and the third and fifth groups overlap. In the Nova computer groups 1 and 3 are unique. But groups 4 and 5 coincide: every arithmetic and logical instruction can test the result for a skip.

The following lists the registers that must be specified and the functions performed by the various instruction classes in the Nova line computers.

Move data	One memory location, one accumulator. Either may be the source of the operand, the other is the destination.
Modify memory	One memory location. Increment or decrement contents; skip if result is zero.
Jump	One memory location from which the next instruction is taken. A return address can be saved in AC3.
Arithmetic and logic	Two accumulators. One or both may be source of operand(s). Perform arithmetic or logical function, with a bit-0 carry affecting the Carry flag as indicated. If desired, swap halves of answer or rotate it with Carry one place right or left, load result into either accumulator, and skip on condition specified for result and/or Carry.

Input-output

One accumulator, one IO device. Transfer word in either direction between any accumulator and one of up to three registers in up to sixty-two devices. Also operate device as specified.

Note: A subclass of these instructions executes no transfer and specifies only a device. The instruction either operates the device or skips on a selected condition in it.

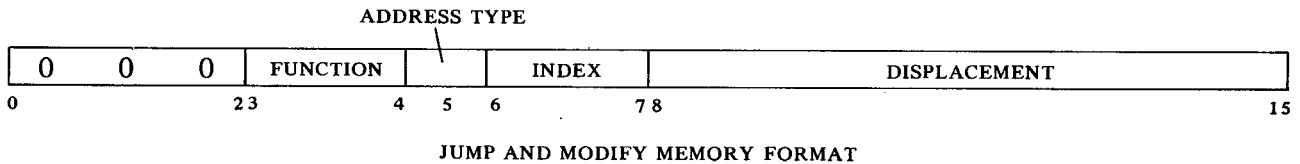
**Addressing.** Instructions in the first three classes must address a memory location. Each instruction word contains information for determining the effective address, which is the actual address used to fetch or store the operand or alter program flow. The instruction specifies an 8-bit displacement which can directly address any location in four groups of 256 locations each. The displacement can be an absolute address, *ie* it may be used simply to address a location in page zero, the first 256 locations in memory. But it can also be taken as a signed number that is used to compute an absolute address by adding it to a 15-bit base address supplied by an index register. The instruction can select AC2 or AC3 as the index register; either of these accumulators can thus be used as an ordinary index register to vary the address computed from a constant displacement, or as a base register for a set of different displacements. The program can also select PC as the index register, so any instruction can address 256 words in its own vicinity (relative addressing).

Now the computed absolute (15-bit) address can be the effective address. However, the instruction can use it as an indirect address, *ie* it can specify a location to be used to retrieve another address. Bits 1–15 of the word read from an indirectly addressed location can be the effective address or they can be another indirect address.

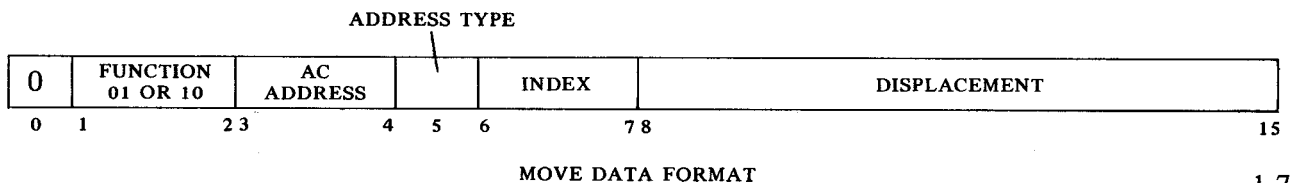
**Automatic Incrementing and Decrementing.** The program can make use of an automatic indexing feature by indirectly addressing any memory location from 00020 to 00037 (addresses are always octal numbers). Whenever one of these locations is specified by an indirect address, the processor retrieves its contents, increments or decrements the word retrieved, writes the altered word back into memory, and uses the altered word as the new address, direct or indirect. If the word is taken from locations 00020–00027, it is incremented by one; if taken from locations 00030–00037, it is decremented by one.

**Instruction Format**

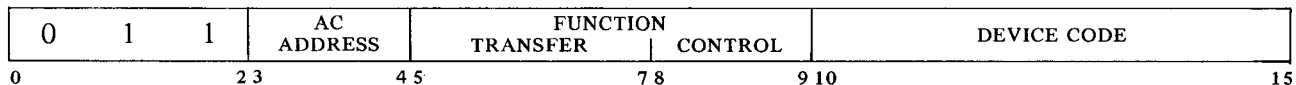
There are four basic formats for instruction words. In all but the arithmetic and logical instructions, bit 0 is 0. If bits 1 and 2 are also 0, bits 3 and 4 specify the function (jump or modify memory) and the rest of the word supplies information for calculating the effective address. Bits 8–15 are the displacement, bits 6 and 7 specify the index register if any, and bit 5 indicates the type of addressing, direct or indirect.



If bits 1 and 2 differ they specify a move data function. Bits 3 and 4 address an accumulator, the rest of the word is as above.



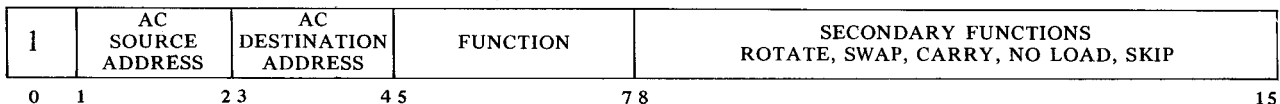
Bits 1 and 2 both being 1 indicate an in-out instruction. In this case the function is specified by bits 5–9, of which bits 5–7 indicate the direction of transfer and select one of three registers in the device. The transfer takes place between the accumulator addressed by bits 3 and 4 and the device selected by bits 10–15. Bits 8



IN-OUT FORMAT

and 9 of the function part specify an action to be performed, such as starting the device. If bits 5–7 are all 0 or all 1, there is no transfer and bits 8 and 9 specify a control or skip function respectively.

If bit 0 is 1, bits 5–7 specify an arithmetic or logical function. One operand is taken from the accumulator addressed by bits 1 and 2; a second operand, if any, from that addressed by bits 3 and 4. The rest of the word specifies the other functions that can be performed, including whether or not the result is to be loaded into the destination accumulator.



ARITHMETIC AND LOGIC FORMAT

The Nova computer assembly programs recognize a number of mnemonics and other initial symbols that facilitate constructing complete instruction words and organizing them into a program [*Appendix D*]. In particular there are three-letter mnemonics for the 2- and 3-bit functions; these mnemonics also represent whatever bits are constant for the class the instruction is in. *Eg* the modify memory mnemonic

**ISZ**

assembles as 010000, the arithmetic mnemonic

**SUB**

assembles as 102400.

**NOTE**

Throughout this manual all numbers representing instruction words, register contents, codes and addresses are always octal, and any numbers appearing in program examples are octal unless otherwise specified. Computer words are represented by six octal digits wherein the left one is always 0 or 1 representing the value of bit 0. The ordinary use of numbers in the text to specify quantities of objects, such as words or locations, to count steps in an operation, or to specify word or byte lengths, bit positions, etc. employs standard decimal notation.

Characters are suffixed to the basic mnemonic to specify the control part of an IO function and most of the secondary functions in the arithmetic and logical class. The displacement and addresses of accumulators and

1-8

index registers are separated from the mnemonic by a space and from each other by commas. Anything written at the right of a semicolon in a program listing is commentary that explains the program but is not part of it.

## 1.4 MEMORY

From the addressing point of view, the entire memory is a set of contiguous locations whose addresses range from zero to a maximum dependent upon the capacity of the particular installation. In a system with the greatest possible capacity, the largest address is octal 77777, decimal 32,767. But the memory is actually made up of a number of core memory modules, each having a capacity of 1024, 2048, 4096 or 8192 words, and can also contain read-only memory modules. The latter may be used for storage of pure (unalterable) programs and constants; they contain 256, 512 or 1024 words. The Supernova computer may also operate with semiconductor random access memory modules; these are available in units of 256, 512 and 1024 words. An address supplied by the program is actually decoded in two parts, the more significant to select a memory module and the less significant to select a location within that module, but this need not concern the programmer. From the point of view of the programmer, memory module size is irrelevant, and read-only memory differs from the others only in that its contents cannot be altered electrically. Common arithmetic and in-out routines are available in standard read-only memory modules; others are available on a custom basis.

**Memory Restrictions.** The use of certain locations is defined by the hardware.

0-1	Program interrupt locations
20-27	Autoincrementing locations
30-37	Autodecrementing locations

## 1.5 SOFTWARE

To support the Nova line computer, Data General supplies a very extensive software package. This package includes assemblers, editors, compilers, and operating systems, as well as numerous utility programs for various devices, debugging programs, data conversion, mathematical and interpretive routines, and a complete set of hardware diagnostics. Some of the major software items are the following.

The **Absolute Assembler** is a two-pass system that produces absolute binary and an assembly listing. The program accepts pseudo commands, and the source input is form free.

The **Extended Assembler** includes all features of the absolute assembler, and it also provides relocation, interprogram communication, conditional assembly, and more powerful number definition facilities.

The **Cross Assemblers for IBM 360, CDC 6600, Univac 1108**, all written in FORTRAN IV, assemble symbolic source code into machine object code for the Nova line computers, using card input to the IBM 360, CDC 6600, or Univac 1108. Output can be in absolute or relocatable binary suitable as input for the Data General binary or relocatable loader.

The **Relocatable Binary Loader** loads relocatable binary tapes produced by the Extended Assembler. It accepts any number of tapes as input, resolves external displacements and normal externals, and maintains an entry symbol table that can be printed on demand.

The **Macro Editor** edits paper tape input to produce updated paper tape output. It is used most commonly to modify program source tapes in preparation for a new assembly. The editor executes simple command-string input using the Teletype to modify text on a character-string basis or a line basis. The user can define command



strings in a special macro register; a string can then be executed repeatedly by specifying the macro register name in other command strings.

**Time Sharing BASIC** is a dedicated interpretive system that allows conversational entry and execution of programs written in the BASIC language. It includes use of all elementary and advanced BASIC statements including matrix and string extensions. The system supports sixteen Teletype terminals and includes a comprehensive list of error messages.

**Single User BASIC** has all the features of Time Sharing BASIC, with the exception of matrix and string manipulation functions.

**Extended BASIC** has all the features of Time Sharing BASIC, plus extended features which allow access to IO peripherals for both data and program files. Supported devices include high speed reader and punch, line printer, fixed head disc, and moving head disc pack and disc cartridge. Disc files can be protected from unauthorized access or can be placed in a system disc library, which is available to all users. Disc versions of Extended BASIC will timeshare core memory among users, allowing each user access to all memory.

**Extended FORTRAN IV** is an implementation of the ANSI FORTRAN IV, with provisions for reentrant object code and many language extensions. The latter include generalized subscript expressions, mixed-mode arithmetic, double precision complex arithmetic, abnormal returns from subroutines via a dummy variable, arrays with 128 dimensions, and array declarations in which the lower bound need not be 1. The code generated provides optimized register and storage allocation and reentrant machine language code, which can be interfaced with assembly language code.

**Extended ALGOL** is a superset of ALGOL 60 with extensions that allow simplified free-form IO or formatted output, bit manipulation, easy manipulation of character-string data, recursive and reentrant procedures, dynamic storage allocation, *n*-dimensional arrays, multiprecision arithmetic, dynamic conversion for full mixed-mode capability, and explicit diagnostics.

**The Real Time Disc Operating System.** (RDOS) is a modular, multitask disc based real time operating system. It can be used in both the development and implementation of programs to be run in a real time environment. RDOS allows user programs to be loaded into fixed partitions within the user address space, thus allowing programs larger than available memory to be executed. RDOS provides a facility for buffered I/O transfers or unbuffered block transfers from the user's area for faster real time applications. Output device spooling maximizes system usage by releasing memory from usages other than message buffering.

**The Real Time Operating System.** (RTOS) provides a flexible, modular interface to user programs operating in a real time environment. Multitasking, time-slicing, and I/O transfers are handled by simple task calls to the system. All standard peripheral devices are supported. RTOS is a compatible subset of the Real Time Disc Operating System.

**The Disc Operating System.** (DOS) provides a comprehensive file system capability, buffered management of all I/O peripherals on a device independent basis, and program execution facilities using disc file overlays. The system operates with both fixed-head and moving-head discs. It supports an extensive library of system software, all executable by means of simple Teletype commands.

**The Stand-Alone Operating System.** (SOS) provides buffered service of I/O peripherals on a device independent basis. It interprets a subset of the system calls of the Disc Operating System to provide stand-alone compatible facilities. Programs may be structured to run under either operating system without modification. SOS provides a magnetic tape or cassette tape operating system allowing users to edit, assemble, or execute programs stored on the tapes.

# Chapter II

## Central Processor

This chapter describes all computer instructions but does not discuss the special effects of the in-out instructions when they address specific peripheral devices. The chapter treats the memory reference instructions and the arithmetic and logical instructions in detail, presents a general discussion of input-output, and describes the effects of the in-out instructions on processor elements, including the program interrupt, the real time clock, multiply-divide, and the memory allocation and protection option. Effects of in-out instructions on particular peripheral devices are discussed with the devices in the remaining chapters.

In the description of each instruction, the mnemonic and name are at the top, the format is in a box below them. The mnemonic assembles to the word in the box, where bits in those parts of the word represented by letters assemble as 0s. The letters indicate portions that must be added to the mnemonic to produce a complete instruction word.

Instruction execution times depend both on the processor and the type of memory; they are therefore given in a table at the end of Appendix D.

**Twos Complement Conventions.** The signed numbers used as displacements in referencing memory and as operands for the arithmetic instructions utilize the twos complement representation for negatives. In a word or byte used as a signed number, the leftmost bit represents the sign, 0 for positive, 1 for negative. In a positive number the remaining bits are the magnitude in ordinary binary notation. The negative of a number is obtained by taking its twos complement, with the sign bit included in the operation as though it were a more significant magnitude bit. If  $x$  is an  $n$ -digit binary number, its twos complement is  $2^n - x$ , and its ones complement is  $(2^n - 1) - x$ , or equivalently  $(2^n - x) - 1$ . Subtracting a number from  $2^n - 1$  (ie, from all 1s) is equivalent to performing the logical complement, ie changing all 0s to 1s and all 1s to 0s. Therefore, to form the twos complement one takes the logical complement — usually referred to merely as the complement — of the entire word including the sign, and adds 1 to the result. A displacement of 89 and its negative would look like this in bits 8–15 of an instruction word where bit 8 is the sign.

$$\begin{array}{rcl}
 +89_{10} & = & +131_8 & = & \boxed{\begin{array}{c} 01\ 011\ 001 \\ 8 \qquad 15 \end{array}} \\
 -89_{10} & = & -131_8 & = & \boxed{\begin{array}{c} 10\ 100\ 111 \\ 8 \qquad 15 \end{array}}
 \end{array}$$

The same numbers used as operands in the accumulators would look like this.

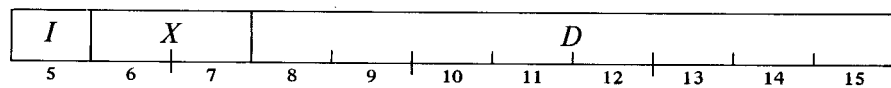
$$\begin{array}{rcl}
 +89_{10} & = & +131_8 & = & \boxed{\begin{array}{c} 0\ 000\ 000\ 001\ 011\ 001 \\ 0 \qquad 15 \end{array}} \\
 -89_{10} & = & -131_8 & = & \boxed{\begin{array}{c} 1\ 111\ 111\ 110\ 100\ 111 \\ 0 \qquad 15 \end{array}}
 \end{array}$$

Bit 0 is now the sign and bits 1–8 are not significant. It is thus evident that expanding an integer into a full word is accomplished simply by filling out the word to the left with the sign.

Zero is represented by a number containing all 0s; complementing this number produces all 1s, and adding 1 to that produces all 0s again. So there is only one zero representation and its sign is positive. Moreover there is one more negative number than there are nonzero positive numbers. Hence there are 256 displacements in an octal range  $-200$  to  $+177$ . (The most negative number has a 1 in only the sign position.)

## 2.1 MEMORY REFERENCE INSTRUCTIONS

Bits 5–15 have the same format in every memory reference instruction whether the effective address is used for storage or retrieval of an operand or to alter program flow. Bit 5 is the indirect bit, bits 6 and 7 are the

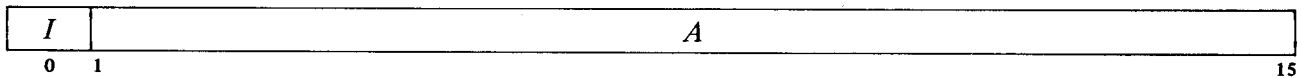


index bits, and bits 8–15 are the displacement. The effective address  $E$  of the instruction depends on the values of  $I$ ,  $X$ , and  $D$ . If  $X$  is 00,  $D$  addresses one of the first  $256_{10}$  memory locations, *ie*  $D$  is a memory address in the range  $00000-00377_8$ . This group of locations is referred to as page zero.

If  $X$  is nonzero,  $D$  is a displacement that is used to produce a memory address by adding it to the contents of the register specified by  $X$ . The displacement is a signed binary integer in twos complement notation. Bit 8 is the sign (0 positive, 1 negative), and the integer is in the octal range  $-200$  to  $+177$  (decimal  $-128$  to  $+127$ ). If  $X$  is 01, the instruction addresses a location relative to its own position, *ie*  $D$  is added to the address in PC, which is the address of the instruction being executed. This is referred to as relative addressing. If  $X$  is 10 or 11 respectively, it selects AC2 or AC3 as a base register to which  $D$  is added.

<i>X</i>	<i>Derivation of address</i>
00	Page zero addressing. $D$ is an address in the range $00000-00377$ .
01	Relative addressing. $D$ is a signed displacement ( $-200$ to $+177$ ) that is added to the address in PC.
10	Base register addressing. $D$ is a signed displacement ( $-200$ to $+177$ ) that is added to the address in AC2.
11	Base register addressing. $D$ is a signed displacement ( $-200$ to $+177$ ) that is added to the address in AC3.

If  $I$  is 0, addressing is direct, and the address already determined from  $X$  and  $D$  is the effective address used in the execution of the instruction. Thus a memory reference instruction can directly address 1024 locations: 256 in page zero, and three sets of 256 in the octal range 200 less than to 177 greater than the address in PC, AC2 and AC3. If  $I$  is 1, addressing is indirect, and the processor retrieves another address from the location



specified by the address already determined. In this new word bit 0 is the indirect bit: bits 1–15 are the effective address if bit 0 is 0; otherwise they specify a location for yet another level of address retrieval. This process continues until some referenced location is found with a 0 in bit 0; bits 1–15 of this location are the effective address *E*.

Specific examples illustrating the various addressing methods are given on the next two pages.

The set of all addresses is cyclic with respect to the operations performed in an effective address calculation; regardless of the true sum or difference in any step, only the low order fifteen bits are used as an address. Hence the next address beyond 77777 is 00000, the next below 00000 is 77777.

**Automatic Incrementing and Decrementing Locations.** If at any level in the effective address calculation an address word is fetched from locations 00020-00037, it is automatically incremented or decremented by one, and the new value is written back in memory. Addresses taken from locations 00020-00027 are incremented, those from locations 00030-00037 are decremented. The next step of the effective address calculation depends on bits 1-15 of the new address word and the *former* value of bit 0 of the address word. If bit 0 of the word originally fetched was 0, the new 15-bit address value is used as the effective address. If bit 0 of the former address word was 1, the new 15-bit address value is used to fetch the next address word in the effective address calculation.

In an auto-increment or auto-decrement address computation, the new value for bit 0 will differ from the former value of the bit only when the address 77777 is incremented or the address 00000 is decremented (in other words, when one of the locations 00020-00027 contains 077777 or 177777 and is addressed indirectly, or when one of the locations 00030-00037 contains 000000 or 100000 and is addressed indirectly). In these cases it is important to remember that it is the former value of bit 0 of the auto-increment or auto-decrement location which determines whether the new 15-bit address value will be treated as the effective address or as another indirect address in the addressing chain. The new value for bit 0 is written back into memory as always, along with the incremented or decremented address value, but it does not participate in an effective address calculation until the next time the auto-increment or auto-decrement location is addressed indirectly.

**Programming Conventions.** All memory reference functions are represented by three-letter mnemonics; eg

ISZ

assembles as 010000. For addressing page zero the displacement is simply an address. Thus

ISZ            344

assembles as 010344. When this word is executed as an instruction it increments the word in location 00344 and skips the next instruction if the incremented word is zero. For relative or base register addressing the displacement is a twos complement integer.

ISZ            - 34,2

assembles as 011344 (0 001 001 011 100 100), in which bits 8–15 have the same configuration as in the previous example, but this time the instruction specifies a location whose address is  $34_8$  less than the address in AC2.

The initial symbol @ preceding the displacement places a 1 in bit 5 to produce indirect addressing. The examples given above use direct addressing, but

ISZ            @ - 34,2

assembles as 013344 (0 001 011 011 100 100), and produces indirect addressing.

For memory reference with an accumulator, the AC address precedes the memory address information and is terminated by a comma. *Eg*

LDA 3, -34, 2

assembles as 035344 (0 011 101 011 100 100).

The assembler also allows the following addressing conventions. A period represents the current address, *ie* the address of the location containing the instruction being executed. Thus

LDA 3, . + 6

is equivalent to

LDA 3, 6, 1

A colon following a symbol indicates that it is a symbolic location name..

A: ADD 2, 3

indicates that the location that contains ADD 2,3 may be addressed symbolically as A. The assembler assigns a 15-bit value to the label A. When A is used in a statement such as

LDA 2, A + 6

the treatment depends on the value of the expression in which A appears. In this case if  $A + 6 < 00400$  its low order eight bits are simply placed in the displacement part of the instruction word and X is set to 00. If  $A + 6$  is within range of PC, the indicated location is represented as a displacement relative to PC. Otherwise the assembler indicates an error as location  $A + 6$  cannot be directly addressed by the instruction.

**Addressing Examples.** Suppose the following registers contain the numbers listed:

Register	Contents
6	100015
12	000035
15	000017
17	000023
23	000011
AC3	000015

Now if the program executes the instruction

LDA 1, 6

which loads AC1 from location 6, AC1 receives the number 100015. AC1 holds the same number after

LDA 1, -7, 3

is executed (effective address =  $C(AC3) - 7 = 15 - 7 = 6$ ). But

LDA 1, @6

which indirectly addresses location 6, which in turn indirectly addresses location 15, which directly addresses location 17, loads 23 into the accumulator. AC1 also contains 23 following execution of

LDA 1, @15

On the other hand, AC1 contains 17 after

LDA 1, 15

or

LDA 1,0,3

is executed. Now

LDA 1,6,3

does not address location 6; it addresses 23 ( $C(AC3)+6=15+6=23$ ) and thus loads 11 into AC1. Note that addressing an autoincrementing location directly does not alter its contents; AC1 simply receives its contents as an operand. AC1 also receives 11 from

LDA 1,23

or

LDA 1,@17

But giving

LDA 1,@23

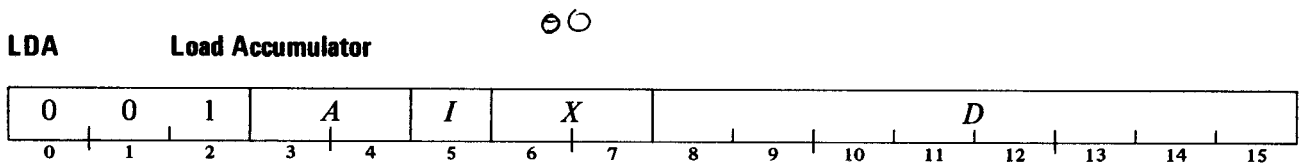
or

LDA 1,@6,3

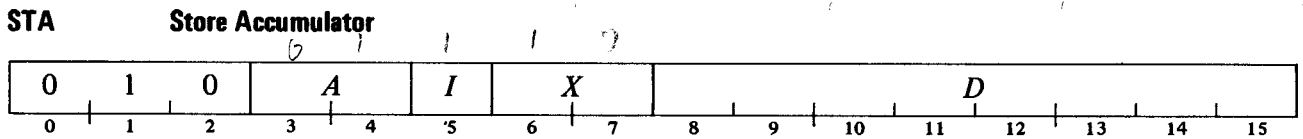
replaces the contents of location 23 with the number 12 and loads 35 (the contents of location 12) into AC1.

### Move Data Instructions

These two instructions move data between memory and the accumulators. In the descriptions of all memory reference instructions, *E* represents the effective address.



Load the contents of location *E* into accumulator *A*. The contents of *E* are unaffected, the original contents of *A* are lost.

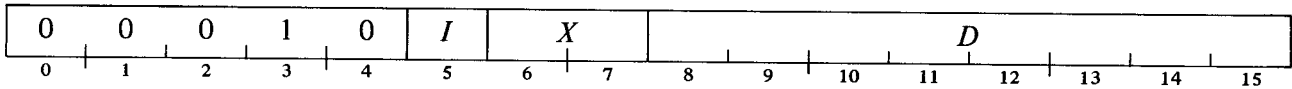


Store the contents of accumulator *A* in location *E*. The contents of *A* are unaffected, the original contents of *E* are lost.

## Modify Memory Instructions

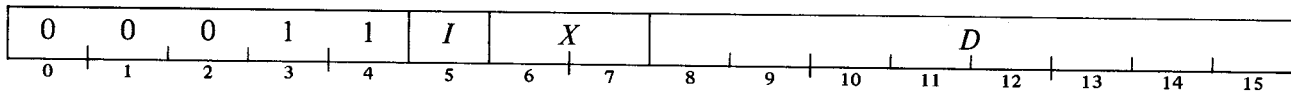
These two instructions alter a memory location and test the result for a skip. They are used to count loop iterations or successively modify a word for a series of operations.

### ISZ            Increment and Skip if Zero



Add 1 to the contents of location *E* and place the result back in *E*. Skip the next instruction in sequence if the result is zero.

### DSZ            Decrement and Skip if Zero



Subtract 1 from the contents of location *E* and place the result back in *E*. Skip the next instruction in sequence if the result is zero.

Consider a block of thirty words in locations 2000–2035 that we wish to move to locations 5150–5205 but in reverse order. We could autoincrement through one set, autodecrement through the other, and decrement a control count to determine when the block transfer is complete.

```

LDA    0,CNT           ;Set up autoincrement location
STA    0,21
LDA    0,CNT+1       ;Set up autodecrement location
STA    0,35

LOOP:  LDA    0,@21    ;Get a word
       STA    0,@35    ;Store it
       DSZ   CNT+2     ;Count down word count
       JMP   LOOP     ;Jump back for next word
       :             ;Skip to here when count is zero
       :

```

CNT:           001777                   ;1 before source block  
                  005206                   ;1 after destination block  
                  000036                   ;Word count:  $30_{10} = 36_8$

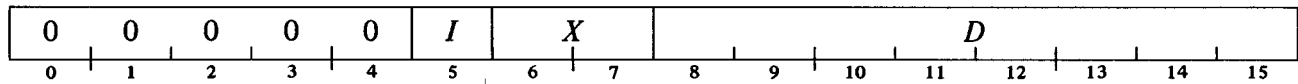
27774 → 31441  
 5 → 775

Of course we could just as well put 177742 (−36) in CNT+2 and replace the DSZ with an ISZ.

### Jump Instructions

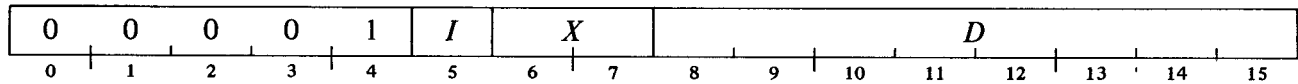
These two instructions allow the programmer to alter the normal program sequence by jumping to an arbitrary location. They are especially useful for calling and returning from subroutines.

#### JMP           Jump



Load *E* into PC. Take the next instruction from location *E* and continue sequential operation from there.

#### JSR           Jump to Subroutine



Load an address one greater than that in PC into AC3 (hence AC3 receives the address of the location following the JSR instruction). Load *E* into PC. Take the next instruction from location *E* and continue sequential operation from there. The original contents of AC3 are lost.

NOTE: The effective address calculation is completed before PC+1 is loaded into AC3. Thus a JSR that specifies AC3 as a base register does execute properly; *ie* the previous contents of AC3 are used in the address calculation.

The usual procedure for calling a subroutine is to give a JSR whose effective address is the starting location of the routine. Since PC+1 is saved in AC3, a subsequent return can be made to the location following the JSR simply by giving a **JMP 0,3**. Note also that PC+1 is saved in an accumulator. Hence the subroutine can be reentrant (pure), *ie* memory is not modified by the act of calling it. If we wish to use AC3 in the subroutine, we can store the return address in a convenient place in page zero, say location B, with an **STA 3,B** and then return with a **JMP @B**.

A convenient way to handle a number of subroutines that are called frequently is to store their starting addresses in page zero. Suppose we have subroutines starting at locations U, V, W, X, . . . . If we store these 15-bit addresses at locations UC, VC, WC, XC, . . . respectively in page zero, then we can call a given routine, say the one beginning at X, simply by giving a **JSR @XC**.



Consider a print subroutine that we wish to use to output fifty words beginning at TAB. The routine begins at PRT, which address is stored in PRTC in page zero. Our main program would contain this.

```

JSR    @PRTC
...           ;Return here

```

We use AC2 as a base register for counting through the table and AC0 to output the data. The starting address of the table is in TAB1, which is in the vicinity of PRT. The subroutine might look something like this.

```

PRT:    LDA    2,TAB1      ;Set up AC2 as base for table
        LDA    0,0,2      ;Load word for output into AC0
        .           ;IO part of routine here
        .
        ISZ    PRT+1      ;Increment displacement in load instruction
        DSZ    CNT        ;Done yet?
        JMP    PRT+1      ;No, get next word
        JMP    0,3        ;Yes, return by AC3

TAB1:   TAB
CNT:    62                ;628 = 5010

```

This routine is incomplete as it destroys itself; to be used again the displacement in location PRT + 1 must be changed back to zero. If we replaced the ISZ with an arithmetic instruction that increments AC2, thus using AC2 as an index register and leaving the LDA displacement alone, the routine would be complete, as AC2 is set up every time it is called, and except in the Nova, it would be faster. It would be even faster if we deleted the ISZ, stored the address TAB-1 in an autoincrementing location, say 23, and loaded AC0 with

```

LDA    0,@23

```

**Argument Passing.** Suppose we have an arithmetic subroutine that operates on arguments in AC0 and AC1, leaving the result in AC1. The subroutine call looks like this:

```

JSR    VS1              ;Call with arguments in AC0, AC1
...           ;Return here with result in AC1

```

and the subroutine looks like this:

```

VS1:   .                ;Arithmetic operations
        .
        .
        JMP    0,3        ;Return to call + 1

```

In the above the program would have to load the accumulators before calling the routine. Now it is often convenient for the program simply to supply the arguments (or the addresses of the locations that contain them) along with the call and have the subroutine take care of the data transfers. With this version the program gives the arguments in the two memory locations immediately following the JSR,

```

JSR    VS2
...           ;Argument 1
...           ;Argument 2
...           ;Return here with result in AC1

```

and the return is made to the location following the second argument with the result in AC1.

```
VS2:    LDA    0,0,3    ;Pick up argument 1
        LDA    1,1,3    ;Pick up argument 2
        .
        .
        JMP    2,3      ;Return to call + 3
```

This version is called with the addresses of the arguments following the JSR; otherwise it is the same as version 2.

```
        JSR    VS3
        ...
        ...          ;Address of argument 1
        ...          ;Address of argument 2
        .
        .
VS3:    LDA    0,@0,3    ;Pick up argument 1
        LDA    1,@1,3    ;Pick up argument 2
        .
        .
        JMP    2,3      ;Return to call + 3
```

The next version is the same as version 3 except that the result replaces the second argument in memory.

```
        JSR    VS4
        ...
        ...          ;Address of argument 1
        ...          ;Address of argument 2 and result
        .
        .
VS4:    LDA    0,@0,3    ;Pick up arguments
        LDA    1,@1,3
        .
        .
        STA    1,@1,3    ;Store result
        JMP    2,3
```

The final version is the same as the fourth but AC0 and AC1 are not disturbed by its execution. The call is exactly the same as for VS4.

```
VS5:    STA    0,TM0    ;Save ACs
        STA    1,TM1
        LDA    0,@0,3    ;Pick up arguments
        LDA    1,@1,3
        .
        .
```

```

STA      1,@1,3      ;Store result
LDA      0,TM0       ;Restore ACs
LDA      1,TM1
JMP      2,3

TM0:     0           ;Temporary storage for ACs
TM1:     0

```

## 2.2 ARITHMETIC AND LOGICAL INSTRUCTIONS

To perform logical operations the hardware interprets operands as logical words. For arithmetic operations, operands are treated as 16-bit unsigned numbers, with a range of 0 to  $2^{16}-1$ . The program however can interpret them as signed numbers in twos complement notation as described at the beginning of this chapter. It is a property of twos complement arithmetic that operations on signed numbers using twos complement conventions are identical to operations on unsigned numbers; in other words the hardware simply treats the sign as a more significant magnitude bit. Suppose an accumulator contains this binary configuration:

1 000 000 001 011 001
0 <span style="float: right;">15</span>

As an unsigned number this would be equivalent to

$$100131_8 = 32857_{10}$$

whereas interpreted as a signed number using twos complement notation it would be

$$-77647_8 = -32679_{10}$$

Insofar as processor operations are concerned, it makes no difference which way the programmer interprets the contents of the accumulators provided only that he is consistent.

Numbers in twos complement notation are symmetrical in magnitude about a single zero representation so all even numbers both positive and negative end in 0, all odd numbers in 1 (a number all 1s represents  $-1$ ). If ones complements were used for negatives, one could read a negative number by attaching significance to the 0s instead of the 1s. In twos complement notation each negative number is one greater than the complement of the positive number of the same magnitude, so one can read a negative number by attaching significance to the rightmost 1 and attaching significance to the 0s at the left of it (the negative number of largest magnitude has a 1 in only the sign position). Assuming the binary point to be stationary, 1s may be discarded at the left in a negative integer, just as leading 0s may be dropped in a positive integer; equivalently an integer can be extended to the left by prefixing 1s or 0s respectively (*ie* by prefixing the sign). In a negative (proper) fraction, 0s may be discarded at the right; as long as only 0s are discarded, the number remains in twos complement form because it still has a 1 that possesses significance; but if a portion including the rightmost 1 is discarded, the remaining part of the fraction is now a ones complement. Truncation of a negative number thus increases its absolute value.

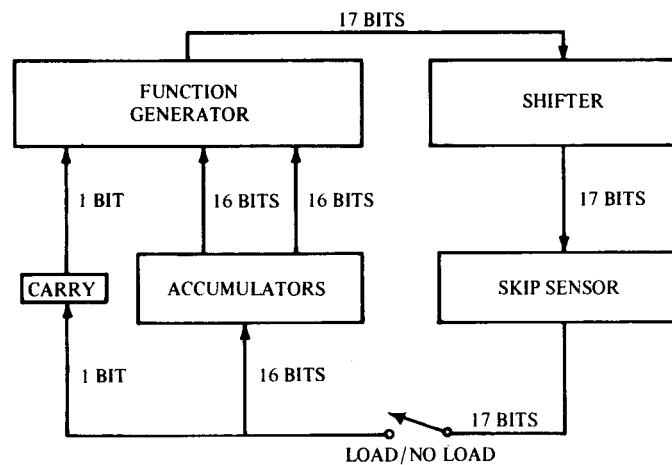
The computer does not keep track of a binary point; the programmer must adopt a point convention and shift the magnitude of the result to conform to the convention used. Two common conventions are to regard a number as an integer (binary point at the right) or as a proper fraction (binary point at the left);

in these two cases the range of signed numbers represented by a single word is  $-2^{15}$  to  $2^{15}-1$  or  $-1$  to  $1-2^{-15}$

Since each bit position represents a binary order of magnitude, shifting a number is equivalent to multiplication by a power of 2, provided of course that the binary point is assumed stationary. Shifting one place to the left multiplies the number by 2. A 0 should be entered at the right, and no information is lost if the sign bit remains the same — a change in the sign indicates that a bit of significance has been shifted out. Shifting one place to the right divides by 2. Truncation occurs at the right, and a bit equal to the sign must be entered at the left.

Associated with the accumulators is the Carry flag, which is used to detect a carry out of bit 0 in an arithmetic operation. The circumstances that generate a carry out of the most significant bit are obvious when dealing with unsigned numbers. If addition or incrementing increases a number beyond  $2^{16}-1$ , a carry is produced. In subtraction the condition is the same if instead of subtracting we add the complement of the subtrahend and add 1 to the result (subtraction is performed by adding the two's complement). In terms of the original operands the subtraction  $A-B$  produces a carry if  $A \geq B$ . Forming the two's complement of zero generates a carry, for complementing zero produces a word containing all 1s, and adding 1 to that produces all 0s again plus a carry. The statement of the carry conditions in terms of signed numbers is more complex, but they are always exactly equivalent to the conditions given above if the numbers are simply interpreted as unsigned. In any event the complete conditions that produce a carry for numbers signed or unsigned are given in the instruction descriptions.

**Arithmetic and Logical Processing.** The logical organization of the arithmetic unit is illustrated below. Each instruction specifies one or two accumulators to supply operands to the function generator, which performs the function specified by the instruction. The function generator also produces a carry bit whose value depends upon three quantities: a base value specified by the instruction, the function performed, and the result obtained. The base value may be derived from the Carry flag, or the instruction may specify an independent value.

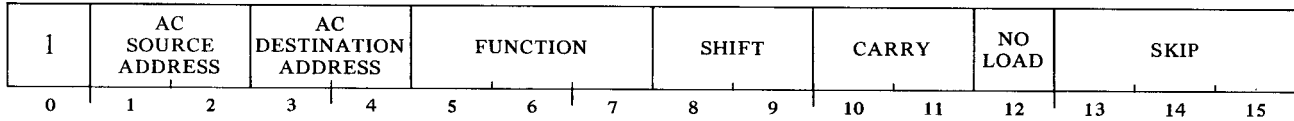


ORGANIZATION OF ARITHMETIC UNIT

The 17-bit output of the function generator, comprising the carry bit and the 16-bit function result, then goes to the shifter. Here the 17-bit result can be rotated one place right or left, or the two 8-bit halves of the 16-bit function result can be swapped without affecting the carry bit. The 17-bit shifter output can then be tested for a skip; the skip sensor can test whether the carry bit or the rest of the 17-bit word is or is not equal to zero. Finally the 17-bit shifted word can be loaded into Carry and one of the accumulators selected by the instruction. Note however that loading is not necessary: an instruction can perform a complicated arithmetic and shifting operation and test the result for a skip without affecting Carry or any accumulator.

### Carry, Shift and Skip Functions

An instruction that has a 1 in bit 0 performs one of eight arithmetic and logical functions as specified by bits 5–7 of the instruction word. The function, which may be anything from a simple move to a subtraction, always uses the contents of the accumulator specified by bits 1 and 2; and if a second operand is required, it comes from the accumulator addressed by bits 3 and 4.



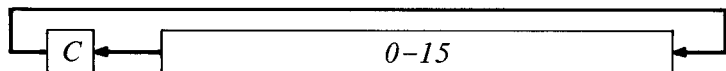
The instruction also supplies a carry bit to the shifter with the result. Bits 10 and 11 specify a base value to be used in determining the carry bit. The instruction supplies either this value or its complement depending upon both the function being performed and the result it generates. The mnemonics and bit configurations and the base values they select are as follows.

<i>Mnemonic</i>	<i>Bits 10–11</i>	<i>Base value for carry bit</i>
	00	Current state of Carry
Z	01	Zero
O	10	One
C	11	Complement of current state of Carry

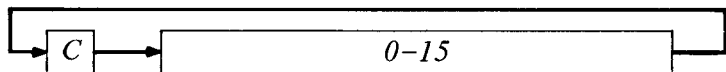
The three logical functions simply supply the listed values as the carry bit to the shifter. The five arithmetic functions supply the complement of the base value if the operation produces a carry out of bit 0; otherwise they supply the value given. The carry bit can be used in conjunction with the sign of the result to detect overflow in operations on signed numbers. But its primary use is as a carry out of the most significant bit in operations on unsigned numbers, such as the lower order parts in multiple precision arithmetic.

The 17-bit word consisting of the carry bit and the 16-bit result is operated on by the shifter as specified by bits 8 and 9.

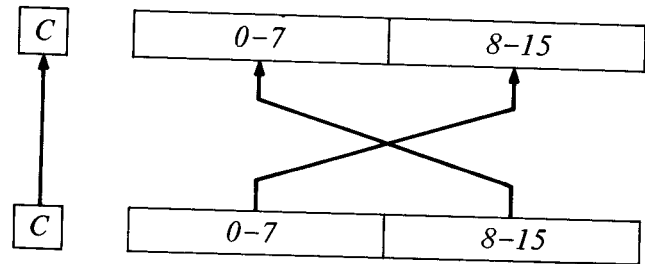
<i>Mnemonic</i>	<i>Bits 8–9</i>	<i>Shift operation</i>
	00	None
L	01	Left rotate one place. Bit 0 is rotated into the carry position, the carry bit into bit 15.



R	10	Right rotate one place. Bit 15 is rotated into the carry position, the carry bit into bit 0.
---	----	--



Swap the halves of the 16-bit result. The carry bit is not affected.



The 17-bit output of the shifter is loaded into Carry and the accumulator addressed by instruction bits 3 and 4 provided bit 12 is 0. A 1 programmed in bit 12 inhibits the loading and prevents the instruction from affecting Carry or the accumulator. Note that it is the shifted result that is loaded: AC receives the result of the function and Carry the carry bit only if bits 8 and 9 are 0.

The shifter output is also tested for a skip according to the condition specified by bits 13–15. The processor skips the next instruction if the specified condition is satisfied.

<i>Bit</i>	<i>Effect of a 1 in the bit</i>
13	Selects the condition that the low order 16 bits of the shifter output are all 0.
14	Selects the condition that the bit in the carry position of the shifter output is 0.
15	Inverts the conditions selected by bits 13 and 14. In other words a 1 in bit 15 causes 1s in the other bits to select nonzero conditions.

The combined effects of bits 13–15 taken together and the mnemonics for the various bit configurations are as follows.

<i>Mnemonic</i>	<i>Bits 13–15</i>	<i>Skip function</i>
	0	Never Skip
SKP	1	Always Skip
SZC	2	Skip on Zero Carry
SNC	3	Skip on Nonzero Carry
SZR	4	Skip on Zero Result
SNR	5	Skip on Nonzero Result
SEZ	6	Skip if Either Carry or Result is Zero
SBN	7	Skip if Both Carry and Result are Nonzero

Remember that the test is made on the shifter output. Thus if the result of an addition is shifted left, SZC and SNC actually test the sign of the sum. Note also that the test is made whether or not the shifter output is loaded. The program can therefore test the result of an arithmetic function without disturbing the original operands or Carry.

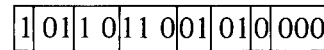
**Programming Conventions.** The instruction

ADD 1,2

1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

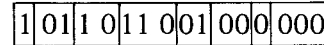
which assembles as 133000, adds the numbers in AC1 and AC2, loads the unshifted result in AC2, and complements Carry if there is a carry out of bit 0. Other carry and shift operations are selected simply by appending the appropriate letters to the function mnemonic, but the carry letter (if any) must appear first. Thus to generate a carry bit of 1 on a carry (0 otherwise) and load Carry and AC2 with the 17-bit result shifted left we give

ADDZL 1,2



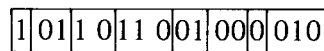
which assembles as 133120. This instruction places the sign of the sum in Carry, the rest of the sum in bits 0–14 of AC2, and a 1 or a 0 in bit 15 depending on whether or not there is a carry out of the sign bit. To use the present state of Carry instead of 0 as the basis for adjusting bit 15, but otherwise produce the same effect, give

ADDL 1,2



which assembles as 133100. The instruction

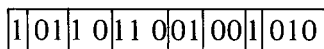
ADDL 1,2,SZC



assembles as 133102, and affects Carry and AC2 in the same manner as the preceding instruction, but also causes the processor to skip the next instruction if the sign of the sum is positive.

The initial symbol # following the expanded function mnemonic places a 1 in bit 12 to prevent the loading of the shifter output. Hence we can skip the next instruction on a positive sum without affecting AC2 or Carry by giving

ADDL# 1,2,SZC

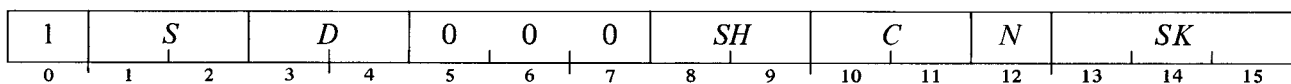


which assembles as 133112.

### Arithmetic and Logical Functions

The eight functions are selected by bits 5–7 of the instruction word. For convenience the source and destination accumulators addressed by the *S* and *D* parts of the instruction are referred to as *ACS* and *ACD*.

#### COM Complement



Place the (logical) complement of the word from *ACS* and place the carry bit specified by *C* in the shifter. Perform the shift operation specified by *SH*. Load the shifter output in Carry and *ACD* unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

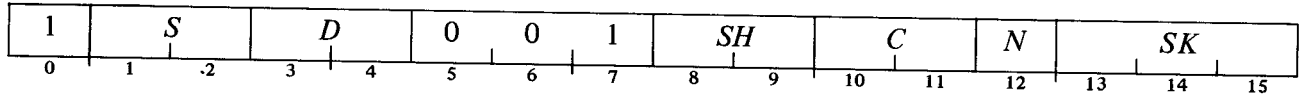
EXAMPLE. Suppose we wish to test AC1 for the unsigned integer  $2^{16}-1$  (177777, signed  $-1$ ). The instruction

COM# 1,1,SZR

skips the next instruction if AC1 contains all 1s. The result is not loaded so we could specify any accumulator as the destination, *eg*

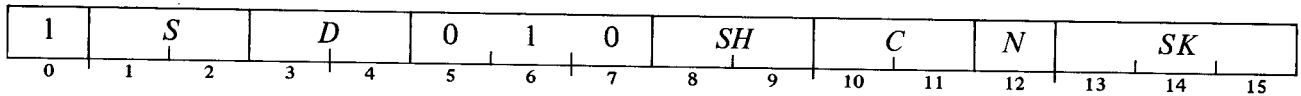
COM# 1,3,SZR

**NEG Negate**



Place the two's complement of the number from ACS into the shifter. If ACS contains zero, supply the complement of the value specified by *C* as the carry bit; otherwise supply the specified value. Perform the shift operation specified by *SH*. Load the shifter output in Carry and ACD unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

**MOV Move**



Place the contents of ACS and the carry bit specified by *C* in the shifter. Perform the shift operation specified by *SH*. Load the shifter output in Carry and ACD unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

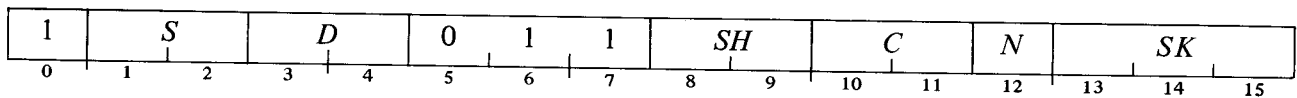
EXAMPLES. The test for a zero word in AC1 is any of these:

MOV 1,1,SZR      MOV 1,1,SNR      MOV# 1,1,SZR      MOV# 1,1,SNR

Suppose we wish to divide the number in AC2 by 2.

MOVL# 2,2,SZC      ;Is it positive?  
 MOVOR 2,2,SKP      ;No, put in a 1 and skip  
 MOVZR 2,2      ;Yes, put in a 0

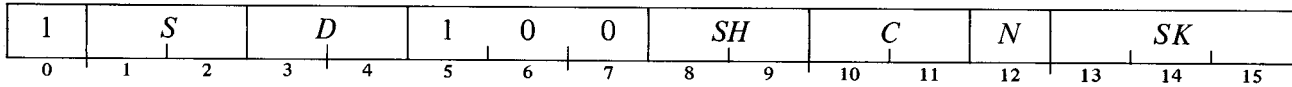
**INC Increment**



Add 1 to the number from ACS and place the result in the shifter. If ACS contains  $2^{16}-1$  (signed  $-1$ ) supply the complement of the value specified by *C* as the carry bit; otherwise supply the specified value. Perform the shift operation specified by *SH*. Load the shifter output in Carry and ACD unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.



**ADC          Add Complement**

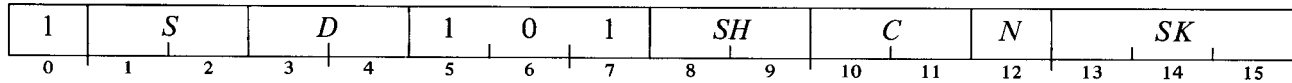


Add the (logical) complement of the number from *ACS* to the number from *ACD*, and place the result in the shifter. If  $ACD > ACS$  (unsigned), supply the complement of the value specified by *C* as the carry bit; otherwise supply the specified value. Perform the shift operation specified by *SH*. Load the shifter output in Carry and *ACD* unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

**NOTE:** For signed numbers the carry condition is that the signs of the operands are the same and *ACD* is the greater, or the signs differ and *ACD* is negative.

This instruction is often used to process high order words in multiple precision subtraction, wherein a negative is usually a ones complement instead of a twos complement. The overflow condition for signed numbers using ones complement conventions is the same as that given for *SUB* below.

**SUB          Subtract**



Subtract by adding the twos complement of the number from *ACS* to the number from *ACD*, and place the result in the shifter. If  $ACD \geq ACS$  (unsigned), supply the complement of the value specified by *C* as the carry bit; otherwise supply the specified value. Perform the shift operation specified by *SH*. Load the shifter output in Carry and *ACD* unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

**NOTE:** For signed numbers the carry condition is that the signs of the operands are the same and  $ACD \geq ACS$ , or the signs differ and *ACD* is negative.

**EXAMPLES.** This instruction can be used to clear an accumulator by subtracting it from itself.

**SUB      2,2**

clears AC2 and complements Carry,

**SUBO    2,2**

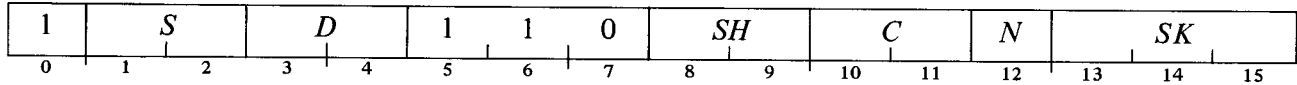
clears both AC2 and Carry.

*SUB* is also useful for comparing quantities, *eg*

**SUB#    2,3,SNR**

skips if AC2 and AC3 are unequal but does not affect either accumulator.

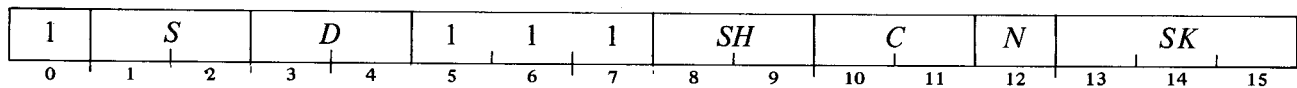
**ADD      Add**



Add the number from *ACS* to the number from *ACD*, and place the result in the shifter. If the unsigned sum is  $\geq 2^{16}$ , supply the complement of the value specified by *C* as the carry bit; otherwise supply the specified value. Perform the shift operation specified by *SH*. Load the shifter output in Carry and *ACD* unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

NOTE: For signed numbers the carry condition is that both summands are negative, or their signs differ and their magnitudes are equal or the positive one is the greater in magnitude.

**AND      And**



Place the logical AND function of the word from *ACS* and the word from *ACD* in the shifter. Supply the value specified by *C* as the carry bit. Perform the shift operation specified by *SH*. Load the shifter output in Carry and *ACD* unless *N* is 1. Skip the next instruction if the shifter output satisfies the condition specified by *SK*.

This instruction operates bitwise on a pair of words, so it actually performs sixteen logical operations simultaneously. A given bit of the result is 1 if the corresponding bits of both operands are 1; otherwise the resulting bit is 0.

<i>ACS<sub>i</sub></i>	<i>ACD<sub>i</sub></i>	<i>Result<sub>i</sub></i>
0	0	0
0	1	0
1	0	0
1	1	1

**Programming Examples**

Together *ADC* and *SUB* allow the program to compare the magnitudes of unsigned numbers in every way. *Eg*

SUBZ# 1,0,SZC

skips if  $AC0 < AC1$ , whereas,

ADCZ# 1,0,SZC

skips if  $AC0 \leq AC1$ .

It is well known that the *n*th perfect square is the sum of the first *n* odd numbers. We can therefore find the largest integer contained in the square root of an integer held in *AC0* by successively subtracting odd numbers in order from *AC0* until overflow occurs, *ie* until *AC0* becomes negative. The desired answer is the number of odd numbers successfully subtracted before a carry occurs. The routine is called by a *JSR* with effective address *SQRT*.

SQRT:	SUBO	1,1	;Clear AC1 and Carry
	MOVOL	1,2	;AC2 gets 1 + twice AC1 (2n + 1)
	SUBZ	2,0,SNC	;Subtract next odd number; still positive?
	JMP	0,3	;No, exit with n one less than number of odd numbers tried
	INC	1,1	;Yes, increment n
	JMP	SQRT+1	;and try next odd number

The instruction set has only one logical function of two variables, but the inclusive and exclusive OR functions can be performed by very simple routines. In an inclusive OR a bit of the result is 1 if either of the corresponding operand bits is 1, otherwise it is 0. The algorithm for full words is

$$A \wedge \sim B + B = A \vee B$$

Taking the arguments as single bits, if  $B$  is 1,  $A \wedge \sim B$  is 0 regardless of the state of  $A$ , and the expression on the right is 1. If  $B$  is 0, the expression is 1 or 0 as  $A$  is 1 or 0. In no case are  $A \wedge \sim B$  and  $B$  both 1, so the full word addition generates no carries. This sequence places the inclusive OR of AC0 and AC1 in AC1 (AC0 =  $B$ , AC1 =  $A$ ).

COM	0,0	;~B
AND	0,1	;~B ^ A in AC1
ADC	0,1	;~~B + ~B ^ A = B + ~B ^ A in AC1

In an exclusive OR a bit of the result is 1 if the corresponding operand bits are different, otherwise it is 0. This is equivalent to the sum if carries from one bit position to the next are ignored. Now a carry out of the  $i$ th position is equal to twice the value of a 1 in the  $i$ th position, and a carry is generated only if the  $i$ th bits of both summands are 1, provided we compensate for any carry into the  $i$ th position. The algorithm is therefore.

$$A \vee B = A + B - 2(A \wedge B)$$

This sequence places the exclusive OR of AC0 and AC1 in AC1, destroying the contents of AC2 and Carry in the process (AC0 =  $B$ , AC1 =  $A$ ).

MOV	1,2	;Move A to AC2
ANDZL	0,2	;2(A ^ B) in AC2
ADD	0,1	;A + B
SUB	2,1	;A + B - 2(A ^ B)

**Double Precision Arithmetic.** A double length number consists of two words concatenated into a 32-bit string wherein bit 0 is the sign and bits 1–31 are the magnitude in twos complement notation. The high order part of a negative number is therefore in ones complement form unless the low order part is null (at the right

+262,146 <sub>10</sub>	=	+2000002 <sub>8</sub>	=	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">0 000 000 000 001 000</td> <td style="padding: 2px 10px;">0 000 000 000 000 010</td> </tr> <tr> <td style="text-align: center; padding: 0 5px;">0</td> <td style="text-align: center; padding: 0 5px;">15 16</td> <td style="text-align: center; padding: 0 5px;">31</td> </tr> </table>	0 000 000 000 001 000	0 000 000 000 000 010	0	15 16	31
0 000 000 000 001 000	0 000 000 000 000 010								
0	15 16	31							
-262,146 <sub>10</sub>	=	-2000002 <sub>8</sub>	=	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">1 111 111 111 110 111</td> <td style="padding: 2px 10px;">1 111 111 111 111 110</td> </tr> <tr> <td style="text-align: center; padding: 0 5px;">0</td> <td style="text-align: center; padding: 0 5px;">15 16</td> <td style="text-align: center; padding: 0 5px;">31</td> </tr> </table>	1 111 111 111 110 111	1 111 111 111 111 110	0	15 16	31
1 111 111 111 110 111	1 111 111 111 111 110								
0	15 16	31							

only 0s are null regardless of sign). Hence in processing double length numbers, twos complement operations are usually confined to the low order parts, whereas ones complement operations are generally required for the high order parts.

Suppose we wish to negate the double length number whose high and low order words respectively are in AC0 and AC1. We negate the low order part, but we simply complement the high order part unless the low order part is zero. Hence

```

NEG      1,1,SNR
NEG      0,0,SKP      ;Low order zero
COM      0,0          ;Low order nonzero

```

Note that the magnitude parts of the sequence of negative numbers from the most negative toward zero are the positive numbers from zero upward. In other words the negative representation  $-x$  is the sum of  $x$  and the most negative number. Hence in multiple precision arithmetic, low order words can be treated simply as positive numbers. In unsigned addition a carry indicates that the low order result is just too large and the high order part must be increased. We add the number in AC2 and AC3 to the number in AC0 and AC1.

```

ADDZ     3,1,SZC
INC      2,2
ADD      2,0

```

In twos complement subtraction a carry should occur unless the subtrahend is too large. We could increment as in addition, but since incrementing in the high order part is precisely the difference between a ones complement and a twos complement, we can always manage with only two instructions. We subtract the number in AC2 and AC3 from that in AC0 and AC1.

```

SUBZ     3,1,SZC
SUB      2,0,SKP
ADC      2,0

```

**Multiply and Divide Subroutines.** In pencil and paper decimal multiplication, one multiplies the multiplicand by each multiplier digit separately to form a set of partial products. Successive partial products are shifted one place to the left (they are multiplied by successive powers of 10) and summed. In the computer it is easier to add each partial product as it is formed and shift the result one place to the right so the running sum is in the correct position to receive the next one. Since the numbers are binary, each partial product is either the multiplicand or zero. Hence at each step we either add the multiplicand and shift or simply shift depending on whether the next bit of the multiplier is 1 or 0.

The multiply subroutine operates on unsigned integers in AC1 and AC2 to generate a double length product whose high and low order parts are left in AC0 and AC1 respectively. If entry is made at .MPYA, the product is added to the number originally in AC0 (the result is  $AC0 + AC1 \times AC2$ ). Carry is left unchanged.

```

.MPYU:   SUBC      0,0          ;Clear AC0, don't disturb Carry
.MPYA:   STA       3,.CB03     ;Save return
         LDA       3,.CB20     ;Get step count

.CB99:   MOVR      1,1,SNC     ;Check next multiplier bit
         MOVR      0,0,SKP     ;0 - shift

```

```

        ADDZR  2,0           ;1 – add multiplicand and shift
        INC    3,3,SZR      ;Count step, complementing Carry on final count
        JMP    .CB99        ;Iterate loop

        MOVCR  1,1         ;Shift in last low bit (which was complemented by final count) and
        JMP    @.CB03       ;restore Carry

.CB03:  0
.CB20:  -20                ;1610 steps

```

The divide subroutine also operates on unsigned integers, using a double length dividend and a single length divisor to produce a single length quotient and remainder. The routine starts by comparing the divisor with the high order half of the dividend: if the divisor is less than or equal to the latter quantity, the division is not performed as the result would be greater than  $2^{16}-1$ , the largest integer than can be held in an accumulator. (The result would be greater than or equal to 1 if the operands are interpreted as proper fractions.) It is not a sensible procedure simply to compute the first sixteen bits of the quotient as it would be impossible to determine the order of magnitude. So it is up to the programmer to shift the dividend to the correct position beforehand. For operations limited to single length integers (referred to as “integer division”) the one-word dividend is treated as the low order half of a double length number whose high order part is null, and the routine fails to perform the division only when the divisor is zero. The worst possible case is the division of  $2^{16}-1$  by 1, whose integral result can be accommodated.

In division on paper, one subtracts out the divisor the number of times it goes into the dividend, then shifts the dividend one place to the left (or the divisor to the right) and again subtracts out. In binary computations the divisor goes into the dividend either once or not at all. Each comparison thus generates a single bit of the quotient. If the divisor does go in, it is subtracted and a 1 is entered into the quotient; if not, a 0 is entered. The test condition is reversed if the dividend shift puts a 1 in Carry; this way Carry is used effectively as an extra magnitude bit and no information is lost in the shift.

The high and low parts of the dividend are in AC0 and AC1, the divisor is in AC2. At completion the remainder is in AC0, the quotient is in AC1, AC2 is unchanged, and Carry is left clear. For integer division entry is at .DIVI with the dividend in AC1. If the division is not performed, Carry is set and the three accumulators are unchanged except that calling .DIVI clears AC0. Note that the result is such that if .MPYA is called, AC0 and AC1 are restored, *ie* divisor times quotient plus remainder equals original dividend. For further information see the subroutine writeup, 093-000016.

```

.DIVI:  SUB    0,0           ;Integer divide – clear high part
.DIVU:  STA    3,.CC03      ;Save return
        SUBZ#  2,0,SZC      ;Test for overflow
        JMP    .CC99        ;Yes, exit (AC0 ≥ AC2)
        LDA    3,.CC20      ;Get step count
        MOVZL  1,1         ;Shift dividend low part

.CC98:  MOVL   0,0           ;Shift dividend high part
        SUB#   2,0,SZC      ;Does divisor go in?
        SUB    2,0           ;Yes
        MOVL   1,1         ;Shift dividend low part
        INC    3,3,SZR      ;Count step
        JMP    CC98         ;Iterate loop

```

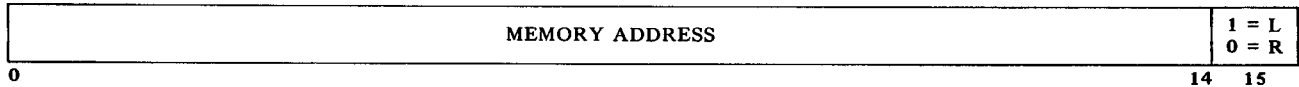
```

SUB0    3,3,SKP    ;Done, clear Carry
.CC99:  SUBZ    3,3    ;Set Carry
        JMP    @.CC03 ;Return

.CC03:  0
.CC20:  -20        ;1610 steps

```

**Byte Manipulation.** For processing 8-bit bytes it is convenient to use a byte pointer in which bits 0–14 are the address of the memory location that contains or will receive the byte, and bit 15 specifies which half (1 left,



0 right). Incrementing a pointer with this format changes bit 15 every count to specify the next byte, but changes the address part only every other count.

The following subroutine picks up a byte, places it in the right half of AC0, and increments the byte pointer in memory. The calling sequence is

```

JSR     PICK
...     ;Address of pointer
...     ;Return here if byte is zero
...     ;Normal return

```

The calling sequence supplies the address of the location containing the pointer. A separate return for a zero byte allows the program to process a sequence of bytes whose length is unspecified, but which terminates with a zero byte.

```

PICK:   LDA    2,@0,3    ;Get byte pointer
        ISZ    @0,3      ;Increment pointer
        MOVZR  2,2       ;Put address in right place (left/right bit to Carry)
        LDA    0,0,2     ;Bring memory word to AC0
        LDA    2,C377    ;Get 8-bit mask
        MOV    0,0,SZC   ;Test Carry for which half
        MOVS   0,0       ;Swap byte from left to right
        AND    2,0,SNR   ;Mask out unwanted byte and test for zero
        JMP    1,3       ;Zero, return to call + 2
        JMP    2,3       ;Nonzero, return to call +3
C377:   377            ;8-bit mask (1s in right half)

```

### 2.3 INPUT-OUTPUT

Instructions in the in-out class govern all transfers of data to and from the peripheral equipment, and also perform various operations within the processor. An instruction in this class is designated by 011 in bits 0–2. Bits 10–15 select the device that is to respond to the instruction. The format thus allows for 64 codes of which 62 can be used to address devices (octal 01–76). The code 00 is not used, and 77 is used for a number of special functions including reading the console data switches and controlling the program interrupt. A table in

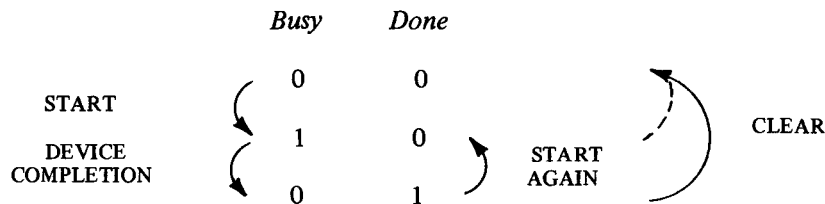
Appendix E lists all devices for which codes have been assigned, and gives their mnemonics and DGC option numbers.

Every device has a 6-bit device selection network, an Interrupt Disable flag, and Busy and Done flags. The selection network decodes bits 10–15 of the instruction so that only the addressed device responds to signals sent by the processor over the in-out bus. The Busy and Done flags together denote the basic state of the device. When both are clear the device is idle. To place the device in operation, the program sets Busy. If the device will be used for output, the program must give a data-out instruction that sends the first unit of data — a word or character depending on how the device handles information. (The word “output” used without qualification always refers to the transfer of data from the processor to the peripheral equipment; “input” refers to the transfer in the opposite direction.) When the device has processed a unit of data, it clears Busy and sets Done to indicate that it is ready to receive new data for output, or that it has data ready for input. In the former case the program would respond with a data-out instruction to send more data; in the latter with a data-in instruction to bring in the data that is ready. If the Interrupt Disable flag is clear, the setting of Done signals the program by requesting an interrupt; if the program has set Interrupt Disable, then it must keep testing Done or Busy to determine when the device is ready.

In all in-out instructions bits 8 and 9 either control or sense Busy and Done. In those instructions in which bits 8 and 9 specify a control function, the mnemonics and bit configurations and the functions they select are as follows.

<i>Mnemonic</i>	<i>Bits 8–9</i>	<i>Control function</i>
	00	None
S	01	Start the device by clearing Done and setting Busy
C	10	Clear both Busy and Done, idling the device
P	11	Pulse the special in-out bus control line — the effect, if any, depends on the device

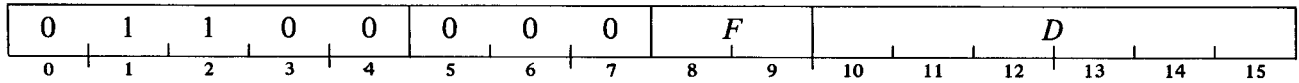
The overall sequence of Busy and Done states is determined by both the program and the internal operation of the device.



The data-in or data-out instruction that the program gives in response to the setting of Done can also restart the device. When all the data has been transferred the program generally clears Done so the device neither requests further interrupts nor appears to be in use, but this is not necessary. Busy and Done both set is a meaningless situation.

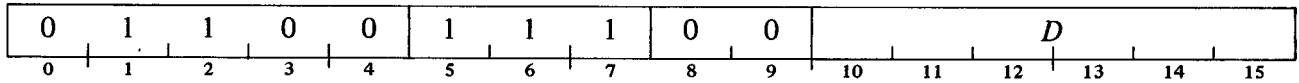
Bits 5–9 specify the complete function to be performed. If there is no transfer (bits 5–7 all alike), bits 3 and 4 are ignored and bits 8 and 9 may specify a control function or a skip condition.

**NIO      No IO Transfer**



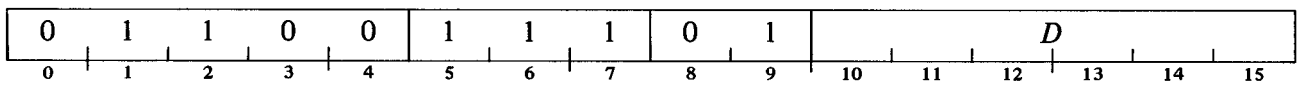
Perform the control function specified by *F* in device *D*.

**SKPBN      Skip if Busy is Nonzero**



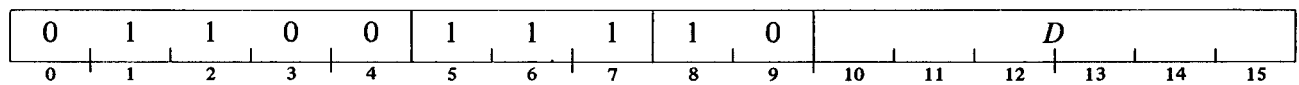
Skip the next instruction in sequence if the Busy flag in device *D* is 1.

**SKPBZ      Skip if Busy is Zero**



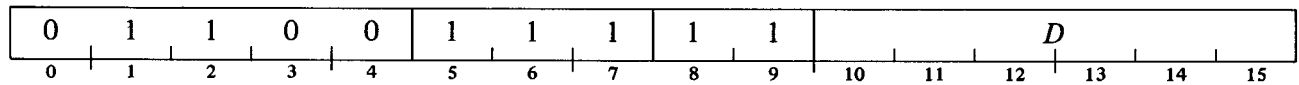
Skip the next instruction in sequence if the Busy flag in device *D* is 0.

**SKPDN      Skip if Done is Nonzero**



Skip the next instruction in sequence if the Done flag in device *D* is 1.

**SKPDZ      Skip if Done is Zero**



Skip the next instruction in sequence if the Done flag in device *D* is 0.

The letter for the control function is appended to the basic mnemonic; NIO alone with any device code is a no-op. To place say the high speed tape reader in operation we could give

NIOS      12

which assembles as 060112 (0 110 000 001 001 010) and causes the reader to read one line from tape into its buffer. There are mnemonics for the device codes so we could also give the equivalent

NIOS      PTR

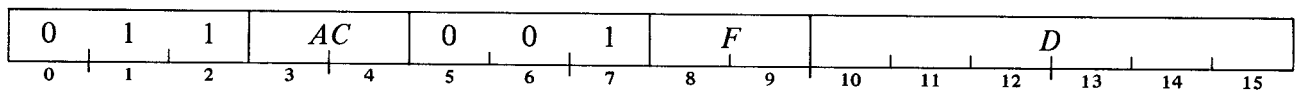


To determine when the character is in the buffer without using the program interrupt we can wait for either Busy to clear or Done to set, eg by giving

```
SKPDN  PTR
JMP    .-1
```

If bits 5–7 are not all alike the instruction calls for an in-out transfer. Bits 3 and 4 specify the accumulator that supplies or receives the data, bits 8 and 9 specify a control function (if any) as listed above.

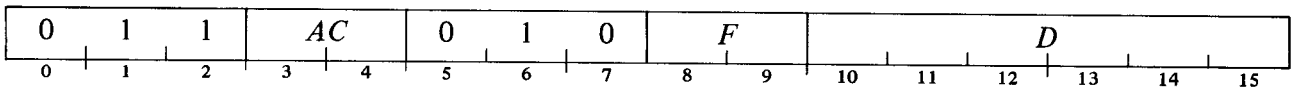
**DIA Data In A**



Move the contents of the A buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

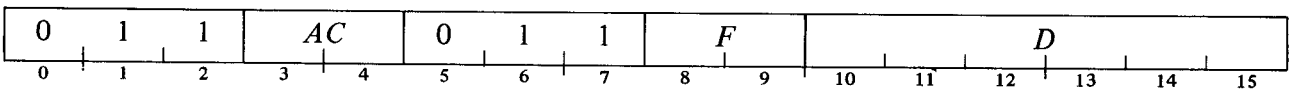
**DOA Data Out A**



Send the contents of accumulator *AC* to the A buffer in device *D*, and perform the function specified by *F* in device *D*.

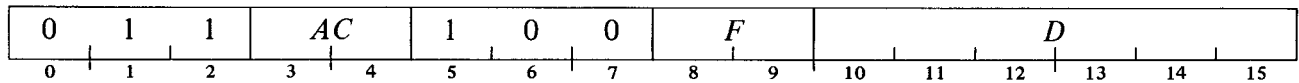
The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

**DIB Data in B**



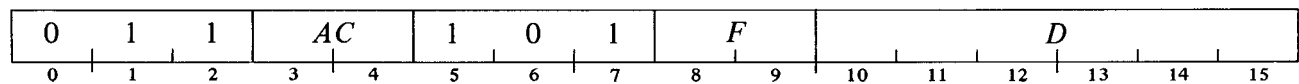
Move the contents of the B buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

**DOB      Data Out B**

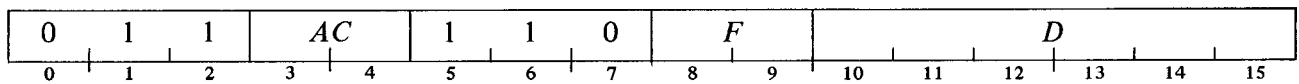
Send the contents of accumulator *AC* to the B buffer in device *D*, and perform the function specified by *F* in device *D*.

The amount of data actually accepted by the device depends on the size its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

**DIC      Data in C**

Move the contents of the C buffer in device *D* to accumulator *AC*, and perform the function specified by *F* in device *D*.

The number of data bits moved depends on the size of the device buffer, its mode of operation, etc. Bits in *AC* that do not receive data are cleared.

**DOC      Data Out C**

Send the contents of accumulator *AC* to the C buffer in device *D*, and perform the function specified by *F* in device *D*.

The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of *AC* are unaffected.

A device may require no IO transfers, such as the real time clock, which uses only NIOS and NIOC to turn it on and off. All of the simpler data handling devices have only an A buffer, *eg* to hold a single character in the teletypewriter, tape reader and tape punch, or to receive incremental plotting data for a single point in the plotter. Suppose the reader has read a line from tape into its buffer. We can bring the character into the right half of AC2 by giving

DIA      2,PTR

If we want to read another line we can make the transfer with a

DIAS     2,PTR

which brings the character into AC2, clears Done and sets Busy causing the reader to read the next line. If the buffer contains the final character to be read from tape we might give

DIAC     2,PTR

which retrieves the character and clears Done. Data is moved in and out in characters of various sizes or in

full 16-bit words. Generally a device uses only DIA and/or DOA for data but it may use other IO transfer instructions to handle status and control information. A high speed device, such as magnetic tape or disk, may require IO transfer instructions *only* for status and control information with data moving directly between the device and memory via the data channel.

Most peripheral devices involve motion of some sort, usually mechanical. In this respect there are two types of devices, those that stay in motion and those that do not. Magnetic tape is an example of the former type. Here the device executes a command (such as read, write, space forward) and Done sets when the entire operation is finished. A separate flag requests a data channel transfer each time the device is ready for direct data access to memory, but the tape keeps moving until an entire record or file has been processed. Paper tape, on the other hand, stops after each line is read, but if the program gives another DIAS within a critical time the tape moves continuously.

Other devices operate in one or the other of these two ways but differ in various respects. The tape punch and teletype output are like the reader. Teletype input is initiated by the operator striking a key rather than by the program. Once started the card reader reads an entire card, with a DIA required for each column.

In the remainder of this manual the discussion of each device treats only the control functions and the applicable IO transfer instructions. The skips apply to all and are the same in all cases. Giving a data-in instruction that does not apply to a device (either because the device is output only or does not have the buffer specified) clears the addressed accumulator but does do the specified control function. Similarly a data-out that does not apply is a no-op except for control functions. When the device code is undefined or the addressed device is not in the system, any data-out, an SKPBN or an SKPDN is a no-op, an SKPBZ or SKPDZ is an absolute skip, and any data-in simply clears the addressed AC.

All instructions discussed in the rest of this manual are in-out instructions with various device codes. For the transfer instructions the mnemonics are given with a dash in the position occupied by an accumulator address, as the assembler indicates an error if the programmer fails to specify an accumulator. The programmer must substitute a valid address for the dash. In the format box for each instruction the accumulator address part is represented by *AC*. In the instruction description, "AC" refers to the accumulator specified by the *AC* part of the instruction word.

### Special Code-77 Functions

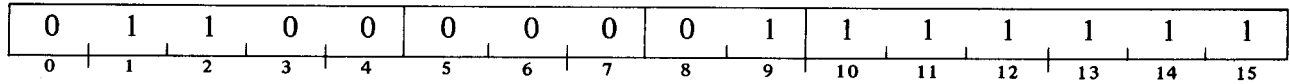
In-out instructions with the code 77 in bits 10–15 perform a number of special functions rather than controlling a specific device. In all but the skip instructions bits 8 and 9 are used to turn the interrupt on and off. The mnemonics are the same as those for controlling Busy and Done in IO devices, but with code 77 they select the following special functions.

<i>Mnemonic</i>	<i>Function</i>
S	Set the Interrupt On flag to enable the processor to respond to interrupt requests.
C	Clear the Interrupt On flag to prevent the processor from responding to interrupt requests.
P	None

Most of these instructions perform functions associated with processor elements so the mnemonic for 77 is CPU. For the transfer type instructions that use no accumulator, the mnemonics are given with an accumulator address included, as the assembler indicates an error if the programmer fails to specify an accumulator

even when none is used. A zero address is given, but any valid address would suffice. Instructions for the program interrupt and power failure detection are treated in greater detail in later sections.

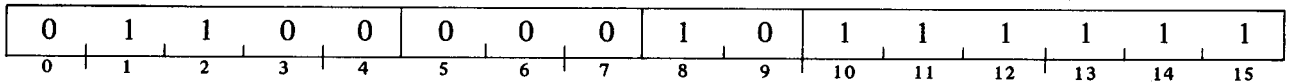
**NIOS CPU            Interrupt Enable**



Set the Interrupt On flag to allow the processor to respond to interrupt requests.

NOTE: The assembler recognizes the mnemonic INTEN as equivalent to NIOS CPU.

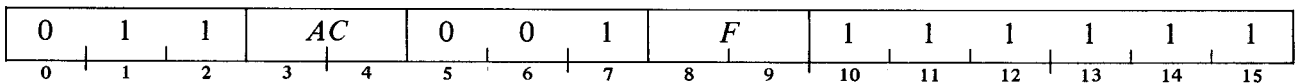
**NIOC CPU            Interrupt Disable**



Clear the Interrupt On flag to prevent the processor from responding to interrupt requests.

NOTE: The assembler recognizes the mnemonic INTDS as equivalent to NIOC CPU.

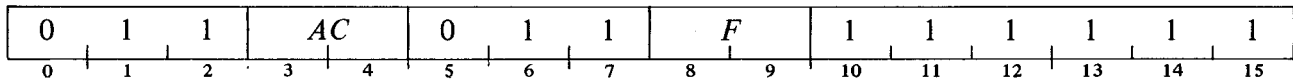
**DIA -,CPU            Read Switches**



Read the contents of the console data switches into AC, and perform the function specified by *F*.

NOTE: The assembler recognizes the mnemonic READS as equivalent to DIA -,CPU.

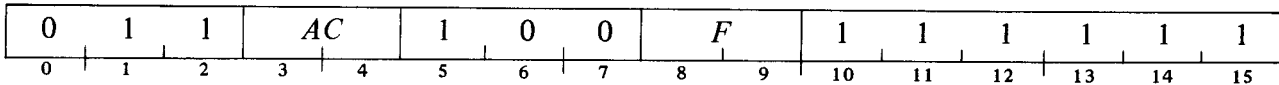
**DIB -,CPU            Interrupt Acknowledge**



Place in AC bits 10–15 the device code of the first device on the bus that is requesting an interrupt (“first” means the one that is physically closest to the processor on the bus). Perform the function specified by *F*.

NOTE: The assembler recognizes the mnemonic INTA as equivalent to DIB -,CPU.

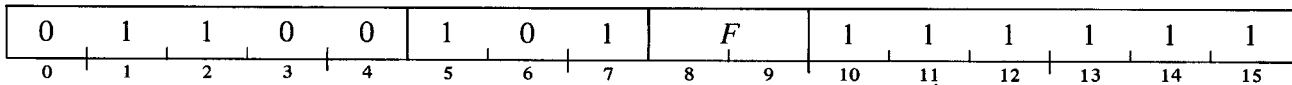
**DOB -,CPU      Mask Out**



Set up the Interrupt Disable flags in the devices according to the mask in AC. For this purpose each device is connected to a given data line, and its flag is set or cleared as the corresponding bit in the mask is 1 or 0. Perform the function specified by *F*.

NOTE: The assembler recognizes the mnemonic MSKO as equivalent to DOB -,CPU.

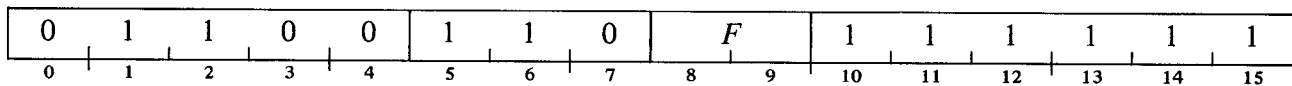
**DIC 0,CPU      Clear IO Devices**



Clear the control flipflops, including Busy, Done and Interrupt Disable, in all devices connected to the bus. Perform the function specified by *F*.

NOTE: The assembler recognizes the mnemonic IORST as equivalent to DICC 0,CPU — *ie* as the instruction defined here with *F* set to 10.

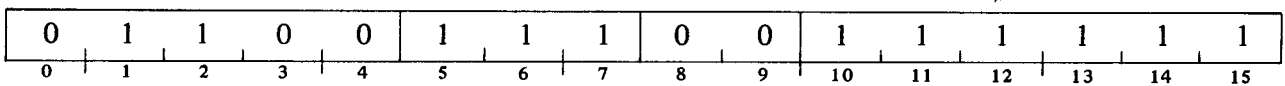
**DOC 0,CPU      Halt**



Perform the function specified by *F* and then halt the processor. When the processor stops, the instruction and data lights display the halt instruction, the address lights point to the location following the halt instruction.

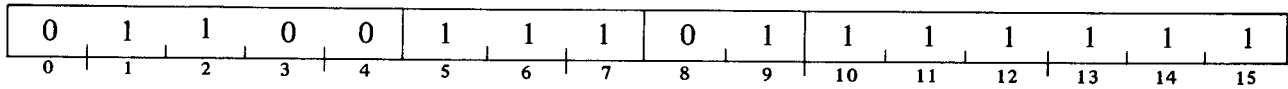
NOTE: The assembler recognizes the mnemonic HALT as equivalent to DOC 0,CPU.

**SKPBN CPU      Skip if Interrupt On is Nonzero**



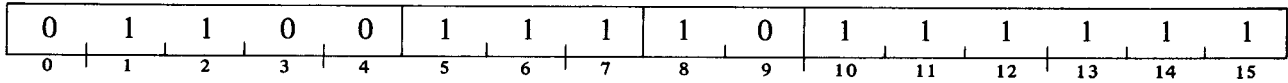
Skip the next instruction in sequence if the Interrupt On flag is 1.

**SKPBZ CPU      Skip if Interrupt On is Zero**



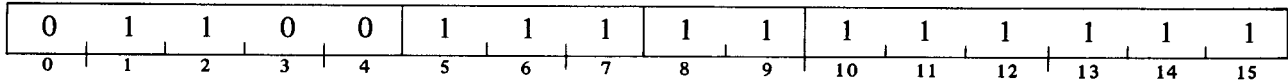
Skip the next instruction in sequence if the Interrupt On flag is 0.

**SKPDN CPU      Skip if Power Failure is Nonzero**



Skip the next instruction in sequence if the Power Failure flag is 1.

**SKPDZ CPU      Skip if Power Failure is Zero**



Skip the next instruction in sequence if the Power Failure flag is 0.

The assembler recognizes a number of convenient mnemonics for instructions with device code 77.

<i>Mnemonic</i>	<i>Meaning</i>	<i>Mnemonic Equivalent</i>	<i>Octal Equivalent</i>
READS	Read Switches	DIA    -,CPU	060477
IORST	IO Reset	DICC   0,CPU	062677
HALT	Halt	DOC    0,CPU	063077
INTEN	Interrupt Enable	NIOS   CPU	060177
INTDS	Interrupt Disable	NIOC   CPU	060277
INTA	Interrupt Acknowledge	DIB    -,CPU	061477
MSKO	Mask Out	DOB    -,CPU	062077

*Eg* to read the switches into AC3 we could simply give

```
READS 3
```

instead of

```
DIA    3,CPU
```

However, there is one important difference between these special mnemonics and the standard ones: mnemonics for turning the interrupt on and off cannot be appended to them! Thus to set Interrupt On while reading the switches we must give

```
DIAS   3,CPU
```

Note that IORST clears Interrupt On along with the devices on the bus. We can set it while clearing the devices by giving

DICS 0,CPU

## 2.4 PROGRAM INTERRUPT

Many in-out devices must be serviced infrequently relative to the processor speed and only a small amount of processor time is required to service them, but they must be serviced within a short time after they request it. Failure to service within the specified time (which varies among devices) can often result in loss of information and certainly results in operating the device below its maximum speed. The program interrupt is designed with these considerations in mind, *ie* the use of interruptions in the current program sequence facilitates concurrent operation of the main program and a number of peripheral devices. The hardware also allows conditions internal to the processor to signal the program by requesting an interrupt.

**Interrupt Requests.** Interrupt requests by a device are governed by its Done and Interrupt Disable flags. When a device completes an operation it sets Done, and this action requests a program interrupt if Interrupt Disable is clear — if Interrupt Disable has been set by the program the device cannot request an interrupt. At the beginning of every memory cycle the processor synchronizes any requests that are then being made. Once a request has been synchronized the device that made it must wait for an interrupt to start. The request signal is a level so once synchronized it remains on the bus until the program clears Done or sets Interrupt Disable. If the program does set the Interrupt Disable flag in a device, that device not only cannot request an interrupt when its Done flag sets, but any request it has already made and had synchronized is disabled, so it is no longer waiting for an interrupt. However, if Done is left set, clearing Interrupt Disable restores the request.

**Starting an Interrupt.** The processor starts an interrupt if all four of the following conditions hold:

- The processor had just completed an instruction or a data channel transfer [see §2.5].
- At least one device is waiting for an interrupt to start (*ie* it was requesting an interrupt at the beginning of the last memory cycle).
- Interrupts are enabled, *ie* Interrupt On is set.
- No device is waiting for a data channel transfer, *ie* there are no data channel requests that the processor has synchronized but not yet fulfilled. 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 synchronized while data channel transfers are being made. When no more devices are waiting for data channel transfers, the processor starts an interrupt if Interrupt On is set and a device was requesting an interrupt at the beginning of the last data channel transfer.

The processor starts an interrupt by clearing Interrupt On so no further interrupts can be started, saving PC (which points to the next instruction) in location 0, and simulating a JMP @1 to jump to the interrupt service routine. Location 1 should contain the address of the routine or an indirect address that will get there.

**Servicing an Interrupt.** The interrupt service routine should determine which device requires service, save the contents of any accumulators that will be used in the routine, save Carry if it will be used, and service the device. The routine can identify the device by testing with IO skips or by giving an interrupt acknowledge instruction (INTA). This instruction determines which is the first device on the bus that is waiting for service by reading its device code into an accumulator. The program can simply leave the interrupt off while servicing the device (by leaving Interrupt On clear), or it can enable interrupts and establish a priority structure that allows higher priority devices to interrupt the current device service routine. This priority is determined by a mask that controls the states of the Interrupt Disable flags in the various devices. If this final course is taken

the routine must save location 0, so the return address to the interrupted program will not be lost should another interrupt occur.

**Device Priority.** There are several ways in which priorities are determined for or assigned to devices on the bus. An elementary priority is established by the hardware for devices that are requesting interrupts simultaneously in that the interrupt acknowledge instruction reads the code of one and only one device: among those that are waiting it reads the code of that one which is physically closest to the processor on the bus. This however applies only to those devices that are waiting at the time the acknowledgement is given. Using IO skips to determine which device to service establishes a priority by the order in which the devices are tested, but again this applies only to those that are waiting at the time.

The most significant method is by specifying which devices can interrupt a service routine currently in progress. This is done through the use of a mask that sets up the Interrupt Disable flags. Every device is wired to a particular data line on the bus and hence to a particular bit of the mask. Although slower devices are assigned to the higher numbered bits in the mask, there is no established priority as the program can use any mask configuration. All devices whose Interrupt Disable flags are set cannot cause an interrupt to start (setting Interrupt Disable causes the withdrawal of any request that has already been made and prevents the setting of Done from making a request) and are therefore regarded by the program as being of lower priority. Those devices in which Interrupt Disable is left clear can interrupt the current routine and therefore are regarded by the program as being of higher priority.

By means of the mask the program can establish any priority structure with one limitation: in some cases two or more devices are assigned to the same bit in the mask and are thus all at the same priority level. When an interrupt is in progress for a device, the rest of the devices assigned to the same mask bit must be regarded as all of lower priority or all of higher priority depending upon whether they are disabled or not.

**Dismissing an Interrupt.** After servicing a device the routine should restore the pre-interrupt states of the accumulators and Carry, turn on the interrupt, and jump to the interrupted program. The instruction that enables the interrupt sets Interrupt On, but the flag has no effect until the next instruction begins. Thus after the instruction that turns the interrupt back on, the processor always executes one more instruction (assumed to be the return to the interrupted program) before another interrupt can start.

If the service routine allows interrupts by higher priority devices, then before dismissing as indicated above, the routine should turn off the interrupt to prevent further interrupts during dismissal. In dismissing, the routine should reenable lower priority devices that were not allowed to interrupt the current routine but will be allowed to interrupt the program to which the processor is returning.

**Instructions.** The instructions for the program interrupt use special device code 77. Bits 8 and 9 of the skip instructions sense whether the interrupt is on or off; in the other instructions these bits turn the interrupt on or off by setting or clearing the Interrupt On flag (these are respectively the start and clear IO control functions).

**NIOS CPU                      Interrupt Enable**

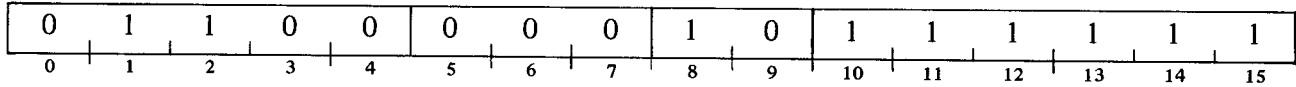
0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Set Interrupt On to allow the processor to respond to interrupt requests. If Interrupt On actually changes state (0 → 1) the processor will execute one more instruction before it can start an interrupt.

NOTE: The assembler recognizes the mnemonic INTEN as equivalent to NIOS CPU.



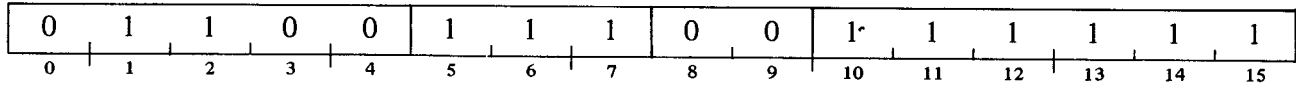
**NIOC CPU      Interrupt Disable**



Clear Interrupt On to prevent the processor from responding to interrupt requests.

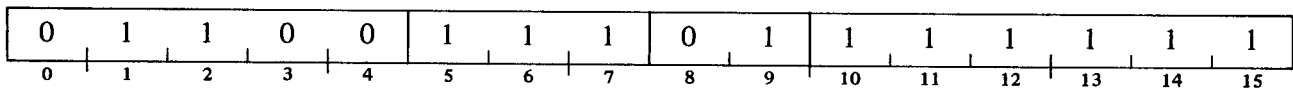
NOTE: The assembler recognizes the mnemonic INTDS as equivalent to NIOC CPU.

**SKPBN CPU      Skip if Interrupt On is Nonzero**



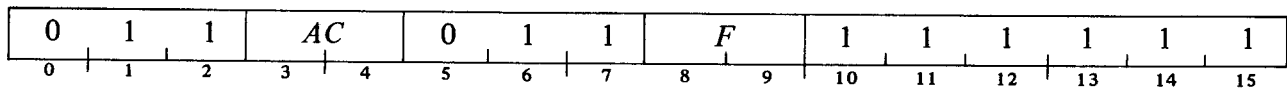
Skip the next instruction in sequence if Interrupt On is 1.

**SKPBZ CPU      Skip if Interrupt On is Zero**



Skip the next instruction in sequence if Interrupt On is 0.

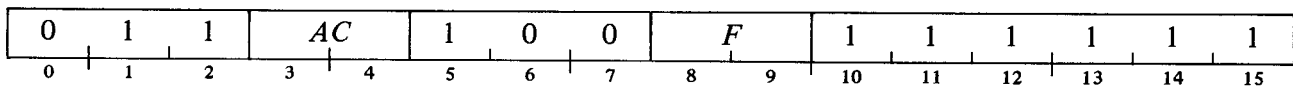
**DIB -,CPU      Interrupt Acknowledge**



Place in AC bits 10–15 the device code of the first device on the bus that is requesting an interrupt, and perform the function specified by *F*.

NOTE: The assembler recognizes the mnemonic INTA as equivalent to DIB -,CPU.

**DOB -,CPU      Mask Out**



Set up the Interrupt Disable flags in the devices according to the mask in AC (a 1 in a mask bit sets the flags in all devices assigned to that bit; a 0 clears them). Perform the function specified by *F*.

The following lists the devices assigned to the bits in the mask, and for each bit gives the mask for disabling all devices assigned to that and all higher numbered bits. [Complete information on all devices is given in Appendix E.]

<i>AC Bit</i>	<i>Devices</i>	<i>Mask</i>
0	Data communications multiplexor	177777
1		77777
2		37777
3		17777
4		7777
5		3777
6		1777
7		777
8	A-D converter, high speed communications controller	377
9	Disc	177
10	Card reader, magnetic tape	77
11	Paper tape reader	37
12	Plotter, line printer, multiprocessor communications adapter	17
13	Real time clock, paper tape punch, display, IBM 360 interface	7
14	Teletype in	3
15	Teletype out	1

A zero mask clears all Interrupt Disable flags. In general the devices are in order by speed, with the fastest ones (those requiring the quickest service) assigned to the lower numbered bits.

NOTE: The assembler recognizes the mnemonic MSKO as equivalent to DOB -,CPU.

The assembler recognizes special mnemonics for some of the above instructions.

INTEN	NIOS	CPU	Interrupt Enable	060177
INTDS	NIOC	CPU	Interrupt Disable	060277
INTA	DIB	-,CPU	Interrupt Acknowledge	061477
MSKO	DOB	-,CPU	Mask Out	062077

To turn the interrupt on or off while acknowledging or masking, the programmer must use the DIB and DOB forms – the S and C mnemonics cannot be appended to INTA and MSKO.

**Timing.** The time a device must wait for an interrupt to start depends on how many devices are using interrupts, how long the service routines are for devices of higher priority, and whether the data channel is in use. A single device will shut out all others of lower priority if every time its service routine dismisses the interrupt, it is already waiting with another request; and the data channel shuts out all interrupts when it operates at the maximum rate. If the data channel is not in use and only one device is using interrupts, it need never wait longer than the time required for the processor to finish the instruction that is being performed when the request is synchronized. Without delays caused by indirect addressing, the maximum interrupt waiting time is the latency given in the table at the end of Appendix D.

To start an interrupt the processor uses two cycles to store PC in location 0 and retrieve the address from location 1. The time given in Appendix D assumes location 1 contains a direct address.

**Sample Master Interrupt Routine.** Suppose we are using only the teletype and the high speed reader and punch. We shall allow higher priority devices to interrupt a lower priority service routine; but since the reader

is the highest priority device, we shall simply leave the interrupt off while servicing it. Because of the small number of devices we can use flag testing to identify the one that is requesting service and we can treat the teletype input and output as the same priority. For illustration let us assume that the reader and punch routines use all the accumulators but the teletype routines use only ACO.

```
.LOC      0          ;This pseudoinstruction causes the assembler to put the next statement in
          0          ;the location specified
          0          ;Clear location 0 – will be used for saving PC
INTRP    ;Put address of master interrupt processor routine in location 1
CMASK:   0          ;Will save current mask here (initially zero)
```

```
:
```

```
;When the processor is interrupted the interrupt is disabled and there is an automatic jump to INTRP.
;First find source of interrupt.
```

```
INTRP:   SKPDZ   PTR          ;Try reader first
          JMP     PTRIN       ;Yes, service it
          SKPDZ   PTP         ;No, try punch
          JMP     PTPIN       ;Jump to punch service
          STA     0,TTSAV     ;Neither, must be teletype; save ACO
          LDA     0,0        ;Save return address from location 0
          STA     0,TTSAV+1
          LDA     0,CMASK    ;Save current mask
          STA     0,TTSAV+2
          LDA     0,CN3      ;Set mask bits 14, 15 (disable teletype interrupts)
          STA     0,CMASK    ;Set new current mask
          DOBS    0,CPU      ;MSKO and enable interrupts

          SKPDZ   TTO         ;Test teletype output
          JMP     TTOIN       ;Jump to output service
          SKPDN   TTI        ;Test input
          JMP     ERROR       ;Something wrong – nobody wants service
          :                ;Service teletype in
          :
          JMP     TTDSM      ;Must dismiss

TTOIN:   :                ;Service teletype out
          :

TTDSM:   INTDS          ;To dismiss, first disable interrupts
          LDA     0,TTSAV+2  ;Restore previous mask
          STA     0,CMASK
          MSKO    0
          LDA     0,TTSAV    ;Restore ACO
          INTEN   ;Enable interrupts
          JMP     @TTSAV+1   ;Return to interrupted program

TTSAV:   0                ;Save ACO here
          0                ;Save PC (from location 0) here
```

```

0                                     ;Save current mask here
CN3: 3

;Punch routine

PTPIN:  STA 0,PPSAV                 ;Save accumulators
        STA 1,PPSAV+1
        STA 2,PPSAV+2
        STA 3,PPSAV+3
        MOVL 0,0                    ;Save Carry
        STA 0,PPSAV+4
        LDA 0,0                     ;Save location 0
        STA 0,PPSAV+5
        LDA 0,CMASK                 ;Save current mask
        STA 0,PPSAV+6
        LDA 0,CN7                   ;Set mask bits 13,14,15 (punch, teletype in and out)
        STA 0,CMASK                 ;Set new current mask
        DOBS 0,CPU                  ;MSKO and turn on interrupt
        :
        :
        INTDS                       ;Turn off interrupt
        LDA 0,PPSAV+6               ;Restore previous mask
        STA 0,CMASK
        MSKO 0
        LDA 0,PPSAV+4               ;Restore Carry
        MOVR 0,0
        LDA 0,PPSAV                 ;Restore ACs
        LDA 1,PPSAV+1
        LDA 2,PPSAV+2
        LDA 3,PPSAV+3
        INTEN                       ;Turn on interrupt
        JMP @PPSAV+5                ;Restore PC

PPSAV: .LOC .+7                     ;Reserve 7 locations
CN7: 7

;Reader routine
PTRIN:  STA 0,PRSAV                 ;Save ACs and Carry, but don't bother with PC or mask, and leave inter-
        STA 1,PRSAV+1               ;rupt off
        STA 2,PRSAV+2
        STA 3,PRSAV+3
        MOVL 0,0
        STA 0,PRSAV+4
        :
        :
        LDA 0,PRSAV+4               ;Restore Carry and ACs

```

```

MOVW    0,0
LDA     0,PRSAV
LDA     1,PRSAV+1
LDA     2,PRSAV+2
LDA     3,PRSAV+3
INTEN                   ;Turn on interrupt
JMP     @0              ;Restore PC

```

```

PRSAV:
.LOC     .+5            ;Reserve 5 locations

```

**When to Use the Interrupt.** If the program has little computing to do and is using only one or two fast in-out devices or several slow ones, it may not be necessary to use the interrupt at all. On the other hand, if there are many calculations to perform and the program is using a fast device or is processing data using several slower devices, then the interrupt is necessary. The critical factors in determining whether to use the interrupt, and beyond that its priority structure, are what the program is doing besides in-out and the time required by the service routines. Suppose the program is doing nothing but processing data using reader, punch and teletype, and further suppose that no service routine requires more than say half a millisecond. In these circumstances the program could dispense with the interrupt and test all the devices with the following loop:

```

TEST:    SKPDZ   PTR
         JMP     PTRSER
         SKPDZ   PTP
         JMP     PTPSER
         SKPDZ   TTO
         JMP     TTOSER
         SKPDZ   TTI
         JMP     TTISER
         :
         :
         :           ;Fast test that determines whether IO is finished
         :
         JMP     TEST      ;Do this if more IO
         :
         :           ;Skip to here and continue if IO done
         :

```

where the reader service routine returns to TEST + 2 and all others return to TEST. The fastest device, the reader, will never be delayed too much. But suppose the program has a significant amount of computing to do. Then we must use the interrupt, but what about the priority structure? If input-output service for the teletype (as in the sample master routine above) requires 1 ms and punch service requires .8 ms, then reader service will never be delayed more than 1 ms if we simply turn the interrupt off while servicing each device. But if teletype service requires 30 ms per character, then neither reader nor punch will be able to run at full speed unless we use the priority structure as illustrated in the sample routine.

**Programming Suggestions.** A convenient method for handling a large number of priority levels is to use a pushdown list for saving the machine state. This obviates setting aside so many specific locations for saving accumulators and the like, and makes it very easy for a routine at any level in a sequence of nested routines to restore the state for the interrupted program. If many devices are in use it may frequently happen that when one routine is dismissing an interrupt, a device of lower priority is already waiting. Thus much time might be wasted in restoring the machine state only to have to save it again as soon as the interrupt is turned back on.

The devices of concern in this situation are those with priority less than or equal to the device presently being serviced, but of priority greater than that of the device whose routine is about to be resumed (to which the current dismissal will return). The usual dismissal procedure (as illustrated in the sample master routine given above) begins by disabling the interrupt and restoring the previous mask. If the program then gives an

INTA AC

a device code will be read into AC if any device of priority higher than that of the interrupted routine has requested service. Since this means that the device will interrupt before the interrupted program can restart, the current program can save a great deal of time by servicing the higher priority device without bothering to restore and resave the machine state. If AC is clear after the INTA is given, no device of appropriate priority has requested service, and the current routine can proceed with the usual dismissal.

Remember the following when programming an interrupt routine:

- An interrupt cannot be started until the current instruction is finished. Therefore do not use lengthy indirect address chains if devices that require very fast service can request an interrupt.
- The routine must save the accumulators and the Carry flag if these will be used by it.
- If this interrupt routine can itself be interrupted, then it must save location 0 so PC can later be restored properly.
- The principal function of an interrupt routine is to respond to the situation that caused the interrupt. *Eg* computations that can be performed outside the routine should not be included within it.
- The routine should restore the accumulators and Carry when returning to the interrupted program.

## 2.5 DATA CHANNEL

Handling data transfers between external devices and memory under program control requires an interrupt plus the execution of several instructions for each word transferred. To allow greater transfer rates the processor contains a data channel through which a device, at its own request, can gain direct access to memory using a minimum of processor time. At rates lower than the maximum the channel frees processor time to allow execution of a program concurrently with data transfers for a device. The channel is multiplexed — many devices may be active at the same time.

Besides the straightforward transfer of a word between memory and a device in either direction, the data channel also allows a device to increment by one a word already in memory and, in the Nova or Supernova computers, to add a word to the contents of a memory location. In these two cases involving an arithmetic operation, the processor sends the result back to the device; and if the operation should increase the contents of the memory location above  $2^{16}-1$ , it also sends an overflow signal to the device. The data channel is used by devices requiring very high data transfer rates, such as magnetic tape or disc, and by devices requiring the specialized transfer functions. *Eg* the memory increment feature would be used for pulse height analysis, the add-to-memory feature for signal averaging.

The program cannot affect the data channel directly because there are no instructions for it; instead the program sets up the device to use it. When the device requires data service, it requests access to memory via the channel. At the beginning of every memory cycle the processor synchronizes any requests that are then being made. Except in the Nova 800 series, the processor completes the current instruction and then takes care of all requests that have been synchronized or are synchronized while it is handling transfers. In the 800 series the data channel is capable of operating at two different speeds (standard and high speed) and does not require that a device wait until the completion of an instruction — the processor can pause to handle transfers

at certain points within an instruction. If several devices are waiting for service simultaneously, the first to receive it is the one that is physically closest to the processor on the bus. When an 800 series processor pauses within an instruction, it handles all data channel requests of either speed (handling high speed requests first) and then continues with the interrupted instruction. Following completion of an instruction, any processor handles all data channel requests, and then starts a program interrupt if a device is waiting for one, or otherwise resumes the execution of instructions. The two-speed channel is available as an option on the Supernova computer, which can pause within an instruction to handle a high speed transfer, but waits until the end of the instruction for transfers at the standard speed.

Operating the 800 data channel at standard speed allows data transfer rates of half a million words per second, but at this rate all other processing activity is suspended. Use of the high speed capability not only allows data transfer rates at essentially the full memory speed (in excess of a million words per second), but at speeds in the standard range its use allows considerable processing activity unrelated to the channel (each transfer takes less time). Hence choice of the standard or high speed depends on the degree of interference with the program caused by channel operations and the maximum time within which the device must make the transfer. When a rate of 100,000 or more words per second is required, both the device and the program will benefit noticeably through use of the high speed capability. To use the high speed the interface for a device must be mounted inside the main frame and must be designed so that it can both respond to the shorter control signals presented to it and operate within the extremely limited time available [*timing specifications for all data channel operations are given in Appendix A, Part II*]. Moreover all high speed interfaces must be grouped at the beginning of the bus: all interfaces closer to the processor than the last high speed one automatically operate at high speed, whereas all devices farther out on the bus operate at standard speed. The processor examines the priority determining signal on the IO bus to determine which way to handle each transfer. A computer that has the two-speed capability is shipped with the high speed enabled for all interfaces mounted inside the main frame (interfaces on an external bus are always limited to standard speed).

**Timing.** The time a device must wait for data channel access depends on when its request is made within an instruction and how many devices of higher priority are also requesting access. Once the processor reaches a point at which it can pause to handle transfers, a given device must wait until all devices closer than it on the bus have been serviced (hence all devices connected for the high speed are serviced first). The highest priority device can preempt all processor time if it requests access at the maximum rate. At less than the maximum rate the closest device need never wait longer than the time required for the processor to finish the instruction that is being performed when the request is synchronized, but indirect addressing can extend this beyond the normal instruction execution time. The latency given in the timing table at the end of Appendix D is the maximum data channel waiting time for the highest priority device exclusive of any delay caused by indirect addressing. With an 800 series channel or a Supernova computer high speed channel, the closest device, once synchronized, need never wait beyond the next point at which the processor can pause within the instruction, but the maximum that this can be depends on whether the program includes IO instructions *ie* the device may have to wait longer when the program is also using the bus). In some cases the time taken for a single isolated transfer is less than the minimum time between transfers.

#### CAUTION

Devices that use the data channel often require service very quickly. In those cases where a device must wait for the current instruction to end, do not use lengthy indirect addressing chains when the data channel is in use.

Maximum rates in transfers per second are as follows.

Function	800 Series		1200 Series	Supernova		Nova
	High Speed	Standard		High Speed	Standard	
Data In	1,250,000	500,000	833,333	1,250,000	434,700	285,500
Data Out	1,000,000	500,000	555,555	1,000,000	357,100	227,500
Increment memory	833,333	454,545	416,666	833,333	357,100	227,500
Add to memory				833,333	357,100	187,500

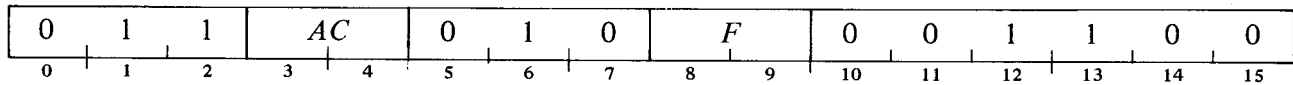
## 2.6 PROCESSOR OPTIONS

Optional equipment for the processor includes a real time clock, a power monitor with facility for automatic restart after power failure, multiply-divide, and memory allocation and protection (the last is available only on the 800 series and the Supernova computer). The program load option for the 1200 series and 800 series is discussed in § 2.8.

### Real Time Clock

The clock generates a sequence of pulses that is independent of processor timing. It uses only one IO transfer instruction to set the clock frequency. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 14, mnemonic RTC. Interrupt Disable is controlled by interrupt priority mask bit 13.

#### DOA —,RTC Data Out A, Real Time Clock



Perform the function specified by *F* and select the clock frequency by *AC* bits 14 and 15 as follows.

AC bits 14-15	Frequency
00	Ac line frequency
01	10 Hz
10	100 Hz
11	1000 Hz

Setting Busy allows the next pulse from the clock to set Done, requesting an interrupt if Interrupt Disable is clear. A DOA to select the frequency need be given only once; following each interrupt an NIOS sets up the clock for the next pulse.

When Busy is first set the first interrupt can come at any time up to the clock period. But once one interrupt has occurred, further interrupts are at the clock frequency provided that the program always sets Busy before the next period expires.

The clock is used primarily for low resolution timing (compared to processor speed) but it has high long-term accuracy. Power up and the I/O reset function generated by the program or from the console reset the clock to line frequency. Following power turnon the line frequency pulses are available immediately, but 5 seconds must elapse before a steady pulse train is available from the crystal for other frequencies.



## Power Monitor and Auto Restart

When ac power is applied to the central processor, core memory is unaltered, the initial states of PC, the accumulators and flags are indeterminate, and the processor is halted. If ac power in the chassis should fail there is a minimum delay of 1 to 2 milliseconds after the power fail interrupt before the processor shuts down. In so doing, the processor always completes a memory cycle and sequences power off so the contents of memory are unaffected. The optional power monitor warns the program when dc power in the chassis is failing by setting the Power Failure flag. This action automatically requests an interrupt—there is no interrupt disable flag for the power monitor. Of course the interrupt must be on if a power failure is to produce an interrupt.

The power monitor does not respond to the INTA instruction. Thus when an interrupt occurs in a machine equipped with the power monitor, the program should test the Power Failure flag before giving INTA or testing other devices. The flag corresponds to the Done flag and is tested by either of these instructions.

### SKPDN CPU      Skip if Power Failure is Nonzero

0	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Skip the next instruction in sequence if Power Failure is 1.

### SKPDZ CPU      Skip if Power Failure is Zero

0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Skip the next instruction in sequence if Power Failure is 0.

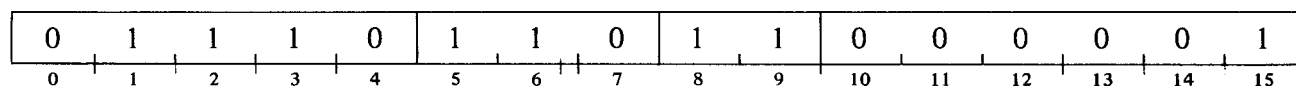
If the power does fail the program should save the accumulators and Carry in memory, save location 0 (for restoring PC in the interrupted program), put a JMP to the desired restart location in location 0, and then HALT.

The action taken by the processor when an adequate power level is restored depends on the power switch on the operator console. If the switch is on, power comes back on with the machine stopped. If the switch is in the lock position, then 50 ms after power comes back on the processor executes a JMP 0, which causes it to begin executing instructions in normal sequence at location 0.

## Multiply-Divide

Multiplication and division can be performed by the subroutines given on pages 2-19 and 2-20, but in all machines except the Nova computer, an option that is added right into the processor hardware is also available for these operations. This option provides two pseudo-IO instructions that duplicate exactly the effects of the subroutines (the writeups of the multiply and divide subroutines and 093-000015 and 093-000016 respectively).

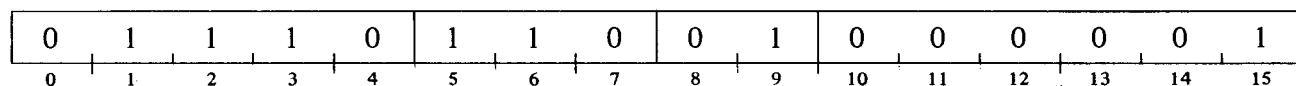
## MUL Multiply



Multiply the unsigned integers in AC1 and AC2 to generate a double length product; add the product to the unsigned integer in AC0, and place the high and low order parts of the result respectively in AC0 and AC1 (in other words the result left in AC0 and AC1 is  $AC0 + AC1 \times AC2$ ). AC2 is unaffected, the original contents of AC0 and AC1 are lost.

Note that the mnemonic MUL is equivalent to DOCP 2, 1. The AC field must be 10. (The hardware requires this, but it is done to be compatible with the Nova computer.)

## DIV Divide



If the unsigned integer in AC0 is greater than or equal to the unsigned integer in AC2, set Carry and go immediately to the next instruction without affecting the original contents of the accumulators. Otherwise clear Carry and divide the double length unsigned integer in AC0 and AC1 by the unsigned integer in AC2, producing a single length quotient including leading zeros, and then clear Carry. Place the quotient in AC1 and the remainder in AC0. AC2 is unaffected, the original contents of AC0 and AC1 are lost.

Note that the mnemonic DIV is equivalent to DOCS 2, 1. The AC field must be 10. (The hardware requires this, but it is done to be compatible with the Nova computer.)

### Nova Computer Multiply-Divide

The hardware multiply-divide option for the Nova computer is actually a peripheral device connected to the in-out bus, although it has no flags or interrupt capability. It contains A, B and C registers, which are loaded and read by the standard IO transfer instructions, and which correspond in use respectively to accumulators 0, 1 and 2 with respect to the multiply and divide software routines and the processor hardware option in the other computers. Bits 8 and 9 in a transfer instruction or an NIO perform control functions as follows.

<i>Mnemonic</i>	<i>Bits 8-9</i>	<i>Function</i>
	00	None
S	01	Divide the double length unsigned integer in A and B by the unsigned integer in C, producing a single length quotient including leading zeros. Place the quotient in B and the remainder in A. C is unaffected.
C	10	Clear the A register.
P	11	Multiply the unsigned integers in B and C to generate a double length product; add the product to the unsigned integer in A and place the high and low order parts of the result respectively in A and B (in other words the result left in A and B is $A + B \times C$ ). C is unaffected.

The multiply-divide device code is 01, mnemonic MDV. With this device code the instructions are those given on pages 2-23 to 2-25, with the exception that the skips are meaningless since the device has no flags.

Following the IO instruction that starts the multiply or divide, the program must wait until the result is available in the A and B registers. Multiplication takes 6.4  $\mu$ s, division takes either 6.8 or 7.2  $\mu$ s depending on the operands. Of course the program can do something useful with the time (such as loading an accumulator for the next operation), but usually one simply gives a couple of no-ops to pass the time.

Generally it is best to set up the accumulators just as one would for the software or the processor option. If they are set up for multiplication, we could give this sequence to multiply and place the result in the same place the subroutine would.

```
DOA    0,MDV    ;AC0 to A (AC)
DOB    1,MDV    ;AC1 to B (MQ)
MUL                    ;= DOCP 2,MDV = AC2 to C, multiply
NIO    0                    ;Wait for result (6.8  $\mu$ s)
JMP    .+1
DIA    0,MDV    ;Put double length product in AC0
DIB    1,MDV    ;and AC1
```

With this procedure, programming for all the computers is compatible. If a program containing the above sequence is run on a Supernova computer, the first two instructions are ignored, the MUL is executed, the two no-ops result in a small amount of lost time, and the DIA and DIB are ignored as the hardware is gated so that calling for input from device 01 cannot affect the accumulators.

Similarly, if the accumulators are set up for software division we would give this sequence to divide.

```
DOA    0,MDV
DOB    1,MDV
DIV                    ;= DOCS 2,MDV but no overflow check
MOV#   0,0            ;Wait for result (7.2  $\mu$ s)
JMP    .+1
DIA    0,MDV
DIB    1,MDV
```

Here the AC configuration is the same but there is no check to determine whether division is possible—the program must do that first and properly adjust the operands. (Carry has no connection with the operation of the device and is unaffected.) For integer division the program need not clear AC0: instead the first two instructions can be replaced by

```
DOBC    1,MDV
```

but compatibility with the other machines is then lost.

### Memory Allocation and Protection

Without memory allocation and protection the system executes a single program that has no restrictions except those inherent in the hardware: the programmer must stay within the memory capacity, and observe the restrictions placed on the use of certain memory locations by the hardware [§ 1.4]. Optional hardware for the 800 series and the Supernova computer can restrict processor operation to permit time sharing by a number of programs. Each user program is run with the processor in user mode, in which the program must

operate within an assigned area in memory and certain operations are illegal. A program that runs unrestricted — the executive — is responsible for scheduling user programs, servicing interrupts, handling input-output needs, and taking action when control is returned to it from a user program.

Every user has a memory area allocated to him and he cannot gain access to the rest of memory for either storage or retrieval of information. Moreover part of his allocated area may be protected from him, *ie* the executive may set aside part of his allocated area so that he can access it but cannot alter its contents, *ie* he cannot write anything in it. The executive would do this when part of the allocated area contains a pure procedure to be used reentrantly by several users. While the processor is in user mode, the program is further restricted in that it is illegal to issue any IO instruction (except MUL and DIV) or to use more than two levels of indirect addressing. The violation of any restriction by a user program causes the processor to terminate the instruction immediately and return control to the executive (by requesting an interrupt, which returns the processor to the supervisor mode).

For allocation purposes the entire memory is divided into blocks of 4096 words each, defined by the three high order address bits. For each user the executive establishes a map of the logical blocks (those defined by the addresses given in the user program) into the physical blocks of memory, and validates those logical blocks that are available to the given user. The most convenient procedure is for the executive to allow all users to write programs beginning at location 0. Thus one user may be limited to a single block, and the executive would validate logical block 0 and assign it to say physical block 4; for another user allowed two blocks, the executive would validate blocks 0 and 1 and assign them to say physical blocks 5 and 6. The first user would use addresses 0-7777 and these would be mapped into addresses 40000-47777; the second would use addresses 0-1777 and these would be mapped into 50000-67777. The programmed addresses are retained in the object program but are mapped by the hardware into the physical area assigned to the user as each access is made while the program is running.

For protection purposes memory is divided into pages of 256 words each. The executive establishes a protection scheme for all of the physical memory, and although a given user can access any location in his allocated blocks, he simply cannot write in any page that is protected. To save swapping time, a Page Written flag is associated with each page. When setting up a user program, the executive should clear all the flags. Whenever the user writes in a given page, its associated Page Written flag is set. Then when that user goes on the inactive list, the executive need rewrite on the swapping disk or drum only those pages that have actually changed.

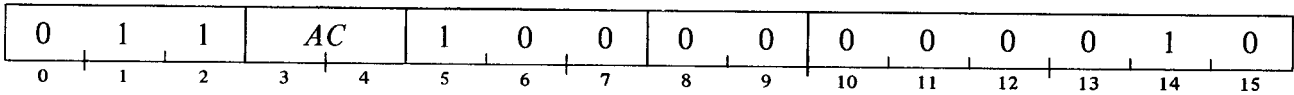
Note that the restrictions apply only to the user program. Data channel transfers can occur while the processor is in user mode, and access is made to the physical locations addressed. An interrupt always returns the processor to supervisor mode—the executive handles all interrupts.

**User Programming.** The user must observe the following rules when programming on a time shared basis.

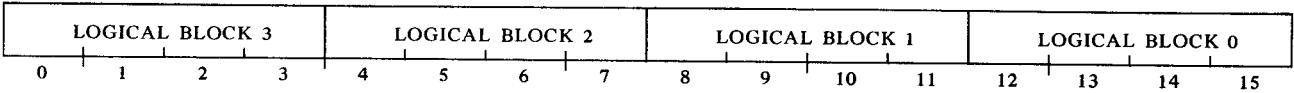
- Use addresses only within the allocated logical blocks for all purposes—retrieval of instructions, retrieval of addresses, storage or retrieval of operands. The method of allocating blocks will depend of course on the executive program used at a particular installation, but usually the executive will be set up so that the user begins at location 0 and can write any size program, *ie* the executive will assign enough memory for his needs. Basically the user must write a sensible program; if he uses absolute addresses scattered all over memory his program cannot be run on a time shared basis with others.
- Do not attempt to store anything in pages that are protected.
- Do not execute a JMP or JSR outside of the logical blocks assigned in any allocation procedure.
- Use IO instructions only for communication with the executive in the manner prescribed for the installation.
- Do not use more than two levels of indirect addressing.

**Executive Programming.** The executive program uses the following instructions to supervise time shared operation.

**DOB - ,MAP0 Assign Lower Logical Memory Map**

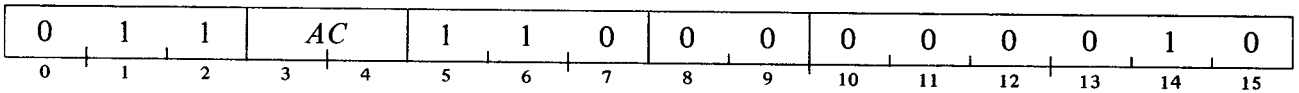


Assign logical memory blocks 0-3 to the physical blocks selected by the contents of AC and establish the validity of user addressing in these logical blocks as shown.

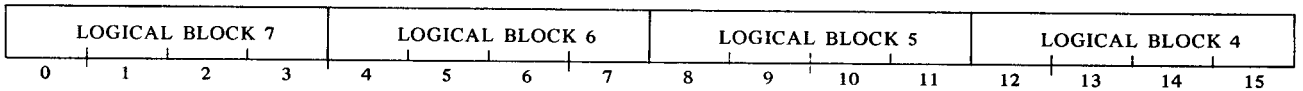


In each set of four bits, a 1 in the left bit validates user addresses within the corresponding logical block (a 0 makes such addresses invalid); the right three bits specify the physical block to which user addresses in the corresponding logical block will be mapped.

**DOC - ,MAP0 Assign Upper Logical Memory Map**

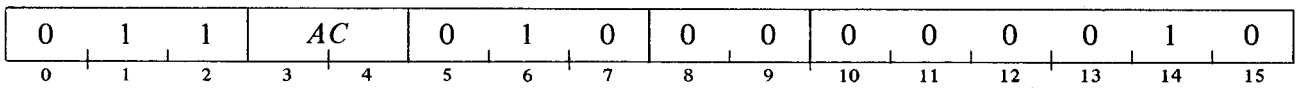


Assign logical memory blocks 4-7 to the physical blocks selected by the contents of AC and establish the validity of user addressing in these logical blocks as shown.

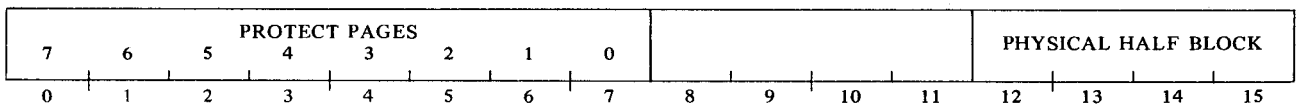


In each set of four bits, a 1 in the left bit validates user addresses within the corresponding logical block (a 0 makes such addresses invalid); the right three bits specify the physical block to which user addresses in the corresponding logical block will be mapped.

**DOA - ,MAP0 Write Protect**



Set up the protection scheme for a half block according to the contents of AC as shown.



Bits 12-15 specify the physical half block, *ie* bits 12-14 specify the physical block and a 0 or 1 in bit 15 selects the half containing the lower or upper addresses in that block. A 1 in any bit from 0-7 protects the

corresponding 256-word page from writing by the user (a 0 allows the user to write in the page if it is in one of his allocated blocks). Page 0 contains the lowest addresses in the half block.

**DOA 0, MAP 2 Clear Page Written Flags**

0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Clear all Page Written flags and select physical block 0 for page-written checking.

**DIA -,MAP1 Read Violation Status**

0	1	1	AC	0	0	1	F	0	0	0	0	1	1		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the status of the allocation and protection option into AC as shown.

USER	INDIRECT ERROR	IO ERROR	VALIDITY ERROR	PROTECTION ERROR	PHYSICAL BLOCK ADDRESSED
0	9	10	11	12	13 14 15

*Bit*

*Meaning of a 1 in the Bit*

- 0 The processor was in user mode when the last interrupt occurred.
- 9 The last user instruction attempted more than two levels of indirect addressing.
- 10 The last user instruction was an IO instruction (not MUL or DIV).
- 11 The last address mapped was invalid.
- 12 The last valid address mapped was for a reference that attempted to write in a protected page.

The setting of bit 9, 10, 11 or 12 requests an interrupt which has priority over all other devices connected to the bus and which cannot be disabled (but these bits cannot cause an interrupt when the processor is in supervisor mode). Bits 13–15 specify the physical block addressed by the last address mapped.

(Perform the function specified by *F*.)

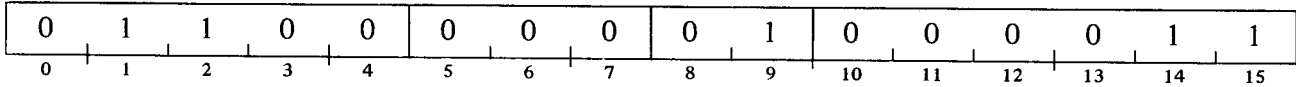
**DOA -,MAP1 Select Mode**

0	1	1	AC	0	1	0	F	0	0	0	0	1	1		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Load AC bit 0 into bit 0 of the status register and clear the rest of the register.

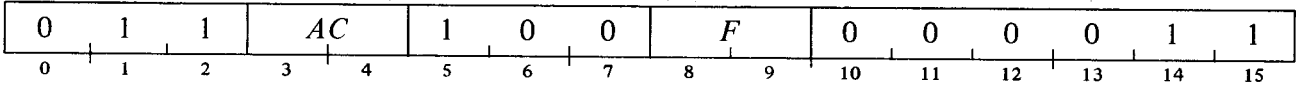
If *F* is 01 (S), turn on the interrupt and place the processor in the mode specified by bit 0 of the status register. If bit 0 is 1 the processor will execute one more instruction before entering user mode. If Interrupt On actually changes state (0 → 1) the processor will execute one more instruction before an interrupt can start.

**NIOS MAP1      Enter User Mode**



Turn on the interrupt and place the processor in the mode specified by bit 0 of the status register. If bit 0 is 1 the processor will execute one more instruction before entering user mode. If Interrupt On actually changes state (0 → 1) the processor will execute one more instruction before an interrupt can start.

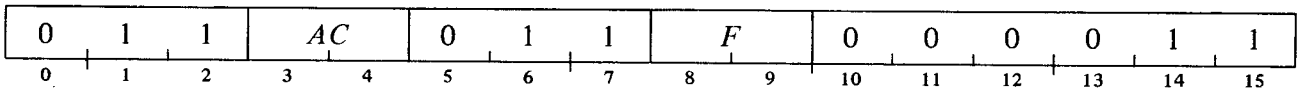
**DOB -,MAP1      Map an Address**



Map the address contained in AC bits 1-15, interpreting it as a user address for a write reference (in other words, indicate any violations in the status register).

(Perform the function specified by *F*.)

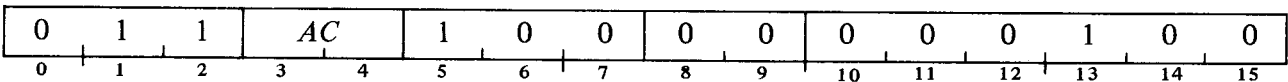
**DIB -,MAP1      Read Mapped Address**



Read the mapped address derived from the address supplied by the last DOB -,MAP1 into AC bits 1-15.

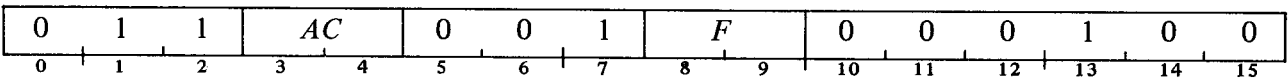
(Perform the function specified by *F*.)

**DOB -, MAP 2      Select Page Written Check**

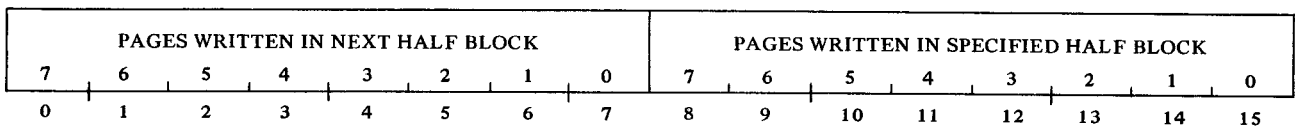


Select, for page-written checking, the pair of contiguous physical half blocks consisting of the half block specified by AC bits 12-15 and the next higher-numbered half block. (If AC bit 15 is 0, this instruction selects the physical block specified by bits 12-14.)

**DIA -,Map 2      Read Page Written Status**



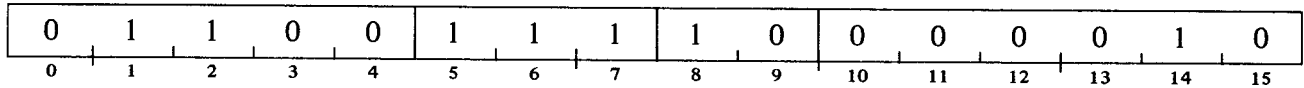
Read the Page Written flags associated with the currently selected pair of contiguous physical half blocks into AC as shown (a 1 in an AC bit indicates the user wrote in the corresponding page).



If *F* is 11 (P), select the next pair of contiguous half blocks following this pair for page-written checking.

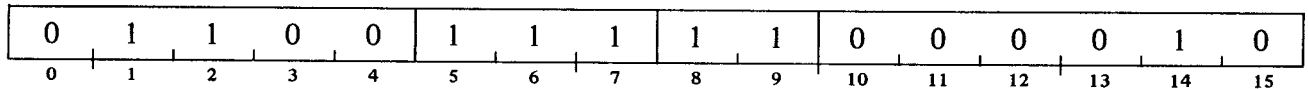
Note: If the user allocation being checked is larger than one block, the executive should use this instruction in the form DIAP so that a string of them can check all user blocks. A single block can of course be checked by a DIA. But if the first in a series of blocks were checked by a DIA, and there were no intervening DOB -,MAP2 or NIOP MAP2, a subsequent DIA would check the status of the higher half block already checked and the next half block after that (*ie* the sixteen flags checked would overlap the previous set by eight).

**SKPDN MAP0      Skip if Any Violation**



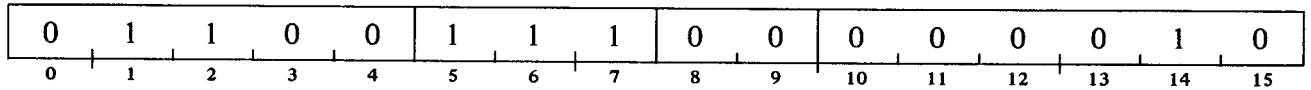
Skip the next instruction in sequence if any of bits 9-12 of the violation status register is 1.

**SKPDZ MAP0      Skip if No Violation**



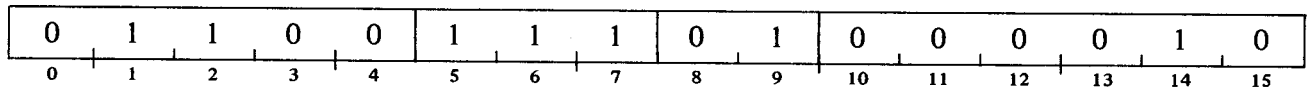
Skip the next instruction in sequence if bits 9-12 of the violation status register are all 0.

**SKPBN MAP0      Skip if IO Violation**



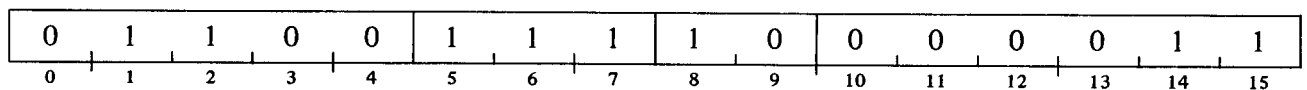
Skip the next instruction in sequence if bit 10 of the violation status register is 1.

**SKPBZ MAP0      Skip if No IO Violation**



Skip the next instruction in sequence if bit 10 of the violation status register is 0.

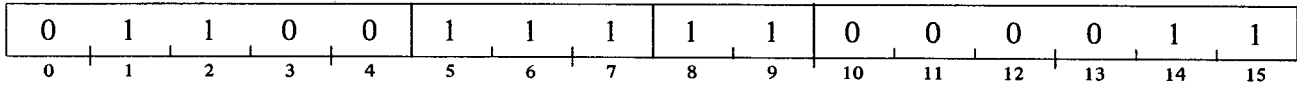
**SKPDN MAP1      Skip if Validity Violation**



Skip the next instruction in sequence if bit 11 of the violation status register is 1.

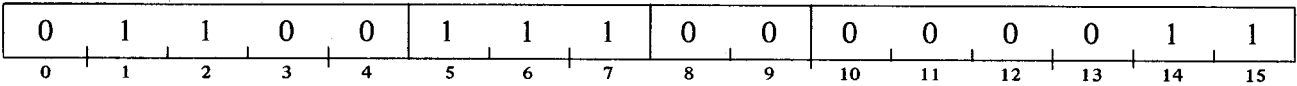


**SKPDZ MAP1      Skip if No Validity Violation**



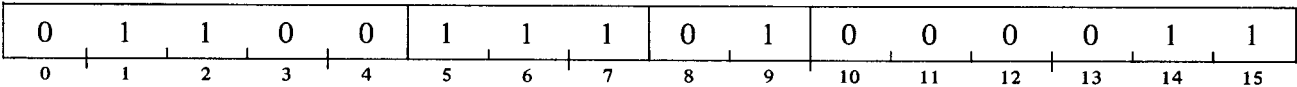
Skip the next instruction in sequence if bit 11 of the violation status register is 0.

**SKPBN MAP1      Skip if Protection Violation**



Skip the next instruction in sequence if bit 12 of the violation status register is 1.

**SKPBZ MAP1      Skip if No Protection Violation**



Skip the next instruction in sequence if bit 12 of the violation status register is 0.

At power turnon the processor is in supervisor mode and the mapping and protection data are indeterminate. The IO reset switch places the processor in supervisor mode but does not affect the mapping and protection data. To run a user program without write-protection, the executive must put 0s in the protection bits for the pages in the user blocks.

Note that the executive may not be able to trace a violation to its source. *Eg*, a JMP to an invalid address is not detected until the next instruction is fetched, and by then the location of the JMP cannot be determined.

## 2.7 OPERATION

The various consoles are illustrated on page 1-2. The lights in the upper right on the programmer's console display control conditions, the rows of lights in the upper center display the processor registers. Below the latter is a register of toggle switches through which the operator can supply addresses and data to the processor (the up position of a switch represents a 1). The register can be used in conjunction with some of the operating switches, and its contents are read by the READS instruction.

In the row at the bottom of the panel are the operating switches. Each switch lever is actually two momentary-contact logical switches with a common off position in the center. Lifting the lever up turns on the switch whose name is printed above it; pressing it down turns on the switch whose name is written below.

At the upper left is a 3-position key-operated rotary switch that controls power and locks the console. Turning it to ON simply turns on power. Turning it to LOCK keeps power on and disables the operating switches so no one can interfere with the operation of the processor (the operator can still use the data switches to supply information to the program).

The turnkey console has the key switch described above, a RUN light, and four operating switches, PROGRAM LOAD, CONTINUE, START, and RESET, which have exactly the same effect as they do on the programmer's console. The remainder of this section describes all of the indicators and switches on the various programmer's consoles (an asterisk indicates a light or switch that is also on the turnkey console).

**Indicators.** When any indicator is lit the associated flipflop is in the 1 state or the associated function is true. A few indicators display useful information while the processor is running, but most change too frequently and are therefore discussed in terms of the information they display when the processor has stopped.

The address lights display the contents of PC. The numbered data lights display the data written in the last memory reference, except following a Supernova computer memory step when they display the address for the next reference. The instruction lights (Nova and Supernova computers only) display the left eight bits of the instruction being executed or just completed; these lights are all off if the processor stops following a program interrupt (in the Nova computer they are also off following a data channel cycle).

RUN*	The processor is in normal operation with one instruction following another. When the computer stops, the light goes off.
ION	The program interrupt is enabled (this is the Interrupt On flag).
FETCH	The next processor cycle will be used to fetch an instruction from memory.
DEFER	The next processor cycle will be used to fetch an address word in an indirectly addressed memory reference instruction.
EXECUTE	The next processor cycle will be used to reference memory for an operand in a move data or modify memory instruction.
DCH	(Nova and Supernova computers only.) The next processor cycle will be used by the data channel for direct access to memory by an in-out device.
PI	(Nova and Supernova computers only.) The next processor cycle will be used to start an interrupt by storing PC in location 0.
OVERLAP	(Supernova computer only.) Arithmetic and logical class instructions are being executed out of read-only memory and the processor is overlapping the execution of one with the fetching of the next. (This light is always off when the computer stops.)
PROTECT	(Supernova computer only.) The processor is in user mode. An 800 series computer that has the memory allocation and protection option can generally be assumed to be in user mode when ION is lit.

FETCH, DEFER, EXECUTE, DCH and PI are the state indicators: they specify the state (the type of cycle) the processor will enter if operations are continued by pressing the CONTINUE or MEMORY STEP switch (see below). At most one light is lit; no light lit on the Supernova computer panel is equivalent to FETCH; on the Nova computer panel one and only one light must be lit. Unless otherwise indicated, use of any operating switch leaves the processor ready to enter the fetch state.

**Operating Switches.** All of the switches in the bottom row except STOP and RESET are interlocked so that they have no effect if RUN is lit. The four pairs of switches at the left are for depositing data in the accumulators and examining their contents. Lifting a switch lever up loads the contents of the data switches into the specified accumulator; pressing it down displays the contents of the accumulator in the data lights. At completion the instruction lights are off.

\*Indicates a light or switch that is also on the Turnkey Console.

The switches at the right perform the following functions when turned on.

EXAMINE	Load the address contained in the data switches into PC (which is displayed in the address lights) and display the contents of the addressed location in the data lights.
DEPOSIT	Deposit the contents of the data switches in the memory location specified by the address lights. At completion the data lights display the word deposited.
EXAMINE NEXT	Add 1 to the PC address displayed in the address lights and display the contents of the location specified by the incremented address in the data lights.
DEPOSIT NEXT	Add 1 to the PC address displayed in the address lights and deposit the contents of the data switches in the memory location specified by the incremented address. At completion the data lights display the word deposited.

The above four switches can be used for a sequence of operations on consecutive memory locations. The sequence must begin with EXAMINE to supply the initial address unless PC already points to the right location. Suppose we set the data switches to octal 100 initially. Then the following sequence of switch settings produces the effects listed.

EXAMINE	Display location 100.
EXAMINE NEXT	Display location 101.
EXAMINE NEXT	Display location 102.
DEPOSIT	Load data switches into 102.
EXAMINE NEXT	Display location 103.
DEPOSIT	Load data switches into 103.
DEPOSIT NEXT	Load data switches into 104.
EXAMINE NEXT	Display location 105.

START*	Load the address contained in the data switches into PC, light FETCH and RUN, and begin normal operation by executing the instruction at the location specified by PC.
STOP	Stop before fetching the next instruction. Thus the processor finishes the current instruction, and then stops with the instruction lights displaying the instruction, unless a device is waiting for data channel access or a program interrupt, in which case it performs all such operations before stopping with the instruction lights off. The address lights point to the next instruction.

#### CAUTION

If the current instruction contains an infinitely long indirect addressing chain or there are continuous data channel requests, pressing STOP will *not* stop the computer (see RESET, below).

CONTINUE\*

Turn on RUN and begin normal operation in the state indicated by the lights.

INST STEP

Begin operation in the state indicated by the lights but then stop as though STOP had been pressed at the same time. If the stop occurs at the end of an instruction, the data displayed by the data lights depends on the instruction as follows.

LDA, STA	Operand
ISZ, DSZ	Operand
JMP	1200 series and 800 series, direct: instruction Otherwise: effective address
JSR	Nova 1200 series and 800 series, direct: instruction Nova 1200 series and 800 series, indirect: effective address Otherwise: address loaded into AC3 (old PC + 1)
Arithmetic and logical	Except Supernova computer: instruction Supernova computer: unshifted result
In-out	1200 series, 800 series: data Supernova computer: zero; Nova computer: instruction

Note that the AC switches can be used between instruction steps without requiring any readjustment.

MEMORY STEP

Perform a single processor cycle in the state indicated by the lights and then stop. At completion the lights indicate the next state to be executed. The address lights display PC; the data lights on the Nova computer display the data for the last memory step, on the Supernova computer they display the address for the next memory step.

*CAUTION*

Using the AC switches between memory steps within an instruction usually destroys information necessary for the execution of the rest of the instruction.

RESET\*

Stop at the end of the current processor cycle. Clear the flags in all IO devices, clear Interrupt On, place the processor in supervisor mode, and set the clock to line frequency.

*CAUTION*

Information deposited in an accumulator from the console is displayed in the lights but is not actually entered into the accumulator until the processor performs some other operation. Hence pressing RESET after an AC deposit prevents the data from actually reaching AC.

PROGRAM LOAD\*

Nova 1200 series, 800 series: If the processor has the program load option, deposit the contents of the bootstrap read-only memory into locations 0-37, then light RUN

and begin normal operation at location 0. The bootstrap program allows the use of this switch for either program load or channel start; for complete details refer to §2.8.

Supernova computer: Read 33 words from the device selected by data switches 10-15 into locations 0-40, then light **RUN** and begin normal operation at location 40 [§ 2.8].

**CHANNEL START** (Supernova computer only.) Issue a **DIAS** to the device selected by data switches 10-15, store **JMP 377** in location 377, then light **RUN** and begin normal operation by executing the instruction at location 377 [§ 2.8].

Note: On the 800 series the function of this switch can be duplicated by **PROGRAM LOAD** as described in §2.8.

**EXAMINE** can be used to load PC for beginning any single step procedure. Instruction stepping can also be begun by pressing **START** *while* holding **STOP** on.

To use the various examine and deposit switches between instruction steps, simply remember what PC is and restore it before continuing.

## 2.8 PROGRAM LOADING

Before a program can be executed it must be brought into memory. This requires that a loading program already reside in memory. If the memory is empty, the operator can use the automatic loading switches on a computer that is so equipped; otherwise he must use the data switches to deposit a bootstrap loader. Automatic loading is available as an option on the Nova 800 series and 1200 series and is standard on the Supernova computer.

### Automatic Loading

There are two types of automatic loading. The "program load" procedure uses programmed transfers for a low speed device such as teletype or paper tape reader. The "channel start" procedure uses the data channel for a high speed device such as magnetic tape or disc. Options 8108 and 8208 for the Nova 1200 series and 800 series respectively provide both of these procedures even though there is only one console switch. For this option the processor has two LSI chips that contain thirty-two words of read-only memory. Pressing the program load switch on the console starts the processor in a special sequence that deposits the read-only words into locations 0-37 and then begins normal program execution at location 0. The bootstrap loader generally supplied in these chips is capable of operating either with programmed transfers or the data channel. In the Supernova computer, these procedures are built in and are initiated by separate console switches. The channel start procedure can read an extensive loading program, but the program load procedure is ordinarily used to read in a short loader program that is then used for loading other information.

Below is the standard version of the bootstrap associated with the program load switch on the Nova 1200 series and 800 series computers. This program is capable of executing both the program load and channel start procedures. To load information, first set up the device that is to be used and set its code into data switches 10-15. For a high speed device such as magnetic tape or disc (which use the data channel), turn on data switch 0 (up); for a low speed device such as teletype or paper tape reader, turn off switch 0. Then press program load. The processor will automatically deposit the contents of the read-only LSI chips into locations 0-37 and then begin normal operation at location 0.

The bootstrap reads the data switches, sets up its own IO instructions with the specified device code, and then performs the program load or channel start type of operation as indicated by data switch 0. If the switch is

on, the bootstrap performs the channel start procedure: it starts the device for data channel storage beginning at location 0 and then sits at location 377 executing a JMP 377 until a data word loaded into 377 causes it to do something else. In other words location 377 eventually receives a data word, which the processor then executes as an instruction; this is typically a jump into the data just read or a halt.

#### NOTE

For proper channel operation, the device selected by the data switches must be initiated for reading by the combination of the IO reset and the START issued by the processor. Moreover it is up to the device to stop the transfer after 256 words have been read. The IO reset clears the location and word counters in the channel interface of the device so the transfer begins at location 0, but since the word counter is also zero the transfer will continue and fill all of memory unless the device stops it. The fixed-head disc is designed to read exactly 256 words; the magnetic tape stops at the end of the record and it is therefore up to the programmer to write a record of the proper length in the first place.

If switch 0 is off, the bootstrap reads low speed input in the program load manner. The device must supply 8-bit data bytes, and each pair of bytes is stored as a single word in memory, wherein the first and second bytes read become the left and right halves of the word. To simplify the positioning of the tape in the reader, the program ignores tape leader, *ie* it does not begin storing any words until it reads a nonzero synchronization byte. The first word following the sync byte must be the negative of the total number of words to be read (including the first word), for a maximum of 192 words. The program stores the words beginning at location 100; after reading all the data, it jumps to the last word stored.

Some of the techniques used here result from the fundamental restriction that the program be no longer than thirty-two words. Time, on the other hand, is not at all critical, as it is assumed that program load will be used only when some catastrophe wipes out the binary loader at the top of memory.

00000	062677	BEG:	IORST		;Reset all IO
00001	060477		READS	0	;Read switches into AC0
00002	024026		LDA	1,C77	;Get device mask (000077)
00003	107400		AND	0,1	;Isolate device code
00004	124000		COM	1,1	; - device code - 1
00005	010014	LOOP:	ISZ	OP1	;Count device code into all
00006	010030		ISZ	OP2	;IO instructions
00007	010032		ISZ	OP3	
00010	125404		INC	1,1,SZR	;Done?
00011	000005		JMP	LOOP	;No, increment again
00012	030016		LDA	2,C377	;Yes, put JMP 377 into location 377
00013	050377		STA	2,377	
00014	060077	OP1:	060077		;Start device; (NIOS 0) - 1
00015	101102		MOVL	0,0,SZC	;Low speed device? (test switch 0)
00016	000377	C377:	JMP	377	;No, go to 377 and wait for channel
00017	004030	LOOP2:	JSR	GET + 1	;Get a frame
00020	101065		MOVC	0,0,SNR	;Is it nonzero?

00021	000017		JMP	LOOP2	;No, ignore and get another
00022	004027	LOOP4:	JSR	GET	;Yes, get full word
00023	046026		STA	1,@C77	;Store starting at 100 (autoincrement)
00024	010100		ISZ	100	;Count word – done?
00025	000022		JMP	LOOP4	;No, get another
00026	000077	C77:	JMP	77	;Yes – location counter and jump to last ;word
00027	126420	GET:	SUBZ	1,1	;Clear AC1, set Carry
		OP2:			
00030	063577	LOOP3:	063577		;Done?: (SKPDN 0) – 1
00031	000030		JMP	LOOP3	;No, wait
00032	060477	OP3:	060477		;Yes, read in AC0: (DIAS 0,0) – 1
00033	107363		ADDCS	0,1,SNC	;Add 2 frames swapped – got second?
00034	000030		JMP	LOOP3	;No, go back after it
00035	125300		MOVS	1,1	;Yes, swap them
00036	001400		JMP	0,3	;Return with full word
00037	000000		0		;Padding

The usual procedure is to use the above bootstrap to bring in a larger program that sizes memory and then reads in the binary loader, storing it at the top.

Pressing the channel start switch on the Supernova computer console starts the processor in a special hardware sequence that simulates a DIAS that addresses the device whose code is selected by data switches 10-15, and then marks time while the channel is reading data. To start the channel, the operator must set up the device he is using, set its code into data switches 10-15, press the IO reset switch to clear the IO system, and press the channel start switch. The processor places the device in operation, then stores the instruction JMP 377 in location 377 and begins normal program execution at that location. Hence the processor keeps repeating the instruction in 377 while the channel stores data beginning at location 0. Eventually location 377 receives a data word, which is then executed by the processor as an instruction this is typically a jump into the data just read or a halt.

Pressing the program load switch on the Supernova computer console starts the processor in a special hardware sequence that simulates a series of sixty-six DIAS instructions, all of which address the device whose code is selected by data switches 10-15. The device must supply 8-bit data bytes, right justified. Each pair of bytes is stored as a single word in memory wherein the first and second bytes read become the left and right halves of the word. To simplify positioning of the tape in the reader, the processor ignores the tape loader, *ie* it does not begin counting the instructions it issues until the first nonzero byte is read.

To load a program automatically, the operator must set up the device he is using, set its code into data switches 10–15, press the IO reset switch to clear the IO system, and press the program load switch. The processor places the device in operation and upon encountering the first nonzero byte reads thirty-three pairs of bytes and stores the resulting words in memory beginning at location 0. Upon storing the thirty-third word in location 40, the processor executes the contents of that location; the last word in the block is thus normally a jump instruction into the body of code just read (or a halt to stop the processor). If the block contains fewer than thirty-three words the processor simply reads the trailing blank tape as zeros. In this case the word stored in location 40 is also zero and is executed as JMP 0. Typically the program is the same one used with the 800 and 1200 program load with the addition of a zero word (JMP 0) in location 40.

## Manual Program Loading

If an 800 series or 1200 series computer does not have the program load option, then to place information in memory without relying on a program already in memory, the operator must use the data switches to load one word at a time manually. The same procedure must be used for the Nova computer. The information loaded manually is usually a bootstrap loader, which is ordinarily used only to bring in a more extensive binary loader. This latter program is then used to read the object tapes of all other programs. The binary loader usually resides in high core where it is not disturbed by any of the standard software. But if an undebugged user routine inadvertently destroys the binary loader, it can be restored by first reloading the bootstrap manually.

Below are two versions of the standard bootstrap loader, one for the teletype reader, the other for the high speed reader (the programming for these devices is discussed in §§3.1 and 3.2). This program loads data relatively to its own position in memory. Although the bootstrap can be placed anywhere, the usual procedure is to place it in high core, beginning at the seventeenth (twenty-first octal) location from the top, so that the binary loader also resides in high core. The program is shown here for placement at the top of a 4K memory.

The bootstrap loader reads a tape in a special format in which each word is divided into four 4-bit characters. Each character occupies channels 1-4 (the right half) of a line on the tape. The first character of a word, containing bits 0-3, is indicated by a 1 in channel five. The tape can begin with any number of blank lines. The first two words are STA 1, +1 and JMP .-4, which are stored in the final two loader locations as indicated in the listing. The third, fifth, . . . words are STA instructions that address AC1, the fourth, sixth, . . . words are data. The bootstrap executes each odd-numbered word to store the succeeding data word in the location specified by the STA instruction. The final odd-numbered word is a HALT, which stops the processor.

In the following listings the first two columns at the left give each memory location and its contents for a 4K memory. The remaining columns are a standard program listing. To load the program simply use the switches to place the octal numbers in the locations specified. For a memory of any other size, load the bootstrap beginning at a location whose address is  $20_8$  less than the largest address.

### BOOTSTRAP LOADER, TELETYPE VERSION

07757	126440	GET:	SUBO	1,1	;Clear AC1, Carry
07760	063610		SKPDN	TTI	
07761	000777		JMP	.-1	;Wait for Done
07762	060510		DIAS	0,TTI	;Read into AC0 and restart reader
07763	127100		ADDL	1,1	;Shift AC1 left 4 places
07764	127100		ADDL	1,1	
07765	107003		ADD	0,1,SNC	;Add in new word
07766	000772		JMP	GET+1	;Full word not assembled yet
07767	001400		JMP	0,3	;Got full word, exit
07770	060110	BSTRP:	NIOS	TTI	;Enter here, start reader
07771	004766		JSR	GET	;Get a word
07772	044402		STA	1, +2	;Store it to execute it
07773	004764		JSR	GET	;Get another word
4	170000		...		;This will contain an STA (first STA 1, +1)
5	063077		...		;This will contain JMP .-4
6	000000				



## BOOTSTRAP LOADER, HIGH SPEED READER VERSION

07757	126440	GET:	SUBO	1,1	;Clear AC1, Carry
07760	063612		SKPDN	PTR	
07761	000777		JMP	.-1	;Wait for Done
07762	060512		DIAS	0,PTR	;Read into AC0 and restart reader
07763	127100		ADDL	1,1	;Shift AC1 left 4 places
07764	127100		ADDL	1,1	
07765	107003		ADD	0,1,SNC	;Add in new word
07766	000772		JMP	GET+1	;Full word not assembled yet
07767	001400		JMP	0,3	;Got full word, exit
07770	060112	BSTRP:	NIOS	PTR	;Enter here, start reader
07771	004766		JSR	GET	;Get a word
07772	044402		STA	1,+2	;Store it to execute it
07773	004764		JSR	GET	;Get another word
			...		;This will contain an STA (first STA 1,+1)
			...		;This will contain JMP .-4

To use the bootstrap to load the binary loader or any other program in the special format, follow these steps:

1. Put the special format tape in the reader and turn it on.
2. Press RESET.
3. For a 4K system set the data switches to 007770 (7 less than the largest address).
4. Press START.

### Binary Loader

A standard loader for loading program tapes in the type of object tape format generated by the assembler [refer to the assembler manual] is available in several forms. Program tape number 091-000036 (write-up 093-000055) is the binary loader prefaced by the sizing and loading program for use with the Nova 800 series and 1200 series program load; 091-000041 (writeup 093-000055) is the binary loader prefaced by both the equivalent Supernova computer bootstrap and sizing and loading program; 091-000004 (writeup 093-000003) is the binary loader for use with the manually loaded bootstrap given above. Following an automatic load, the operator can read an object tape on the same device simply by pressing CONTINUE. To load an object tape in any other circumstances, follow this procedure.

1. Put the object tape in the paper tape reader or teletype.
2. Set the data switches to x7777.
3. If you are using the paper tape reader, turn on data switch 0; otherwise turn it off.
4. Press START.

If a starting address is given on the object tape, control will be transferred to that location when loading is complete. Otherwise, the loader will halt with the address lights displaying x7740, and the user must start the program from the console.

The binary loader computes a checksum over every data block and start block read. If a checksum failure occurs over a block, the loader halts with x7726 displayed in the address lights. Reposition the tape to the beginning of the last block read and press CONTINUE. If the checksum failure again occurs, the object tape is probably in error. Generate a new tape before attempting to load the program again.

# Chapter III

## Basic I/O Equipment

This chapter discusses the simpler peripheral devices: teletypewriter, tape reader, tape punch, card reader, card punch, plotter, line printer, and DGC Cassette. These devices are used primarily for communication between computer and operator using either a paper medium: tape, cards, form paper, or graph paper, or a magnetic tape cassette. All transfers for them are made by the program through accumulators, except with the DGC Cassette, which uses the data channel.

The program can type out characters on the teletypewriter and can read characters that have been typed in at the keyboard. This device has the slowest transfer rate of any, but it provides a convenient means of man-machine interaction. The KSR teletypewriters comprise only a keyboard and printer; the ASR models also have a slow tape reader and punch. This punch and the separate high speed punch supply output in the form of 8-channel perforated paper tape. The information punched in the tape can be brought into the processor by the high speed tape reader or the one mounted in the teletypewriter.

The card equipment processes standard 12-row 80-column cards. Many programmers find cards a convenient medium for source program input and for supplying data that varies from one program run to another. Cards and paper tape are both convenient to prepare manually, but card input is much faster than tape, and simple changes are easier to make: individual cards can be repunched, and cards can be added or removed from the deck. A possible consideration in using cards is that many installations do not include an online card punch.

The line printer provides text output at a relatively high rate. The program must effectively typeset each line; upon command the printer then prints the entire line. With the plotter, the program can produce ink drawings by controlling the incremental motion of pen on paper in a cartesian coordinate system. Curves and figures of any shape can be generated by proper combinations of motion in  $x$  and  $y$ .

The DGC Cassette provides input and output to single channel magnetic tapes which are housed in handy, portable, interchangeable cassettes.

### 3.1 TELETYPEWRITER

Five teletypewriter models are regularly available: the ASR33, KSR33 and KSR35, all of which are capable of speeds up to ten characters per second, and the KSR37 and ASR37, which can handle up to fifteen characters per second. The program can type out characters and can read in the characters produced when keys are struck at the keyboard. With an ASR the program can also punch characters in a tape and read characters from a tape.

The teletype separates its input and output functions and is really two distinct devices. Each has its own device code, its own Busy, Done and Interrupt Disable flags, and its own interrupt priority mask assignment. Placing a code for a character in the output buffer and setting Output Busy causes the teletype to print the character or perform the designated control function. Striking a key places the code for the associated char-

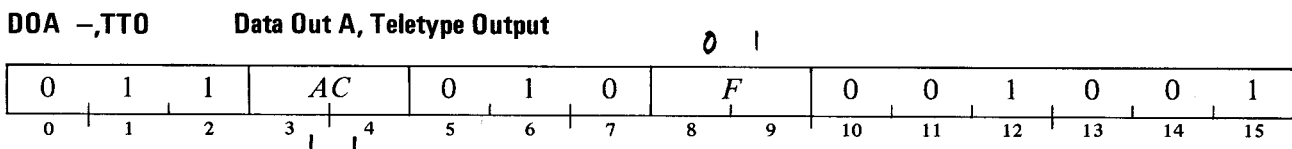
acter in the input buffer where it can be retrieved by the program, but it does nothing at the teletype unless the program sends the code back as output.

Character codes received from the keyboard have eight bits wherein the most significant is an even parity bit. The Model 33 and 35 printers ignore the parity bit in characters transmitted to them. The model 37 ignores the parity bit in a code for a printable character, but it performs no function when it receives a control code with incorrect parity.

The Model 37 has the entire character set listed in the table in Appendix E. Lower case characters are not available on the Model 33 or 35, but transmitting a lower case code to the teletype causes it to print the corresponding upper case character. (There are, of course, no restrictions on the codes that can be punched in or read from tape.) To go to the beginning of a new line the program must send both a carriage return, which moves the type block or box to the left margin, and a line feed, which spaces the paper. The horizontal and vertical tabs and form feed have no effect on the Model 33 printer.

### Teletype Output

The teletype output uses only one IO transfer instruction. Output Busy and Output Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 11, mnemonic TTO. Output Interrupt Disable is controlled by interrupt priority mask bit 15.



Load the contents of AC bits 8-15 into the teletype output buffer, and perform the function specified by *F*.

Setting Output Busy turns on the transmitter, causing it to send the contents of the output buffer serially to the teletype (the buffer is cleared during transmission). The printer prints the character or performs the indicated control function. If the punch is on, the character is also punched in the tape, with AC bit 15 corresponding to channel 1 (a 1 in AC produces a hole in the tape). Completion of transmission clears Output Busy and sets Output Done, requesting an interrupt if Output Interrupt Disable is clear.

#### NOTE

Although the buffer clears during transmission, giving an NIOS without loading it again does not transmit a zero character. So do not give an NIOS without first loading the buffer. To transmit any character including null, either give a DOAS or give a DOA followed by an NIOS.

#### CAUTION

Clearing Output Busy while the transmitter is running (as with an NIOC) terminates the transmission. But the printer still prints whatever character is represented by the indeterminate code it receives.

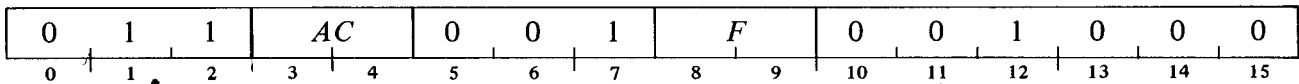
**Timing.** Models 33 and 35 can type or punch up to ten characters per second. After Output Done is set, the program has 4.55 ms to give a DOAS to keep typing or punching at the maximum rate. The 37 can handle fifteen characters per second, 66.7 ms per character. After Output Done is set, the program has 3.33 ms to send a new character to maintain the maximum typing rate.

The sequence carriage return-line feed, when given in that order, allows sufficient time for the type block to get to the beginning of a new line. After tabbing, the program must wait for completion of the mechanical function by sending one or two rubouts. If the time is critical, the programmer should measure the time required for his tabs. Tabs are normally set every eight spaces (columns 9, 17, . . .) and require one rubout.

### Teletype Input

The teletype input uses only one IO transfer instruction. Input Busy and Input Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 10, mnemonic TTI. Input Interrupt Disable is controlled by interrupt priority mask bit 14.

**DIA - ,TTI      Data In A, Teletype Input**



Transfer the contents of the input buffer into AC bits 8–15, and perform the function specified by *F*. Clear AC bits 0–7.

Reception from the keyboard requires no initiating action by the program; striking a key transmits the code for the character serially to the input buffer. However, if the reader is under program control, giving the start function (NIOS or DIAS) sets Busy and causes the reader to read all eight channels from the next line on tape and transmit the line serially into the buffer (the presence of a hole produces a 1 in the buffer). In either case completion of reception clears Input Busy and sets Input Done, requesting an interrupt if Input Interrupt Disable is clear. When the character is brought into AC, tape channel 1 corresponds to AC bit 15.

**Timing.** After Input Done is set by a Model 33 or 35, the character is available for retrieval by a DIA for 21.59 ms before another key strike can destroy it. If the reader is in use, the program has 3.41 ms to give a DIAS (or DIA and NIOS) and keep the tape in continuous motion. With the 37, the character is available for 9.17 ms after Input Done is set.

### Programming Examples

There are basically two procedures for using the skip instructions in a loop to process a series of characters. Consider this loop for typing out (we assume the printer is not in use).

```

OUT:   DOAS   AC,TTO       ;Type out
       SKPDN  TTO         ;Wait till transmission done
       JMP    .-1
       .
       .
       JMP    OUT        ;Go back
  
```

This procedure is very poor as most of the time is spent waiting during the transmission, and there is very little time to do anything afterwards if we are to go back to type out the next character at full speed. But with this arrangement:

```

OUT:      SKPBZ   TTO           ;Wait till printer free
          JMP     .-1
          DOAS   AC,TTO       ;Type out character
          .
          .                   ;Compute, etc
          .                   ;Get next character
          JMP    OUT         ;Go back

```

we have almost all of the time for worthwhile program and we can run at full speed provided only that we jump back to OUT before the entire teletype cycle time is over. Also, the first time into the loop we wait until any previous (perhaps unknown to us) teletype output operation is finished.

The same dichotomy exists for input operations. This is bad:

```

IN:       NIOS    TTI           ;Read character
          SKPDN   TTI           ;Wait till reception done
          JMP     .-1
          DIA     AC,TTI        ;Bring in character
          .
          .                   ;Decide whether to read another character, etc
          .
          JMP    IN            ;Go back

```

but this is good:

```

IN:       NIOS    TTI           ;Read first character
          SKPDN   TTI           ;Wait till reception done
          JMP     .-1
          DIAS AC,TTI          ;Bring in character and read another
          .
          .                   ;Compute, etc
          .
          JMP    IN            ;Go back

```

Of course the last program does not allow us to inspect a character to determine whether to get another one. So for the best of all possible worlds we combine the procedures.

```

IN:       NIOS    TTI           ;Read character
          .
          .                   ;Lots of time to compute
          .
          SKPDN   TTI           ;Wait till reception done
          JMP     .-1
          DIA     AC,TTI        ;Bring in character
          .
          .                   ;Decide whether to get another
          .
          .
          JMP    IN            ;Do this if want another
          .
          .                   ;Skip to here if not
          .

```

## Operation

A KSR is actually two independent devices, keyboard and printer, which can be operated simultaneously. An ASR is really four devices, keyboard, printer, reader and punch, which can be operated in various combinations. Power must be turned on by the operator. On the 33 and 35 the switch is beside the keyboard and is labeled LINE/OFF/LOCAL or ON/OFF and has an unmarked third position opposite ON. A similar switch is located beneath the stand on the 37. When this switch is set to LOCAL or the unmarked position, power is on but the machine is off line and can be used like a typewriter. Moreover, in an ASR, turning on the punch allows the operator to punch a tape from the keyboard, and running the reader allows a tape to control the printer (if the punch is also on, it duplicates the tape).

Turning the switch to LINE or ON connects the unit to the computer and separates its input and output functions. Thus any information transmitted to the computer from the keyboard affects the printer only insofar as the computer sends it back. Turning on the reader places it under program control, and turning on the punch causes it to punch whatever is sent to the printer by the computer.

The only control on the reader is a 3-position switch. When the switch is in the FREE position, the tape can be moved by hand freely through the reader mechanism. The STOP position engages the reader clutch so the tape is stationary but the reader is still off. Turning the switch to START causes the reader to read the tape if the unit is in local, but places it under program control if on line.

The operator controls the punch by means of four pushbuttons. The two on the right turn the punch on and off. Pressing the REL. button releases the tape so it can be moved by hand through the punch mechanism. Pressing B. SP. moves the tape backward one frame so the operator can delete a frame that is incorrect by striking the rubout key. Pressing HERE IS with the keyboard in local punches twenty lines of blank tape (lines with only a feed hole punched).

The keyboard resembles that of a standard typewriter. Codes for printable characters on the upper parts of the key tops on the 33 and 35 are transmitted by using the shift key; most control codes require use of the control key. Those familiar with the 33 or 35 who are using the 37 for the first time should take a close look at the keyboard. On the 37 the shift is used for real upper case characters. The control key is used for some control characters, but many have separate keys. Note also that both the keyboard arrangement and the labels differ somewhat. On all models the line feed (labeled "new line" on the 37) spaces the paper vertically at six lines to the inch, and must be combined with a return to start a new line. The local advance (feed) and return keys affect the printer directly and do not transmit codes. Appendix E lists the complete teletype code, ASCII characters, key combinations, and differences among the several models.

On the 33 and 35 is a repeat button REPT. Pressing this button and striking any character key causes transmission of the corresponding code so long as REPT is held down. Characters that require the shift key may also be repeated in this manner, but there is no repetition of control characters.

Teletype manuals supplied with the equipment give complete, illustrated descriptions of the procedures for loading paper and tape, changing the ribbon, and setting horizontal and vertical tabs. Setting tabs is usually left for maintenance personnel; in any event, the best and easiest way to learn how to do any of these things is to have someone who knows show you how. However, as a precautionary measure we describe here the things you may have to do yourself.

**Tape.** The tape moves in the reader from back to front with the feed holes closer to the left edge. To load tape, set the switch to FREE, release the cover guard by opening the latch at the right, place the tape so that the sprocket wheel teeth engage the feed holes, close the cover guard, and set the switch to STOP.

To load tape in the punch, raise the cover, feed the tape manually from the top of the roll into the guide at the back, move the tape through the punch by turning the friction wheel, then close the cover. Turn on the punch with the unit in local and punch about two feet of leader by pressing HERE IS or the control, shift and P keys to generate null codes.

**Paper.** The 33 printer has an 8½-inch roll of paper at the back. Printed sections can be torn off against the edge of the glass window in front of the platen. To replenish the paper, snap open the cover, remove the old roll and slip a new one in its place. Draw the paper from the roll around the platen as in an ordinary typewriter.

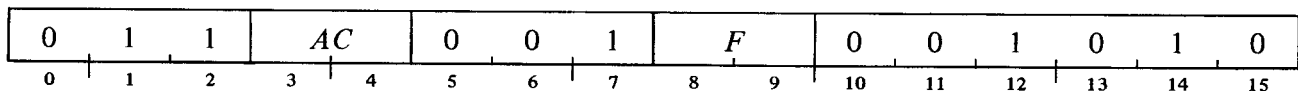
The 35 and 37 printers have a sprocket feed and use 8½ × 11 fanfold form paper. The supply is held in a tray at the back. To replenish it, first remove the upper cover by pressing the cover release button on the right side. To free the remaining old paper for removal, lift the paper guides by pushing the handle marked PUSH at the right of the platen. To insert new paper from the tray, bring it up below the platen at the rear, line up the holes at the edges of the paper with the sprockets, and press line feed (in local) to draw the paper under the platen.

**Ribbon.** Replace the ribbon whenever it becomes worn or frayed or the printing becomes too light. Disengage the old ribbon from the ribbon guides on either side of the type block, and remove the reels by lifting the spring clips on the reel spindles and pulling the reels off. Remove the old ribbon from one of the reels and replace the empty reel on one side of the machine; install a new reel on the other side. Push down both reel spindle spring clips to secure the reels. Unwind the fresh ribbon from the inside of the supply reel, over the guide roller, through the two guides on either side of the type block, out around the other guide roller, and back onto the inside of the takeup reel. Engage the hook on the end of the ribbon over the point of the arrow in the hub. Wind a few turns of the ribbon to make sure that the reversing eyelet has been wound onto the spool. Make sure the ribbon is seated properly and feeds correctly in operation.

### 3.2 PAPER TAPE READER

The high speed reader processes 8-channel perforated paper or mylar tape photoelectrically at speeds up to 400 lines per second. It uses only one IO transfer instruction to retrieve data from an 8-bit buffer in the interface. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 12, mnemonic PTR. Interrupt Disable is controlled by interrupt priority mask bit 11.

**DIA - ,PTR      Data In A, Paper Tape Reader**



Transfer the contents of the reader buffer into AC bits 8–15, and perform the function specified by *F*. Clear AC bits 0–7.

Setting Busy causes the reader to read all eight channels from the next line on tape into the buffer (the presence of a hole produces a 1 in the buffer). When the operation is complete the reader clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. When the character is brought into AC, tape channel 1 corresponds to AC bit 15.

Clearing Busy stops the reader.

**Timing.** At 400 lines per second the reader takes 2.5 ms per character. After Done is set, the program has 100 μs to retrieve the character and set Busy to keep the tape in continuous motion. Waiting longer forces

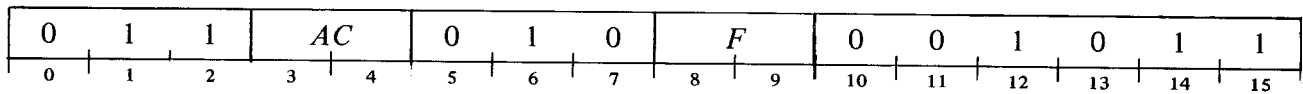
the reader to stop and restart, and the program should not attempt to operate the reader in this manner at rates above 150 lines per second. Faster start-stop rates produce reader chatter, which is not only rather annoying but also conducive to less reliable reader operation.

**Operation.** An OFF/RUN/FEED switch is located in the center of the front of the paper tape reader. On either side of the front are bins to hold fanfold paper or mylar tape. To load the reader, press the switch to its OFF position and raise the brake keeper lever to its upper position. Place the perforated tape vertically in the right-hand tape bin so that the sprocket holes are to the rear and the beginning of the tape is on the top. Remove three or four folds of tape from the bin and place them in the left-hand bin, slipping the portion between the bins into the horizontal slit and under the brake keeper. Lower the brake keeper and press the switch to RUN. After the tape has been read, press and hold the switch in the FEED position until all of the tape has been fed into the left-hand bin. Press the switch to OFF and remove the tape.

### 3.3 PAPER TAPE PUNCH

The punch perforates 8-channel paper tape at speeds up to 63.3 lines per second. It uses one IO transfer instruction to load data into an 8-bit buffer in the interface. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 13, mnemonic PTP. Interrupt Disable is controlled by interrupt priority mask bit 13.

**DOA  $\bar{}$ ,PTP**      **Data Out A, Paper Tape Punch**



Load the contents of AC bits 8–15 into the punch buffer, and perform the function specified by *F*.

Setting Busy causes the punch to punch the contents of the buffer in the tape with AC bit 15 corresponding to channel 1 (a 1 in AC produces a hole in the tape). After punching is complete, the device clears Busy sets Done, requesting an interrupt if Interrupt Disable is clear.

**Timing.** While the punch motor is on, punching is synchronized to a punch cycle of 15.8 ms. After Done sets, the program has 11.3 ms to give a new DOAS to keep punching at the maximum rate; after 11.3 ms punching is delayed until the next cycle.

The standard punch must be left on all the time that it might be used as it otherwise will not respond to the program. With the power option the punch can be left off. Then if Busy is set when the motor is off, punching is automatically delayed 1 second while the motor gets up to speed. While the motor is on, timing is as given above. It can be assumed that the motor will remain on throughout any normal punching run. But if Busy remains clear for 5 seconds the motor turns off.

**Operation.** Fanfold tape is fed from a box behind the punch inside its enclosure. After it is punched, the tape moves into a storage bin from which the operator may remove it through a slot in the front. Pushing the feed button beside the slot clears the buffer and punches blank tape (tape with only feed holes punched) as long as it is held in, provided either the power toggle switch is on or the punch has the power option. The power switch overrides the logic and keeps the motor on continuously.

To load tape, first empty the chad box. Then tear off the top of a box of fanfold tape (the top has a single flap; the bottom of the box has a small flap in the center as well as the flap that extends the full length of the box). Set the box in the frame and thread the tape through the punch mechanism. The arrows on the tape should be on top and should point in the direction of tape motion. If they are underneath, turn the box



around. If they point in the opposite direction, the box was opened at the wrong end; remove the box, seal up the bottom, open the top, and thread the tape correctly.

To facilitate loading, tear or cut the end of the tape diagonally. Thread the tape under the out-of-tape plate, open the guide plate (over the sprocket wheel), push the tape beyond the sprocket wheel, and close the guide plate. Press the feed button long enough to punch about a foot and a half of leader. Make sure the tape is feeding and folding properly in the storage bin.

To remove a length of perforated tape from the bin, first press the feed button long enough to provide an adequate trailer at the end of the tape (and also leader at the beginning of the next length of tape). Remove the tape from the bin and tear it off at a fold within the area in which only feed holes are punched. Make sure that the tape left in the bin is stacked to correspond to the folds; otherwise, it will not stack properly as it is being punched. After removal, turn the tape stack over so the beginning of the tape is on top, and *label it with name, date, and other appropriate information.*

### 3.4 LINE PRINTER

Two line printers are regularly available; these are Models 4034A and 4034B, which output hardcopy composed of lines 80 and 132 characters long respectively. The printing speed in lines per minute is a function of the number of columns printed from the left edge of the paper as follows.

<i>Model 4034A</i>		<i>Model 4034B</i>	
<i>Columns</i>	<i>Lines per minute</i>	<i>Columns</i>	<i>Lines per minute</i>
20	1110	24	1110
40	650	48	650
60	460	72	460
80	356	96	356
		120	290
		132	245

There are sixty-four printing characters available to the program. The characters and codes are the figure and upper case sets, codes 040–137, in the teletype code [*Appendix E*] with the exception that codes 134, 136 and 137 respectively are an open diamond, the AND symbol ( $\wedge$ ) and an open heart. Besides accepting printing characters, the printer responds to three control characters, CR, LF and FF. All other codes are interpreted as **space characters**.

Each line is printed from left to right in zones, and the printer has a buffer that holds the image of a single zone. The 4034A has a 20-character buffer and printing is in four zones of twenty columns each; the 4034B has a 24-character buffer and printing is in six zones, where the first five are twenty-four columns each, the sixth is twelve columns. To print a line, the program must first load the buffer one character at a time for zone 1 even if all the characters are spaces. Once the buffer is full, the characters are printed automatically and at the completion of the print cycle, the program can fill the buffer for zone 2. However, for each full line the program need send out characters, including spaces, only as far as the rightmost nonspace character; giving a control character at this point prints the current zone with only the filled portion of the buffer producing a printout. When printing is caused by a control character or the filling of the buffer in the rightmost zone (in zone 6 on the 4034B the buffer is “full” when twelve characters are loaded), the printer then returns to zone 1; in other words, in the next print cycle the contents of the buffer will be printed at the left edge of the paper.

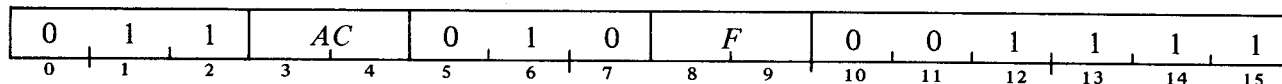
The standard paper has 11-inch pages. Spacing is six lines per inch and the image area is sixty-three lines (there is automatically a half-inch space across the perforation between pages). Paper spacing is produced by the control characters. An LF spaces the paper one line after the zone is printed; an FF spaces the paper to the top line of the next page. If the program prints a whole line without spacing (either by giving a CR or filling

the buffer in the final zone), subsequent print cycles can overprint, *ie* print other characters in column positions already printed. With this technique the program can produce a character such as “≠” by overprinting a slash on an equal sign (or vice versa). Programmers commonly use the combination CR plus LF to print and space for compatibility with the teletype. Just as horizontal tabbing is accomplished by giving strings of spaces, vertical tabbing is produced by strings of line feeds.

In a print cycle the characters are printed in the order that they pass the print hammers, and a given character is printed simultaneously in all positions that require it. In other words the drum has a row of 80 or 132 *Ms*, a row of *Ns*, etc; all *Ms* are printed together, all *Ns* together, and so forth. The first character printed depends only upon the position of the drum when the print cycle begins. The drum has sixty-four rows of characters of which only sixty-three are used; the printer produces spaces in a zone by not printing anything in the columns corresponding to the buffer positions that hold space characters.

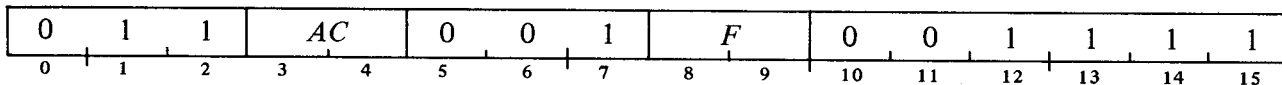
**Instructions.** The printer uses two of the IO transfer instructions, one to load a single character into a 7-bit buffer in the interface, the other to read a single status bit. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 17, mnemonic LPT. Interrupt Disable is controlled by interrupt priority mask bit 12.

**DOA  $\rightarrow$  LPT      Data Out A, Line Printer**



Load *AC* bits 9–15 into the character buffer and perform the function specified by *F*.

**DIA  $\rightarrow$  LPT      Data In A, Line Printer**



Read the Ready status into *AC* bit 15, clear *AC* bits 0–14, and perform the function specified by *F*. A 1 read into *AC* bit 15 indicates that the printer is available to the program (*eg* it is on line, with power on and paper loaded).

At the beginning of a print run the program should check Ready and send a form feed to get rid of anything that may have been left in the zone buffer and start on a new page. The program can then set each zone and print by giving DOASs that send the appropriate characters. Start sets Busy and sends the contents of the character buffer to the printer. If the character sent neither fills the buffer nor is a valid control character, the printer clears Busy after 6  $\mu$ s without setting Done; the program can then supply another character to the printer (the contents of the character buffer remain until a new DOA is given). If the character sent fills the zone buffer or is a valid control character, Busy remains set while the printer prints the contents of the buffer. When the buffer again becomes available, Busy clears and Done sets, requesting an interrupt if Interrupt Disable is clear, and subsequent characters will be loaded starting in the first buffer position. At the completion of the print cycle, the printer either advances to the next zone or returns to zone 1 with or without spacing the paper depending upon the condition that initiated the print cycle as explained above. If printing is caused by a CR or a full buffer in the final zone, the next line will overprint unless the paper is advanced before any nonspace characters are loaded into the zone buffer.

**Timing.** The program can load the buffer and print by giving DOASs separated by at least 6  $\mu$ s. The most convenient way to produce this delay is simply to give the necessary number of no-ops to wait and then check Busy to determine whether the printer can accept another character or has entered a print cycle. The program must load the zone buffer within 200  $\mu$ s to keep the printer going at the maximum rate. The overall time required for a print run is the total printing and spacing time for all lines. Buffer loading time is generally not a factor in total printer operating time because the buffer becomes available in time for the program to load it before the next print cycle can start or the paper stops.

Each print cycle takes 34 ms, spacing one line requires 20 ms. If before the paper stops, the program gives another spacing character without first loading any printing characters in the buffer, the paper will move at the slew rate of 13 ms per line (13 inches per second). The paper also moves at the slew rate when it is spacing to a top of form.

**Operation.** On the top of the cabinet are three toggle switches and three indicators, including a red power light. The right toggle has a center null position and two momentary-contact positions. Pushing the switch toward the back of the printer places it on line, lighting the ON LINE indicator, provided the READY light is on. This last light indicates that power is on, paper is loaded, the drum gate is closed, and the drive motor is not overheated. The Ready status flag is set when both READY and ON LINE are lit. When the unit is off line, the operator can use the other two toggles to step the paper a single line or run it to the top of the next page. The main power circuit breaker is at the lower left behind the front panel, which can be opened by pushing the button at the right.

The printer uses 11-inch fanfold form paper with edge holes a half-inch apart. The minimum single copy weight is 15 pound bond, but the printer can also handle multiple copies of up to six parts of 12 pound bond with carbons. Paper width can be 4 to 9 $\frac{1}{8}$  inches on the 4034A, 4 to 14 $\frac{1}{8}$  inches on the 4034B. To load paper, open the front of the printer. At the left edge inside the printer is a lever with a black knob: push this lever to the left and up, and swing the drum gate out to the right. Press TOP OF FORM to position the tractors and form cam. Open the tractor guides, and place the paper on the tractor teeth with a perforation aligned with the red arrow on the left just above the hammer bank. Close the tractor guides, and if necessary, adjust the perforation to the arrow by means of the black vernier knob in the upper left (moving the knob left and right moves the paper up and down). Close the drum gate, push the gate latch down and to the right, close the front panel, and place the printer on line.

For information on ribbon changing, maintenance controls, test operation, and paper position and tension adjustments, refer to the printer manual.

#### *CAUTION*

On models 4034A and 4034B, when changing the ribbon, make sure to put the fat roll at the top.

### 3.5 PLOTTER

The plotter control interfaces the processor to various plotters that use cartesian coordinates. The following lists the type and paper size of the most commonly supplied models.

<i>Model</i>	<i>Type</i>	<i>Paper size in inches</i>
4017D	Bed	31 × 34
4017C	Drum	29½ × 1440
4017E	Bed	11 × 1734

These are high accuracy, incremental digital plotters that produce fine quality ink plots of computer-generated data. Bidirectional stepping motors provide individual increments of motion in either coordinate or both at once. The program draws a continuous sequence of line segments by controlling the relative motion of pen and paper with the pen lowered, and it can raise the pen for repositioning. The 4017E uses fanfold paper perforated for 11 X 8½ or 17.

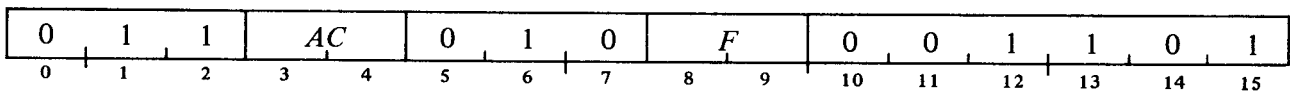
Motion in *y* is movement of the pen carriage along a rod or pair of rods. Motion in *x* is movement of the entire carriage-and-rod mechanism on the 4017D bed plotter, movement of the paper underneath the carriage on the drum type or the 4017E. On a bed plotter the coordinate directions are the standard ones when viewing the device from the front: positive *x* to the right, positive *y* to the back. The coordinate system on a drum is in the standard orientation when the viewer is standing at the right side, unrolling the paper from the drum with his left hand. In other words positive *y* is movement of the pen from right to left across the drum, positive *x* is drum rotation downward at the front (drawing a line toward the paper supply roll at the back).

The step sizes and plotting speeds available with the various models are the following.

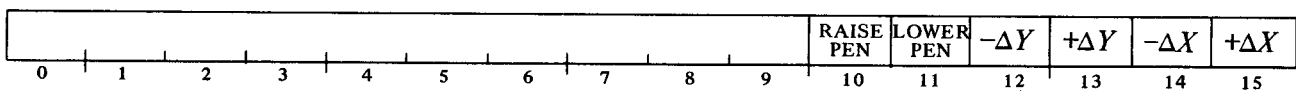
<i>Model</i>	<i>Step size</i>	<i>Plotting speed in steps per second</i>	<i>Time per step in ms</i>
4017D	.01 inch	300	3.3
	.005 inch		
	.002 inch		
	.1 mm		
4017C	.05 mm	200	5
	.01 inch		
	.005 inch		
	.1 mm		
4017E	.01 inch	300	3.3
	.005 inch		
	.25 mm		
	.1 mm		

The plotter uses only one IO transfer instruction, and the program can draw any complete figure by giving a string of them, with each supplying the information for one step. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions with device code 15, mnemonic PLT. Interrupt Disable is controlled by interrupt priority mask bit 12.

**DOA - ,PLT      Data Out A, Plotter**



Load plotting information from AC bits 10-15 into the plotter command register as shown, and perform the function specified by *F*.



Setting **Busy** causes the plotter to execute the plotting command given by the last DOA. After sufficient time has elapsed for the device to carry out the specified action, the control clears **Busy** and sets **Done**, requesting an interrupt if **Interrupt Disable** is clear.

To avoid drawing line segments shorter than one step, do not raise or lower the pen in the same DOA that calls for  $xy$  motion. Specifying contradictory movements results in no motion in the given dimension.

**Timing.** Raising or lowering the pen takes 100 ms. The time required to move one step in either or both coordinates depends on the plotting speed as given in the above table.

**Operation.** On a drum plotter the supply roll is behind the drum. Bring the paper over the drum, down in front, and above and behind the pickup roll underneath the drum (use a piece of masking tape to attach the paper, or roll some onto the tube).

To put the plotter on line simply turn on the power and the chart drive. The remaining controls are for manual operation: raising and lowering the pen, moving the carriage and drum in either direction, rapidly or single step. The 4017D bed plotter has similar controls.

To load paper in the 4017E plotter, lift open the cover by lifting the lid knob while pressing its center. At the right rear is a support that can be snapped into place to hold the lid open. Pull the paper over the plotter bed from a supply pile at the right, making sure that the sprocket teeth engage the holes in all four corners (the round holes should be at the back), and snap the lid shut. Photographs and drawings of the plotter and information on the types of pens and how to change them are given in the plotter instruction manual.

To put the plotter on line simply press the power button, turn the pen switch to **REMOTE**, and turn the chart and pen axis switches to **PLOT**. The **POWER** and **READY** lights should be lit. For manual operation the pen switch can be used to move the pen up and down, the other two switches can be used to enable the motion pushbuttons at the left of each switch. Pressing a button produces motion of the pen or chart in the direction of the arrow. Note that chart motion is diametrically opposed to motion in the  $x$  coordinate: moving the chart to the right plots a line toward the left, *ie* in the  $-x$  direction. The movement produced by a button depends on the position of the associated axis switch: with the switch set to **JOG** each button push produces a motion of one step in the corresponding coordinate; the **SLEW** position produces motion at full plotting speed as long as the button is held down.

### 3.6 CARD READER

The card readers handle standard 12-row 80-column punched or Mark Sense cards at speeds up to 1,000 cards per minute. Once started, an entire card is read column by column. The reader supplies each column to the processor as twelve bits, and the program can translate in any way it wishes; the standard DGC character representations and the translation to ASCII made by the software are given in Appendix E. Of course the data can simply be in binary (a 7 and 9 punch in the first column is the standard indication that the rest of the card contains binary data).

The card reader has device code 16, mnemonic **CDR**, and uses two IO transfer instructions, one to retrieve each column from a 12-bit buffer in the interface, the other to read status. **Busy** and **Done** are sensed by bits 8 and 9 in the IO skip instructions and are controlled in the usual fashion by the **Clear** and **Start** functions, but the IO Pulse function ( $F = 11$ ) is also used to clear **Done** without affecting **Busy**. **Interrupt Disable** is controlled by interrupt priority mask bit 10.

**DIA -CDR Data In A, Card Reader**

0	1	1	AC	0	0	1	F	0	0	1	1	1	0		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Transfer the contents of the column buffer into AC bits 4-15 where the correspondence of card rows to bit positions is as shown, and perform the function specified by *F*. Clear AC bits 0-3.

				ROW 12	ROW 11	ROW 0	ROW 1	ROW 2	ROW 3	ROW 4	ROW 5	ROW 6	ROW 7	ROW 8	ROW 9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

**DIB -CDR Data In B, Card Reader**

0	1	1	AC	0	1	1	F	0	0	1	1	1	0		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the status of the reader into AC bits 11-15 as shown, and perform the function specified by *F*. Clear AC bits 0-10.

	HOPPER EMPTY STACKER FULL	PICK FAILURE	TROUBLE	READY	CARD IN READER
10	11	12	13	14	15

- 12 The reader has received a read command but has failed to bring in a card from the hopper.
- 13 A card has failed to move properly through the reader (it has probably slipped) or an error has been detected in the photoelectric circuitry. When Trouble sets the reader stops at the end of the current card, and the program should be dubious of any data taken from it.
- 14 The reader is ready to accept a read command (all other status bits are 0).
- 15 The reader has brought a card in from the hopper and has not yet finished reading it.

Before trying to read a deck the program should check Ready. To start every card the program must give Start, either in an NIOS or while checking status with a DIBS. Setting Busy causes the reader to pick a card; movement of the card in from the hopper sets Card in Reader. As each column is loaded into the buffer (the presence of a hole produces a 1 in the buffer), Done sets, requesting an interrupt if Interrupt Disable is clear. The program must respond with a DIAP to bring in the column and clear Done.

After all eighty columns have been read, the card moves out to the stacker, and Card in Reader goes off, clearing Busy and setting Done, again requesting an interrupt.

Note that Done does double duty as both a column ready flag and a card done flag, and thus sets eighty-one times per card. In this case Busy and Done both set is legitimate: Busy remains set throughout the card even though Done sets on each column and the program must respond to each column with IO Pulse to clear Done without affecting Busy.

**Timing.** The timing of the sequence of operations that process a card depends upon whether the reader handles 225 or 400 cards per minute (figures for the latter are given in parentheses). After Busy sets, 65 (37) ms elapse before Card in Reader goes on. The first column Done occurs 7.5 (4.2) ms later. Subsequent columns are ready every 2.4 (1.35) ms — the program must give a DIAP within 2.175 (1.25) ms after each Done or data will be missed. Total time from first to last column Done is 189.6 (107) ms. After Done sets for

the eightieth column, 7.2 (4.05) ms elapse before Card in Reader clears, clearing Busy and again setting Done. The program then has 150 (84)  $\mu$ s within which to give a new Start to keep the reader going at the maximum rate. These times are determined by mechanical operations and may therefore vary by as much as 20 percent.

**Operation.** The reader has a hopper and stacker capacity of 500 cards. To load a deck, first fan and flex the cards and jog them on top of the reader. Turn the deck over and put the first hundred cards (about an inch of the deck) into the hopper (at the right) with the 9 edge against the back so column 1 is read first. Place the rest of the deck on top of the first part and put the card weight on top of the deck. Cards can be added to the hopper while the reader is running provided at least a half-inch of the deck is left, but always stop the reader before removing cards from the stacker.

The reader is operated by the buttons in front of the hopper. The two at the left turn on power and the reader motor. Pushing START places the reader on line so the program can read cards. Pushing STOP turns off the reader, taking it off line. An empty hopper, a full stacker, or any error condition indicated by the lights in front of the buttons also stops the reader, but it always finishes the current card before stopping.

The four error lights indicate a pick failure, a card motion error, and a photocell output that is too weak or that exists when there should be none (a photocell error may be caused by a hardware malfunction but can also be caused by an obstruction in the read station or a damaged card). The last three error conditions set Trouble. Do not attempt to reread a worn or damaged card that has caused an error — duplicate it first. After correcting the trouble, press START to allow the program to continue reading the deck.

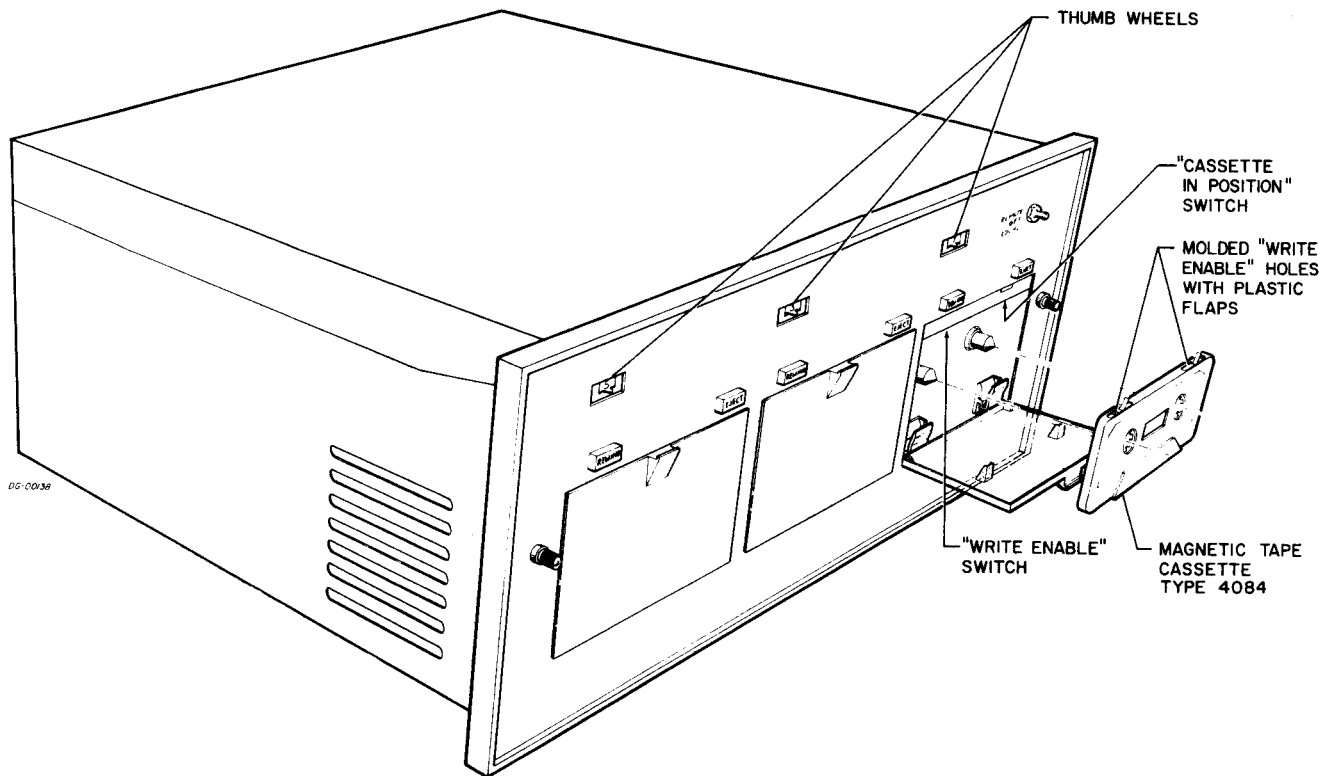
### 3.7 DGC CASSETTE

The DGC Cassette system is a computer peripheral which stores digital information on a single track of .15 inch magnetic tape. The DGC Cassette consists of a control and up to eight cassette transports. The transports, their drive electronics and power supply are housed in a 19 inch chassis in banks of up to three; the control is housed in the computer's chassis or its extension.

The computer governs the control and the control governs all transports. The transports are numbered from 0 to 7 and identified visually by the operator and logically by the control by thumb wheels mounted on the chassis PC board, and available through the front panel. Data is transferred at 800 words per second between the transports and the computer through the control via the computer's data channel; the data channel has 28 $\mu$ sec to respond to a data channel request. Each transport reads and writes at 800 sixteen bit words a second; spaces (counts records) forward or reverse at 30 inches per second; rewinds completely in 85 seconds, and writes an End of File (EOF) mark on command. Writing is done through a Write Head and reading through a Read Head which sees data on the tape shortly after it has been written (Read After Write). Only one transport can be reading or writing at any time, and a transport can only read or write data when the tape is moving forward (from supply reel to takeup reel). A cassette can be rewound manually by the operator with a switch on the transport, or automatically under computer control.

Each transport drives one magnetic tape cassette. The magnetic tape cassettes store an average of 50,000 sixteen bit words in blocks of 2 to 4096 on 200 feet of .15 inch magnetic tape. The magnetic tape in the cassette begins and ends with 22 inch reflective leader/trailers which stop the transport when they pass over a photodiode mounted near the heads. Two holes molded into the top of each cassette determine the state of the transport's Write Enable switch. The control cannot write on a cassette unless the left hand hole (facing the cassette when it is in the transport) is covered to depress the Write Enable switch. When the cassette is properly mounted in its transport, it depresses the transport's "Cassette-in-Position" switch. A control cannot use the transport unless this switch is depressed.

Each chassis has a three-position power-on switch: REMOTE puts the power supply under computer control so that the chassis turns on or off with the computer; OFF turns off the chassis's power supply and LOCAL turns on the chassis's power supply without regard to the computer.



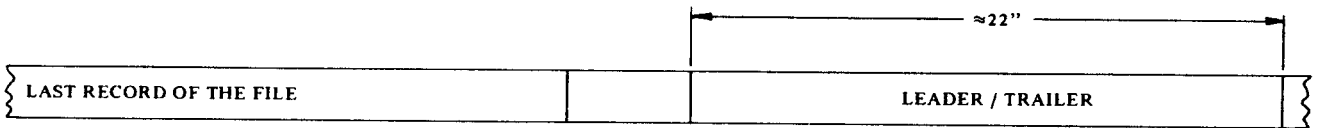
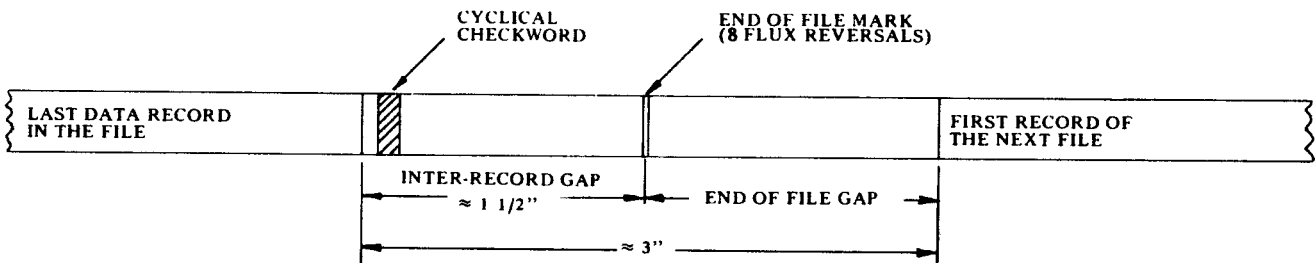
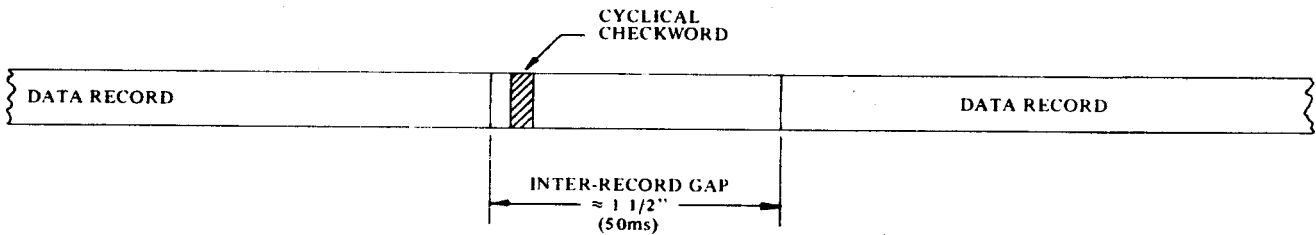
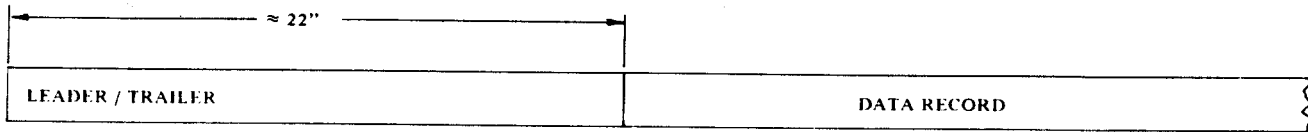
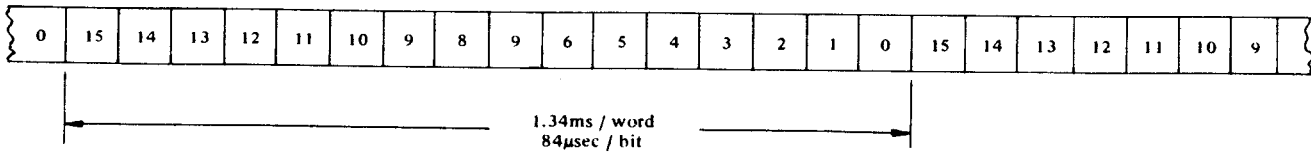
The DGC Cassette Chassis

### Tape Format

The control writes serially onto the tape in groups of bits called words, groups of words called records and groups of records called files. The words are always 16 bits long, the records are 2 to 4096 words long (determined by the program), and the files are 2 to 4096 blocks long (usually limited by the length of tape). Words need no terminators; records are terminated automatically by a 16 bit cyclical checkword and a 1½" gap; files are terminated by two 1½" gaps and an End of File (EOF) marker. The tape itself begins and terminates with 22" reflective leader/trailer. The control automatically detects and responds to these markers and the checkword.

The cyclical checkword is calculated and tested during both write and read operations. During a write operation, it is calculated from the data stream that is input from the control through the Write Head to the magnetic tape, and then re-calculated from the (same) data stream that is input (ms later) from the magnetic tape to the Read Head. If the checkword that is finally written at the end of the record is not the same as the checkword calculated through the Read Head, an error flag called **PARITY ERROR** is posted. During a read operation, the checkword is calculated from the data stream that is entering the Read Head and then compared to the version that was written at the end of the record. If the cyclical checkword written is not the same as that calculated, the **PARITY ERROR** flag is posted.





1. The leader/trailer is the End of Tape mark. If the computer writes into the leader then the record and possibly its file must be erased and started on another cassette.
2. A cassette will record approximately two-hundred, 256-word blocks or twelve 4096 word-blocks.

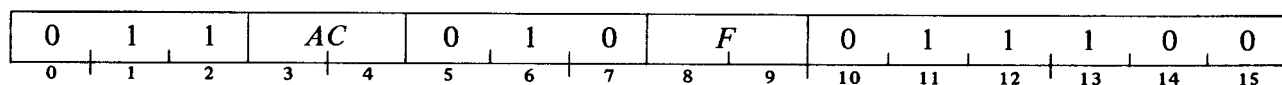
The DGC Cassette Magnetic Tape Format

## Instructions

To run the tape, the program must select a transport and a command. A command usually requires an initial address (to the 15-bit address counter) for data channel access, and the (two's complement) negative of a word count. Space commands use the 12-bit word counter for counting records.

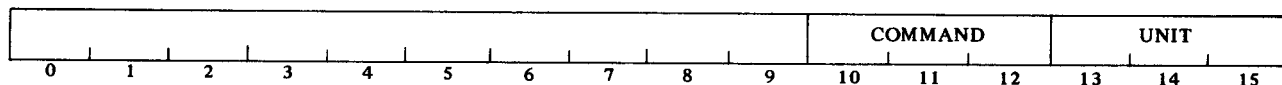
The tape system uses five of the IO transfer instructions. Busy and Done are controlled or sensed by bits 8 and 9 in all IO instructions, with device code 34, mnemonic CAS. Interrupt Disable is controlled by interrupt priority mask bit 10. A second tape system connected to the bus would have device code 74. The Clear function ( $F = 10$ ) clears Busy and Done and also clears the command register and the status flags in the control. Start ( $F = 01$ ) clears Done, sets Busy, and places the control and the selected transport in operation.

### DOA $-$ ,CAS      Data Out A, Cassette Tape



Load the contents of AC bits 10–15 into the command register as shown, and perform the function specified by  $F$ .

The contents of the AC remain unchanged and must be in the following format.

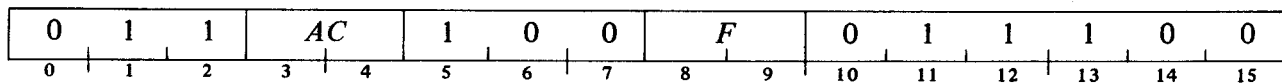


10–12 These bits select the command as follows

- 0 Read
- 1 Rewind
- 2 No Effect
- 3 Space Forward
- 4 Space Reverse
- 5 Write
- 6 Write End of File
- 7 Erase

13–15 Numbers 0–7 address transports 0–7.

### DOB $-$ ,CAS      Data Out B, Cassette Tape



Load the contents of AC bits 1–15 into the address counter (AC bit 0 should be 0), and perform the function specified by  $F$ .

**DOC -CAS      Data Out C, Cassette Tape**

0	1	1	AC		1	1	0	F	0	1	1	1	0	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Load the contents of AC bits 4-15 into the word counter, and perform the function specified by *F*. The maximum block size is 4096 words and the minimum block size is 2 words. The word counter becomes the block counter during spacing. Minimum count is 1.

**DIA -CAS      Data In A, Cassette Tape**

0	1	1	AC		0	0	1	F	0	1	1	1	0	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the status of the tape system into AC as shown, and perform the function specified by *F*.

Upon completion the contents of the AC are formatted as follows:

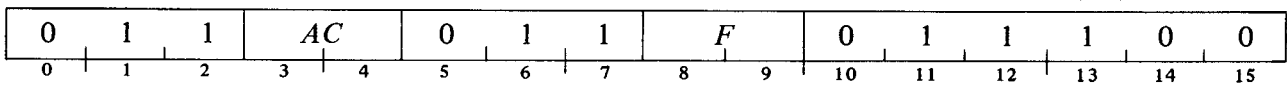
ERROR	DATA LATE	RE-WINDING	ILLEGAL	1	PARITY ERROR	END OF TAPE	END OF FILE	BEGIN OF TAPE	1	WRITE FAIL	0	0	WRITE LOCK	0	CASSETTE UNIT READY
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Start clears Error, Data Late, Parity Error, End of File, and Write Fail; Clear, clears these, plus Illegal; the remaining flags are supplied by the addressed transport (which is automatically unit 0 after Clear is given).

- 0      Bit 1, 3, 5, 6, 7, 8 or 10 is 1.
- 1      The data channel has failed to respond in time to a request for access (e.g., because of a long indirect addressing chain or pre-emption of the channel by faster devices).
- 2      The addressed transport is now rewinding.
- 3      This bit sets if the program gives Start when any of the following conditions holds:
  - the command is Write, Erase or Write End of File, and Write Lock (bit 13) is 1.
  - the command is Space Reverse and Begin of Tape (bit 8) is 1.
  - Busy is 0 but Unit Ready (bit 15) is also 0, which means that the cassette is not in place or is rewinding.
 The setting of Illegal prevents the tape control from going into operation and sets Done, requesting an interrupt if Interrupt Disable is clear. The program must give Clear before proceeding (Start does not clear Illegal).
- 4      Always in the 1 state.
- 5      During Write, this bit is set if the cyclical checkword written at the end of the record differs from the cyclical checkword calculated by the read logic, indicating that a Write or Read error has occurred. During Read this bit sets if the cyclical checkword calculated by the Read logic differed from the cyclical checkword read at the end of the record, indicating that a Write or Read error occurred. During Read or Write this bit sets if the correct number of flux reversals was not detected after the block has been completed.
- 6      The addressed tape has moved into the leader/trailer at the end of the tape. (Reverse motion clears this bit.)

- 7 The control has written a file mark or has encountered one in reading or spacing. If there is an error in a file mark it is not recognized as such, i.e., the control interprets it as very short data record.
- 8 The addressed tape has moved into the leader at the beginning of tape.
- 9 Always in the 1 state.
- 10 The write current has ended in the wrong state due to a catastrophic failure.
- 11, 12 Not Used.
- 13 The selected transport's Write Enable switch is disabled.
- 14 Not Used.
- 15 The addressed transport is ready for operation by the program.

**DIB -CAS      Data In B, Cassette Tape**



Read the present contents of the address counter into AC bits 1-15, and perform the function specified by *F*.

Clear AC bit 0.

During Spacing, the address counter increments at the end of each block. It therefore holds the file end block count when Spacing terminates.

**Automatic Loading**

Should the core image loader which programs the computer to accept data from the DGC Cassette be destroyed by program debugging, it can easily be restored from the cassette.

The core image loader is two blocks long, each block having 257 words. The first block starting at location zero, is used to read in the second block. Only the second block of 257 words, however, stays in core, at the highest memory location. The core image loader should be the first file on a tape cassette mounted on transport zero.

In a Supernova computer the core image loader is brought in from tape simply by pressing RESET, setting 000034 in the switches and then CHANNEL START at the computer console. In a Nova 1200, 1230, 800, 1210, 1220 or 820 computer with the PROGRAM LOAD option switch press RESET, setting 100034 in the switches, then press PROGRAM LOAD. To bring the core image loader into memory without automatic loading, the operator must use the following procedure:

1. Press RESET.
2. Set 376 into the data switches and press EXAMINE.
3. Set the instruction NIOS CAS (060134) into the data switches and press DEPOSIT.
4. Set 000377 into the data switches (JMP 377) and press DEPOSIT NEXT.
5. Set 376 into the data switches and press START.

## Tape commands

To perform any operation, the program must select the unit while giving a command, and all commands are initiated by giving Start. The Rewind command does not actually place the control in operation; but, for all other commands, Start clears Done and sets Busy. At the termination of the command, the control clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. Before any instruction is given read Status to see if the given command is valid. No forward command should be given after reaching the End Of Tape leader. Following this section are flow charts that show the actual procedures for programming the tape commands properly.

**Write.** The program must specify a (negative) word count, and an initial address. If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise, the control makes an immediate data request for the first word, and writes the words it receives via the data channel from the locations specified by the address counter. When the word counter overflows Data Late or EOT sets, the control terminates the record and sets Done.

**Write End of File.** If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise, the control writes a file mark and then sets Done.

**Erase.** If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise, the control erases 2½ inches of tape and then sets Done.

This command is used primarily to skip sections of tape on which the program has found it impossible to write data without parity errors.

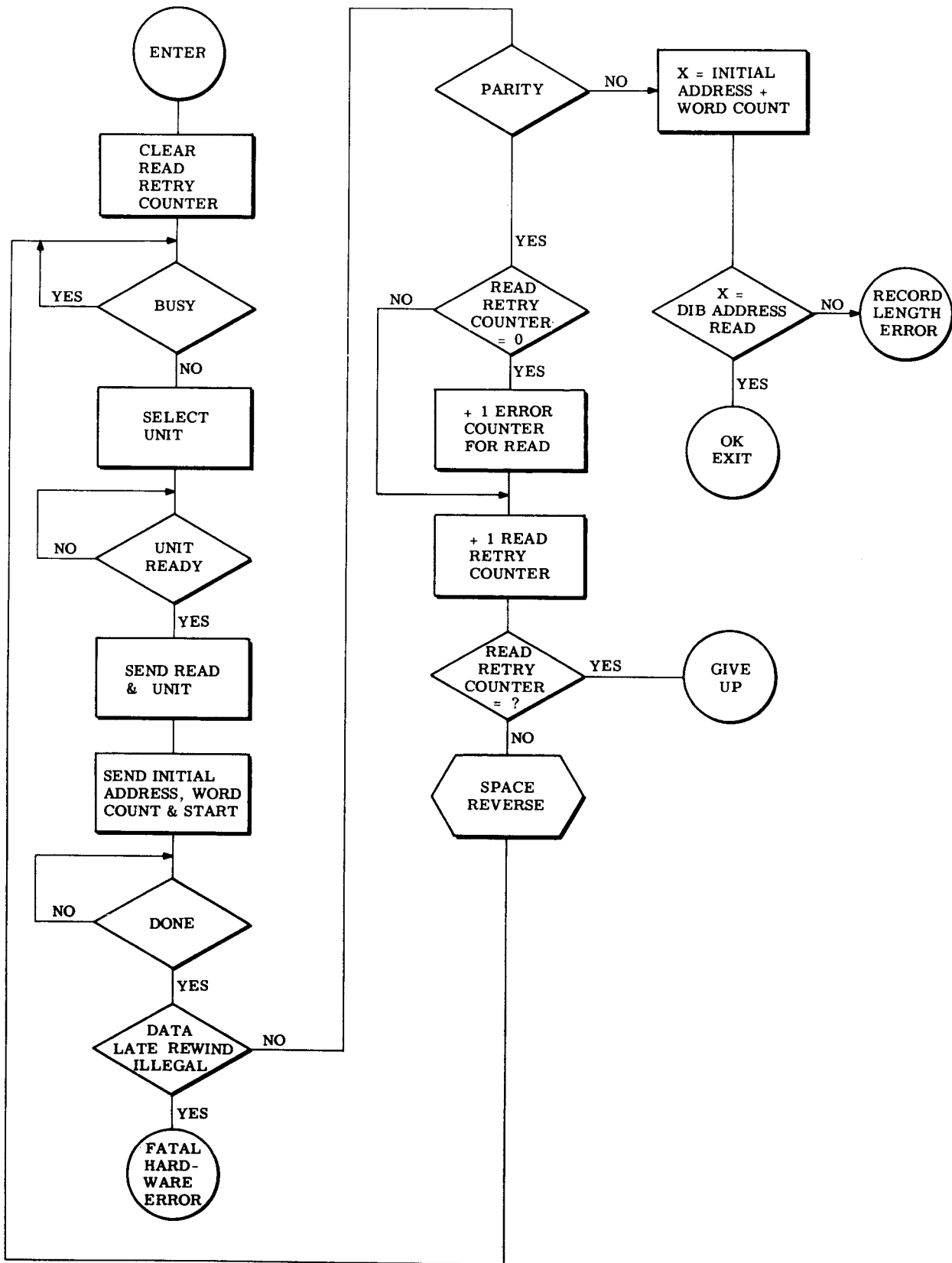
**Read.** The program must specify a (negative) word count, and an initial address. The control reads a single record from tape, and sends the data via the data channel to the locations specified by the address counter until it encounters the EOR gap or the word counter overflows, whichever occurs first. Giving a large word count (eg giving zero) ensures that the entire record will be read even if its length is unknown. The setting of Data Late during the record indicates that information has been lost, but data transfers continue until overflow or the record ends. After completing the record, the control sets Done.

The length of a record of unknown size can be determined after it is read by giving a DIB to check the contents of the address counter, which will be one greater than the address to which the last word in the record was sent (provided, of course, the word count was large enough).

**Space Forward.** The program should give a (negative) word count equal to the number of records to be spaced. The control spaces forward over the given number of records unless it encounters a file mark or the end of tape, in which case it stops at the mark or at the end of the record in which the EOT marker is encountered. To space a file of up to 4096 blocks the program can simply give a zero word count.

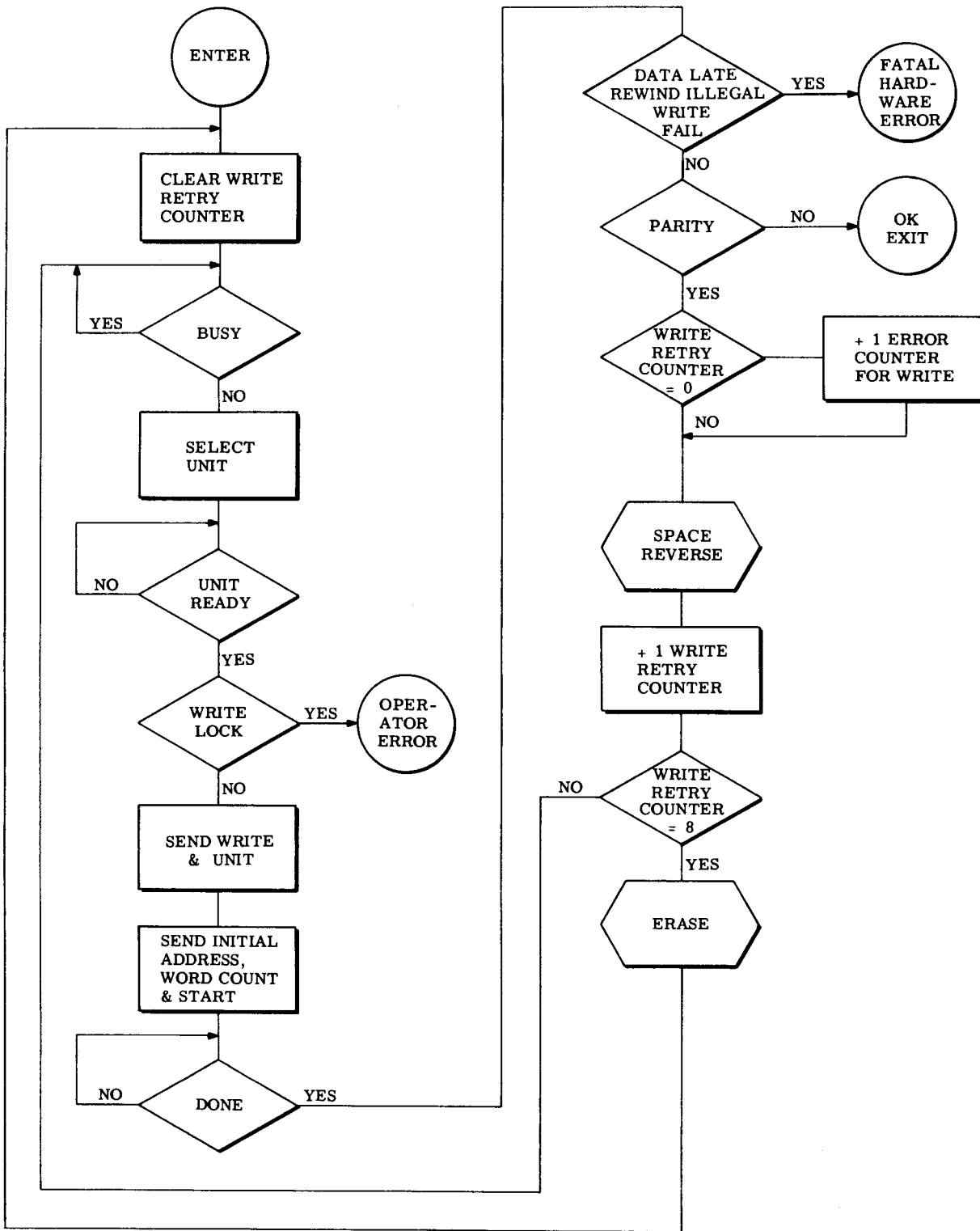
**Space Reverse.** The program should give a (negative) word count equal to the number of records to be spaced. If Begin of Tape is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise, the control spaces reverse over the given number of records, but stops the tape automatically upon encountering a file mark or the beginning of tape. To space a file, (up to 4096 blocks) the program can simply give a zero word count. It is not advisable to Space Reverse at the beginning of the tape leader.

**Rewind.** A Rewind command causes the transport to space forward the length of the leader/trailer (22 inches), space backward until it encounters the beginning of tape leader/trailer, and then forward again the length of the leader/trailer. During this last spacing from the beginning of tape, the transport erases the tape. Once a Rewind command is issued to a transport, it carries on automatically, so that any number of transports can be rewinding at once.



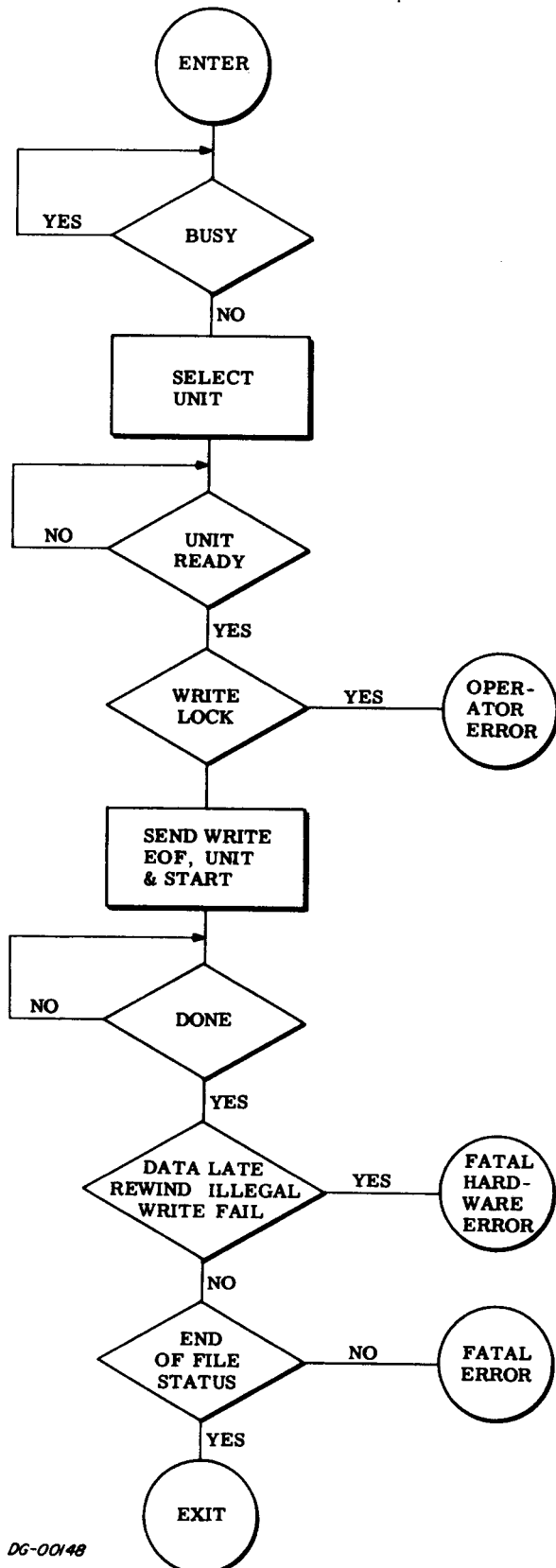
DG-00146

DGC Cassette READ Flowchart



DG-00147

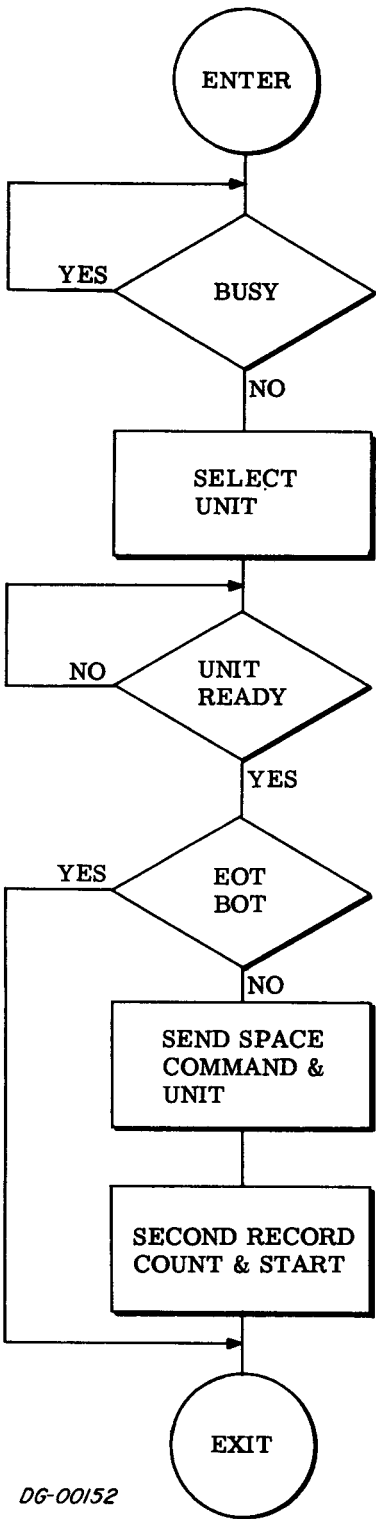
DGC Cassette WRITE Flowchart



DG-00148

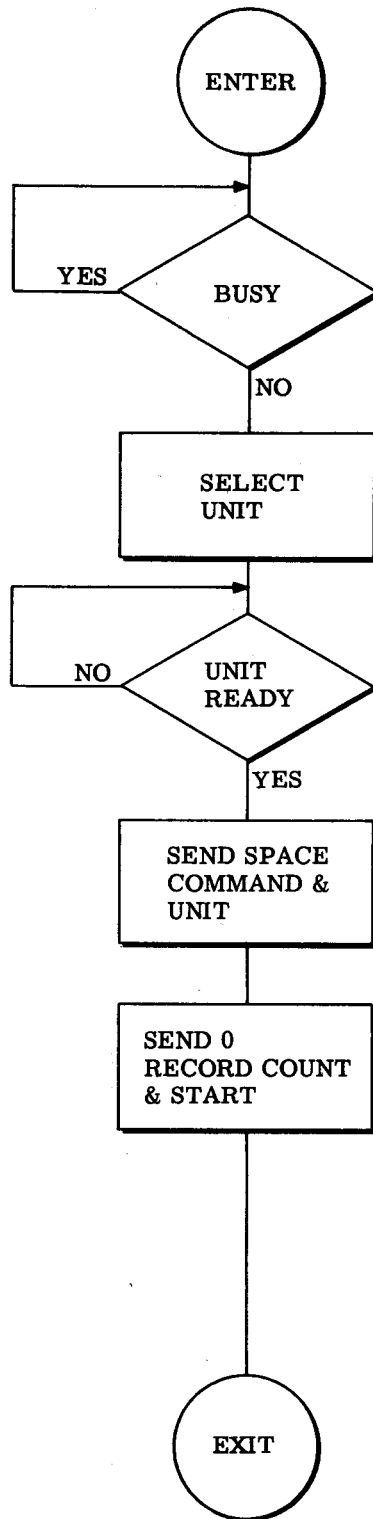
DGC Cassette WRITE END OF FILE Flowchart



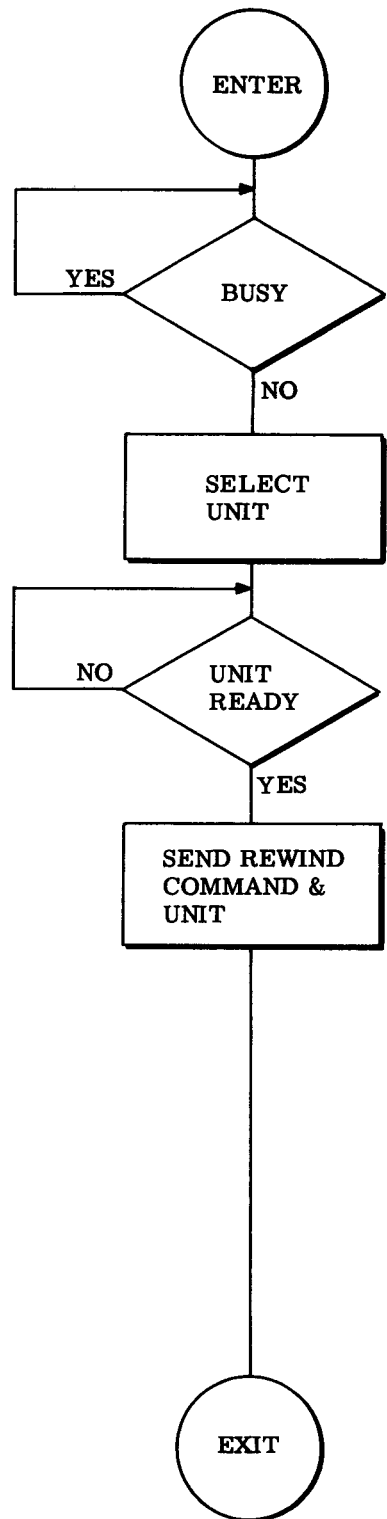


DG-00152

SPACE FORWARD/REVERSE



SPACE FILE FORWARD/REVERSE



REWIND

# Chapter IV

## Magnetic Tape

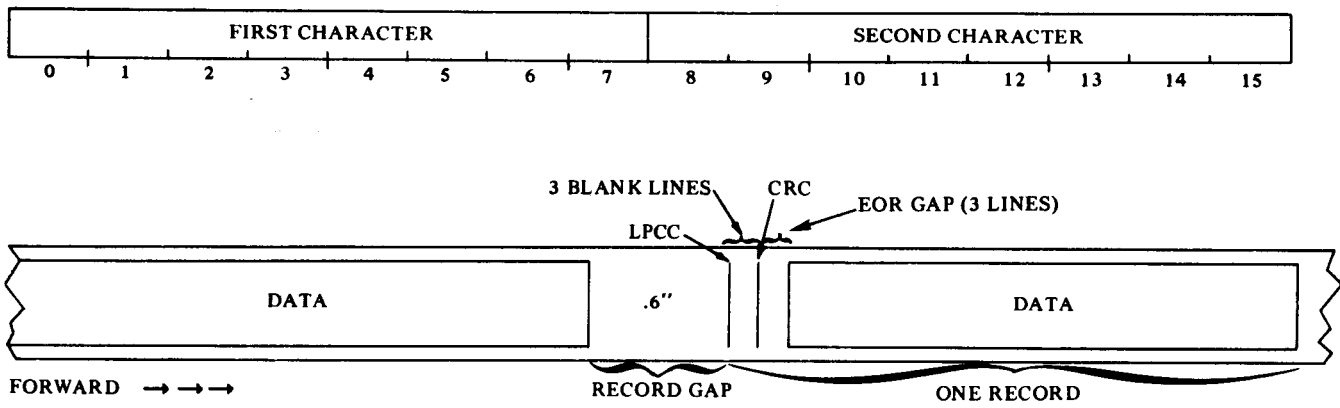
The magnetic tape equipment handles the large reels of half-inch tape that are standard throughout the industry. A tape system consists of a control and up to eight tape transports. The control is connected to the data channel, so the program need only set up the tape for a particular operation and all transfers to and from memory are then handled automatically. To operate with the data channel the control has an address counter and a word counter as well as data buffers. Data General supplies several types of transports that differ in tape speed and tape handling characteristics. Each type is available in two versions, for recording information in nine tracks and seven tracks. Thus data transfer rates and timing depend on the transport, but each transport supplies information to the control such that transports of different speeds and recording formats can be operated by a single control. Every transport accommodates two reels (one for supply, one for takeup) and can record information in both low and high densities, 556 and 800 bytes per inch. A full 10½-inch reel has 2400 feet of half-inch tape and at high density can store over 180 million bits of data in the 9-track format, over 135 million bits in the 7-track format.

The program communicates with the tape control, which in turn governs all tape transports but operates only one at a time. Reading and writing (recording) can occur only when tape is moving forward (from supply reel to takeup reel), but the control can space the tape (*ie* move it to a new position) in either direction. Although only one transport can be reading, writing or spacing at a time, rewinding the entire tape onto the supply reel at high speed requires only initiation by the control, and the transport then proceeds automatically while the control can operate another.

### 4.1 TAPE FORMAT

The control writes lateral characters, *ie* it writes transverse lines on tape with nine or seven bits of information per line, one bit in each track. The density of the information written is determined by a switch at the transport. Every character is in either a data record or a file mark. A data record contains both data characters and error-checking characters; every data character consists of a data byte and a parity bit, which the control generates so that the number of 1s in the line is odd or even as specified by the program. The data bytes in a record taken together correspond to a block of words sent from memory to the control. To separate adjacent records the control automatically erases a segment of tape, a record gap, between them. The control always stops tape in a gap.

Transfers between memory and control are of full words even though the tape characters may contain 8-bit or 6-bit data bytes. The minimum length of a record is two words (four data characters), the maximum length is 4096 words. To write, the control divides the words into data bytes, and when reading, it reassembles them. There are two ways in which this is done. For 9-track format the control writes each word as two characters, each containing an 8-bit data byte. After the control writes the last data line for a record, it writes three blank

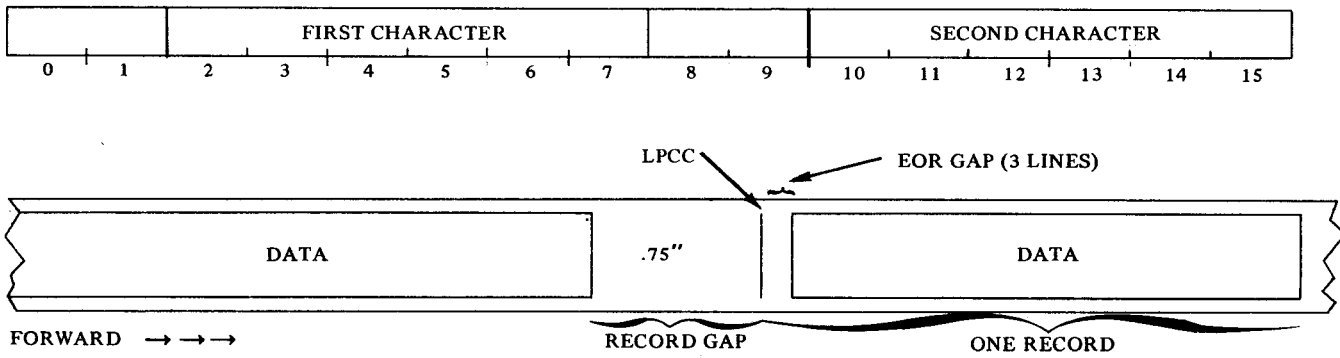


9-TRACK FORMAT

lines, a cyclic redundancy character (CRC), three more blank lines, and a longitudinal parity check character (LPCC). The first three zero characters constitute the end-of-record gap (EOR), so called because the control uses it to detect the end of record; this is so even in writing, as the tape encounters the write head first, and the control detects everything shortly after writing it. The control generates the CRC as described in §6 of USAS X3.22-1967, *USA Standard, Recorded Magnetic Tape for Information Interchange*. Taking the CRC bits as numbered in that document, CRC bit 1 corresponds to the parity track, bits 2-9 correspond to the tracks that receive the bits from left to right in each data byte. The LPCC (which may be zero) produces even longitudinal parity in each of the tracks along the length of the record. The minimum record gap is .6 inch. For compatibility with IBM format, a record must be written in high density and odd parity.

Whenever the control reads or writes a data record, it checks that the lateral parity of every data line agrees with the parity specified by the program and checks that every track has even longitudinal parity.

For 7-track format the control writes bits 2-7 and bits 10-15 of each word in two characters, ignoring bits 0, 1, 8 and 9 altogether. After writing the last data line, the control writes an EOR gap and an LPCC. The



7-TRACK FORMAT

minimum record gap is .75 inch. When reading 7-track tape, the control assembles pairs of bytes into words in the positions shown above, with 0s in the unused bits.

When writing in even parity, the program must take care not to supply a word containing a zero data byte in the recording format selected, as this would result in a missing character (a blank line), and no words beyond

that point would reassemble correctly. The control does not check for missing characters when reading, but two or more contiguous missing characters would be interpreted as an EOR gap, so the command would terminate with the Bad Tape flag set.

To facilitate tape processing the program can group sets of data records into files. The end of a file is indicated by a 3-inch gap followed by a file mark, which is a special record containing a single, special data character and its (equivalent) LPCC. A space command automatically terminates when a file mark is encountered.

Every tape has two physical markers to indicate its extremities. These markers are reflective strips that are sensed by photoelectric cells in the transport. At least ten feet in from the beginning of the reel is the loadpoint marker, which is the logical beginning of tape (BOT). Reverse commands stop automatically at this marker. A loadpoint gap of at least three inches precedes the first record on the tape. The end-of-tape marker (EOT) is at least fourteen feet from the physical end of the tape; the final ten feet of tape should be left for trailer, *ie* the program should not record more than a few feet beyond the EOT (this is more than enough for a record of maximum length at low density). A status bit indicates when the tape is beyond the EOT, but this condition stops the tape automatically only when it is spacing forward.

An annular groove is molded into the back of every reel. The control cannot write on the tape unless the supply reel has a plastic (write enable) ring in this groove. By leaving the ring out, the operator can protect the data on the tape from accidental destruction (overwriting or erasure).

While the control is actually processing the data part of a record, the data transfer rate is fixed. But in a lengthy tape run the effective (average) transfer rate depends on record length, which determines the percentage of tape taken up by gaps (at the higher density each record gap could hold an additional 240 words). The effective transfer rate is therefore a function of record length as well as tape speed and density.

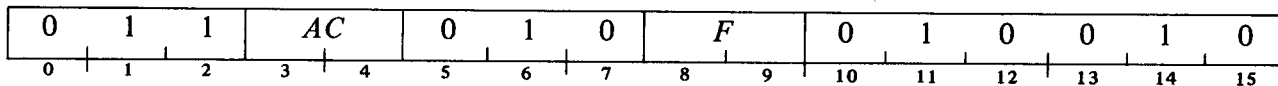
## 4.2 INSTRUCTIONS

The tape control has two 16-bit buffer registers to provide double buffering of data between tape and data channel; hence the channel has almost three character times in which to respond to requests by the tape control.

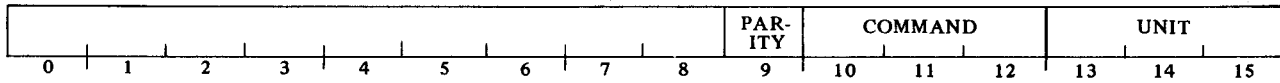
To run the tape, the program must select a transport and a command; most of the latter also require specification of parity, an initial address (to the 15-bit address counter) for data channel access, and the (twos complement) negative of a word count. Space commands use the 12-bit word counter for counting records.

The tape system has device code 22, mnemonic MTA, and uses five of the IO transfer instructions. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions. The Clear function ( $F = 10$ ) clears Busy and Done and also clears the command register and the status flags in the control. Start ( $F = 01$ ) clears Done, sets Busy, clears many of the flags, and places the control and the selected transport in operation. Interrupt Disable is controlled by interrupt priority mask bit 10. A second tape system connected to the bus would have device code 62.

### DOA $\rightarrow$ MTA      Data Out A, Magnetic Tape



Load the contents of AC bits 9–15 into the tape command register as shown, and perform the function specified by  $F$ .



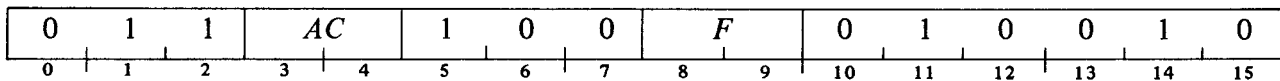
9        0 selects odd parity, 1 selects even.

10-12    These bits select the command as follows.

- 0    Read
- 1    Rewind
- 2    No Effect
- 3    Space Forward
- 4    Space Reverse
- 5    Write
- 6    Write End of File
- 7    Erase

13-15    Numbers 0-7 address transports 0-7.

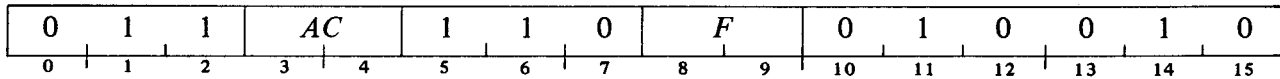
**DOB - ,MTA      Data Out B, Magnetic Tape**



Load the contents of AC bits 1-15 into the address counter (AC bit 0 should be 0), and perform the function specified by *F*.

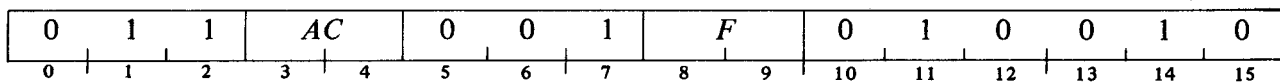
Note: If this instruction is given with a 1 in AC bit 0 and if the control then executes a Read command in which the word counter does not overflow, the control reads the CRC at the end of the record and sends it to the next memory location specified by the address counter. This is primarily for maintenance, for the program to check whether the CRC is being generated properly.

**DOC - ,MTA      Data Out C, Magnetic Tape**



Load the contents of AC bits 4-15 into the word counter, and perform the function specified by *F*.

**DIA - ,MTA      Data In A, Magnetic Tape**



Read the status of the tape system into AC as shown, and perform the function specified by *F*.

ERROR	DATA LATE	RE-WINDING	ILLEGAL	HIGH DENSITY	PARITY ERROR	END OF TAPE	END OF FILE	LOAD POINT	9 TRACK	BAD TAPE	SEND CLOCK	FIRST CHAR	WRITE LOCK	ODD CHAR	UNIT READY
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 11 and 12 are for maintenance only and are not discussed further here. Start clears Error, Data Late, Parity Error, End of File, and Bad Tape; Clear clears these plus Illegal; the remaining flags are supplied by the addressed transport (which is automatically unit 0 after Clear is given).

0 Bit 1, 3, 5, 6, 7, 8, 10 or 14 is 1.

1 The data channel has failed to respond in time to a request for access (*eg* because of a long indirect addressing chain or preemption of the channel by faster devices).

2 The addressed transport is now rewinding.

3 This bit sets if the program gives Start when any of the following conditions holds:

- The command is Write, Erase or Write End of File, and Write Lock (bit 13) is 1.
- The command is Space Reverse and Loadpoint (bit 8) is 1.
- Busy is 0 but Unit Ready (bit 15) is also 0.

The setting of Illegal prevents the tape control from going into operation and sets Done, requesting an interrupt if Interrupt Disable is clear. The program must give Clear before proceeding (Start does not clear Illegal).

4 The addressed transport is set to high density (0 indicates low density).

5 In Read or Write the control has encountered a data character whose lateral parity differs from that specified with the command or has discovered a track with odd parity the length of a record. Incorrect parity in a CRC or LPCC does not set this bit, but specifying the wrong parity when reading a file mark does.

6 The addressed tape is beyond the EOT marker. (Reverse motion clears this bit.)

7 The control has written a file mark or has encountered one in reading or spacing. If there is an error in a file mark it is not recognized as such, *ie* the control interprets it as a very short data record.

8 The addressed tape is at loadpoint.

9 The addressed transport handles 9-track tape (0 indicates 7-track).

10 The control has encountered either data in a record gap or a false end of record (two or more contiguous blank characters). Spacing reverse over an unrecognized file mark also sets Bad Tape.

13 The write enable ring is not in the supply reel on the addressed transport.

14 An odd number of characters were detected while reading or writing.

15 The addressed transport is ready for operation by the program.

#### DIB —,MTA Data In B, Magnetic Tape

0	1	1	AC		0	1	1	F	0	1	0	0	1	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the present contents of the address counter into AC bits 1–15, and perform the function specified by *F*. Clear AC bit 0.

### 4.3 TAPE COMMANDS

To perform any operation the program must select the unit while giving a command, and all commands are initiated by giving Start. The rewind commands do not actually place the control in operation, but for all other commands Start clears Done and sets Busy, and at the termination of the command the control clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. Following this section are flow charts that show the actual procedures for programming the tape commands properly. The timing in all cases is dependent upon the transport speed, tape handling characteristics and density, and is therefore treated in the discussion of each transport.

**Write.** The program must specify parity, a (negative) word count, and an initial address. If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise the control makes an immediate data request for the first word, and it writes the words it receives via the data channel from the locations specified by the address counter until either the word counter overflows or Data Late sets, at which time the control terminates the record and sets Done.

**Write End of File.** The program *must* specify even parity for a 7-track tape, odd parity for a 9-track tape, or the control will not write a file mark properly. If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise the control erases 2½ inches of tape (*ie* it extends the present record gap to three inches, writes a file mark and then sets Done.

**Erase.** If Write Lock is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise the control erases 2½ inches of tape and then sets Done.

This command is used primarily to skip sections of tape on which the program has found it impossible to write data correctly, *ie* without parity errors or a bad tape indication.

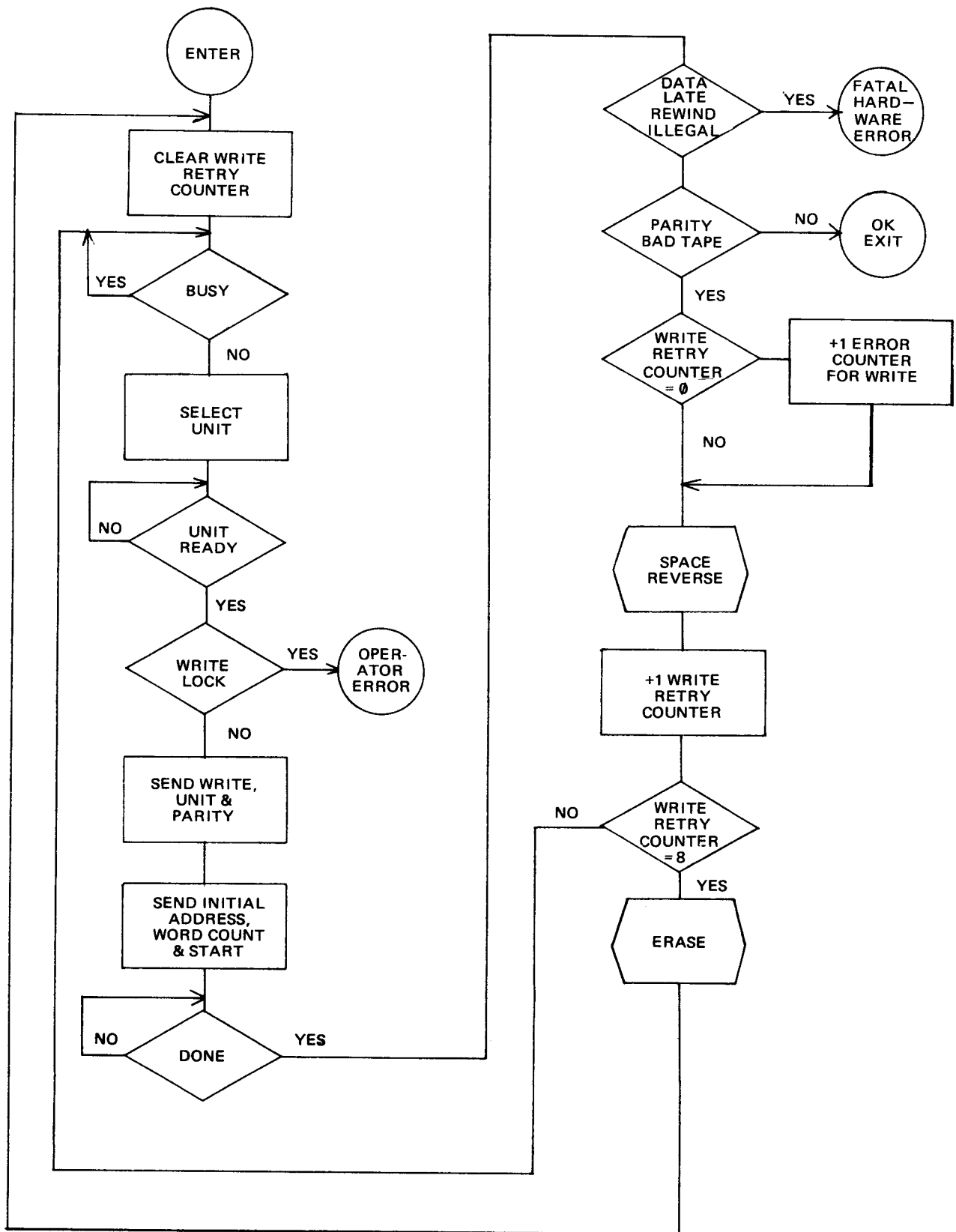
**Read.** The program must specify parity, a (negative) word count, and an initial address. The control reads a single record from tape, and sends the data via the data channel to the locations specified by the address counter until it encounters the EOR gap or the word counter overflows, whichever occurs first. Giving a large word count (*eg* giving zero) ensures that the entire record will be read even if its length is unknown. If the record contains an odd number of data characters, the final one is sent to memory in the left half of a separate word. The setting of Data Late during the record indicates that information has been lost, but data transfers continue until overflow or the record ends. After completing the record, the control sets Done.

If the record read is a file mark, its single "data" character is sent to memory via the data channel. The length of a record of unknown size can be determined after it is read by giving a DIB to check the contents of the address counter, which will be one greater than the address to which the last word in the record was sent (provided of course the word count was large enough).

**Space Forward.** The program should give a (negative) word count equal to the number of records to be spaced. The control spaces forward over the given number of records unless it encounters a file mark or the end of tape, in which case it stops at the mark or at the end of the record in which the EOT marker is encountered. To space a file of up to 4096 records, the program can simply give a zero word count.

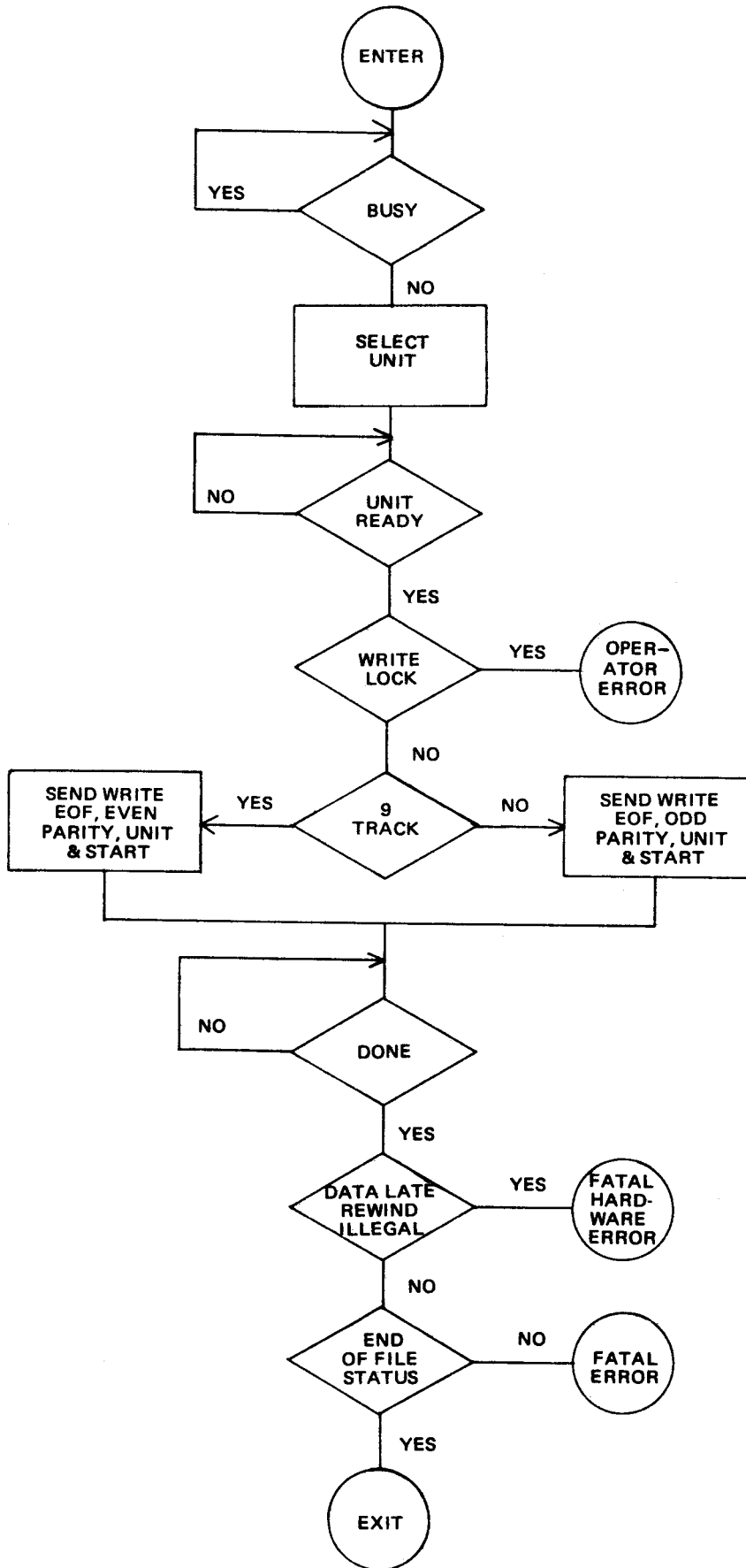
**Space Reverse.** The program should give a (negative) word count equal to the number of records to be spaced. If Loadpoint is 1, Start sets Illegal and Done, and the control does not go into operation. Otherwise the control spaces reverse over the given number of records, but it stops the tape automatically upon encountering a file mark or the loadpoint. To space a file of up to 4096 records, the program can simply give a zero word count.

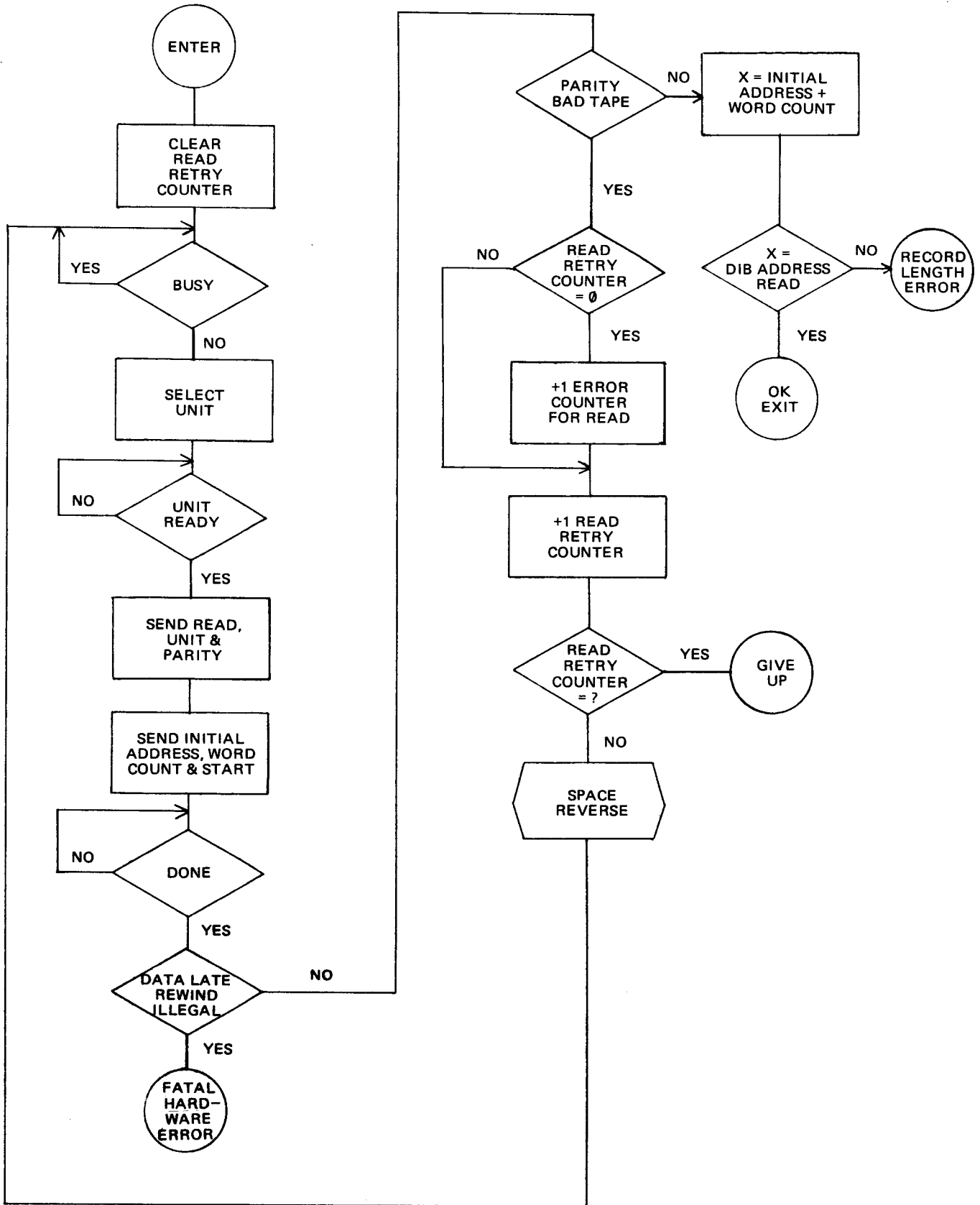
**Rewind.** Start does not affect the control but simply initiates the rewind in the addressed transport and the control is free for further use by the program. The addressed transport rewinds the tape at high speed onto the supply reel and stops at loadpoint.



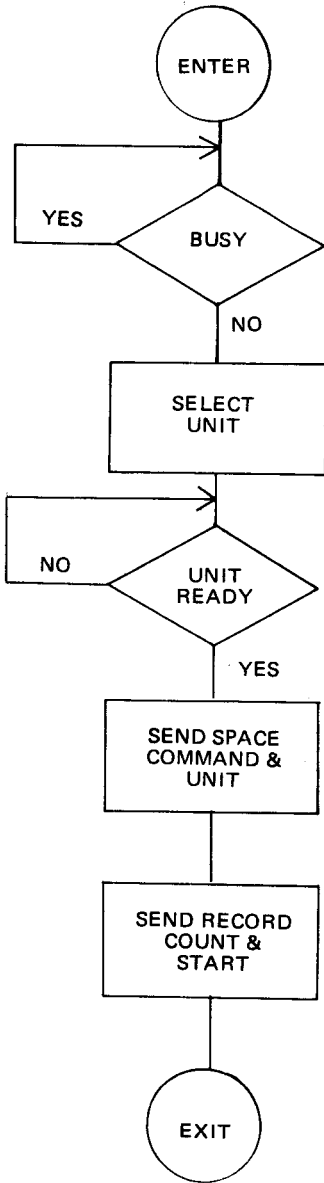
Magnetic Tape WRITE Flowchart



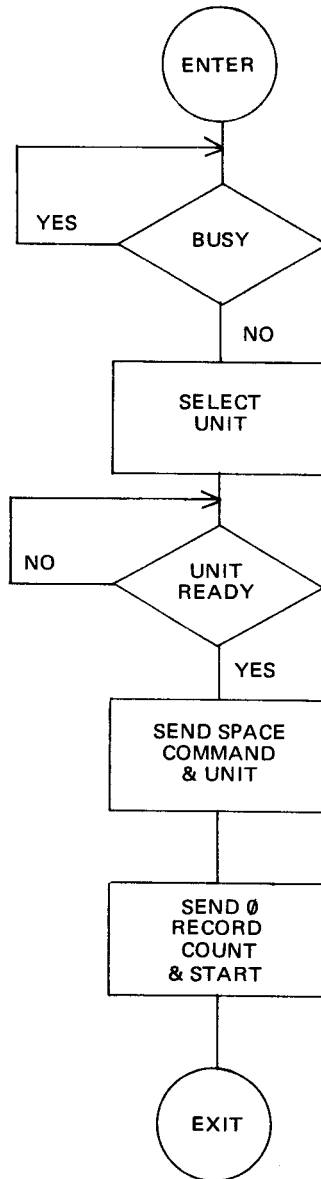




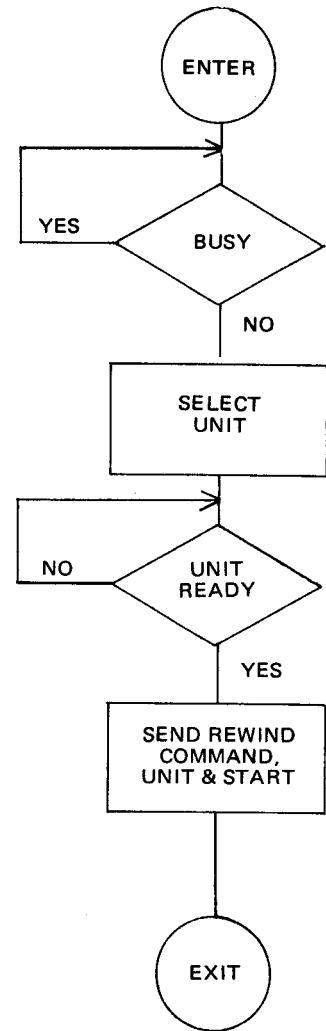
Magnetic Tape READ Flowchart



SPACE FORWARD/REVERSE



SPACE FILE FORWARD/REVERSE



REWIND

Magnetic Tape Flowcharts

### Automatic Loading

Should the binary loader in core be destroyed by program debugging it can easily be restored from tape. The loader should be in 9-track format, odd parity, in the first record on a reel mounted on transport 0 (the record must be at least 256 words long). To bring the loader into memory automatically, set device code 22 into data switches 10-15 at the computer console. Then in a Nova 1200 series or 800 series computer with the program load option, press RESET, turn on data switch 0, then press PROGRAM LOAD; in a Supernova computer press RESET and then CHANNEL START. To bring the loader into memory without automatic loading, the operator must use the following procedure:

1. Press RESET.
2. Set 376 into the data switches and press EXAMINE.
3. Set the instruction NIOS MTA (060122) into the data switches and press DEPOSIT.
4. Set 000377 into the data switches (JMP 377) and press DEPOSIT NEXT.
5. Set 376 into the data switches and press START.

#### 4.4 TAPE TRANSPORTS—Models 4030A through 4030H

Several types of transports are available for use with the Nova line computers. Each discussion below gives the speed and word processing time, but because of double buffering in the control, the data channel has almost 50 percent more than the word time to respond to a request (ie three character times). Since all transfers are made through the channel, transfer timing is not usually critical to the program; however, in order to determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing (in the latter, Start triggers an immediate request to load the buffers). In each case all relevant times are given.

Operating information for each transport is given in Section III of the appropriate manual; said section contains illustrations of the panels and controls, a photograph of a tape reel showing the write enable ring, and a drawing showing the location of the tape markers. Every transport requires a Data General adapter, which is mounted below the transport in all models except models 4030E and 4030F, which require mounting in a separate rack. On the adapter are a power button and a thumbwheel switch for selecting the unit address.

The most important consideration in tape operations is cleanliness. Nothing can ruin a tape run more easily than ash, dust or a piece of dirt. The tape path should be cleaned at least once every eight hours. Cleaning instructions are given in the manual.

#### Models 4030A, 4030B, 4030C, 4030D

This transport accommodates 10½-inch reels and may have a tape processing speed of either 37.5 or 24 inches per second. At the faster speed, the time required to process each word at high density is 67  $\mu$ s, at low density 96  $\mu$ s; equivalent times at the slower speed are 104 and 149  $\mu$ s respectively. Interrecord times for 9-track tape are as follows.

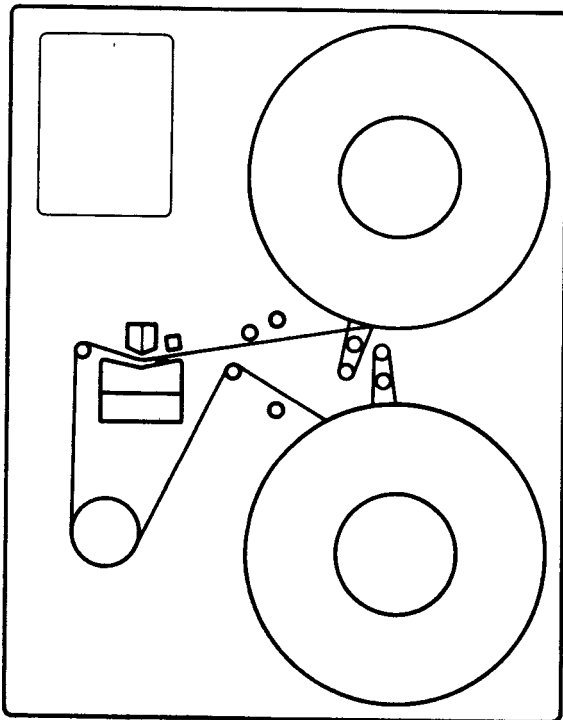
	<i>Write interrecord times in ms</i>			<i>Read interrecord times in ms</i>	
	<i>24 ips</i>	<i>37.5 ips</i>		<i>24 ips</i>	<i>37.5 ips</i>
Start	19	12	Start	29	18
Last character to stop	6.2	4.2	Stop	3.2	2

Stop	6.4	4	Settle down	<u>20</u>	<u>10</u>
Settle down	<u>20</u>	<u>10</u>	Total	52.2	30
Total	51.6	30.2	For 7-track add	10	6
For 7-track add	6.3	4.2			

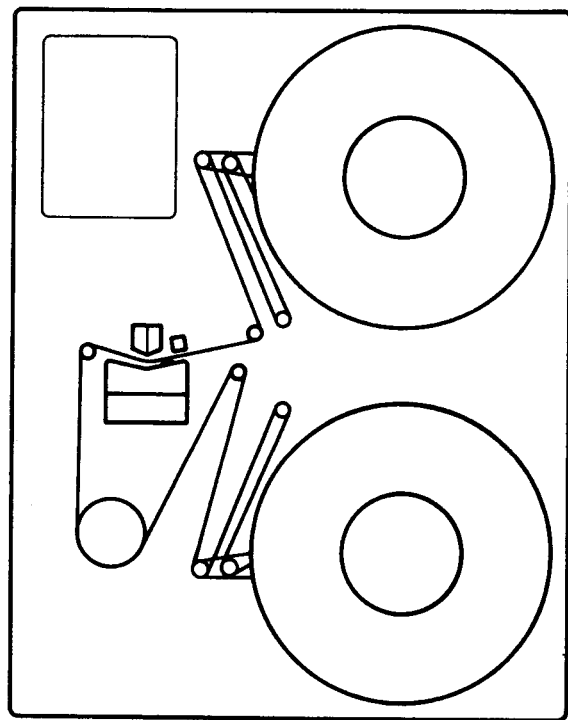
The rewind and fast forward speed is 150 inches per second; rewinding an entire reel takes about three minutes.

Controls for the transport are located on the adapter and at the upper left on the transport. The file-protect light at the top indicates when the data on the supply reel is protected from action by the program (the write enable ring is not in place). The remaining three controls at the top and bottom of the panel are alternate-action buttons which illuminate when on: POWER allows the operator to control transport power independently of the adapter; pressing REMOTE places the unit on line if the door is closed and tape is properly loaded; pressing STOP-RESET stops the tape and takes the unit off line. With the transport off line, holding down one of the four buttons in the center moves the tape at the speed and in the direction indicated; forward motion automatically terminates at the EOT marker, reverse motion at loadpoint. At the lower left corner inside the door are the density switch, an interlock, and a LOAD/UNLOAD button. When loading a tape, the operator must set the first switch to the density at which the tape will be processed; the program has no control over the density. Should the door be opened while the unit is running, the interlock stops the tape and takes it off line. Pulling the switch out overrides the interlock, allowing operation with the door open. The LOAD/UNLOAD button is alternate-action and moves the tension arms to the loading or operating position. Cleaning instructions are given in §3.5.1 of the manual.

The illustrations below show the loading and operating tape configurations (supply reel at the top). Before loading a reel make sure it has no write enable ring if the data on the tape is not to be changed by the program;



LOADING CONFIGURATION



OPERATING CONFIGURATION

otherwise place a ring in the reel so the transport can respond to write commands. To load a reel, turn the retainer knob on the reel hub to its counterclockwise limit, slip a reel onto the hub with the groove toward the tape deck, and holding the reel firmly, turn the retainer knob to its clockwise limit. Unwind about a foot of tape from the supply reel, thread it (as shown in the illustration) under the first edge guide, the tape cleaner head and the read-write head, and over the second edge guide. Bring the tape down and around the capstan and over the third edge guide. Pull the tape to unwind another foot, and wind about three turns around the takeup reel. Press LOAD/UNLOAD to generate tape tension, shut the door, and press FORWARD to locate the loadpoint; press REMOTE to put the unit on line. To unload the tape press STOP-RESET, rewind the tape to loadpoint, open the door, press LOAD/UNLOAD to release the tension arms, and turn the supply reel by hand counterclockwise to unwind the rest of the tape. Turn the supply reel retainer knob counterclockwise and remove the reel from the hub.

### Models 4030G and 4030H

This transport accommodates 8½-inch reels containing 1600 feet of tape (7-inch reels can be used, but they have only half the capacity). The tape processing speed is 12.5 inches per second; the time required to process each word at high density is 200  $\mu$ s, at low density 288  $\mu$ s. Interrecord times for 9-track tape are as follows.

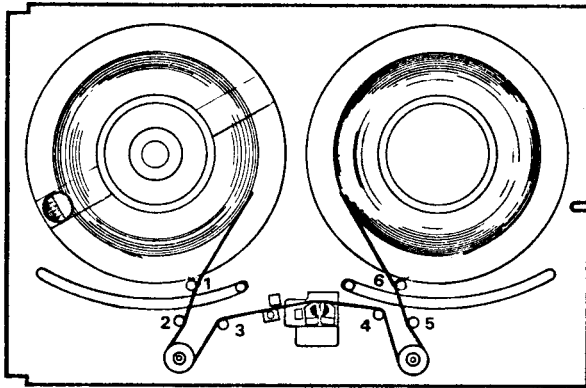
<i>Write interrecord times in ms</i>		<i>Read interrecord times in ms</i>	
Start	37	Start	60
Last character to stop	12.5	Stop	5.7
Stop	12	Settle down	30
Settle down	<u>30</u>	Total	<u>95.7</u>
Total	91.5	For 7-track add	19
For 7-track add	12.5		

The rewind speed is 75 inches per second; rewinding an entire reel takes about four minutes.

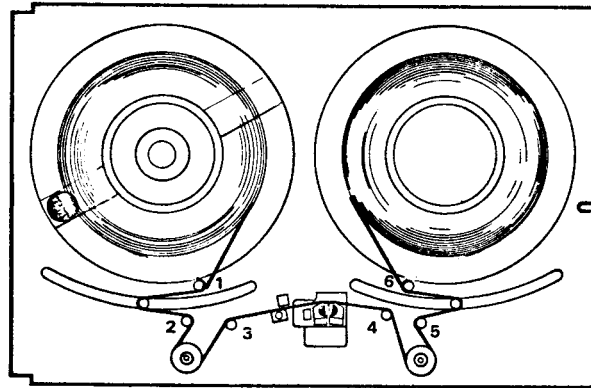
Controls for the transport are located on the adapter and on the left and right at the bottom of the transport front panel. When loading a tape, the operator must set the left toggle switch to the density at which the tape will be processed; the program has no control over the density. The tape can be threaded when the three-position toggle on the right is latched into the DISABLE position; pressing the toggle to LOAD generates tape tension; it must be latched into RUN for normal transport operation. On the left panel are three lights that indicate when power is on, when the transport is ready for operation, and when the data on the supply reel is protected from action by the program (the write enable ring is not in place). The remaining controls are momentary-contact buttons. Pressing REMOTE places the unit on line if READY is lit; pressing STOP stops the tape and takes the unit off line (in either case the appropriate button is illuminated to indicate the transport condition). With the transport off line, holding down one of the remaining three buttons (on the right) moves the tape at the speed and in the direction indicated; forward motion automatically terminates at the EOT marker, reverse motion at loadpoint. Cleaning instructions are given in paragraph 3.5.1 of the manual.

The illustrations below show the loading and operating tape configurations (supply reel at the left). Before loading tape make sure there is no write enable ring in the reel if the data on it is not to be changed by the program; otherwise install a write enable ring so the transport can respond to write commands. To load a reel, set the right toggle switch to DISABLE, turn the retainer knob on the reel hub counterclockwise several turns, slip a reel onto the hub with the groove toward the tape deck, and holding the reel firmly, turn the retainer knob to its clockwise limit. Push the tape tension arms as far as they will go toward the center of the tape deck. Unwind about a foot of tape from the supply reel, thread it (as shown in the illustration) by the

first two tape guides (between them and the supply reel tension arm guide), around the left capstan, over the third guide, between the tape cleaner and photosense heads, over the read-write head and the fourth guide, around the right capstan and by guides 5 and 6 (between them and the takeup reel tension arm guide). Pull the tape to unwind about another foot, and wind about three turns around the takeup reel, making sure the tape is taut against the guides. Hold the right toggle switch to LOAD until tension arm motion ceases, and then set the switch to RUN. Shut the door and press FORWARD to locate the loadpoint; press REMOTE to put the unit on line. To unload the tape press STOP, rewind the tape to loadpoint, and press REVERSE to wind all the tape onto the supply reel. Open the door, turn the supply reel retainer knob counterclockwise several turns, and remove the reel from the hub.



LOADING CONFIGURATION



OPERATING CONFIGURATION

#### Models 4030E and 4030F

This transport uses 10½-inch reels and has a tape processing speed of 120 inches per second. The time required to process each word at high density is 21  $\mu$ s, at low density 30  $\mu$ s. Interrecord times for 9-track tape are as follows.

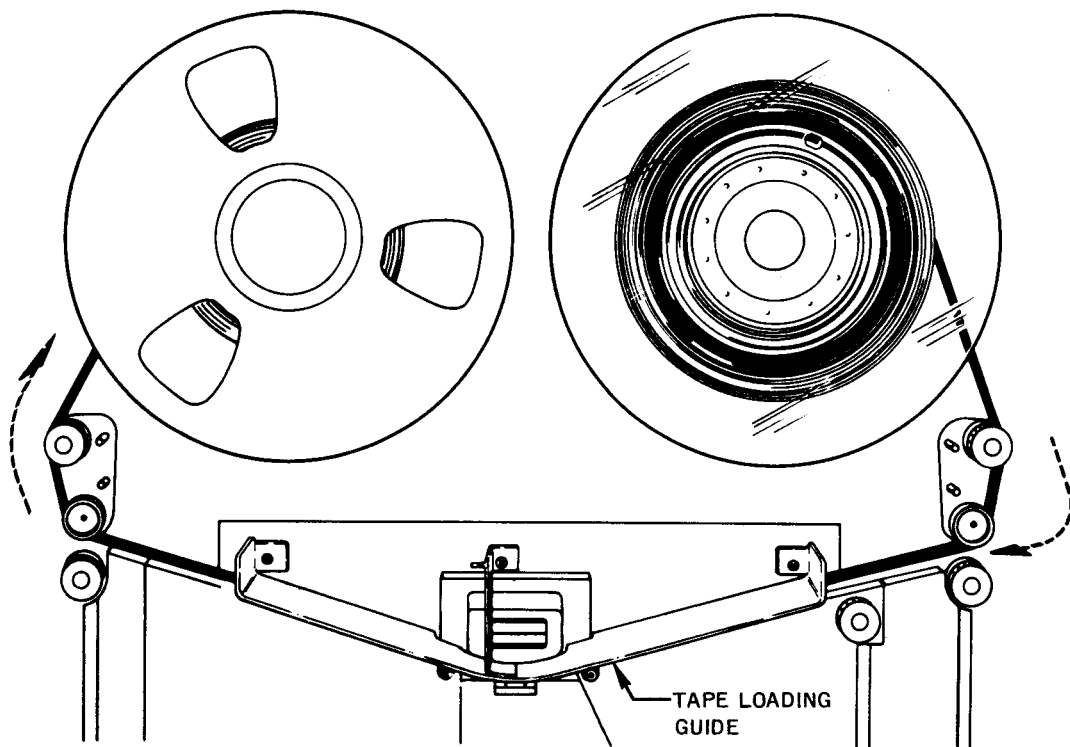
<i>Write interrecord times in ms</i>		<i>Read interrecord times in ms</i>	
Start	4	Start	6.5
Last character to stop	1.2	Stop	.6
Stop	1.2	Settle down	<u>5</u>
Settle down	<u>5</u>	Total	12.1
Total	11.4	For 7-track add	2
For 7-track add	1.2		

Rewinding an entire reel takes about 90 seconds.

The entire front of the cabinet is a door that covers the tape deck and vacuum columns, but the operator can gain access to the deck by lowering the window in the door. Controls for the transport are located on the adapter and on a panel at the top of the transport. The upper row on the panel contains three illuminated buttons and the file-protect light, which indicates when the data on the supply reel is protected from action by the program (the write enable ring is not in place). DENSITY is an alternate-action button containing two lights that indicate the density selected by the operator; when loading a tape, the operator must specify the density at which the tape will be processed, as the program has no control over it. The alternate-action POWER button allows the operator to control transport power independently on the adapter. The remaining buttons are all momentary-contact. Pressing REMOTE places the unit on line (lighting the button) if the window is closed

and tape is properly loaded. Pressing RESET stops the tape and takes the unit off line, enabling the remaining buttons in the bottom row and allowing RESET to be used to raise (close) the window. After a tape has been threaded and attached to the takeup reel, pressing LOAD/REWIND loads it into the vacuum columns and moves it forward to loadpoint; if the tape is already loaded, this button rewinds it to loadpoint. Pressing UNLOAD rewinds the tape, pulls it out of the vacuum columns, winds it entirely on the supply reel, and lowers the window. Holding down either of the remaining buttons moves the tape at normal processing speed in the direction indicated; forward motion automatically terminates at the EOT marker, reverse motion at loadpoint. Cleaning instructions are given in §2.5.1 of the manual.

The illustration below shows the loading tape configuration (supply reel at the right). Before loading a reel make sure it has no write enable ring if the data on the tape is not to be changed by the program; otherwise place a ring in the reel so the transport can respond to write commands. To load a reel, press UNLOAD to lower the window, press the narrow part of the hub operating lever to release the hub lock, press a reel onto the hub with the groove toward the tape deck, and holding the reel firmly, press the wide part of the lever to lock the hub. Unwind several feet of tape from the supply reel, thread it outside the tape guides as shown in the illustration, and wind about three turns around the takeup reel. Press LOAD/REWIND to load the tape and press RESET to raise the window. Once the tape is in the vacuum columns and positioned properly, press REMOTE to put the unit on line. To unload a tape press RESET, press UNLOAD to lower the window and rewind the tape entirely on the supply reel, press the narrow part of the hub operating lever to release the lock, and remove the reel from the hub.



LOADING CONFIGURATION



## 4.5 TAPE TRANSPORTS—Models 4030I and 4030J

This transport accommodates 10½-inch reels and has a tape processing speed of 45 inches per second. The time required to process each word at high density is 56  $\mu$ s; at low density, which is available only on a 7-track transport, the time is 80  $\mu$ s. Interrecord times for 9-track tape are as follows.

<i>Write interrecord times in ms</i>		<i>Read interrecord times in ms</i>	
Start	10	Start	15
Last character to stop	3.5	Stop	1.7
Stop	3.3	Settle down	<u>8.3</u>
Settle down	<u>8.3</u>		
		Total	25.0
Total	25.1		
		For 7-track add	5
For 7-track add	3.5		

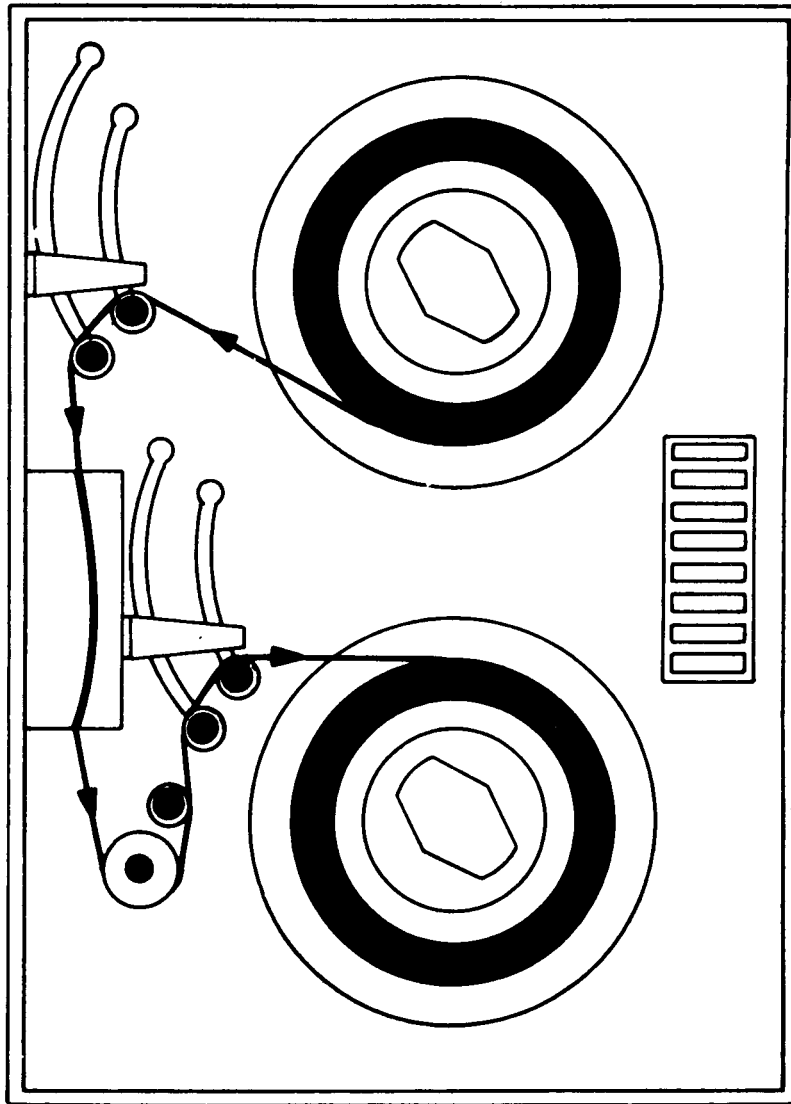
The rewind speed is 150 inches per second; rewinding an entire reel takes about three minutes.

Because of double buffering in the control, the data channel has almost 50 percent more than the word time to respond to a request (*ie* three character times). Since all transfers are made through the channel, transfer timing is not usually critical to the program; however, in order to determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing (in the latter, Start triggers an immediate request to load the buffers).

Operating information for the transport, including illustrations of the transport and its controls, is given in Sections I and II of the manual. The required Data General adapter is mounted below the transport; on the adapter are a power button and a thumbwheel switch for selecting the unit address. The remaining controls are located at the right on the transport front panel. The file-protect light indicates when the data on the supply reel is protected from action by the program (the write enable ring is not in place). The remaining controls are alternate-action buttons, all of which, except for REWIND and RESET, illuminate when on: POWER allows the operator to control transport power independently of the adapter; pressing ON LINE places the unit on line if the door is closed and tape is properly loaded; pressing RESET stops the tape and takes the unit off line. The remaining buttons are enabled only when the transport is off line. With the tape threaded pressing LOAD advances the tape to loadpoint and places the transport on line. At the completion of this operation LOAD and ON LINE will both be lit; moving the tape turns off the LOAD light, but even though the switch is disabled the light goes on again whenever the tape is positioned at loadpoint. Pressing REWIND moves the tape at high speed to loadpoint; pressing REWIND again unthreads the tape and winds it entirely onto the supply reel. Pressing FORWARD or REVERSE moves the tape in the indicated direction at normal speed; forward motion automatically terminates at the EOT marker, reverse motion at loadpoint. On a 7-track transport there is no reverse button: it is replaced by a DENSITY button, which selects the high density when on.

The most important consideration in tape operations is cleanliness. Nothing can ruin a tape run more easily than ash, dust or a piece of dirt. The tape path should be cleaned at least once every eight hours. Cleaning instructions are given in Section III of the manual.

The illustration below shows the loading tape configuration (supply reel at the top). Before loading a reel make sure it has no write enable ring if the data on the tape is not to be changed by the program; otherwise place a ring in the reel so that the transport can respond to write commands. In the reel hub is a tab, one end of which is marked PRESS. To load a reel, press in the marked end of the tab, slip a reel onto the hub with the groove toward the tape deck, press the reel firmly against the hub (touch only the center part of the reel — never press the reel flanges against the tape pack), and press the protruding end of the tab back down so it is flush with the face of the hub. Unwind about a foot of tape from the supply reel, thread it (as shown in the illustration)



LOADING CONFIGURATION

over the top buffer arm guides, down through the slot guide in the head assembly cover, under and around the capstan, and over the lower buffer arm guides. Pull the tape to unwind another foot, and wind about three turns around the takeup reel. Close the door and press **LOAD**; the transport will generate the proper tape tension, move the tape to loadpoint, and go on line. For a 7-track transport select the density at which the tape will be processed as the program has no control over this. To unload tape, press **RESET**, rewind the tape to loadpoint, press **REWIND** again to unload the tape, and open the door. Press in the marked end of the tab on the reel hub and remove the reel.

# Chapter V

## Discs

A disc is generally the largest random-access storage device in a computer system (a single disc usually holds more bits than all of core), and it also provides the fastest storage outside of core. This makes the disc exceptionally desirable for backup storage for memory generally, and in particular, for swapping in time-sharing systems: while the currently active user programs are in core, inactive programs are stored on the disc. Unlike magnetic tape, a disc is constantly in motion and has a predetermined format with data blocks of fixed length (256 16-bit words). Hence individual data blocks are addressable, and at the simplest level reading and writing may be the only functions the system need perform.

As a storage medium a disc is similar to a phonograph record. Data is stored in tracks that are concentric circles on the disc surface (there may be tracks on both surfaces or on only one). The disc is further divided into sectors, *ie* pie-shaped sections. Hence the tracks are divided into arcs, each arc being the intersection of a track with a sector (referred to as a "track-sector"). Besides data, a track-sector may contain information for synchronization, error checking, and addressing. The disc also may have extra tracks for timing and addressing information.

Disc systems are of two basic types: those with fixed heads and those with movable heads. In a fixed-head disc system there is a separate read/write head for each track. Since the storage medium is continuous, has a fixed format, and is in constant motion (both in speed and direction), functions are limited to read and write, with an automatic search for an initially specified track-sector; the average search time for a random sector (the "latency" time) is slightly over half a revolution. If the heads are movable, then the drive is usually constructed so that the discs themselves are removable. Since additional time is required for head positioning, a drive usually has a number of surfaces with one head per surface, where all heads are positioned simultaneously. In a disc pack drive the storage medium is a removable stack of discs; this allows greater storage capacity and the data can literally be stored on the shelf like magnetic tape while the drive is being used with another pack. Each disc surface has tracks and sectors. However there are many surfaces and the set of identically numbered tracks on the various surfaces constitutes a cylinder. The basic data block is a track-sector, and the track is addressed as the intersection of a cylinder and a surface. In terms of the addressing scheme used in continuous data processing, the disc pack is treated as though it were a pack of cylinders; the hardware counts through the tracks (surfaces) in one cylinder at a time. (A cylinder consists of all the tracks that can be processed without repositioning the heads.)

A single removable disc is a disc cartridge; it has movable heads and is treated like a disc pack, *ie* the pairs of tracks on opposite surfaces are treated as cylinders. A fixed-head disc drive may use both surfaces of a disc, and may have more than one disc, but no cylinder addressing scheme is used; since there is a separate head for every track, the ordering of the tracks can be arbitrary, as switching from one track to another simply requires switching from one head to another.

A disc system consists of a control and a number of disc drives; each drive is a separate unit, and the control is contained on one standard circuit board that can be mounted in the computer chassis. The program communi-

cates with the control, which in turn governs all discs over a disc bus but communicates with only one at a time. The control is connected to the data channel, so the program need only set up the disc system for reading or writing, and all transfers to and from memory are then handled automatically. To operate with the data channel the control has an address counter as well as data buffers (since all transfers are of fixed-length blocks, no word counting is necessary). A bank of drives can also be connected to a second control, which in turn is connected to the IO bus and data channel of another computer, thus allowing communication between the two computers through disc storage.

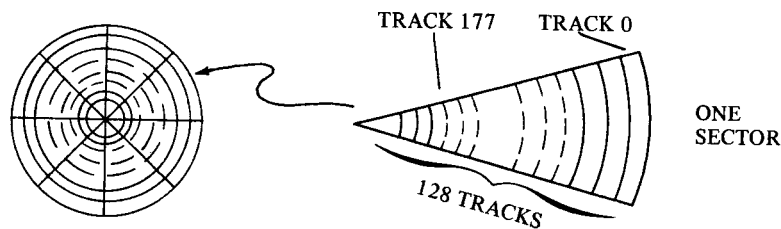
### 5.1 FIXED HEAD DISC SYSTEM

The fixed-head system consists of a 4019 control and up to eight logical units, each of which can store 262,144 words. Drives available for this system include the 6002, 6003 and 6004 which have respectively one, two and three logical units each (256K, 512K and 768K words). The same series includes a half-unit disc, the 6001 (128K words), and the control can also handle the 4019A, B and C drives, which contain respectively quarter-unit, half-unit and single discs (64K, 128K and 256K words).

All fixed-head drives run at 3600 rpm, giving an average latency time of 8.4 ms. While a block is being processed, data transfers are at the rate of one word every  $8\mu\text{s}$ ; once an initial block is found, the average transfer rate over a number of blocks is 57,835 words per second.

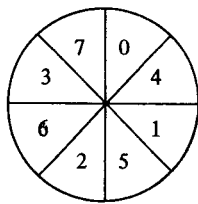
#### Data Format

Each disc has 128 circular data tracks numbered from the outside in (actually both surfaces are used but tracks on alternate surfaces are numbered consecutively). A quarter- or half-unit disc has fewer heads and hence a smaller number of tracks. Every track is divided into eight sectors, each of which contains 256 words of data. Each track-sector also contains a cyclic check word, which is generated and checked automatically by the control, as well as other information for the internal use of the control. At 3600 rpm a given sector passes the read-write heads in 2.085 ms, of which 2.05 ms are used for processing data.

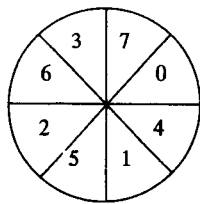


DISC CONFIGURATION

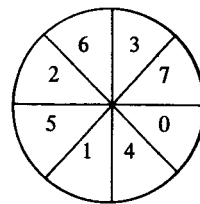
The control cannot process physically adjacent sectors consecutively; in other words after processing a given track-sector, the control can process another removed from it as soon as the other is encountered, but can process the next adjacent sector only after waiting for a complete disc revolution. To simplify programming and to minimize waiting time, the sectors are numbered alternately and the numbering scheme changes from one track to the next as shown here. Hence the program can process consecutively numbered sectors in a given track, and upon processing the last sector in a track, can switch to the first sector of the next track, all with minimum waiting time. Time between sectors is thus 2.085 ms except when switching from sector 3 to sector 4, for which the waiting time is 4.17 ms.



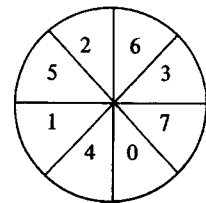
TRACK XX0



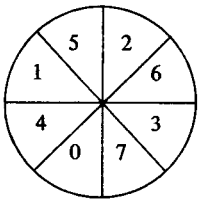
TRACK XX1



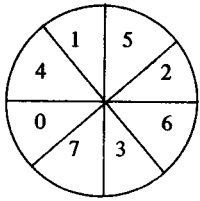
TRACK XX2



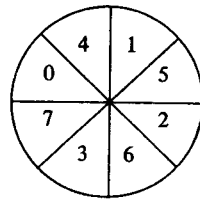
TRACK XX3



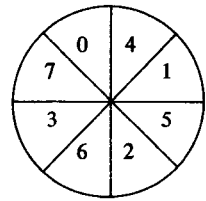
TRACK XX4



TRACK XX5



TRACK XX6



TRACK XX7

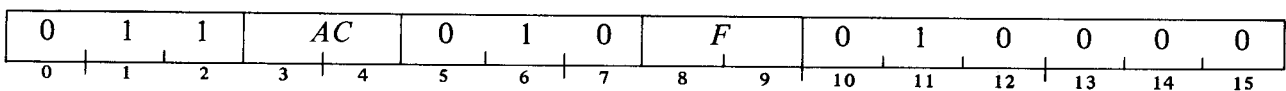
TRACK-SECTOR CONFIGURATION

To provide protection for data on a disc, sets of tracks can be locked against writing. For this purpose the 128 tracks on a disc are divided into eight sets of sixteen each. Located on the rear of the disc cabinet is a three-position switch which allows the operator to lock out none of the tracks, all of the tracks, or only those sets of tracks selected by jumpers located in the disc logic.

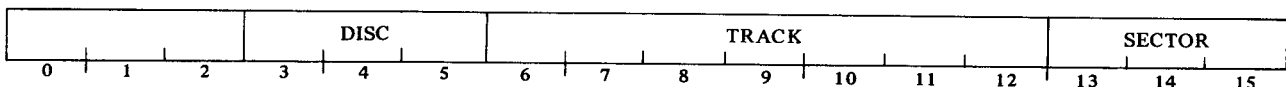
To use a fixed-head disc the program must select the disc, track and sector, specify whether data is to be read or written, and supply an initial address (to the 15-bit address counter) for data channel access. The disc system has device code 20, mnemonic DSK, and uses five of the IO transfer instructions, one of which is strictly for maintenance and can be used only when the disc control is in special diagnostic mode. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions; these flags are controlled in the usual way by Clear and Start, but the IO Pulse function ( $F = 11$ ) is also used. Interrupt Disable is controlled by interrupt priority mask bit 9. A second system connected to the bus would have device code 60.

The Clear function clears Busy and Done, and thus terminates data transfers if a track-sector is currently being processed. Start and Pulse both clear Done and set Busy, but these functions also specify the disc operation: Start selects Read, Pulse selects Write. All three functions clear the status flags.

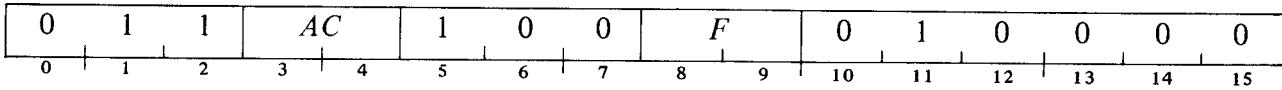
**DOA —,DSK Data Out A, Disc**



Select the disc, track and sector according to the contents of AC bits 3–15 as shown, and perform the function specified by  $F$ .



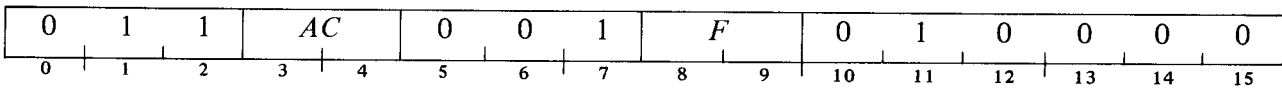
**DOB --,DSK      Data Out B, Disc**



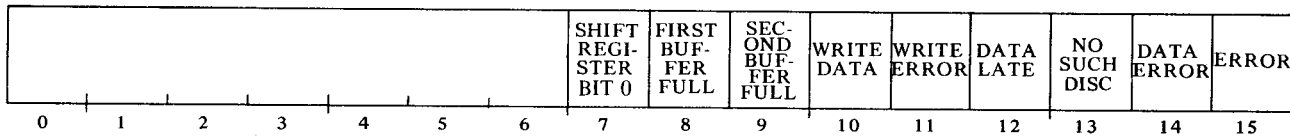
Load the contents of AC bits 1–15 into the address counter (AC bit 0 should be 0), and perform the function specified by *F*. If *F* is 01 (S), select Read; if *F* is 11 (P), select Write. If *F* is nonzero, clear the status flags.

Note: Giving this instruction with a 1 in AC bit 0 places the control in diagnostic mode.

**DIA --,DSK      Data In A, Disc**



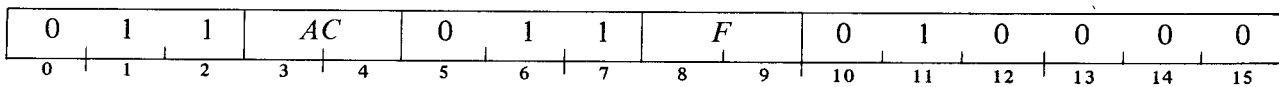
Read the status of the disc system into AC bits 7–15 as shown, and clear AC bits 0–6. (Perform the function specified by *F*.)



Bits 7-10 are for maintenance only and are not discussed further here. "Clear, Start and Pulse clear all of these flags except for bits 12 & 13 which are valid only after a Start or Pulse operation is initiated to the desired logical unit."

- 11 The program has specified Write and the selected track-sector is write-protected. The setting of this bit clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear.
- 12 The data channel has failed to respond in time to a request for access (*eg* because of a long instruction or preemption of the channel by faster devices).
- 13 The disc selected by the program is not connected to the bus. The setting of this bit clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear.
- 14 In Read, the cyclic check word read from the disc differed from that computed by the control for the data in the block.
- 15 Bit 11, 12, 13 or 14 is 1.

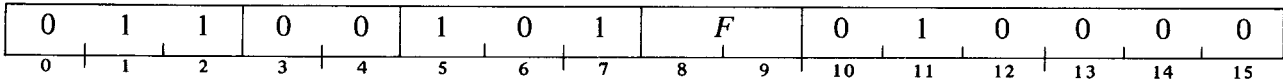
**DIB --,DSK      Data In B, Disc**



Read the present contents of the address counter into AC bits 1–15, and clear AC bit 0. (Perform the function specified by *F*.)

This instruction can be used to determine how many words have been transferred, but it is ordinarily used only for diagnostic purposes.

**DIC 0, DSK      Data In C, Disc Maintenance**



If the disc control is in diagnostic mode, supply a single clock pulse to the control logic. Perform the function specified by *F*.

Setting Busy places the disc control in operation to read or write depending upon whether the program gave Start or Pulse; in Write, the control immediately makes three data channel requests to fill the two buffers and the shift register before writing begins. If the disk selected by the DOA is not connected to the bus, or the program specified Write and the selected track-sector is write-protected, the control sets the appropriate status flag, clears Busy, and sets Done, requesting an interrupt if Interrupt Disable is clear.

If there is no error the control waits until the selected track-sector is encountered; it then processes the block, making data channel requests whenever it has a word ready for memory in reading or one of the buffers is free in writing. The setting of Data Late during a block indicates that information has been lost, but data transfers continue until the control processes the entire block. At the completion of the data block in Write, the control writes a computed check word at the end of the track-sector; in Read, the control compares the check word read from the disc with one it has computed from the data read, and sets Data Error if they differ. At completion the control clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear.

**Timing.** After Start or Pulse is given for the first operation with a given disc, the control may wait up to 16.7 ms before the selected track-sector is encountered; moreover, no operation can be performed until 1 ms after DOA is given. While processing the block, the control makes data requests every 8 μs, but because of double buffering in the control the processor may take up to 14 μs to respond in an isolated case without being late. Once an operation has been performed with a given disc, the program then knows the disc orientation and thus knows exactly the waiting time required to reach any desired track-sector. The time required to traverse any sector is 2.085 ms, which is also the time taken between consecutively numbered sectors except between sectors 3 and 4, which are separated by 4.17 ms.

**Programming Considerations**

After one Read, no further DOBs need be given if subsequent operations are also Read and are to access consecutive pages in memory.

*CAUTION*

For Write, always give both a DOA and a DOB. The address counter does not count properly from one block to the next in writing.

At the completion of each operation the program should check status, and if data was late or in error, the operation should be repeated. Do not check status before starting an operation with a disc — the status is not valid until an operation has been performed.

The word sent by a DOA is set up so that the program can process consecutive sectors and tracks (and even discs) simply by incrementing. Suppose we wish to process tracks 10–13, all sectors, on disc 1. We load 002100 into AC2 and give a DOA 2,DSK for the first track-sector. For subsequent track-sectors we simply increment AC2 before giving the DOA.

**Automatic Loading.** Ordinarily sector 0, track 0 of disc 0 is reserved for a binary loader. Should the loader in core be destroyed by program debugging, it can easily be restored from the disc. To bring the loader into memory automatically, set device code 20 into data switches 10–15 at the computer console. Then in a Nova 1200 series or 800 series computer with the program load option, press RESET, turn on data switch 0, then press PROGRAM LOAD; in a Supernova computer press RESET and then CHANNEL START. To bring the loader into memory without automatic loading, the operator must use the following procedure:

1. Press RESET.
2. Set 376 into the data switches and press EXAMINE.
3. Set the instruction NIOS DSK (060120) into the data switches and press DEPOSIT.
4. Set 000377 into the data switches (JMP 377) and press DEPOSIT NEXT.
5. Set 376 into the data switches and press START.

**Multiprocessor Operation.** When two controls from different computers are connected to the same disc bus, access is alternated between them whenever there is a conflict. When one control finishes a track-sector, access is automatically given to the other control if it is making a request. If not, the first control can continue.

The restriction on processing adjacent sectors still applies: if both processors are doing disc runs simultaneously, each can process at most one track-sector every half revolution (8.35 ms).

## 5.2 DISC CARTRIDGE SYSTEM 4047

The movable-head disc cartridge system is based on the 4046 control, which can handle four drives via a 4047 adapter. The adapter can also be connected to a second control so that its bank of drives can be made available to another computer. The 4047A drive has a single removable 4047C cartridge, and there is also a 4047B double drive, which is two drives in a single package. One drive in the package has a removable cartridge and the other has a fixed disc (albeit with movable heads).

The system stores data in blocks of 256 words, and all discs (whether fixed or removable) rotate at 1500 rpm. The capacity of each disc is 1,247,232 words. In the data part of a block, words are processed at the rate of one every 11.1  $\mu$ s. The control can process up to sixteen consecutive sectors at a time within a single cylinder, and can therefore process 4096 words in 53.3 ms.

### Data Format

Each disc has two surfaces divided into twelve sectors addressed as octal 0–13. Each surface has 203 tracks, addressed as 0–312 octal. Each track-sector contains a data block and a cyclic check word, which is generated and checked automatically by the control. Since a block contains 256 words, each disc with twelve sectors and two surfaces has 3072 words per track, 6144 words per cylinder; 203 cylinders gives a total capacity of 1,247,232 words. Often data operations are limited to 200 tracks per surface with the other three kept as spares.

The drive has one head per disc surface, with both heads mounted in a single carriage so that they are positioned simultaneously. The maximum time required to move the heads from one cylinder to the next is 20 ms,



and at most 135 ms are required for motion from one extremity to the other. Once the positioning operation is complete, the control must wait for the specified sector to reach the heads; this requires about half a revolution on average (20 ms).

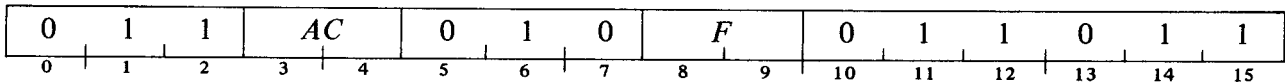
### Instructions

Before reading or writing, the program must give a Seek command to position the heads at the desired cylinder; once the Seek is done the program can then give a Read or Write command. Information the program must supply for a disc operation is the drive, cylinder, surface and sector, the (two's complement) negative of the number of sectors to be processed, and an initial address (to the 16-bit address counter) for data channel access. In a single Read or Write, the hardware automatically counts from one sector to the next and from the first surface to the second until the sector count is zero. However data operations always stop at the end of a cylinder, and the program must reposition the heads to go on to the next cylinder. Throughout this discussion it is assumed that the system is operating from a single processor; multiprocessor operation is treated at the end of the section on programming considerations.

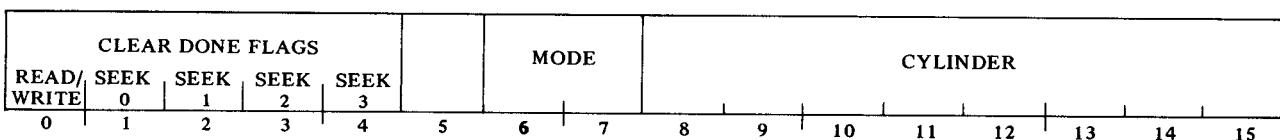
The disc cartridge system has device code 33, mnemonic DKP, and uses all six of the IO transfer instructions. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions. The Clear and Start functions control these flags in the usual fashion, but the IO Pulse function ( $F = 11$ ) is also used. Interrupt Disable is controlled by interrupt priority mask bit 7. A second system connected to the bus would have device code 73.

The Clear function clears Busy, Done and the other flags in the control, and terminates operations if the control is currently processing data. Start and Pulse are both used to start disc commands, but the command is determined by the mode of the control; in other words the program specifies a command by placing the control in the appropriate mode and actually begins the command by giving Start or Pulse. When the Seek or Recalibrate mode is selected, the program starts the command by giving Pulse, as this function does not set Busy (although it does clear Done); hence while the heads are being positioned on one drive the control is free for a command on another. For Read or Write mode, the program gives Start to set Busy, and thus prevents the control from beginning any other command while processing data. An interrupt is requested as usual by the setting of Done, but there are also separate done flags that indicate the completion of a Seek command in any drive or the completion of a Read or Write command in the drive currently selected by the control. The setting of any Seek Done flag or the Read/Write Done flag requests an interrupt, and Read/Write Done also clears Busy and sets Done in the control.

### DOA --DKP      Data Out A, Disc Pack



Clear the Done flags selected by 1s in AC bits 0-4 and select the mode and cylinder according to the contents of AC bits 6-15 as shown. Perform the function specified by *F* as explained above.

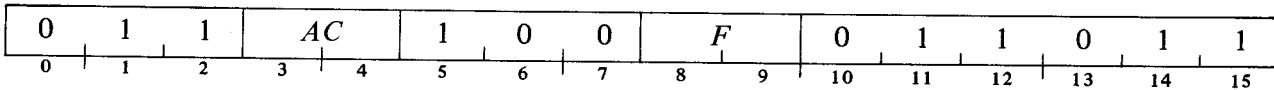


6-7 These bits select the mode:

- 00 Read
- 01 Write
- 10 Seek — position the heads to the cylinder specified by bits 8-15
- 11 Recalibrate — force the heads to cylinder 0 independently of the head-positioning circuits

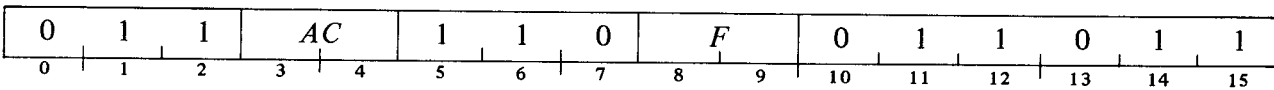
8-15 Numbers 0-312 octal address the cylinder.

**DOB —,DKP Data Out B, Disc Pack**

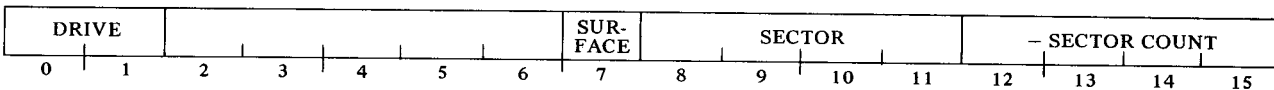


Load the contents of AC into the address counter (AC bit 0 should be 0), and perform the function specified by *F* as explained above.

**DOC —,DKP Data Out C, Disc Pack**



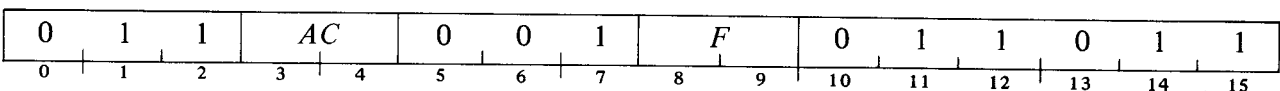
Select the drive, surface and sector, and the number of sectors to be processed, according to the contents of AC as shown; perform the function specified by *F* as explained above.



Bits 2-6 must be all zeroes.

- 0-1 Numbers 0-3 select the drive.
- 7 Numbers 0 and 1 select the surface.
- 8-11 Numbers 0-13 select the sector.
- 12-15 The two's complement negative of the number of sectors to be processed by Read or Write (maximum: sixteen).

**DIA —,DKP Data in A, Disc Pack**



Read the status of the disc system into AC as shown. (Perform the function specified by *F*.)

COMMAND DONE					SEEKING ON DRIVE				DISC	SEEK	END	UN-	CHECK	DATA	ERROR
READ/ WRITE	SEEK 0	SEEK 1	SEEK 2	SEEK 3	0	1	2	3	READY	ERROR	ERROR	SAFE	ERROR	LATE	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Clear function clears all of the done and error flags; Start and Pulse clear the error flags.

- 0-4 A drive has completed a command as indicated by 1s in these bits; a Seek Done flag can be set from any drive, Read/Write Done can be set only from the drive currently selected. The setting of any of these flags requests an interrupt.
- 5-8 1s in these bits indicate drives on which the heads are presently being positioned.
- 9 The selected drive is available to the program.
- 10 The selected drive failed to position its heads as requested.
- 11 The control has reached the end of a cylinder but the sector counter is not zero, and the data operation has terminated anyway.
- 12 There is a malfunction in the drive.
- 13 In Read, the cyclic check word read from the disc differed from that computed by the control for the data in the block.
- 14 The data channel has failed to respond in time to a request for access.
- 15 Any of bits 10-14 is 1.

**DIB -,DKP Data in B, Disc Pack**

0	1	1	AC	0	1	1	F	0	1	1	0	1	1		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the present contents of the address counter into AC. (Perform the function specified by F.)

At the end of a Write command, the address counter is 2 greater than the address of the last word written.

**DIC -,DKP Data in C, Disc Pack**

0	1	1	AC	1	0	1	F	01	1	1	0	1	1		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the address status of the presently selected drive into AC as shown. (Perform the function specified by F.)

DRIVE							SUR-	SECTOR				- SECTOR COUNT			
0	1	2	3	4	5	6	FACE	8	9	10	11	12	13	14	15

The normal programming procedure for operating a single drive is to give a DOC to select the drive, surface, sector and sector count, and then give a DIA to check whether the disc is ready (status bit 9). If it is, the program should give a DOAP to select the cylinder, and to select and initiate the Seek command. When the heads are properly positioned the drive sets its Seek Done flag requesting an interrupt. The program should respond with a DIA to check status bit 10, Seek Error, to ensure that the heads have positioned properly. A DOA is then given to clear Seek Done and select the Read or Write mode as desired. Finally a DOBS specifies the initial address and starts the Read or Write. For Write the control immediately makes two data channel requests to fill two buffers before writing begins.

The control waits until the drive encounters the selected sector on the selected surface; it then processes the block, making a pair of data channel requests whenever it has two words ready for memory in reading or both buffers are free in writing. As each sector is completed, the sector counter is incremented by one; and upon completing the final sector in the track, the surface counter is incremented by one while the sector counter is returned to zero, so the operation continues from surface 0 to surface 1 in the cylinder. For each sector, the control also increments the sector count (*ie* the negative count of the number of sectors still to be processed). When the sector count reaches zero, the control terminates the command (with the counters pointing to the last block processed) and sets Read/Write Done, which in turn clears Busy and sets Done requesting an interrupt.

The setting of Data Late during a block indicates that information has been lost, but data transfers continue until the control processes the entire block, at which time it terminates the command. At the completion of a data block in Write, the control writes a computed check word at the end of the sector; in Read, the control compares the check word read from the disc with one it has computed from the data read, and if they differ, sets Check Error and terminates the command. If the control reaches the end of the cylinder (*ie* the end of the last sector on surface 1) and the sector count is not zero, the control sets End Error and terminates the command.

Although the program can process data with only one drive at a time, it can position the heads on several drives simultaneously. After giving a Seek for one drive, the program can select another and give a Seek for it. This means that the surface, sector and sector count information cannot be given when the drive is selected initially for seeking, and each time the program wishes to handle a particular drive it must reselect that drive. *Eg* if several drives are seeking simultaneously and there is an interrupt, the program should give a DIA to determine which drive interrupted, then give a DOC to select that drive, and give another DIA to check Seek Error for that drive. To read or write, the program must give all three output commands, DOA, DOB, and DOC, to supply all the necessary information (the order of the instructions is not important, but Start should be given with the last of the three in order for the command to start properly).

**Timing.** The maximum time required to move the heads from one cylinder to the next is 20 ms, and at most 135 ms are required for motion from one extremity to the other; average random positioning time is therefore at most 70 ms. These times are of course for the drive; a command that moves the heads takes about 50  $\mu$ s of control time (the program must wait 50  $\mu$ s before giving any other disc instruction).

At 1500 rpm the cartridge takes 40 ms per revolution. Although the time to traverse one complete sector is 3.33 ms, traversing the data block within a sector takes only 2.84 ms. Search time for a random data block is 40.5 ms maximum, 20.5 ms average. During processing, data channel requests for pairs of words occur every 22.2  $\mu$ s, and the processor has almost this much time to supply both words.

### **Programming Considerations**

If a Seek Error is indicated, give the Recalibrate command and do the Seek over again. Be careful not to give an address for a cylinder or sector that does not exist (in other words a number too large for the system). Selecting a nonexistent cylinder causes a Seek Error. The drive searches forever for a sector that does not exist (the system can be freed by giving the IO reset function).

In Write the address counter is always two words ahead of the disc as the control always asks for two words to have ready when the previous two are shifted to the drive. Hence at the end of a Write, the address counter

points to the location two beyond the last word that was actually written in the final sector. To start a new Write for consecutive operation from memory, the program must give a new initial address for the correct location in memory (two less than the address read by a DIB). In Read the control requests access only for words actually read, so the program can give a new Read without supplying a new initial address.

On a Data Late or Check Error the program should reprocess the sector in which the error occurred. If information cannot be read correctly after several tries, rewrite the bad block. If this fails to correct the error, the sector can no longer be used.

When processing to the end of a cylinder, the program should give the correct sector count, as a nonzero count at the end of a cylinder is regarded as an error. The nonzero count causes the control to act as though the command were continuing, so it returns the sector address to zero, increments the surface counter to 2, and increments the sector count by one. Should the program inadvertently cause an End Error by attempting continuous processing across the boundary between cylinders, it can continue the operation by positioning the heads to the next cylinder, adjusting the initial address (as explained above) if the command is Write, and giving a DOB that selects the same drive, surface 0, sector 0, and specifies the negative of a sector count one greater than that indicated by the status read by a DIC.

Within a given cylinder, the control can process sixteen consecutive sectors (4096 words). Upon completion of a data command however the control cannot process the next consecutive sector; if the program immediately gives a new command for the next sector, the control will wait for a complete disc revolution. The fastest way to process an entire cylinder is to use three commands this way: the first command processes thirteen sectors, the entire track on surface 0 and sector 0 of surface 1; the second command skips to sector 2 on surface 1 to process the ten remaining sectors in the track; and finally the third command processes sector 1 of surface 1. This does not mean that the program need jump around in memory, as there is no need to have consecutive memory blocks correspond to physically consecutive sectors; in other words the program can write consecutive memory blocks using this technique, and reading by the same procedure brings the data back into memory in the original order.

**Automatic Loading.** Ordinarily sector 0, surface 0, cylinder 0 of drive 0 is reserved for a binary loader. Should the loader in core be destroyed by program debugging, it can easily be restored from the disc. To position the heads at cylinder 0, the operator should first turn the drive power off and then back on again. To bring the loader into memory automatically, set device code 33 into data switches 10-15 at the computer console. Then in a Nova 1200 series or 800 series computer with the program load option, press RESET, turn on data switch 0, then press PROGRAM LOAD; in a Supernova computer press RESET and then CHANNEL START. To bring the loader into memory without automatic loading, the operator must use the following procedure:

1. Press RESET.
2. Set 376 into the data switches and press EXAMINE.
3. Set the instruction NIOS DKP (060133) into the data switches and press DEPOSIT.
4. Set 000377 into the data switches (JMP 377) and press DEPOSIT NEXT.
5. Set 376 into the data switches and press START.

The disc control will read sixteen consecutive sectors unless the loader stops it.

**Multiprocessor Operation.** When two controls from different computers are connected to the same disc adapter, access is alternated between them whenever there is a conflict. When one control finishes a data command, access is automatically given to the other control if it is making a request. If not, the first control can continue.

The program must give a Seek for every data command, as the other processor might have changed the head position. If the adapter is being used by the other processor when a Seek is given, the control waits until the adapter is free to start it; in the meantime the Seeking flag will be on, and the program must not give another command (with multiprocessor operation, seeking on more than one drive at a time is ineffective). Once the adapter starts the Seek, it is unavailable to the other control until either Read/Write Done is set or six seconds have elapsed.

If a disc is known to be available, loss of Disc Ready can be taken to mean that the other control is using the drive. The programs in both computers should allow for operator intervention.

### **Operation**

On the adapter is a 3-position key-operated rotary switch, which controls power for both the adapter and the drives. With the switch in the LOCK position the key can be removed with the power on.

To load a cartridge set the rocker switch on the front of the drive to LOAD and lift the drive door. Hold the cartridge with its door toward the drive and slide it into the slot. Close the drive door and set the switch to RUN. To remove a cartridge set the switch to LOAD and wait about 15 seconds. Then open the drive door and pull the cartridge out.

The lights at the right on the drive indicate when it is in the load state, when it is ready for use by the program, and when a check error has occurred (the protect light is not used). The number of a drive (for calling by the program) is determined by the drive's two internal jumpers.

## **5.3 DISC PACK SYSTEMS**

Data General has equipment for handling two sizes of movable-head disc packs. Both systems use the 4046 control, which can handle four drives via an adapter. The adapter can also be connected to a second control so that its bank of drives can be made available to another computer. The 4048 adapter utilizes the 4048A drive, and the 4057 adapter utilizes the larger 4057A drive.

Both drives store data in blocks of 256 words and operate at the same rotational speed, 2400 rpm. The 4057A has four times the storage capacity and processes data at twice the speed of the 4048A. The capacity of the 4048A is 3,118,080 words. Within a data block words are processed at the rate of one every 12.8  $\mu$ s, and the entire cylinder of 15,360 words can be processed in 250 ms. The 4057A stores 12,472,320 words, processed at the rate of one every 6.4  $\mu$ s; a single cylinder containing 61,440 words can be processed in 500 ms.

### **Data Format**

The 4048A disc pack has ten surfaces divided into six sectors, addressed as octal 0-11 and 0-5 respectively; the 4057A has twenty surfaces divided into twelve sectors, numbered 0-23 and 0-13 octal. Each surface in either pack has 203 tracks, addressed as 0-312 octal. Each track-sector contains a data block and a cyclic check word, which is generated and checked automatically by the control. As each block contains 256 data words, the 4048A with ten surfaces and six sectors has 1536 words per track, 15,360 words per cylinder; 200 cylinders gives a total capacity of 3,118,080 words. The 4057A with twenty surfaces and twelve sectors, has 3,072 words per track, 61,440 words per cylinder, and a total capacity of 12,472,320 words. Often data operations are limited to 200 tracks per surface with the other three kept as spares.

Both drives have one head per pack surface, with all heads mounted in a single carriage so that they are positioned simultaneously. The maximum time required to move the heads from one cylinder to the next is 10 ms, and at most 60 ms are required for motion from one extremity to the other. Once the positioning operation is complete, the control must wait for the beginning of the specified track for an address check; on average this requires half a revolution (12.5 ms). Then the control must wait for the specified sector to reach the heads; since the address check is at the beginning of the track, the waiting time depends entirely on which sector is selected by the program.

### **Instructions**

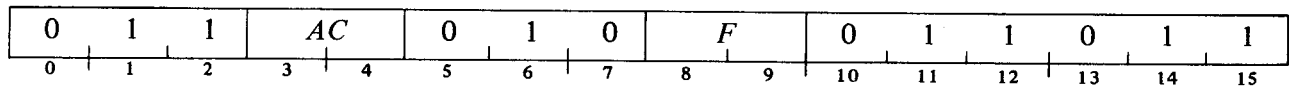
Before reading or writing, the program must give a Seek command to position the heads at the desired

cylinder; once the Seek is done the program can then give a Read or Write command. Information the program must supply for a disc operation is the drive, cylinder, surface and sector, the (twos complement) negative of the number of sectors to be processed, and an initial address (to the 16-bit address counter) for data channel access. In a single Read or Write, the hardware automatically counts from one sector to the next and from one surface to the next until the sector count is zero. However data operations always stop at the end of a cylinder, and the program must reposition the heads to go on to the next cylinder. Throughout this discussion it is assumed that the system is operating from a single processor; multiprocessor operation is treated at the end of the section on programming considerations.

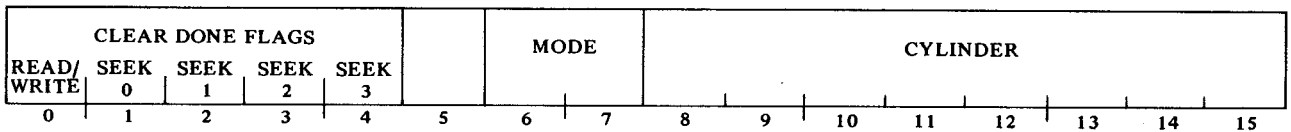
A disc pack system has device code 33, mnemonic DKP, and uses all six of the IO transfer instructions. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions. The Clear and Start functions control these flags in the usual fashion, but the IO Pulse function ( $F = 11$ ) is also used. Interrupt Disable is controlled by interrupt priority mask bit 7. A second system connected to the bus would have device code 73.

The Clear function clears Busy, Done and the other flags in the control, and terminates operations if the control is currently processing data. Start and Pulse are both used to start disc commands, but the command is determined by the mode of the control; in other words the program specifies a command by placing the control in the appropriate mode and actually begins the command by giving Start or Pulse. When the Seek or Recalibrate mode is selected, the program starts the command by giving Pulse, as this function does not set Busy (although it does clear Done); hence while the heads are being positioned on one drive the control is free for a command on another. For Read or Write mode, the program gives Start to set Busy, and thus prevents the control from beginning any other command while processing data. An interrupt is requested as usual by the setting of Done, but there are also separate done flags that indicate the completion of a Seek command in any drive or the completion of a Read or Write command in the drive currently selected by the control. The setting of any Seek Done flag or the Read/Write Done flag requests an interrupt, and Read/Write Done also clears Busy and sets Done in the control.

**DOA —,DKP      Data Out A, Disc Pack**



Clear the Done flags selected by 1s in AC bits 0-4 and select the mode and cylinder according to the contents of AC bits 6-15 as shown. Perform the function specified by  $F$  as explained above.



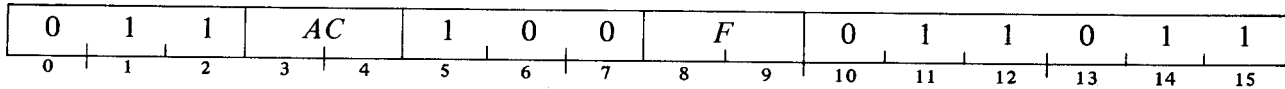
6-7      These bits select the mode:

- 00 Read
- 01 Write
- 10 Seek — position the heads to the cylinder specified by bits 8-15

11 Recalibrate-position heads at cylinder 0 independently of the head-positioning circuits

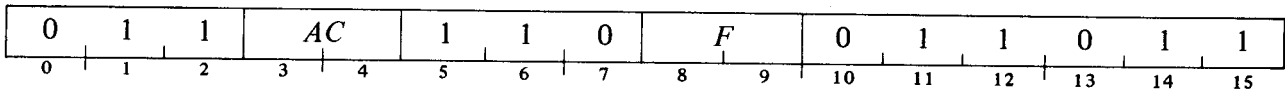
8-15 Numbers 0-312 octal address the cylinder.

**DOB - ,DKP Data Out B, Disc Pack**

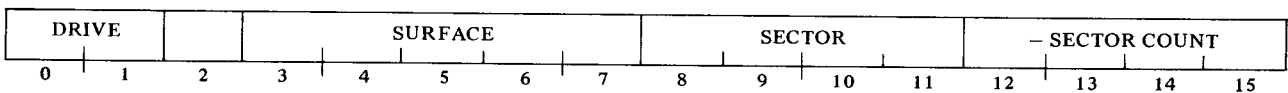


Load the contents of AC into the address counter (AC bit 0 should be 0), and perform the function specified by *F* as explained above.

**DOC - ,DKP Data Out C, Disc Pack**



Select the drive, surface and sector, and the number of sectors to be processed, according to the contents of AC as shown; perform the function specified by *F* as explained above.



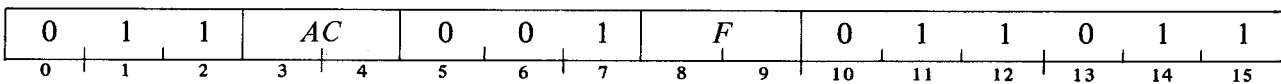
0-1 Numbers 0-3 select the drive.

3-7 Numbers 0-11 select the surface in the 4048; numbers 0-23 select the surface in the 4057.

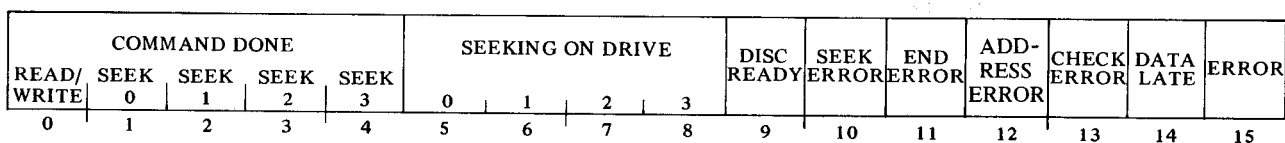
8-11 Numbers 0-5 select the sector in the 4048; numbers 0-13 select the sector in the 4057.

12-15 The two's complement negative of the number of sectors to be processed by Read or Write (maximum: sixteen).

**DIA - ,DKP Data in A, Disc Pack**



Read the status of the disc system into AC as shown. (Perform the function specified by *F*.)

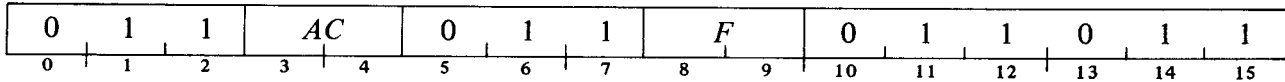




The Clear function clears all of the done and error flags; Start and Pulse clear the error flags.

- 0-4 A drive has completed a command as indicated by 1s in these bits; a Seek Done flag can be set from any drive, Read/Write Done can be set only from the drive currently selected. The setting of any of these flags requests an interrupt.
- 5-8 1s in these bits indicate drives on which the heads are presently being positioned.
- 9 The selected drive is available to the program.
- 10 The selected drive failed to position its heads as requested.
- 11 The control has reached the end of a cylinder but the sector counter is not zero, and the data operation has terminated anyway.
- 12 Address information read from the disc did not agree with that specified by the control, or there is a malfunction in the drive.
- 13 In Read, the cyclic check word read from the disc differed from that computed by the control for the data in the block.
- 14 The data channel has failed to respond in time to a request for access.
- 15 Any of bits 10-14 is 1.

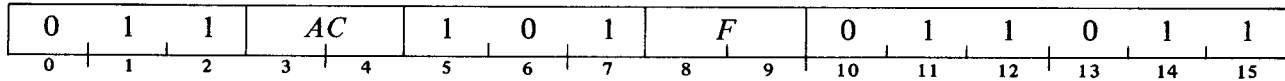
**DIB -,DKP Data in B, Disc Pack**



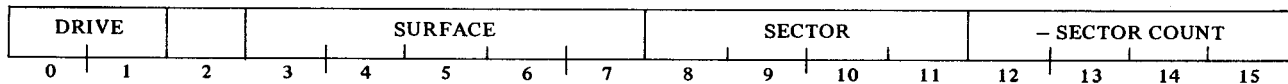
Read the present contents of the address counter into AC. (Perform the function specified by *F*.)

At the end of a Write command, the address counter is 2 greater than the address of the last word written.

**DIC -,DKP Data in C, Disc Pack**



Read the address status of the presently selected drive into AC as shown. (Perform the function specified by *F*.)



The normal programming procedure for operating a single drive is to give a DOC to select the drive, surface, sector and sector count, and then give a DIA to check whether the disc is ready (status bit 9). If it is, the program should give a DOAP to select the cylinder, and to select and initiate the Seek command. When the heads are properly positioned the drive sets its Seek Done flag requesting an interrupt. The program should respond

with a DIA to check status bit 10, Seek Error, to ensure that the heads have positioned properly. A DOA is then given to clear Seek Done, select the desired cylinder again, and select the Read or Write mode as desired. Finally a DOBS specifies the initial address and starts the Read or Write. For Write the control immediately makes two data channel requests to fill two buffers before writing begins.

The control waits for the drive to do an address check and search for the selected sector. When the disc reaches the index point — the end of the final sector — the drive reads the address information written at the beginning of the track that is at the head corresponding to the surface selected by the control. If the cylinder and surface addresses written in the track do not match the cylinder and surface addresses supplied by the last DOA and DOC respectively, Address Error sets and the command terminates. If the address information is correct, the control waits until the drive encounters the selected sector; it then processes the block, making a pair of data channel requests whenever it has two words ready for memory in reading or both buffers are free in writing. As each sector is completed the sector counter is incremented by one; and upon completing the final sector (*ie* upon reaching the index point) the surface counter is incremented by one while the sector counter is returned to zero, so the operation continues from one surface to the next in the cylinder. For each sector, the control also increments the sector count (*ie* the negative count of the number of sectors still to be processed). When the sector count reaches zero, the control terminates the command (with the counters pointing to the last block processed) and sets Read/Write Done, which in turn clears Busy and sets Done requesting an interrupt.

The drive always does an address check before starting a command, and since the address check is at the index point, the control cannot end one command and start another within a track. However if the program always gives a sector count such that every command terminates at the end of a track, then the disc can be processed continuously. As each command ends, the program can give another that begins processing at sector 0 since the address check is made immediately.

The setting of Data Late during a block indicates that information has been lost, but data transfers continue until the control processes the entire block, at which time it terminates the command. At the completion of a data block in Write, the control writes a computed checkword at the end of the sector. In Read, the control compares the checkword read from the disc with one it has computed from the data read, and if they differ, sets Check Error and terminates the command. If the control reaches the end of the cylinder (*i.e.*, the end of the last sector on the last surface) and the sector count is not zero, the control sets End Error and terminates the command.

Although the program can process data with only one drive at a time, it can position the heads on several drives simultaneously. After giving a Seek for one drive, the program can select another and give a Seek for it. This means that the surface, sector and sector count information cannot be given when the drive is selected initially for seeking, and each time the program wishes to handle a particular drive it must reselect that drive. *Eg* if several drives are seeking simultaneously and there is an interrupt, the program should give a DIA to determine which drive interrupted, then give a DOC to select that drive, and give another DIA to check Seek Error for that drive. To read or write, the program must give all three output commands, DOA, DOB, and DOC, to supply all the necessary information (the order of the instructions is not important, but Start should be given with the last of the three in order for the command to start properly).

**Timing.** The maximum time required to move the heads from one cylinder to the next is 10 ms, and at most 60 ms are required for motion from one extremity to the other; average random positioning time is therefore at most 35 ms. These times are of course for the drive; a command that moves the heads takes about 10  $\mu$ s of control time (the program must wait 10  $\mu$ s before giving any other disc instruction).

At 2400 rpm the pack takes 25 ms per revolution. Since the drive must check the address at the index point before searching for a specified sector, the search time for a random sector can be as much as 50 ms; specifically, the total wait is the time to reach the index point (maximum 25 ms, average 12.5 ms) plus the known time required from the index to the desired sector. In continuous processing, waiting time can be reduced to a minimum by judicious ordering of sectors to be processed.

On the 4048, the time to traverse one complete sector is 4.17 ms, and to traverse the data block within a sector takes 3.28 ms. During processing, data channel requests for pairs of words occur every 25.6  $\mu$ s, and the processor has almost this much time to supply both words. If a command terminates following the final sector in a track, the program has approximately .5 ms to give a new command to process the first sector without waiting for a complete disc revolution.

On the 4057 the time to traverse one complete sector is 2.08 ms, and to traverse the data block within a sector takes 1.64 ms. During processing, data channel requests for pairs of words occur every 12.8  $\mu$ s, and the processor has almost this much time to supply both words. If a command terminates following the final sector in a track, the program has approximately .2 ms to start a new command to process the first sector without waiting for a complete disc revolution.

### **Programming Considerations**

If a Seek Error is indicated, give the Recalibrate command and do the Seek over again. Be careful not to give an address for a cylinder, surface or sector that does not exist (in other words a number too large for the system). Giving a Seek for a nonexistent cylinder causes a Seek Error. With a data command, selecting a nonexistent cylinder or surface, or specifying a cylinder different from the one at which the heads are positioned, sets Address Error and terminates the command. The drive searches forever for a sector that does not exist (the system can be freed by giving the IO reset function).

Within a given cylinder, the control can process sixteen consecutive sectors (4096 words). Since the drive must go to the index point between commands, to process an entire cylinder it is best for each command to handle one or two tracks (six or twelve sectors) on the 4048, a single track (twelve sectors) on the 4057.

In Write the address counter is always two words ahead of the disc as the control always asks for two words to have ready when the previous two are shifted to the drive. Hence at the end of a Write, the address counter points to the location two beyond the last word that was actually written in the final sector. To start a new Write for consecutive operation from memory, the program must give a new initial address for the correct location in memory (two less than the address read by a DIB). In Read the control requests access only for words actually read, so the program can give a new Read without supplying a new initial address.

On a Data Late or Check Error the program should reprocess the sector in which the error occurred. If information cannot be read correctly after several tries, rewrite the bad block. If this fails to correct the error, the sector can no longer be used. A Check Error may also occur in the address field at the beginning of each track.

When processing to the end of a cylinder, the program should give the correct sector count, as a nonzero count at the end of a cylinder is regarded as an error. The nonzero count causes the control to act as though the command were continuing, so it returns the sector address to zero, increments the surface counter to one beyond the final surface in the pack, and increments the sector count by one. Should the program inadvertently cause an End Error by attempting continuous processing across the boundary between cylinders, it can continue the operation by positioning the heads to the next cylinder, adjusting the initial address as explained above if the command is Write, and giving a DOB that selects the same drive, surface 0, sector 0, and specifies the negative of a sector count one greater than that indicated by the status read by a DIC.

**Automatic Loading.** Ordinarily sector 0, surface 0, cylinder 0 of drive 0 is reserved for a binary loader. Should the loader in core be destroyed by program debugging, it can easily be restored from the disc. To position the heads at cylinder 0, the operator should first turn the drive power off and then back on again. To bring the loader into memory automatically, set device code 33 into data switches 10–15 at the computer console. Then in a Nova 1200 series or 800 series computer with the program load option, press RESET, turn on data switch 0, then press PROGRAM LOAD; in a Supernova press RESET and then CHANNEL START. To bring the loader into memory without automatic loading, the operator must use the following procedure:

1. Press RESET.

2. Set 376 into the data switches and press EXAMINE.
3. Set the instruction NIOS DKP (060133) into the data switches and press DEPOSIT.
4. Set 000377 into the data switches (JMP 377) and press DEPOSIT NEXT.
5. Set 376 into the data switches and press START.

The disc control will read sixteen consecutive sectors unless the loader stops it.

**Multiprocessor Operation.** When two controls from different computers are connected to the same disc adapter, access is alternated between them whenever there is a conflict. When one control finishes a data command, access is automatically given to the other control if it is making a request. If not, the first control can continue.

The program must give a Seek for every data command, as the other processor might have changed the head position. If the adapter is being used by the other processor when a Seek is given, the control waits until the adapter is free to start it; in the meantime the Seek flag will be on and the program must not give another command (with multiprocessor operation, seeking on more than one drive at a time is ineffective). Once the adapter starts the Seek, it is unavailable to the other control until either Read/Write Done is set or six seconds has elapsed.

If a disc is known to be available, loss of Disc Ready can be taken to mean that the other control is using the drive. The programs in both computers should allow for operator intervention.

### Operation

A three-position, OFF/LOCK/ON, key-operated switch is provided on the front panel of the adapter cabinet; this switch controls power to the drive and may be locked in the ON state by removal of the key in the LOCK position.

At the left of the access door on the drive are three alternate-action pushbutton switch/indicators and one indicator. With the adapter power on, a disc pack loaded and the access door closed, depressing the POWER ON pushbutton switch applies power to the drive and initiates the power-up sequence; when power is applied to the unit, the white POWER ON indicator lamp is illuminated. Depressing the green pushbutton switch (Enable On) places the disc drive on-line at the completion of the power-up sequence; the green indicator lamp will illuminate when the disc drive is ready to accept and execute the commands of the program. After initially depressing this pushbutton switch to place the disc drive on-line, it is unnecessary to activate this switch each time power is applied to the unit. If the disc drive is placed off-line by depressing the green pushbutton switch, causing the indicator lamp to extinguish, the on-line mode is restored by again depressing the green pushbutton switch. To power down the disc drive, depress the POWER ON pushbutton switch; the indicator lamp will extinguish. Note: the access door contains a window for visual inspection of the disc drive; allow the disc pack to stop revolving before opening this door.

Depressing the READ ONLY pushbutton switch until the yellow indicator lamp illuminates causes a write-lock condition. To restore the write mode, depress the pushbutton switch until the indicator lamp extinguishes.

The red SELECT LOCK indicator, when illuminated, indicates an unsafe drive condition exists.

To load a pack, hold it by its handle and unscrew the bottom of the container. Open the drive door, set the pack down over the spindle, and turn it clockwise until it stops. Lift off the cover and close the door. To keep dust out of the empty-case screw the bottom back into the cover. To remove a pack turn off the drive and wait about twelve seconds. Then open the door, place the cover over the pack, and turn it counterclockwise until a click is heard; the pack is now free of the spindle and is held tightly inside the cover, so it can be lifted out. Screw the bottom back onto the case.

# Chapter VI

## Analog Conversion Equipment

Equipment is available with the Nova line computers for both analog-to-digital (A-D) and digital-to-analog (D-A) signal conversion. The user has great latitude in configuring the system to meet his needs by selecting from among a large number of modular building blocks. The first four sections of this chapter describe these systems. The first section treats the overall configuration of the combined system. The second and third sections describe the organization and programming of the A-D and D-A equipment respectively, whereas §6.4 discusses the conversion scale and gives the complete physical, electrical and environmental specifications for the equipment. One device that requires a D-A converter is the 4053 oscilloscope control, which is treated in §6.5.

### 6.1 A-D, D-A SYSTEM CONFIGURATIONS

Data General supplies A-D converters with resolutions of 8, 10, 12, 13, 14 and 15 bits and D-A converters with resolutions of 8, 10, 12, 13 and 14 bits. To provide a number of analog outputs from the computer, a separate D-A converter is required for each channel. Multiplexing however allows the computer to handle up to 256 analog input channels through a single A-D converter. The conversion equipment is contained entirely in chassis that are 3½ inches high, 17 inches deep, and can be mounted on slides in a standard 19-inch rack. The main chassis is a 4055A or 4055B, which contains a power supply and a large printed circuit mother board at the bottom. Individual pc boards are plugged directly into the mother board, which has four edge connectors that are available at the back of the chassis. Two of the connectors are for the analog channels in and out and for connecting to expansion chassis; the other two connect the basic chassis to the computer interfaces.

The two main chassis differ in the capacity of their power supplies and therefore in the amount of equipment they can handle. The 4055A low capacity chassis can contain an A-D converter with sample and hold and a multiplexor with thirty-two single-ended or sixteen differential channels; *or* eight D-A converters; *or* an A-D converter with sample and hold and a multiplexor with sixteen single-ended or eight differential channels *and* two D-A converters. The 4055B high capacity chassis can contain a sample and hold, multiplexed A-D converter with sixty-four single-ended or thirty-two differential channels *and* eight D-A converters. In all cases the converters can be any size. The A-D system can be expanded to 256 channels; this maximum system may have sample and hold for each channel separately just prior to conversion, or sample and hold for all channels simultaneously followed by a string of conversions. The multiplexor expander and multichannel sample and hold are housed in chassis separate from the 4055B. The D-A part of the combined system is limited to eight channels, but Data General also has available a separate 24-channel unit, the 4056H, which contains its own power supply and control and is completely independent of the combined system.

A-D conversion requires a 4014 interface subassembly and an A-D interface 4032, with or without a channel scanner 4033; D-A conversion requires a 4036 interface subassembly and a D-A interface 4037. These units are connected to the IO bus and can be installed in the computer chassis. The basic 4032 interface allows the sampling of one channel at a time by IO instructions; the 4033 scanner (which is mounted on the same board as the 4032) allows automatic sampling of all channels sequentially with the digital results going directly to memory via the data channel. For a combined system, the interfaces are connected to the 4055A or B; for the larger D-A configuration, the 4056H is connected directly to a 4037 interface.

A complete system for either or both types of conversion is built up from among the following items.

	<i>Unit</i>	<i>Prerequisite</i>
4055A	Basic enclosure, low capacity	4032 or 4037
4055B	Basic enclosure, high capacity	4032 or 4037
4055C	A-D converter, 8 bits	4055A or B
4055D	A-D converter, 10 bits	4055A or B
4055E	A-D converter, 12 bits	4055A or B
4055F	A-D converter, 13 bits	4055A or B
4055G	A-D converter, 14 bits	4055A or B
4055H	A-D converter, 15 bits	4055A or B
4055I	Buffer amplifier, single ended	Any of 4055C-H
4055J	Buffer amplifier, differential	Any of 4055C-H
4055K	Control for multiplexor and sample and hold	Any of 4055C-H
4055L	Sample and hold	4055K
4055M	Multiplexor, 8 single-ended channels	4055I and K
4055N	Multiplexor, 16 single-ended or 8 differential channels	4055I or J and K
4055O	Enclosure, power supply and control for multiplexor with 128 single-ended or 64 differential channels	4055M or N
4055P	Enclosure, power supply and control for 32 single-ended or 16 differential simultaneous sample and hold channels.	4055M or N
4055Q	Dual simultaneous sample and hold	4055P
4056A	D-A converter control	4055A or B
4056B	D-A converter, 8 bits	4056A or H
4056C	D-A converter, 10 bits	4056A or H
4056D	D-A converter, 12 bits	4056A or H
4056E	D-A converter, 13 bits	4056A or H
4056F	D-A converter, 14 bits	4056A or H
4056H	Enclosure, power supply and control for 24 D-A converters	4037

The standard analog voltage ranges available are  $\pm 2.5$ ,  $\pm 5$ ,  $\pm 10$ , 0 to 5 and 0 to +10 volts, which must be specified at the time of order. The 15 bit converter (A-D) only is available only in the 0 to 10 and  $\pm 10$  volt range. Each D-A channel supplies 10 ma of output current. Accuracy and settling time depend upon the type of converter; for these and other D-A specifications refer to paragraph 6.4.

The A-D equipment can withstand analog input voltages within the range of  $\pm 15$  volts without damage. Warmup time for a 15 or 14-bit converter is 25 minutes; for other converters, 15 minutes. A sample and hold amplifier (with a maximum decay rate of  $15\mu\text{V/ms}$ ) may be used to hold the acquired analog signal constant during the conversion and thus reduce errors due to variations in the applied input voltage. Using the sample and hold, the aperture (measurement time) uncertainty is 50 ns maximum. Both the converter and the single-channel sample and hold have low input impedance. For an application requiring high input impedance, the converter should be preceded by a buffer amplifier, which is required in any multiplexed system in order to maintain high input impedance and ensure proper operation of the multiplexer. Channel to channel crosstalk is 74 db (typical) rejection at 1 KHz on selected channel with 15 adjacent unselected channels connected to 20V p-p sine-wave source. 1 kilohm source resistance on selected channel.

The analog input channels may be single-ended or differential. The maximum number of channels is the same — 256 — but more equipment is required for differential channels. With differential channels the signal plus common mode voltage is  $\pm 10$  volts maximum, and common mode rejection is 86 db typical at 60 Hz and

with 1 kilohm source imbalance. Accuracy, stability and various other characteristics of the system depend upon the type of converter used; complete A-D specifications are given in §6.4. System accuracy is given as a percent error in excess of the inherent resolution error of  $\frac{1}{2}$  LSB. Such inaccuracy can be caused by noise, variation dependent upon temperature, static errors caused by slight inaccuracies in calibration, and dynamic errors caused by the processing of moving waveforms.

The illustrations on the following two pages show the types of A-D configurations. The simplest possible system is a single analog channel fed directly into an A-D converter whose output is available to the interface. If needed, the setup may also include a buffer amplifier and/or a sample and hold (which is optional in any configuration). Even in this case the interface can use the channel scanner, for a time scan of the single input signal.

Any configuration for handling more than one channel requires a multiplexor, whose output is fed to the converter through a buffer amplifier. The main chassis can accommodate sixty-four single-ended channels using six address bits from the interface or thirty-two differential channels using five address bits. Note that a multiplexor of any size is always made up of the basic 16-channel 4055N multiplexors, with at most one 4055M for the eight (or fewer) highest numbered channels. To increase the capacity of the system beyond sixty-four single-ended or thirty-two differential channels requires that all channels input to multiplexors external to the main chassis, and their outputs in turn undergo higher level multiplexing through an 8-channel multiplexor (single-ended or differential) in the main chassis. The high level multiplexing uses the three high order address bits, and the five low order bits are supplied to the expansion enclosures for the low order multiplexing. This allows a maximum of 256 channels, utilizing two 4055O enclosures for single-ended channels, four for differential channels.

The multiplexed configurations may have simultaneous sample and hold. This setup differs in that a single sample and hold at the converter is not used, and each set of thirty-two single-ended or sixteen differential analog inputs is fed to the multiplexer through a 4055P enclosure containing one 4055Q dual simultaneous sample and hold for each pair of single-ended channels or two 4055Qs for each pair of differential channels (where one 4055Q handles the low sides of both channels and the other handles the high sides). Although this configuration requires more equipment, it allows simultaneous sampling on all channels, so that a subsequent string of conversions can provide digital data reflecting the state of all analog channels at the same point in time. It is also faster since there is only one sample and hold for an entire set of conversions, which are then executed at the same rate as in a multiplexed system without sample and hold (the interface must include the 4033 scanner).

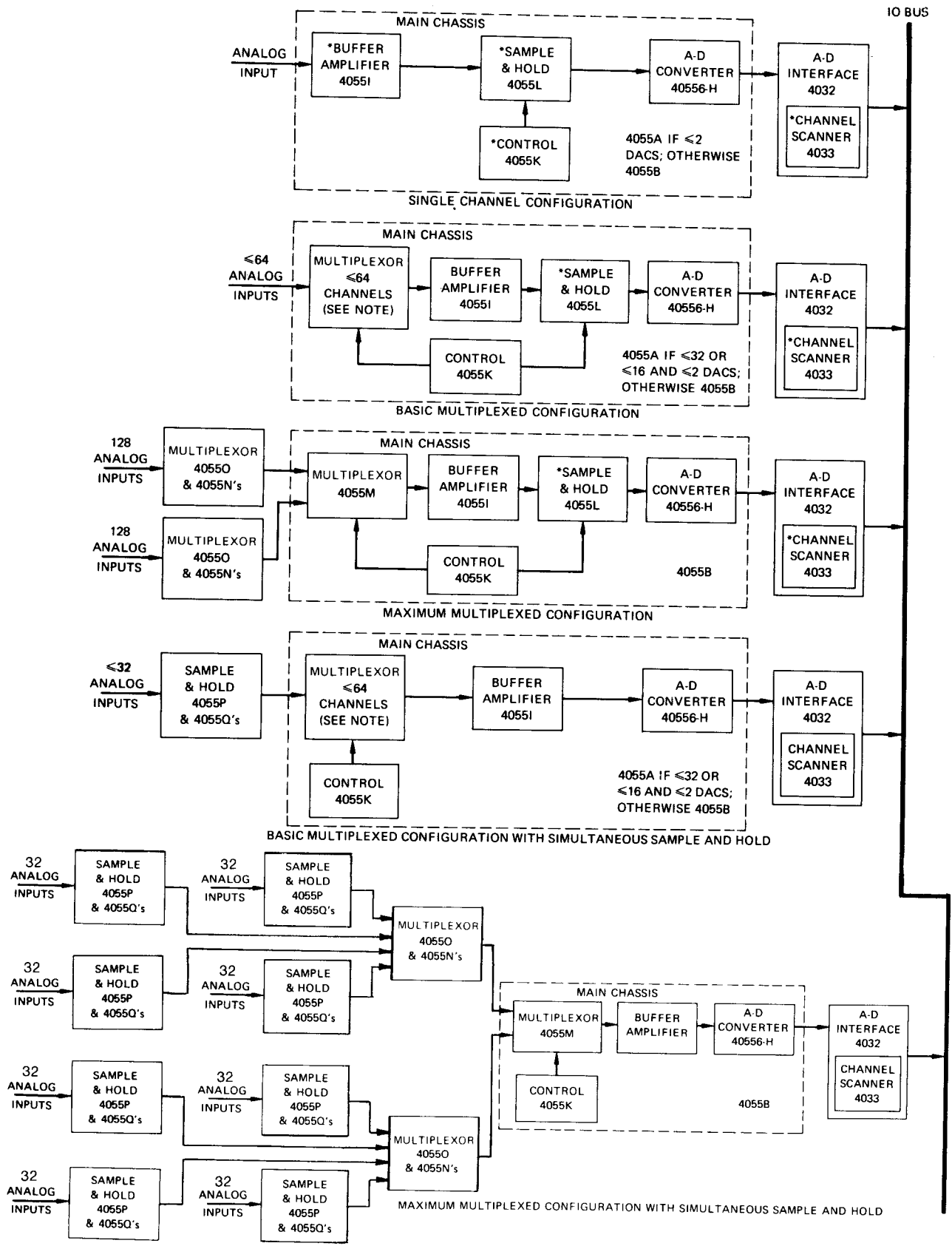
## 6.2 A-D CONVERSION

The 4032 A-D interface without the 4033 channel scanner allows the program to select a single channel, trigger a conversion on it, and read the digital result. To implement this the interface contains an 8-bit channel select register and a 16-bit data buffer that receives the output of the converter whenever a conversion is completed.

The program produces each conversion by giving the Start function, but operation can be synchronized to a clock. Without the clock, each Start triggers a conversion, but with it each Start allows the next clock pulse to trigger a conversion. The user can supply an external clock, but mounted on the interface board is an internal clock that can be set to any pulse interval from 10  $\mu$ s to 100  $\mu$ s.

The basic interface uses three of the IO transfer instructions with device code 21, mnemonic ADCV. Busy and Done are sensed by bits 8 and 9 in IO skip instructions; Clear and Start control these flags in the usual fashion. Interrupt Disable is controlled by interrupt priority mask bit 8. Start ( $F = 01$ ) clears Done, sets Busy, and triggers a conversion or primes the clock to do so.

For a single conversion the program should give a DOAS to specify the channel and trigger the conversion (an NIOS is sufficient if there is only one channel) either directly or at the next clock pulse. At completion, the converter loads the digital result into the data buffer, and sends an end of conversion pulse that clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. The program can then give a DIC to read the buffer into AC.

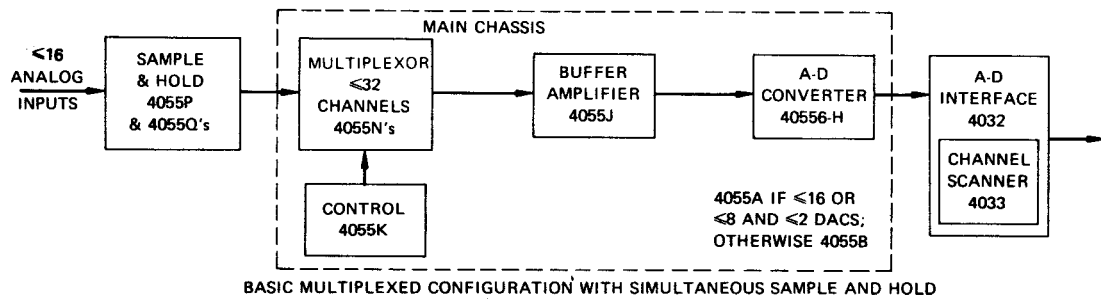
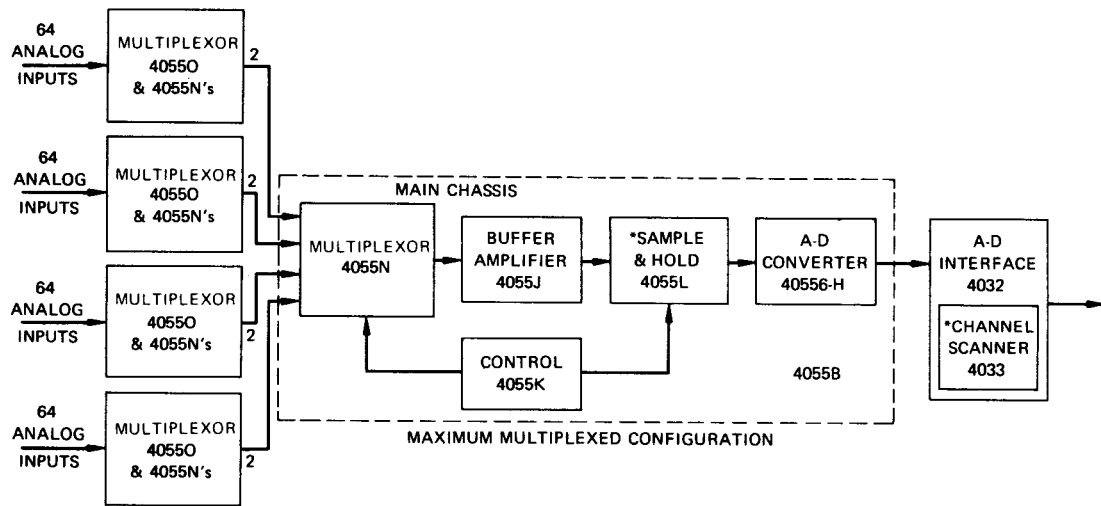
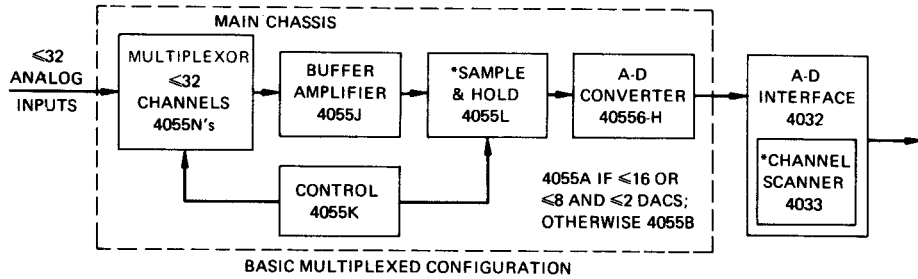
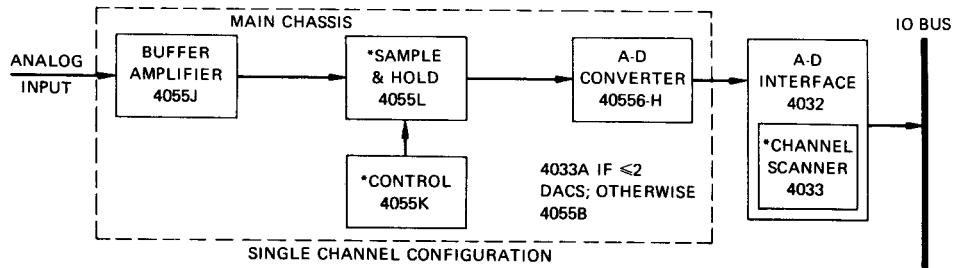


NOTE: 4055M IF  $\leq 8$  CHANNELS; OTHERWISE 4055N's WITH AT MOST ONE 4055M FOR HIGHEST NUMBERED CHANNELS

\*OPTIONAL

### SINGLE-ENDED A-D CONFIGURATIONS

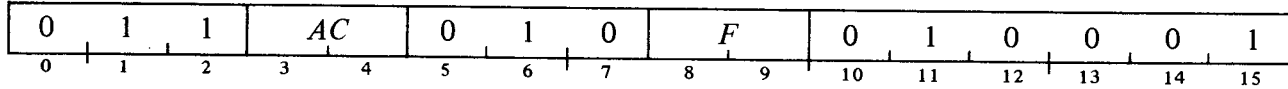




\*OPTIONAL

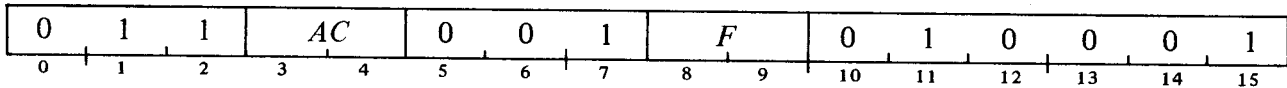
## DIFFERENTIAL A-D CONFIGURATIONS

### DOA —,ADCV Data Out A, A/D Converter



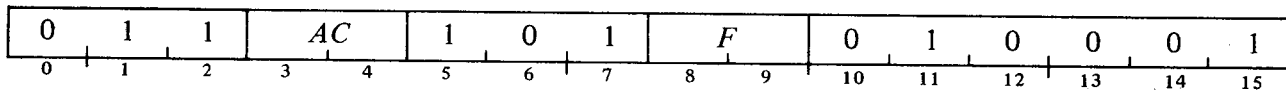
Load the contents of AC bits 8–15 into the channel select register, and perform the function specified by *F*.

### DIA —,ADCV Data In A, A/D Converter



Read the contents of the channel select register into AC bits 8–15, and perform the function specified by *F*. Clear AC bits 0–7.

### DIC —,ADCV Data In C, A/D Converter



Transfer the contents of the data buffer into AC, and perform the function specified by *F*. The digital value, of whatever number of bits supplied by the converter, is right justified in the buffer and the sign is extended to the left.

For a sequence of readings from sequential channels, the program can give a DOAS to select the first channel and start the first conversion, and then respond to each completion with a DICS that reads the converted data, increments the channel select register, and triggers the next conversion.

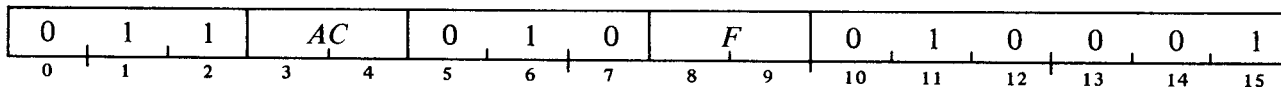
To sample a number of random channels, the program must give both a DIC and a DOAS following each conversion. (A multiplexed system with simultaneous sample and hold cannot be used with the 4032 alone.)

### Interface with Channel Scanner

The combined 4032–4033 interface acts in exactly the manner described above when the Start function is used to convert. But if the special IO Pulse function is used instead, the interface begins automatic operation, synchronized to the clock, in which it samples the channels sequentially and sends the results directly to memory via the data channel. To implement this feature the combined interface has a channel counter (in place of the channel select register), an 8-bit final channel register, a 15-bit address counter and a 12-bit word counter. To run a sequence of conversions automatically, the program must specify the number of the starting channel to the channel counter, the number of the final channel in the sequence to the final channel register, an initial address to the address counter for data channel access, and the (twos complement) negative of a word count.

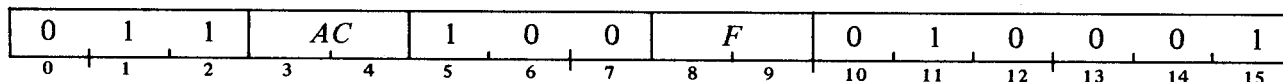
The combined interface uses all six IO transfer instructions, but only five are relevant to the scanning procedure (DIC is only for reading the result of a single conversion using Start). Both Start and Pulse (*F* = 11) clear Done, set Busy and start a conversion, but Pulse places the system in the scanning mode.

**DOA  $\bar{}$ ,ADCV Data Out A, A/D Converter**



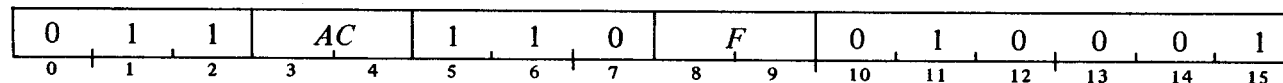
Load the contents of AC bits 0-7 into the final channel register and the contents of AC bits 8-15 into the channel counter. Perform the function specified by *F*.

**DOB  $\bar{}$ ,ADCV Data Out B, A/D Converter**



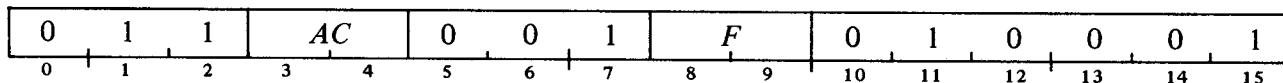
Load the contents of AC bits 1-15 into the address counter, and perform the function specified by *F*.

**DOC  $\bar{}$ ,ADCV Data Out C, A/D Converter**



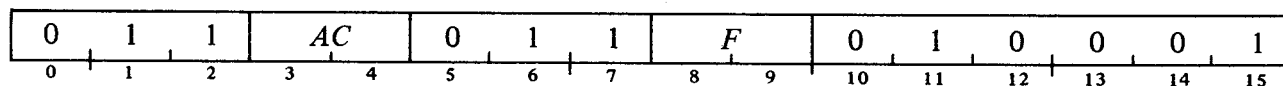
Load the contents of AC bits 4-15 into the word counter, and perform the function specified by *F*.

**DIA  $\bar{}$ ,ADCV Data In A, A/D Converter**



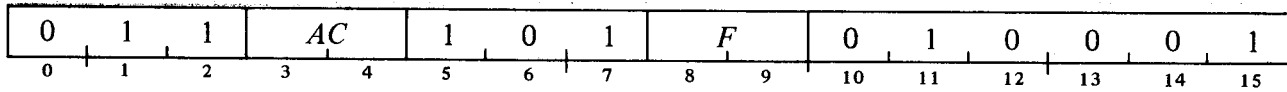
Read the contents of the final channel register into AC bits 0-7 and the channel counter into AC bits 8-15. Perform the function specified by *F*.

**DIB  $\bar{}$ ,ADCV Data In B, A/D Converter**



Read the contents of the address counter into AC bits 1-15, and perform the function specified by *F*. Clear AC bit 0.

## DIC —,ADCV      Data In C, A/D Converter



Read the contents of the data buffer into AC, and perform the function specified by *F*. The digital value, of whatever number of bits supplied by the converter, is right justified in the buffer and the sign is extended to the left.

When Busy is set by the Pulse function, the interface begins the scanning procedure synchronized to the clock. The first conversion is on the channel selected by the DOA, *ie* the channel initially selected by the channel counter. Following each conversion, the result is sent to the location currently addressed by the address counter, and the address, word and channel counters are all incremented by one. When a conversion occurs on the channel specified by the final channel register, the address and word counters are incremented, but the channel counter is reset to zero to begin a new cycle. Cycles from channel 0 to the final channel are repeated until the word counter overflows, which clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. If the system is multichannel with simultaneous sample and hold, Pulse triggers the sample and hold and the word count must limit the scan to one cycle.

The combined interface can be used like the basic interface simply by using DOA and DIC and giving Start. The conversion is on the channel selected by the DOA, *ie* the channel currently selected by the channel counter. However when the DOA is given, AC bits 0–7 should be *all 1s* to ensure that the number of the selected channel cannot be greater than the number specified for the final channel (this would inhibit the conversion).

### Wiring Considerations

To select a synchronizing clock for the interface, insert jumper J1 (located in the vicinity of ICs E44–46 on the circuit board) and input the clock pulse train at connector pin A63. To use the internal clock, jumper A47–A63; the clock frequency is selected by installing an appropriate capacitor and adjusting the trimpot on the clock package located at E43.

Differential analog inputs should be shielded twisted pairs, but if noise is serious, single-ended inputs should be shielded as well. All shields should be terminated at the source ground rather than the data acquisition ground. This is especially important for differential systems, as the difference between source ground and data acquisition ground is considered to be the source of the common mode. The twisted-pair shields should be connected to this common mode source to reduce the effects of cable capacitance, which can in turn affect the CMR characteristics.

Data acquisition signal return ground is available at both analog connectors (J1–A17, B17 and J2–A6, B6) as well as at an insulated stud located on the rear panel. A solid ground wire should be connected between the signal return ground and the signal source ground or grounds in all systems. Without this ground, common mode voltages are undefined and could exceed the  $\pm 10$  volt maximum.

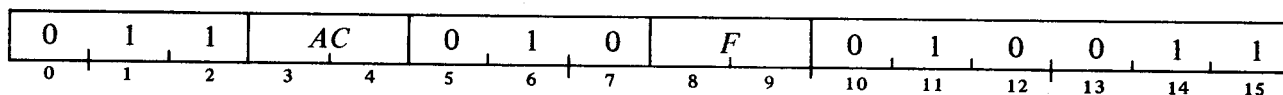
Any unused analog inputs for which multiplexer channels are installed should be grounded to minimize **stray pickup**.

When expansion chassis are used, the five low-order channel address bits must be wired to them, and the multiplexer outputs should be connected to the pins for channels 0–7 at the back of the main chassis (all signals are on connector J2).

### 6.3 D-A CONVERSION

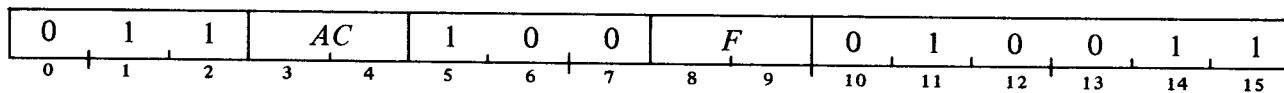
The 4037 D-A interface contains no flags and uses no control functions of the IO instructions. The interface uses only two IO output instructions, one to select the converter (channel), the other to supply the data for conversion. Each converter contains its own data buffer; conversion begins as soon as the buffer contents change, and the analog output is held constant until new digital data is supplied to the buffer. The interface has an 8-bit channel selection register, but only the low order three bits are used with a 4055A or 4055B, and only the low order five are used with a 4056H (in the latter case, specifying a channel number greater than 27 (decimal 23) selects no channel). The device code is 23, mnemonic DACV (which is also used for the 4053 oscilloscope control).

#### DOA -DACV Data Out A, D/A Converter



Convert the right  $n$  bits of *AC* into an analog voltage on the selected channel, where  $n$  is the number of bits resolution of the converter.

#### DOB -DACV Data Out B, D/A Converter



Select the D-A channel addressed by *AC* bits 13-15 (4055A or 4055B) or bits 11-15 (4056H).

D-A outputs are on connector J1. All D-A returns should be connected to the D-A return grounds at J1-A20, B20 or the insulated stud on the rear panel. To minimize interaction between a multiplexed A-D and the D-A, the analog outputs should be bundled separately and perhaps shielded from the analog inputs.

### 6.4 SPECIFICATIONS

For a converter with  $n$  bits resolution, the bipolar analog signal is converted to or derived from signed numbers with  $n - 1$  magnitude bits using twos complement convention for negatives; the unipolar analog range corresponds to the set of positive  $n$ -bit numbers. There are  $2^n$  numbers containing  $n$  bits. This set of numbers defines only  $2^n - 1$  intervals, but the analog range is divided into  $2^n$  increments, where each corresponds to an LSB of the digital resolution. The minimum analog value corresponds to the minimum digital value, but the actual maximum analog voltage is one LSB less than the nominal maximum (LSB used in this sense means the voltage increment corresponding to the digital LSB). Eg for a 12-bit converter with a  $\pm 5$  volt range,  $LSB = 10 \text{ volts} / 4096 = 2.4 \text{ mv}$ . Hence the plus full scale voltage (+FS) is  $5.000 - .0024 = 4.9976 \text{ volts}$ . (-FS is  $-5.000 \text{ volts}$ ). The term "full scale" (FS) usually means the nominal 5 volts, "full scale range" (FSR) refers to the 10 volts.)

The following table gives the minimum, mid range and maximum analog and digital values for all converter resolutions for both the unipolar case and the bipolar case where  $V = 5.000$  or  $10.000$ . The digital values are given as the 16-bit words received from the A-D; on D-A the unused bits at the left are ignored and may have any value.

Analog	15 Bits	14 Bits	13 Bits	12 Bits	10 Bits	8 Bits
$+V - \text{LSB}$	037777	017777	007777	003777	000777	000177
0.000	000000	000000	000000	000000	000000	000000
$-V$	140000	160000	170000	174000	177000	177600
$+10.000 - \text{LSB}$		037777	017777	007777	001777	000377
$+5.000$		020000	010000	004000	001000	000200
0.000		000000	000000	000000	000000	000000

Any analog input signal greater than  $+FS$  converts to the maximum digital value; any less than  $-FS$  converts to the minimum.

The outputs of the D-A are actually  $2^n$  discrete voltage levels corresponding to the digital values, whereas the A-D input is a continuum throughout the input voltage range. Hence except for errors introduced into the D-A by actual circuit characteristics, noise, etc., the analog value corresponding to a particular number is exact, whereas there is an inherent limitation on the accuracy of the A-D equal to  $\pm \frac{1}{2} \text{LSB}$ , i.e. the actual input can be anywhere within one half LSB of the value specified for the digital output. However in spite of appearances, the same error is inherent in the D-A but does not appear in the converter — it occurs in the computer when the program rounds or truncates the digital quantities to  $n$  bits.

### A-D Operating Specifications

Channels	256 single-ended or differential
Resolution	8, 10, 12, 13, 14 or 15 bits
Warmup time	
14, 15 bits	25 minutes
8, 10, 12, 13 bits	15 minutes
Input voltage range	
Standard, except 15 bits	$\pm 5V, 0$ to $5V$ $\pm 10V, 0$ to $10V$
15 bits	$\pm 10V, 0$ to $10V$
Differential input voltage	$\pm 10V$ maximum, signal plus common mode
Input overvoltage range	$\pm 15V$ maximum without damage
Sample and hold	
Aperture uncertainty	50 ns maximum
Settling time	
Single channel	$5 \mu s$
Simultaneous	$10 \mu s$ ( $7 \mu s$ typical)

Input impedance	
Converter	≥1250 ohms
Sample and hold	
Single channel	2000 pf input capacitance in series with 100 ohms (switch)
Simultaneous	10,000 megohms in parallel with 10 pf
Buffer amplifier	10,000 megohms
Multiplexor channel	2000 megohms
Channel-to-channel crosstalk	74 db (typical) rejection at 1 KHz on selected channel with 15 adjacent unselected channels connected to 20V p-p sinewave source.
Maximum source resistance on selected channel	1 kilohm
Channel-to-channel short circuit protection	2 kilohms
Differential common mode rejection	86 db typical at 60 Hz and 66 db typical at 1 KHz with 1 kilohm source imbalance
Line voltage	117/234 vac ±10%, 47-420 Hz; 3AG 1 amp fuse
Dissipation	
4055A	20 watts
4055B	40 watts
40550	25 watts
4055P	50 watts

*A-D Converter Alone*

	<i>15 Bits</i>	<i>14 Bits</i>	<i>13 Bits</i>	<i>12 Bits</i>	<i>10 Bits</i>	<i>8 Bits</i>
Conversion time in $\mu$ s	16.5	84	78	24	10	8
Conversions per second	60.6K	11.9K	12.8K	41K	100K	125K
Accuracy: % FSR $\pm$ 1/2 LSB @ 20°C	±.01	±.01	±.01	±.02	±.05	±.1
Offset stability: % FSR						
24 hours	±.001	±.005	±.005	±.005	±.005	±.005
30 days	±.002	±.01	±.01	±.01	±.01	±.01
Temperature coefficients						
Offset: % FSR/°C	±.001	±.0015	±.0015	±.0015	±.0015	±.0015
Gain: ppm FSR/°C	±7	±10	±10	±10	±10	±10
Noise (3 $\sigma$ ): % FSR	.001	.005	.005	.005	.01	.01

*Multiplexed A-D Converter with Sample and Hold*

	<i>15 Bits</i>	<i>14 Bits</i>	<i>13 Bits</i>	<i>12 Bits</i>	<i>10 Bits</i>	<i>8 Bits</i>
Minimum conversion time in $\mu$ s	21.5	89	83	29	15	13
Maximum A-D conversions per second*	40K	10.5K	11.5K	32K	43K	62K
Accuracy: % FSR $\pm$ $\frac{1}{2}$ LSB @ 20°C	$\pm .02$	$\pm .02$	$\pm .02$	$\pm .025$	$\pm .06$	$\pm .1$
Linearity: % FSR	.006	.01	.01	.02	.05	.1
Offset stability: % FSR						
24 hours	$\pm .003$	$\pm .006$	$\pm .006$	$\pm .006$	$\pm .006$	$\pm .006$
6 months	$\pm .01$	$\pm .012$	$\pm .012$	$\pm .012$	$\pm .012$	$\pm .012$
Temperature coefficients						
Offset: % FSR/°C						
Single-ended	$\pm .001$	$\pm .0015$	$\pm .0015$	$\pm .0015$	$\pm .0015$	$\pm .0015$
Differential	$\pm .0012$	$\pm .0017$	$\pm .0017$	$\pm .0017$	$\pm .0017$	$\pm .0017$
Gain: ppm FSR/°C						
Single-ended	$\pm 10$	$\pm 13$	$\pm 13$	$\pm 13$	$\pm 13$	$\pm 13$
Differential	$\pm 12$	$\pm 15$	$\pm 15$	$\pm 15$	$\pm 15$	$\pm 15$

\*Every A-D converter has a potentiometer for adjusting the bit rate to trade off linearity vs speed. The conversion rates given satisfy the listed linearity specification.

**D-A Operating Specifications**

Channels	8 with A-D, 24 in 4056H
Resolution	8, 10, 12, 13 or 14 bits
Output voltage range	$\pm 5$ V, 0 to 5V $\pm 10$ V, 0 to 10V
Output current	10 ma
Settling time to $\frac{1}{2}$ LSB	
8, 13, 14 bits	5 $\mu$ s
10 bits	7 $\mu$ s
12 bits	10 $\mu$ s
Accuracy	
8 bits	$\pm .1\%$ FSR
10 bits	$\pm .04\%$ FSR
12 bits	$\pm .02\%$ FSR
13, 14 bits	$\pm .01\%$ FSR



<b>Offset stability</b>	
24 hours	$\pm .002\%$ FSR
30 days	$\pm .01\%$ FSR
<b>Temperature coefficients</b>	
<b>Offset</b>	
8, 10 bits	$.002\%$ FSR/ $^{\circ}$ C
12, 13, 14 bits	$.001\%$ FSR/ $^{\circ}$ C
<b>Range</b>	
8 bits	40 ppm/ $^{\circ}$ C
10 bits	17 ppm/ $^{\circ}$ C
12, 13, 14 bits	10 ppm/ $^{\circ}$ C
Recalibration interval	Six months recommended
Line voltage	117/234 vac $\pm 10\%$ , 47-420 Hz; 3AG 1 amp fuse
<b>Dissipation</b>	
4055A	20 watts (8 D-A converters)
4055B	40 watts
4056H	55 watts

### Physical and Environmental Specifications

All units are  $3\frac{1}{2}'' \times 17'' \times 17''$  and mount on slides in a standard 19'' rack.

Weight: 4055A, 16 pounds; all other units, 20 pounds

Operating temperature 0 to  $55^{\circ}$ C

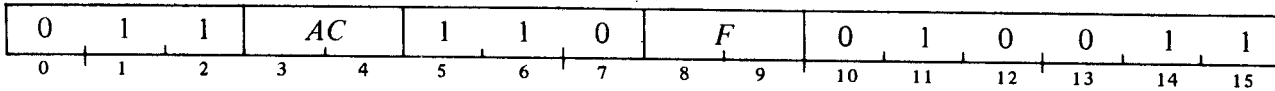
Storage temperature  $-25$  to  $80^{\circ}$ C

### 6.5 OSCILLOSCOPE CONTROL 4053

This interface allows the program to display information by plotting points on any typical storing or nonstoring oscilloscope. The control not only requires a dual D-A converter but is mounted on the converter board and shares its device code. To display each point, the computer must supply two words for the  $x$  and  $y$  coordinates to the converter and give the signal to intensify the beam. The program can also erase all the information that has been stored on the scope face, select nonstorage mode in a storage scope, and display information that is not stored but does not affect information previously stored. Timing and signal characteristics can be adjusted to satisfy the requirements of the scope.

The scope control uses two IO transfer instructions, one to select the scope operating mode, the other to read a single status bit. The control has no busy and done flags or interrupt capability, but Start in any IO instruction with device code 23, mnemonic DACV, intensifies the scope beam. Hence the program need not give a separate instruction to start the scope — the same instruction that supplies the second coordinate to the converter can also intensify the beam. Programming the IO Pulse function ( $F = 11$ ) erases the scope (erasing can also be done by means of a switch at the scope).

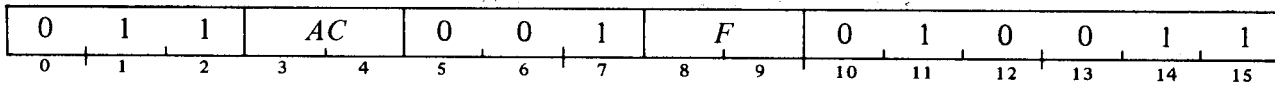
**DOC - ,DACV      Data Out C, Scope Control**



Select the scope mode according to AC bits 14 and 15, and perform the function specified by F. The meaning of the mode bits is as follows.

<i>Bits 14—15</i>	<i>Meaning</i>
00	Standard operation—store or nonstore depending on scope
01	Write through—points may be displayed without storing but without affecting previously stored information
10	Nonstore mode
11	This combination gives conflicting mode information

**DIA - ,DACV      Data in A, Scope Control**



Read the erase status into AC bit 15 and clear AC bits 0-14. A 1 in AC bit 15 indicates the scope is presently erasing (if the jumper W2 is installed.) (Perform the function specified by F.) Erase time is typically in the range one-tenth to one-half second.

**Scope Parameters.** To display each point the program must supply two coordinates and a Start pulse to intensify the beam. Both the time from Start to the beginning of the intensification signal and the duration of that signal can be adjusted to a value in the range 1.4 to 6  $\mu$ s by means of screwdriver pots located beside packages U37 and U38 respectively on the converter board. The user can substitute other time ranges by changing the resistance and/or capacitance associated with the pots. The polarity of the intensification signal is controlled by placing an appropriate jumper at the outputs of U38: with a jumper in position W3 the output of the intensification circuit is normally at a low level and becomes high during the time period defined by the pot; installing a jumper at W4 reverses this polarity. The circuit is set up for high and low levels of +5 volts and ground. The high level can be changed to +15 volts by deleting diode CR2, and the low level can be dropped to -5 volts by cutting out CR1. The output is ac coupled, but dc coupling can be substituted by installing a jumper to bypass the capacitors connected to pin B48.

## Connectors

### J1 44-pin Viking 2VH22/IJN3

A-D channel numbers are given for single-ended and low and high differential (numbers in parentheses are decimal).

A1	A-D 40, 20H (32, 16H)	B1	A-D 50, 20L (40, 16L)
A2	A-D 41, 21H (33, 17H)	B2	A-D 51, 21L (41, 17L)
A3	A-D 42, 22H (34, 18H)	B3	A-D 52, 22L (42, 18L)
A4	A-D 43, 23H (35, 19H)	B4	A-D 53, 23L (43, 19L)
A5	A-D 44, 24H (36, 20H)	B5	A-D 54, 24L (44, 20L)
A6	A-D 45, 25H (37, 21H)	B6	A-D 55, 25L (45, 21L)
A7	A-D 46, 26H (38, 22H)	B7	A-D 56, 26L (46, 22L)
A8	A-D 47, 27H (39, 23H)	B8	A-D 57, 27L (47, 23L)
A9	A-D 60, 30H (48, 24H)	B9	A-D 70, 30L (56, 24L)
A10	A-D 61, 31H (49, 25H)	B10	A-D 71, 31L (57, 25L)
A11	A-D 62, 32H (50, 26H)	B11	A-D 72, 32L (58, 26L)
A12	A-D 63, 33H (51, 27H)	B12	A-D 73, 33L (59, 27L)
A13	A-D 64, 34H (52, 28H)	B13	A-D 74, 34L (60, 28L)
A14	A-D 65, 35H (53, 29H)	B14	A-D 75, 35L (61, 29L)
A15	A-D 66, 36H (54, 30H)	B15	A-D 76, 36L (62, 30L)
A16	A-D 67, 37H (55, 31H)	B16	A-D 77, 37L (63, 31L)
A17	A-D signal return	B17	A-D signal return
A18	D-A 1	B18	D-A 0
A19	D-A 3	B19	D-A 2
A20	D-A return	B20	D-A return
A21	D-A 5	B21	D-A 4
A22	D-A 7	B22	D-A 6

### J2 44-pin Viking 2VH22/IJN3

A-D channel numbers are given for single-ended and low and high differential (numbers in parentheses are decimal).

A1	A-D address bit 15	B1	
A2	A-D address bit 14	B2	
A3	A-D address bit 13	B3	
A4	A-D address bit 12	B4	
A5	A-D address bit 11	B5	Simultaneous sample & hold
A6	A-D signal return	B6	A-D signal return
A7	A-D 0, 0H (0, 0H)	B7	A-D 10, 0L (8, 0L)
A8	A-D 1, 1H (1, 1H)	B8	A-D 11, 1L (9, 1L)
A9	A-D 2, 2H (2, 2H)	B9	A-D 12, 2L (10, 2L)
A10	A-D 3, 3H (3, 3H)	B10	A-D 13, 3L (11, 3L)
A11	A-D 4, 4H (4, 4H)	B11	A-D 14, 4L (12, 4L)
A12	A-D 5, 5H (5, 5H)	B12	A-D 15, 5L (13, 5L)
A13	A-D 6, 6H (6, 6H)	B13	A-D 16, 6L (14, 6L)
A14	A-D 7, 7H (7, 7H)	B14	A-D 17, 7L (15, 7L)

A15 A-D 20, 10H (16, 8H)  
 A16 A-D 21, 11H (17, 9H)  
 A17 A-D 22, 12H (18, 10H)  
 A18 A-D 23, 13H (19, 11H)  
 A19 A-D 24, 14H (20, 12H)  
 A20 A-D 25, 15H (21, 13H)  
 A21 A-D 26, 16H (22, 14H)  
 A22 A-D 27, 17H (23, 15H)

B15 A-D 30, 10L (24, 8L)  
 B16 A-D 31, 11L (25, 9L)  
 B17 A-D 32, 12L (26, 10L)  
 B18 A-D 33, 13L (27, 11L)  
 B19 A-D 34, 14L (28, 12L)  
 B20 A-D 35, 15L (29, 13L)  
 B21 A-D 36, 16L (30, 14L)  
 B22 A-D 37, 17L (31, 15L)

Signals on A1-A5 are for multiplexer expansion.

### J3 60-pin DGC#111000172

A1	A-D data bit 0	B1	D-A data bit 0
A2	A-D data bit 1	B2	D-A data bit 1
A3	A-D data bit 2	B3	D-A data bit 2
A4	A-D data bit 3	B4	D-A data bit 3
A5	A-D data bit 4	B5	D-A data bit 4
A6	A-D data bit 5	B6	D-A data bit 5
A7	A-D data bit 6	B7	D-A data bit 6
A8	A-D data bit 7	B8	D-A data bit 7
A9	A-D data bit 8	B9	D-A data bit 8
A10	A-D data bit 9	B10	D-A data bit 9
A11	A-D data bit 10	B11	D-A data bit 10
A12	A-D data bit 11	B12	D-A data bit 11
A13	A-D data bit 12	B13	D-A data bit 12
A14	A-D data bit 13	B14	D-A data bit 13
A15	A-D data bit 14	B15	D-A address bit 13
A16	A-D data bit 15	B16	D-A address bit 14
A17	End of conversion	B17	D-A address bit 15
A18	Convert	B18	External/internal
A19	A-D address transfer	B19	D-A address transfer
A20		B20	[Multiplexer sequence advance]
A21		B21	[Delayed trigger]
A22		B22	A-D address bit 15 (10)
A23		B23	A-D address bit 14 (9)
A24		B24	A-D address bit 13 (8)
A25	Simultaneous sample & hold	B25	A-D address bit 12
A26	A-D address bit 11	B26	A-D address bit 11
A27	A-D address bit 12	B27	A-D address bit 10
A28	A-D address bit 13	B28	[A-D serial data]
A29	A-D address bit 14	B29	[A-D clock out]
A30	A-D address bit 15	B30	Digital ground

Brackets indicate unused signals. Signals on A26-A30 are for multiplexer expansion; when the multiplexer is expanded, B22-B24 carry the address bits given in parentheses.

J4 60-pin DGC#111000172

A1	B1
A2	B2
A3	B3
A4	B4
A5	B5
A6	B6
A7	B7
A8	B8
A9	B9 Digital ground
A10	B10 D-A address transfer
A11	B11 External/internal
A12	B12 D-A address bit 13
A13	B13 D-A address bit 14
A14	B14 D-A address bit 15
A15	B15 D-A data bit 15
A16	B16 D-A data bit 14
A17	B17 D-A data bit 13
A18	B18 D-A data bit 12
A19	B19 D-A data bit 11
A20	B20 D-A data bit 10
A21	B21 D-A data bit 9
A22	B22 D-A data bit 8
A23	B23 D-A data bit 7
A24	B24 D-A data bit 6
A25	B25 D-A data bit 5
A26	B26 D-A data bit 4
A27	B27 D-A data bit 3
A28	B28 D-A data bit 2
A29	B29 D-A data bit 1
A30	B30 D-A data bit 0

# Chapter VII

## Data Communications

The devices in this category are for transferring data between the computer and a remote station or another computer, as against production of hardcopy locally or storage of information at the periphery of the computer for later retrieval.

### 7.1 SYNCHRONOUS COMMUNICATIONS CONTROLLER 4015 WITH CLOCK OPTION 4020 AND PARITY OPTION 4021

This controller provides complete bidirectional interfacing between a Nova line computer and a Bell 201, Bell 301 or equivalent synchronous data set at speeds up to 50,000 bits per second. Although mounted on a single circuit board, the controller is actually two independent interfaces, allowing full duplex operation (simultaneous reception and transmission of data). Each interface is connected separately to the data channel, so the program need only set up an interface for receiving or sending and all transfers to and from memory are then handled automatically. To operate with the data channel, each interface has an address counter and a word counter as well as a data shift register for handling serial character transfers. The controller also contains equipment for automatic answering of incoming calls. Device codes for the receiver and transmitter are 40 and 41 respectively. Additional controllers connected use device code pairs 42-43, 44-45, . . . , 74-75, where in each case the receiver uses the even code, the transmitter the odd code.

The controller is available in a number of configurations. Characters may contain six, seven or eight data bits; parity option 4021 enables the transmitter to generate and send a parity bit with each character (thus allowing transmission of characters as long as nine bits including parity) and enables the receiver to check parity. The various characteristics are all selectable separately for the two interfaces by means of jumpers on the board. Transmission and reception can be timed by a clock in the local data set or by an internal clock in the controller (option 4020) for use with an externally clocked modem or a data link that is operated without a modem.

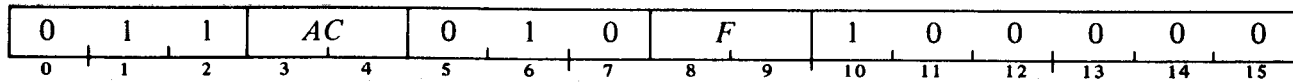
All transfers between controller and memory are in full words containing two characters right-justified in each half word; *eg* 6-bit characters would be in bits 2-7 and 10-15 of a memory word. The transmitter takes two characters from the appropriate bits of each word from memory and transmits them, first the right and then the left. The receiver assembles each pair of characters into the appropriate bits of a word, right to left, for storage in memory. Characters are transmitted and received serially with the least significant bit first (*ie* bit 15 and bit 7).

#### Receiver

To set up the receiver to handle incoming data, the program must specify a sync character, supply an initial address to the 15-bit address counter, and either supply a specific (twos complement) negative word count to the 12-bit word counter or specify an end-of-transmission (EOT) character and a word count large enough (*eg* zero) to receive the entire message.

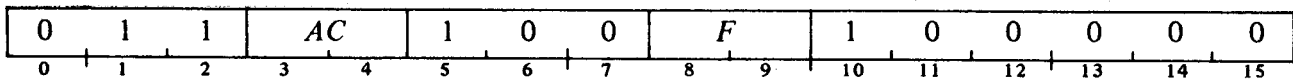
The receiver uses five IO transfer instructions, one of which includes the status bits for the automatic answering feature. The instructions are given here with device code 40 although a receiver can be set up with any even device code 40 or above. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions and are controlled in the usual fashion by Clear and Start. Interrupt Disable is controlled by interrupt priority mask bit 8. For convenience, the mnemonic REC is used in representing the instructions, but it is not recognized by the assembler; the programmer must define his own mnemonics.

**DOA -,REC      Data Out A, Receiver**



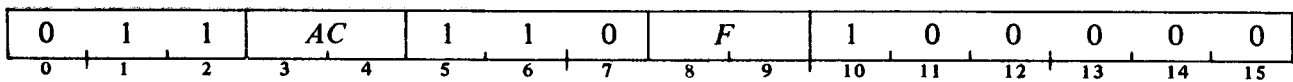
Define the sync character as equal to the contents of AC bits 0-7 and the EOT character as equal to the contents of AC bits 8-15. Perform the function specified by *F*.

**DOB -,REC      Data Out B, Receiver**



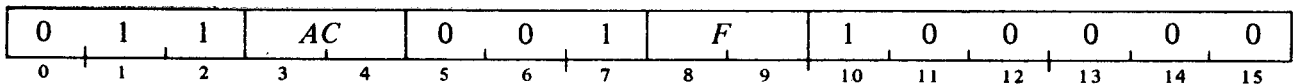
Load the contents of AC bits 1-15 into the receiver address counter, and perform the function specified by *F*.

**DOC -,REC      Data Out C, Receiver**

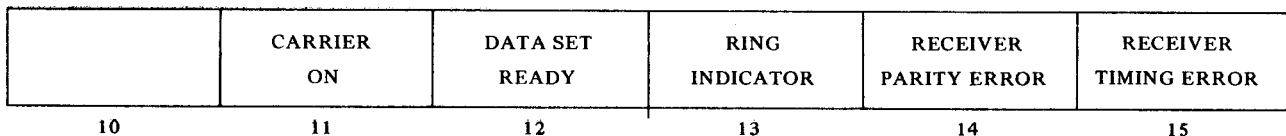


Load the contents of AC bits 4-15 into the receiver word counter, and perform the function specified by *F*.

**DIA -,REC      Data In A, Receiver**



Read the status of the receiver AC bits 11-15 as shown, and perform the function specified by *F*. Clear AC bits 0-10.



Bits 11-13 are for the automatic answering feature described at the end of this section.

- 11 A carrier is being received from a remote station.
- 12 The local data set is connected and is capable of handling data.
- 13 A ringing signal is being received from a remote station.
- 14 The parity option is installed and a character with incorrect parity has been received.
- 15 The data channel has failed to respond in time to a request for access by the receiver and incoming data has been lost.

**DIB -REC      Data In B, Receiver**

0	1	1	<i>AC</i>	0	1	1	<i>F</i>	1	0	0	0	0	0	0	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Read the present contents of the receiver address counter into AC bits 1-15, clear AC bit 0, and perform the function specified by *F*.

Setting Busy causes the receiver to monitor the incoming bit stream continuously until it successively receives two of the sync characters defined by the program. This synchronizes the receiver to the bit stream. It then ignores additional sync characters until some other character is received, at which time it begins assembling pairs of characters into words for transmission to the memory locations specified by the address counter. Since reception is serial the data channel has one-half bit time in which to respond to a request before information is lost; if the channel is late, Timing Error is set but reception continues. If the receiver is so configured, Parity Error sets if a character with incorrect parity is received.

When the EOT character defined by the program appears in the input, the receiver accepts one more character, stores the final word (one or two characters) in memory, and terminates reception. In a message containing an odd number of characters, the final word has the last character on the right (the left half is undefined). If the termination character does not appear, reception ends automatically when the word counter overflows. In either case, at termination the receiver clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear.

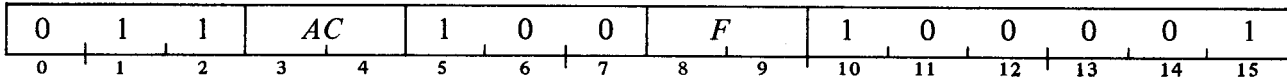
### Transmitter

To set up the transmitter to send data, the program must supply an initial address to the 15-bit address counter and a (two's complement) negative word count to the 12-bit word counter.

The transmitter uses four IO transfer instructions, one of which reads a single status bit. The instructions are given here with device code 41 although a transmitter can be set up with any odd device code 41 or above. Busy and Done are sensed by bits 8 and 9 in the IO skip instructions and are controlled in the usual fashion by Clear and Start. Interrupt Disable is controlled by interrupt priority mask bit 8. For convenience, the mnemonic XMT is used in representing the instructions, but it is not recognized by the assembler; the programmer must define his own mnemonics.

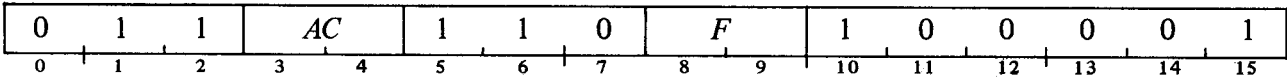


**DOB -,XMT      Data Out B, Transmitter**



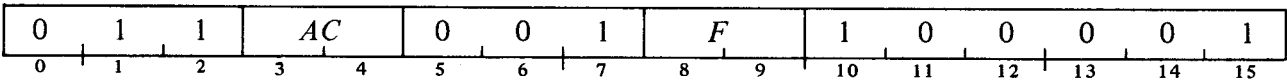
Load the contents of AC bits 1-15 into the transmitter address counter, and perform the function specified by *F*.

**DOC -,XMT      Data Out C, Transmitter**



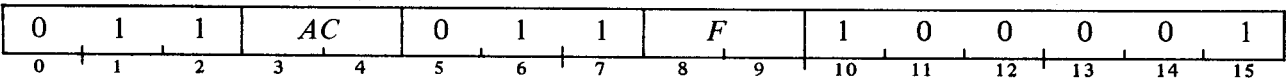
Load the contents of AC bits 4-15 into the transmitter word counter, and perform the function specified by *F*.

**DIA -,XMT      Data In A, Transmitter**



Read the Data Late status into AC bit 15, clear AC bits 0-14, and perform the function specified by *F*. A 1 read into AC bit 15 indicates that the data channel has failed to respond in time to a request for access, and sync has been lost.

**DIB -,XMT      Data In B, Transmitter**



Read the present contents of the transmitter address counter into AC bits 1-15, clear AC bit 0, and perform the function specified by *F*.

Setting Busy causes the transmitter to request data channel access for the first word and raise the Request to Send signal. When the local data set returns the Clear to Send signal, the transmitter begins sending the pairs of characters taken from the memory locations specified by the address counter. Since transmission is serial, the data channel has one bit time in which to respond to a request before sync is lost; if the channel is late, Date Late sets and a garbled character pair may be sent, but transmission continues to complete the block.

If all is well, the word counter overflows as the last word is received from the channel; overflow clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear, even though the transmitter has one more word to send. This provides two character times for the program to supply a new initial address and word count and restart the transmitter without losing sync. If the transmitter is not restarted within this time, the Request to Send signal to the data set is dropped, and sync must be reestablished before further data transmission can take place.

### Automatic Answering

The controller includes equipment that allows the computer to answer incoming calls if the local data set is so configured and operates with EIA standard levels. For this the program makes use of bits 11–13 of the status word read by the DIA for the receiver. Bits 11 and 12 give the status of the communications circuit and the local data set: bit 11 indicates that a carrier is being received from the remote station; bit 12 indicates that the local data set is connected and is capable of handling data. The program detects a ringing signal from a remote station by periodically examining bit 13, the Ring Indicator. The program answers a call by sending a Data Terminal Ready signal to the local data set; the program must also dismiss the call when completed. The program answers and dismisses a call with the following instruction, which uses the transmitter device code.

**DOA -XMT      Data Out A, Transmitter**

0	1	1	AC		0	1	0	F		1	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

If AC bit 15 is 1, send a Data Terminal Ready signal; if AC bit 15 is 0, terminate the Data Terminal Ready signal. (Perform the function specified by *F*.)

## 7.2 ASYNCHRONOUS DATA COMMUNICATIONS MULTIPLEXOR 4026

By means of this device the program can control the transmission of asynchronous serial data on sixteen output lines and can receive asynchronous serial data simultaneously over sixteen input lines. The communication frequency is determined by a clock, but the customer can select several clocks to operate the lines at different speeds up to a maximum of 600 baud. For each transmission line the software not only handles the actual movement of serial data, but selects the character length and selects the speed from among those available. Although the speed of a given input line must be known, the software can recognize the character length while sampling the data. The entire operation is under software control with a minimum of hardware (the device with all line interface modules is contained on a single board). Hence the allocation of a small percentage of processor time results in considerable saving in peripheral hardware. However the standard handler supplied by DGC operates at a single line frequency and character length.

The customer can select the number of communication channels in multiples of four, input and output together, and for each four-line group can select the signal type: line interface module 4027 handles EIA standard levels for a Teletype Model 37 or a Bell 103 Data Set, and line interface module 4028 handles 20 ma signals for a Model 33 or 35 (located within 100 feet).

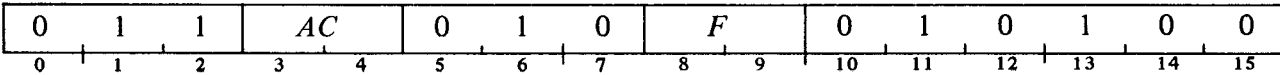
The program handles output by periodically changing the contents of a 16-bit output register in which each bit is connected to a separate output channel; successive changes in the register contents produce bit-by-bit serial transmission over the channels. Data is received simply by sampling the sixteen input lines periodically to pick up the bit-by-bit serial input. In both input and output, 1 is a mark, 0 a space. The channels begin at the most significant end of the bus (in the minimum configuration the four input lines and four output lines are both connected to IO bus data lines 0-3). With each 4027 four-line group (EIA standard levels), a second set of input lines allows the program to sample the control signals for the channels, such as Ring Indicator, Clear to Send, or Data Set Ready.

A single multiplexor has device code 24, but the board actually contains jumpers for selecting any code from 24 to 27. Other multiplexor boards can be used for handling more communication channels, or a second board can be used (say with device code 25) for handling more modem control signals for the communication channels already in use. A second board supplies two sets of inputs, so that altogether the program can sample all three of the control signals listed above, and one set of outputs, either Request to Send or Data Terminal Ready. The ability to sample Ring Indicator and control Data Terminal Ready allows automatic answering of incoming calls (as explained at the end of the preceding section).

If the input were sampled at the bit rate, a bit could easily be missed and a transient could easily be mistaken for a start level (the space that begins a character). Hence each clock has a frequency five times the baud rate for which it will be used. By sampling the input five times per bit time, no bits are missed, and an initial space that lasts less than three sample times is properly recognized as a transient.

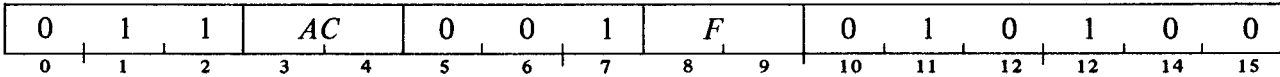
The multiplexor uses three IO transfer instructions, two for input and one for output, with device code 24, mnemonic DCM. Busy and Done are sensed by bits 8 and 9 in IO skip instructions and are controlled in the usual fashion by Clear and Start. Interrupt Disable is controlled by interrupt priority mask bit 0. Other multiplexors for data or additional control signals use the same instructions with the appropriate device codes.

**DOA --,DCM      Data Out A, Data Communications Multiplexor**



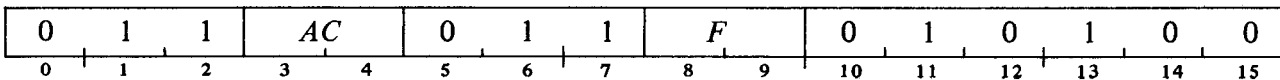
Load the contents of AC into the data output register, and perform the function specified by *F*.

**DIA --,DCM      Data In A, Data Communications Multiplexor**



Transfer the contents of the data input lines into AC, and perform the function specified by *F*.

**DIB --,DCM      Data In B, Data Communications Multiplexor**



Read the signals on the control input lines into AC, and perform the function specified by *F*.

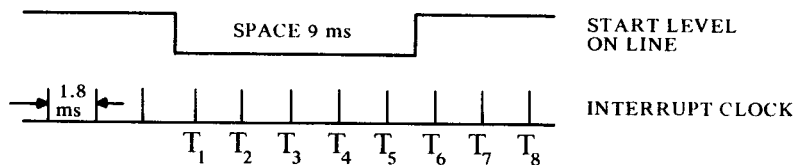
Setting Busy turns on the clock so that the next pulse clears Busy and sets Done, requesting an interrupt if Interrupt Disable is clear. Before beginning any operations, the program should give Start and wait for the first interrupt. Then at each interrupt, the program can update the output, sample the input, and restart the clock. Between interrupts the program should process the input and set up the next output. Output lines that are not in use should be left marking (*ie* unused bits in the output register should be loaded with 1s).

**Timing.** A Model 33 or 35 has a transmission rate of ten characters per second, 110 bits per second. For these devices the clock runs at 550 Hz, so the program can sample the input five times per bit. Clocks are available for channels that operate at frequencies up to 600 baud (3,000 Hz).

Consider a system with sixteen lines, all of which operate at 110 baud with 550 interrupts per second. Suppose that all lines are inactive and then at an interrupt a DIA 1,DCM reads this word into AC1:

1110111111111111

The 0 in bit 3 may be a space indicating that information is coming in on line 3. The relationship between the interrupts and the line signal timing is something like this:



The program has discovered the possible space at T<sub>1</sub>; if a 0 is still read at T<sub>2</sub> and T<sub>3</sub> it can be assumed that the line has a true space rather than a transient and that transmission has started. The program should then sample the line at every fifth interrupt (T<sub>3</sub>, T<sub>8</sub>, T<sub>13</sub>, ...) so that sampling is centered within each bit time. If a number of lines are operating, the program must keep track of them separately, *ie* different lines should be sampled at different interrupts to keep the sampling times centered.

For output the program decides when to transmit, and different lines can be changed at different times; but if DOAs are given more frequently than the transmission speed, information given previously for a line must be repeated so that it is held on the line for a full bit time (five interrupts). To save time it is usually preferable to run all of the output lines together, so the output word need be recomputed and the output register reloaded at only every fifth interrupt.

### 7.3 MODEM CONTROL 4023 AND 4029

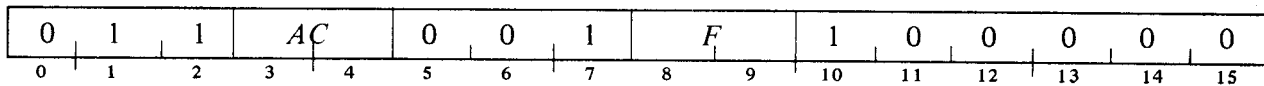
The 4023 is an option that can be added to the 4010 teletypewriter interface to supply EIA standard levels and 150 baud operation for a Model 37, Bell 103 Data Set, or equivalent; and in fact the 4023 is installed for use with the console teletypewriter if that is a Model 37. The instructions and all of the information given for the Model 37 in §3.1 apply to the 4023 whether used for the console or with a modem for remote communication. Of course in the latter case, device codes and mnemonics different from those for the console teletypewriter must be used; the code pairs generally assigned to receiver and transmitter respectively are 40-41 or 50-51.

The 4029 operates with the 4010 interface and the 4023 option to implement use of the Bell Data Set 202C, 202D, or equivalent for communication over telephone lines. The 202C, which is used for dial-up operations, generally operates at 1200 bits per second and has a telephone hand set contained within the unit, allowing

alternate voice/data operation. The 202D is used for permanently connected private lines and generally operates at 1800 bits per second; alternate voice operation requires addition of an optional auxiliary hand set Type 804A. Reverse channel capability, which is optional on the 202, is not available in the 4029.

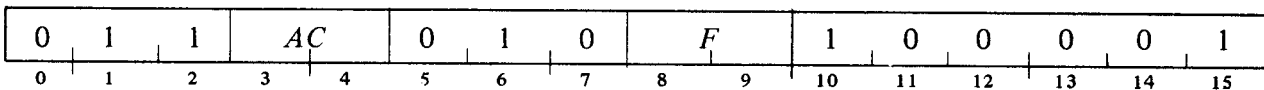
Although mounted on a single circuit board, the 4010 with the 4023 and 4029 features is actually two independent interfaces, allowing full duplex reception and transmission of data. Besides a pair of IO instructions like those of the teletype to handle character transfers, the interface has two IO transfer instructions for checking status and handling the automatic answering of incoming calls. The board has jumpers for the selection of device code pair 40-41 or 50-51 for the receiver and transmitter respectively. Busy and Done for the receiver and transmitter are sensed by bits 8 and 9 in IO skip instructions with the assigned code, and these flags are controlled in the usual fashion by Clear and Start. Receiver Interrupt Disable is controlled by interrupt priority mask bit 14, Transmitter Interrupt Disable by mask bit 15. The special IO Pulse function ( $F = 11$ ) given with the receiver code clears the Break Indicator (status bit 15). For convenience the instructions are given with device codes 40 and 41, and the mnemonics REC and XMT are used in representing the instructions, but they are not recognized by the assembler; the programmer must define his own mnemonics.

**DIA -,REC      Data In A, Receiver**



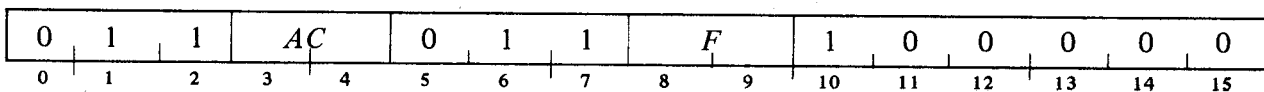
Transfer the contents of the receiver buffer into AC bits 8-15, and perform the function specified by  $F$ . Clear AC bits 0-7.

**DOA -,XMT      Data Out A, Transmitter**

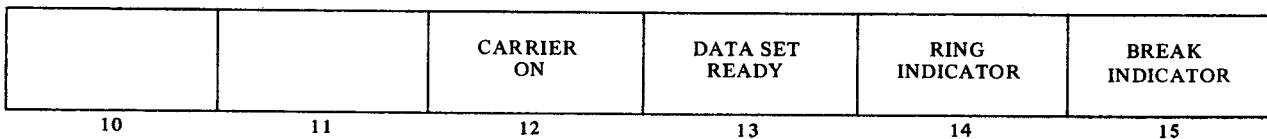


Load the contents of AC bits 8-15 into the transmitter buffer, and perform the function specified by  $F$ .

**DIB -,REC      Data In B, Receiver**



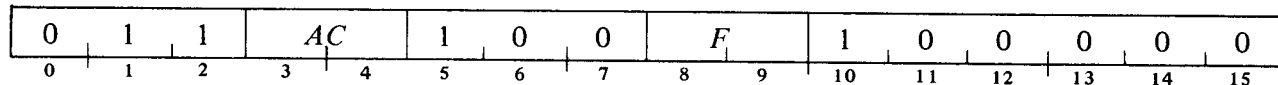
Read the status of the receiver into AC bits 12-15 as shown, and perform the function specified by  $F$ . Clear AC bits 0-11.



Bits 12-14 are for the automatic answering feature described below. Bit 15 is cleared by P.

- 12 A carrier is being received from a remote station.
- 13 The local data set is connected and is capable of handling data.
- 14 A ringing signal is being received from a remote station.
- 15 The line has been opened or a break key struck.

**DOB —,REC Data Out B, Receiver**



If AC bit 15 is 1, send a Data Terminal Ready signal; if AC bit 15 is 0, terminate the Data Terminal Ready signal. (Perform the function specified by *F*.) This instruction is used to answer incoming calls (see below).

Reception from the line requires no initiating action by the program; any character that appears on the line is automatically loaded serially into the buffer (the Reader Busy flag of the 4023 is set by giving Start, but it serves no function here). Completion of reception clears Reader Busy and sets Receiver Done, requesting an interrupt if Receiver Interrupt Disable is clear. Programming the Pulse function (*F* = 11) with device code 40 clears Break Indicator.

When the transmitter is off, setting Transmitter Busy turns it on and generates the Request To Send signal. Once the local modem generates the Clear To Send signal, the contents of the transmitter buffer are sent out serially over the line (the buffer is cleared during transmission). Completion of transmission clears Transmitter Busy and sets Transmitter Done, requesting an interrupt if Transmitter Interrupt Disable is clear. Once the transmitter is on, setting Transmitter Busy sends out the contents of the buffer — the transmitter remains on so long as either Busy or Done is set. Giving Clear (*F* = 10) clears both Busy and Done, terminating the Request To Send signal and turning off the transmitter.

**NOTE**

Although the buffer clears during transmission, giving an NIOS without loading it again does not transmit a zero character. So do not give an NIOS without first loading the buffer. To transmit any character including null, either give a DOAS or give a DOA followed by an NIOS.

The 4029 can handle either 10-unit or 11-unit codes. In the transmitter the code type is selected by means of a jumper; the receiver can handle either type arbitrarily with no change needed in the logic.

**Timing.** The 4029 is normally set to operate at 1200 bits per second, but other speeds are available. To fully utilize the 1200 baud rate, the program must be prepared to handle a 10-unit character every 8.3 ms, an 11-unit character every 9.2 ms. Since the rate of incoming data cannot be known *a priori*, the maximum must be assumed. Hence to avoid the possibility of data loss, the program must retrieve a 10-unit character within 1.25 ms after Receiver Done sets, an 11-unit character within 2.1 ms. After Transmitter Done sets, the program must supply another character within .83 ms to keep the transmitter going at the maximum rate.

The corresponding times for 1800 baud operation are 10-unit characters every 5.55 ms, 11-unit characters every 6.1 ms. The program has .83 ms to retrieve a 10-unit character from the receiver, 1.4 ms to retrieve an 11-unit character. To maintain the maximum transmission rate, the program must respond to Transmitter Done within .55 ms.

**Automatic Answering.** If the local data set is so configured, the computer can answer incoming calls by making use of bits 12-14 of the status word read by the DIB. Bits 12 and 13 give the status of the communications circuit and the local data set: bit 12 indicates that a carrier is being received from the remote station; bit 13 indicates that the local data set is connected and is capable of handling data. The program detects a ringing signal from a remote station by periodically examining bit 14, the Ring Indicator. The program answers a call by sending a Data Terminal Ready signal to the local data set; the program must also dismiss the call when completed. Answering and dismissing a call are effected by using the DOB to control Data Terminal Ready.

#### 7.4 MULTIPROCESSOR COMMUNICATIONS ADAPTER 4038

This option makes it possible to connect up to fifteen Nova line computers into a multiprocessor system by permitting the transfer of blocks of data from one computer to another through their data channels. One adapter is attached to the IO bus of each computer in the system, and the adapters are connected together by a common communication bus. Although mounted on a single circuit board, an adapter (MCA) is actually two independent interfaces, allowing simultaneous reception and transmission of data. Each interface is connected separately to the data channel, so the program need only set up an interface for receiving or sending and all transfers to and from memory are then handled automatically. To operate with the data channel, the receiver and transmitter each have an address counter and a word counter as well as data and status registers.

By means of jumpers on the board, each adapter is assigned a code in the range 1-17 octal, which code is shared by the transmitter and receiver and is used for creating communication links. A processor with an adapter can establish a link between its transmitter and any receiver it designates provided that receiver has been set up for reception. In other words the transmission of a data block between any pair of computers requires program activity at both ends of the link. Once a processor sets up its receiver, that receiver locks onto any transmitter that sends it a word and then accepts data from only that transmitter until the receiver is unlocked by the program. A given block transfer is complete when one of the word counts, in either receiver or transmitter, goes to zero, but the receiver does not unlock from the transmitter without specific action by the program. This way the program can set up the receiver for another block from the same transmitter without worrying that some other transmitter will interfere.

The characteristics of data transmission must be established by convention in the software for the multiprocessor system. At the simplest level all transmission can be in standardized blocks with every receiver simply left enabled to lock onto any transmitter that calls it. A much more flexible system can be achieved by the use of control blocks to specify the characteristics of subsequent data operations. Then whenever a receiver is not engaged in a data transfer, it can simply be left free to receive a control block in some standard format. Upon receipt of a control block, the processor can inspect its contents to determine how to respond: this may involve setting up the receiver for a specific data operation, setting up the transmitter to send a block to another computer, or both.

Each adapter has multiplexer circuitry built into it so that any number of communication links can be held concurrently on the bus, with each receiving an equal share, if needed, of the available time. Links can be added or dropped at any time without affecting the system except in terms of individual transfer rates; even turning off power at one computer does not affect the other computers or the communication network.

**Timing.** The maximum overall transfer rate through the communication bus is half a million words per second. The rate for a single link however is at most 250,000 words per second regardless of the speed capability of the data channel when connected to a single device. A typical data rate for a single link ranges from 70,000 words per second for a pair of Nova computers to 140,000 for Nova 800s or Supernova computers with high speed data channels. The basic cycle time of the network is  $2\mu\text{s}$ , and all transmitters currently executing a block transfer are allowed access to it in round robin fashion. If a given transmitter is not ready when its turn comes, it must wait until the next time around. Whether or not a transmitter is always ready in time for its turn depends

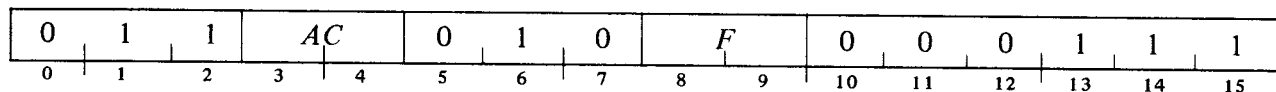
somewhat on the speed of the channel, but primarily on any delay caused by the program or other devices before the transmitter can gain direct access to memory for another word. If a receiver does not accept a word transmitted to it, the sending transmitter must simply try again with the same word the next time around.

### Receiver

To set up the receiver to accept a block of data, the program must supply an initial address to the 15-bit address counter and supply either a specific (two's complement) negative word count to the 16-bit word counter or a word count large enough (eg zero) to receive the entire block regardless of size.

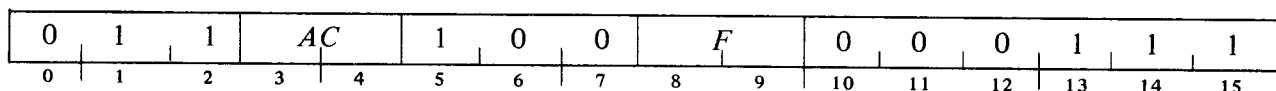
The receiver uses five IO transfer instructions, with device code 7, mnemonic MCAR, for loading and reading the address and word counters and reading status. Busy and Done are sensed by bits 8 and 9 in IO skip instructions and controlled in the usual fashion by Clear and Start. Interrupt Disable is controlled by interrupt priority mask bit 12. A second receiver connected to the IO bus as part of a second independent MCA system would have device code 47.

#### DOA $\bar{M}$ CAR Data Out A, MCA Receiver



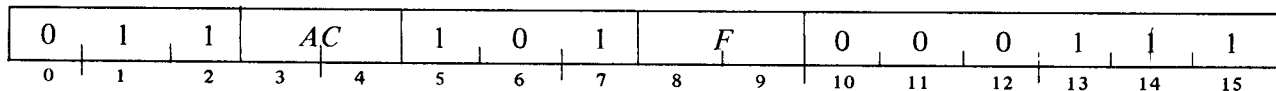
Load the contents of AC bits 1-15 into the address counter, and perform the function specified by F.

#### DOB $\bar{M}$ CAR Data Out B, MCA Receiver

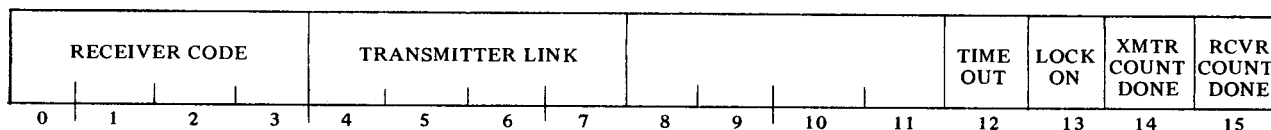


Load the contents of AC into the word counter, and perform the function specified by F.

#### DIC $\bar{M}$ CAR Data In C, MCA Receiver



Read the status of the receiver into AC as shown, and perform the function specified by F.

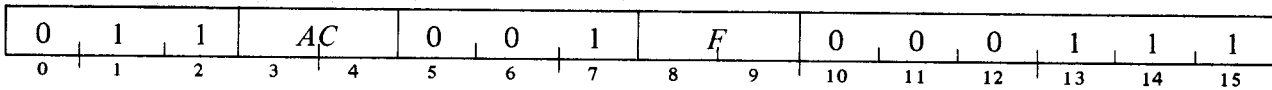


Start clears Time Out, XMTR Count Done and RCVR Count Done; Clear clears these plus Lock On. The setting of Time Out, XMTR Count Done or RCVR Count Done clears Busy, sets Done, and disables (but does not unlock) the receiver.



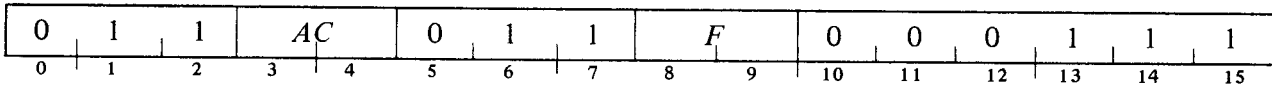
- 0-3 The code of this receiver as determined by its jumpers.
- 4-7 The code of the transmitter to which this receiver is or was connected.
- 12 A block transfer is in progress but no data has been received for 10 ms.  
NOTE: This bit indicates suspicious behavior — it cannot be set by normal termination, *ie* transmitter word count overflow.
- 13 The receiver is locked on the transmitter specified by bits 4-7.
- 14 The transmitter has completed the block transfer as determined by its word counter.
- 15 This receiver has completed its reception as determined by its word counter.

**DIA —,MCAR      Data In A, MCA Receiver**



Read the present contents of the address counter into AC bits 1-15, and perform the function specified by *F*.  
Clear AC bit 0.

**DIB —,MCAR      Data In B, MCA Receiver**



Read the present contents of the word counter into AC, and perform the function specified by *F*.

Besides clearing Done and setting Busy, Start ( $F = 01$ ) clears Time Out, XMTR Count Done and RCVR Count Done (status bits 12, 14 and 15). Setting Busy enables the receiver so that it can accept data. If Lock On is set, the receiver is already locked to the transmitter whose code appears in status bits 4-7 and will accept data from only that transmitter. If Lock On is clear, then as soon as some transmitter sends a word to the receiver, it locks on that transmitter (Lock On sets and the transmitter code appears in status bits 4-7) and will accept data from only that transmitter until it is unlocked. As each word is received it is sent to the memory location specified by the address counter, and both counters are incremented. The receiver does not accept another word until the previous one is stored. Word count overflow at either end of the link or failure of the transmitter to send data for 10 ms sets the appropriate status flag, clears Busy (disabling the receiver without unlocking it), and sets Done, requesting an interrupt if Interrupt Disable is clear. Ordinarily the setting of Time Out indicates program or operator intervention or equipment malfunction at the transmitting processor.

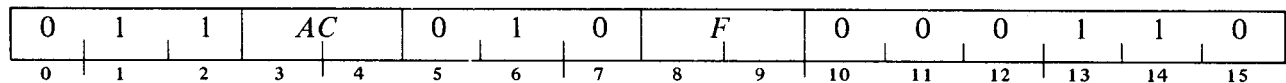
Besides clearing Busy and Done, Clear ( $F = 10$ ) clears status flags 12-15; hence it both disables and unlocks the receiver. This terminates a transfer if one is in progress, and frees the receiver to accept data from any transmitter once the program sets Busy again.

**Automatic Loading.** MCA receivers of revision 03 and later can be used with the channel start procedure for automatic program loading. The initiation of this procedure with device code 7 in data switches 10-15 enables the receiver to accept data into memory locations 0-377 from any transmitter that calls it. The procedure works just as it would were the data taken from a disc or magnetic tape, except that the procedure places the latter devices in operation but with the MCA must wait until some other computer starts transmitting.

## Transmitter

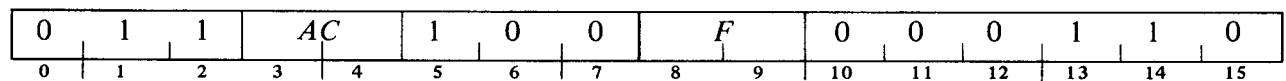
To set up the transmitter to send a block of data, the program must supply an initial address to the 15-bit address counter and supply a specific (two's complement) negative word count to the 16-bit word counter. The transmitter uses all six IO transfer instructions, for loading and reading the address and word counters, for reading status, and for specifying the receiver to which a communication link is desired. These instructions use device code 6, mnemonic MCAT. Busy and Done are sensed by bits 8 and 9 in IO skip instructions and controlled in the usual fashion by Clear and Start. Interrupt Disable is controlled by interrupt priority mask bit 12. A second transmitter connected to the IO bus as part of a second independent MCA system would have device code 46.

### DOA $\text{--},\text{MCAT}$ Data Out A, MCA Transmitter



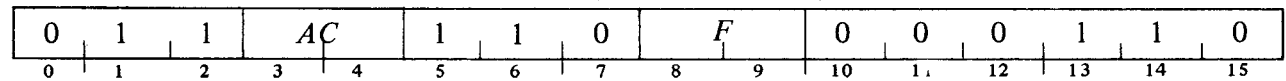
Load the contents of AC bits 1–15 into the address counter, and perform the function specified by *F*.

### DOB $\text{--},\text{MCAT}$ Data Out B, MCA Transmitter



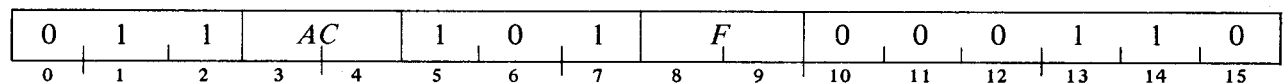
Load the contents of AC into the word counter, and perform the function specified by *F*.

### DOC $\text{--},\text{MCAT}$ Data Out C, MCA Transmitter

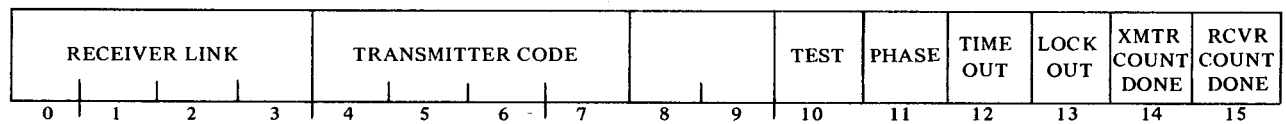


Select the receiver specified by AC bits 0–3 for establishing a communication link, and perform the function specified by *F*. (A 1 in AC bit 11 places the system in diagnostic mode, wherein the instruction NIOP  $\text{--},\text{MCAT}$  can be used to step the system through its clock phases.)

### DIC $\text{--},\text{MCAT}$ Data In C, MCA Transmitter



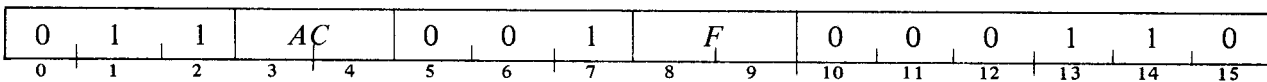
Read the status of the transmitter into AC as shown, and perform the function specified by *F*.



Clear and Start both clear Time Out and XMTR Count Done. The setting of Time Out, XMTR Count Done or RCVR Count Done clears Busy, sets Done, and disables the transmitter. Lock Out and RCVR Count Done are meaningless unless Done is set. (Bits 10 and 11 are for maintenance only.)

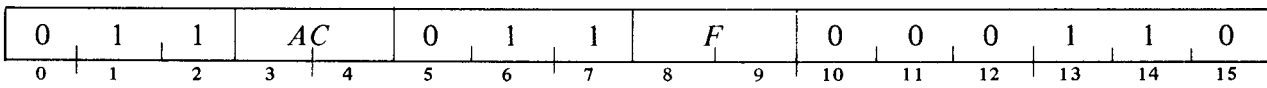
- 0-3 The code of the receiver specified for a link by the last DOC.
- 4-7 The code of this transmitter as determined by its jumpers.
- 12 The transmitter has attempted to begin a block transfer or one is in progress, but the receiver specified by bits 0-3 has accepted no data for 10 ms.  
 NOTE: At the initiation of a block transfer this bit indicates the processor at the other end of the attempted link has not set up its receiver, or the receiver is locked to some other transmitter for an abnormally long time; if a transfer is already in progress, a 1 in bit 12 indicates suspicious behavior — it cannot be set by normal termination, *ie* receiver word count overflow.
- 13 The receiver specified by bits 0-3 is locked to some other transmitter.
- 14 This transmitter has completed the block transfer as determined by its word counter.
- 15 The receiver has completed its reception as determined by its word counter.

**DIA —,MCAT Data In A, MCA Transmitter**



Read the present contents of the address counter into AC bits 1-15, and perform the function specified by *F*. Clear AC bit 0.

**DIB —,MCAT Data In B, MCA Transmitter**



Read the present contents of the word counter into AC, and perform the function specified by *F*.

Besides clearing Done and setting Busy, Start (*F* = 10) clears Time Out and XMTR Count Done (status bits 12 and 14). Setting Busy turns on the transmitter, which in turn retrieves a word from the memory location specified by the address counter and attempts to send this word to the receiver specified by the last DOC. If the receiver is locked to some other transmitter, Lock Out sets, but the transmitter keeps trying and sets Time Out only if the receiver refuses to accept the word within 10 ms. If the receiver does accept the word, Lock Out clears and the receiver locks on this transmitter, which then sends the block of words retrieved from the locations specified by the address counter and increments both counters on each transfer. Word count overflow at either end of the link or failure of the receiver to accept data for 10 ms sets the appropriate status flag. The setting of Time Out, XMTR Count Done or RCVR Count Done clears Busy, turning off the transmitter, and sets Done, requesting an interrupt if Interrupt Disable is clear. Ordinarily the setting of Time Out once a block transfer is in progress indicates program or operator intervention or equipment malfunction at the receiving processor.

Besides clearing Busy and Done, Clear (*F* = 10) clears Time Out and XMTR Count Done. Giving Clear, and thus turning off the transmitter, during a block transfer can cause data loss.

**Installation**

The adapter is mounted on a single 15-inch square printed circuit board that plugs directly into one of the slots in the computer. Each adapter is assigned a 4-bit code by means of jumpers on the board; the adapter is shipped with the jumpers in place (code 17), but the user can change this to any desired code by cutting out jumpers for 0s.

The standard communication network has a bus 40 feet long, allowing a data transfer rate of 500 KHz. Reducing this rate to 300 KHz by installing jumper W5 in every adapter allows lengthening the bus to a maximum of 90 feet. The in connector of each adapter is bussed to the out connector of the next in a chain.

The communication bus requires a terminator network at each end, but a network is included in every adapter. In a system that combines more than two processors, the terminators must be removed from all adapters except those at the ends of the bus. To do this, remove all 270 ohm terminating resistors from the middle adapters, and replace all 180 and 220 ohm resistors with 10K resistors. If an additional adapter is added at either end of an installed system, the terminating configuration must be revised accordingly.

To change the device codes for a second MCA system connected among the same processors, remove jumper W6 and install W7 on all adapters in the second system.

## **7.5 ASYNCHRONOUS DATA COMMUNICATIONS MULTIPLEXOR 4060**

This device handles up to sixty-four full duplex communication channels for the simultaneous serial transmission and reception of data at various speeds and character configurations. All of the serial operations are done by the device, so the program is involved only in the movement of data characters to and from the device via the IO bus. For transmission the program outputs a word that specifies the channel and supplies the data; the transmitter for that channel adds the necessary start and stop bits and sends the complete character out on the line serially. For reception a receiver responds to the appearance of a start bit on its associated channel and assembles the character that comes in serially over the line. To retrieve the data the program reads a word that contains both the data part of the character and the number of the channel on which it was received.

The customer can select the number of communication channels in multiples of four, input and output together, and for each four-line group can select the type of signal, the character configuration, the speed, and the channel numbers as a consecutive set of four (0-3, 4-7, etc). Interface cards 4060 and 4061 handle 20 ma signals for a Teletype Model 33 or 35; cards 4062 and 4063 handle EIA standard levels for a Model 37, a Bell 103 Data Set, or other modem. A system with only four channels and with data sets limited to manual answer generally uses the 4060 or 4062, which has individual connectors for the channels. In a system with a larger number of lines or data sets equipped for automatic answering, 4061s or 4063s are used in conjunction respectively with a 4050 or 4051 junction panel for each four interface cards (a mixed system requires use of the 4051 panel with adapters for the local dc terminals).

Other device characteristics are selected for each four-line group by means of jumpers on the interface card. Transmission and reception can be in characters of from five to eight data bits with one or two stop bits (for characters with five data bits the user can also select a stop code of 1 1/2 units). Available clock frequencies are 75, 110, 134.5, 150, 300, 600, 1200, 2400, 4800 and 9600 Hz, with accuracy better than .6%. Other frequencies or more precise frequency control can be had by adding clock option 4064.

Four lines are mounted on a single board occupying one subassembly slot. The entire system, which can fill as many as 16 slots, consisting of up to 64 lines appears to the program as a single device with a Done flag (but no Busy). Internally however the system is actually 128 devices, as the transmitter and the receiver for each channel can operate independently of and simultaneously with the others. At this level each device has a Ready flag which indicates, in the case of a receiver, that it has an assembled character ready for the program, or in the case of a transmitter, that its buffer is free to accept a character from the program. In terms of interrupt priority, the system acts partly like a single device but partly like sixty-four devices, each of which is the receiver-transmitter pair for a single channel. The boards for the system must be mounted in adjacent slots, and the entire system responds as a single unit to the INTA instruction according to the position of the system on the bus (INTA reads the device code of the closest device that is requesting an interrupt). But analogous to INTA, this system itself has an instruction that reads the number of the highest priority channel that has a Ready flag set. The priority is determined by the position of the board on the bus and the position of the channel on the board.

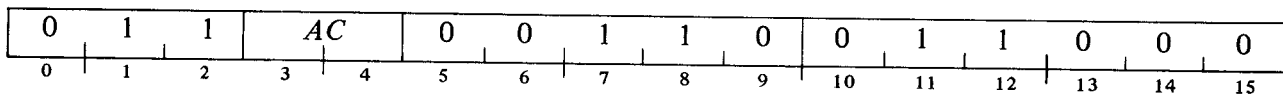
Hence the jumpers in any interface card should be configured so that lines 0–15 are on the first board, lines 20–37 on the second, etc., and on a given board the numbers are assigned in order from highest priority to lowest.

### Instructions

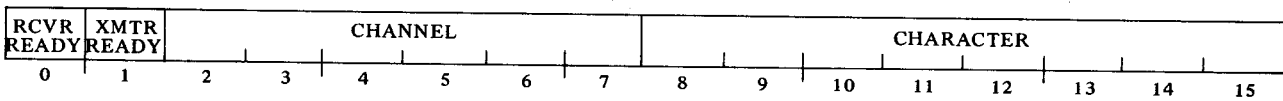
The program executes no initiating operations for the receiver as reception is initiated externally and the program simply responds when data is ready. The program turns on a transmitter simply by supplying a data character to it for transmission. Done is set by the setting of any Ready flag for the transmitter or receiver for any channel. The program has no direct control over Done except by giving an IO reset to clear the entire in-out system. Once Done is set, it remains set so long as any Ready is set, and it clears automatically whenever all Ready flags are clear — in other words the program clearing a particular Ready does not affect Done unless there are no other Ready flags set. Each receiver and transmitter has an 8-bit buffer for handling data via the IO bus as well as a shift register for connection to the line.

The multiplexer uses three IO transfer instructions, one for input and two for output, with device code 30, mnemonic AHM. The single input instruction handles both character input and the determination of the highest priority channel that interrupted. One output instruction handles character output, and the other is used simply to clear transmitter flags. Done is sensed by bit 9 in the two IO skip instructions that have a 1 in bit 8, but the other two skips are not used as there is no Busy. The Clear function is used only to clear receiver flags with the input instruction, and Start is not used at all. Interrupt Disable is controlled by interrupt priority mask bit 14. A second multiplexer connected to the IO bus would have device code 70.

#### DIAC —,AHM Data In A, Asynchronous Multiplexor

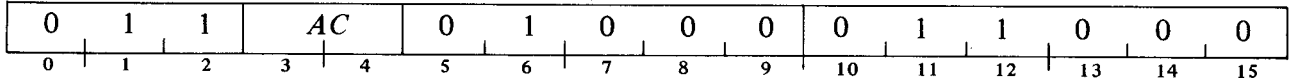


Read the interrupt information from the multiplexer into AC as shown.

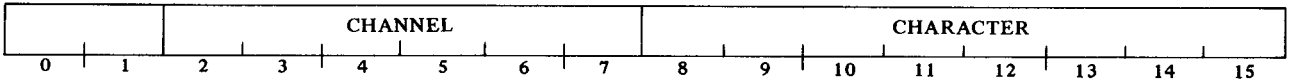


- 0     The receiver for the channel specified by bits 2–7 has assembled a character from the line and the data part of that character is right-justified in bits 8–15. This instruction clears RCVR Ready for the specified channel.
- 1     The transmitter for the channel specified by bits 2–7 has moved a character from its buffer to its shift register and is free to receive another data character from the program.
- 2–7   The number of the highest priority channel which has either Ready flag set.
- 8–15   If bit 0 is 1, these bits contain the right-justified data part of the character just received on the channel specified by bits 2–7.
- 7-16

**DOA -AHM Data Out A, Asynchronous Multiplexor**

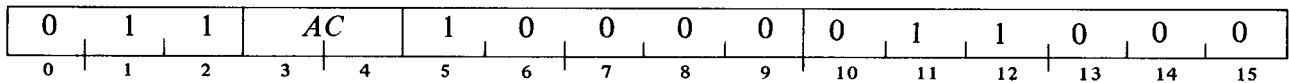


Load the contents of AC bits 8-15 into the transmitter buffer and clear the XMTR Ready flag for the channel specified by AC bits 2-7.



The data bits for the character length handled by the specified channel must be right-justified in the right half of AC.

**DOB -AHM Data Out B, Asynchronous Multiplexor**



Clear XMTR Ready for the channel specified by AC bits 2-7.

Reception is initiated at the other end of the line, but the processor begins transmission on an idle line by giving a DOA that specifies the desired channel and loads the data for a character into the transmitter buffer for that channel. The transmitter assembles a complete character of the appropriate configuration (as determined by the jumpers on the interface card) taking the correct number of data bits from the right of the buffer; hence the program must supply the data for each character right-justified in AC. As soon as the transmitter takes the character into the shift register making the buffer free, it sets XMTR Ready and starts serial transmission of the character over the line. Note however that in an EIA interface the transmitter does not start sending until Clear to Send (circuit CB) is true. (Many typical modems, such as the 103, have the CB-CF common configuration, *ie* Clear to Send and Carrier Detected are equivalent signals.)

Except when initiating transmission on an idle line, the program simply responds to a running system at each interrupt. When a receiver assembles a complete character from the line it moves the data bits of the character right-justified into its buffer and sets RCVR Ready. When a transmitter finishes sending a character it takes the data bits (if any) for the next character from its buffer and sets XMTR Ready. The setting of either type of Ready flag sets Done, requesting an interrupt if Interrupt Disable is clear. Upon recognizing that the multiplexer is the source of the interrupt, the program gives a DIAC that reads the number of the highest priority channel for which either Ready flag is set. If RCVR Ready is set, it is automatically cleared and the right half of the word read contains the right-justified data bits of the character received, so no further action by the program is necessary. If XMTR Ready is set, the program gives a DOA that selects the interrupting transmitter, supplies the data bits for the next character, and clears its Ready flag. Should the program fail to respond before the transmitter finishes sending the character in its shift register, the transmitter simply marks the line until a new character is supplied. Should no further transmission be desired on the interrupting channel, the program should give a DOB for that channel to clear XMTR Ready without supplying a new character.

The program has no control over the character length or bit rate for any channel as these are jumper selected. However the program should be aware of these characteristics for every channel. Supplying an incorrect number of data bits means that the transmitter will either throw away the extras or will fill missing bit positions with Os.

**Timing** Since both receiver and transmitter have a buffer, the program has an entire character time in which to retrieve a character before data can be lost or to send a character and keep the transmitter operating at maximum rate. The character time for any channel depends upon both the frequency and the character configuration selected for that channel. In a system with mixed bit rates, the lower numbers should be assigned to the faster lines so that they will have higher priority. The receiver is set up to begin character assembly only when it receives a start level (space) of duration longer than one-half bit time; a shorter start level is regarded as a transient.

### **Modem Control**

Other than the transmitter sensing of Clear to Send, the 4060 series multiplexor contains no facilities for handling control signals (no control is necessary for a dc channel). To handle control signals for each set of four 4-line EIA interface cards (sixteen lines) requires one 4026 programmed asynchronous multiplexer with a 4027 for each 4-line group. The 4027 allows the program to sense two input signals and control one output signal for each of four communication channels. The manner in which the program senses input signals and turns output signals on and off is treated in the description of the 4026 multiplexer in §7.2.

Suppose the 4060 system has eight channels operating with Bell 103 Data Sets equipped for automatic answering. To allow the program to answer incoming calls on these data sets requires the addition of one 4026 with two 4027s. For these channels the 4027s should be connected to monitor Ring Indicator and Data Set Ready and to control Data Terminal Ready. The program can detect a ringing signal from a remote station by periodically examining Ring Indicator; the program answers the call by sending Data Terminal Ready to the local data set, and when the call is completed the program must dismiss it by turning off that signal.

To use Bell 202 Data Set equipped with automatic answer on a two-wire (half-duplex) channel or in a multi-point network requires the addition of a second 4027 for each 4-line group (in other words each 4026 can handle the modem control for only eight channels). The second 4027 would be used to monitor Data Carrier and Clear to Send and to control Request to Send.

### **7.6 IBM SYSTEM 360/370 INTERFACE 4025**

This device allows a Nova line computer to simulate any peripheral device or group of peripheral devices that can operate through a selector or multiplexor channel on an IBM System 360 or 370 computer. To program the system at the machine level requires not only a knowledge of the IBM part of the system including the peripherals to be simulated, but also a detailed knowledge of the timing and hardware operation of the interface itself. Such a treatment is beyond the scope of this manual. This section is therefore limited to an overall description of the interface and the definition of the IO instructions for it. For the complete information required to program the system effectively, refer to *Technical Reference: 4025 IBM System 360/370 Interface*, Data General document 014-000001.

The 4025 consists of two standard logic boards that mount in the Nova line main frame and a separate power control adapter that requires 5¼ inches of space in a standard 19-inch rack. Connection to the IBM computer is made at any position on either the selector channel for high speed block transfers for one device at a time, the byte multiplexor channel for single bytes interleaved among many devices, or the 370 block multiplexor channel, which is like the byte multiplexor channel except that it handles fast blocks instead of single bytes. The 4025 acts like any standard IBM device in that it is capable of recognizing all IBM bus/tag sequences, making all of the necessary requests, etc. Although all information transfers through the 4025 are in the 8-bit bytes basic to the 360/370 system, this description is from the point of view of the Nova line computer, wherein the program is the Nova computer program, data in is the movement of data into the Nova computer (hence out of the IBM computer), and so forth.

Jumpers in the equipment determine which IBM device addresses the 4025 can recognize; this may be all 256 addresses, only a single address, or any subset of the possible addresses that is selectable by means of jumpers for the address bits with some bits indeterminate. For a recognized device address, the program uses the 4025 to simulate the device by responding to command bytes received from the channel, executing data transfers in either direction, and supplying status bytes to the channel. The IO instructions handle only control information for the interface itself; all information going to or from the IBM channel via the interface is handled by the Nova line data channel, so the program need only set up the 4025 for a particular operation and all transfers to and from memory are then handled automatically.

Command bytes received from the IBM channel are stored along with their associated device addresses in a 256-word circular buffer in memory for subsequent processing by the program. Storage is under the control of a command address counter, which is incremented just before each command byte and device address are stored in the left and right halves of the location specified by the counter. Counting is limited however to the right half of the counter (bits 8–15), so that the circular buffer always begins at a location whose address has 0s in bits 8–15; each time a command-address word is stored in the final location in the buffer, the counter recycles so the next storage is in the first location.

Upon indication by the program that a status byte should be supplied to the IBM channel for a particular device, the 4025 retrieves the status information from the left half of a location in the status table. This table is also exactly 256 words in memory, and begins at a location whose address has 0s in bits 8–15. The program initially locates the table by specifying bits 0–7 of a memory address. Then for a given status transfer the interface accesses that location in the table corresponding to the address of the device it is simulating — in other words the device address is used as bits 8–15 of the memory address for retrieving the status byte.

Data transfers are handled like those for tape or disc except that transfers are in bytes instead of words. Hence the 4025 has a 16-bit byte counter and a 16-bit byte address counter. In the latter counter the high order fifteen bits (1–15) specify the address for the memory location while the counter LSB (bit 0 as supplied by the load instruction) specifies which half of the location, 0 for left, 1 for right. A transfer may begin with either half of a location (as per the initial state of the counter LSB), but bytes are otherwise handled first left then right in each location consistent with standard counting. Data for output is always supplied two bytes per word as just described. Input data is packed two bytes per word under control of the byte address counter only if the Pack Data flag is set. Otherwise counter bits 1–15 function as a word address counter to supply addresses for storage of one byte in each location; which half receives each byte is determined by the now stable state of bit 0 as initially specified by the program.

On the adapter are lights that indicate when the unit is off line and when ac power is on at the Nova line computer. The online/offline state of the 4025 is controlled by a pair of switches, manual and request; the former overrides the latter, which is functional only when the former is in its center null position. Setting either switch to the online position places the unit on line, but the request switch also signals the program by setting Done to produce an interrupt. Setting the manual switch to off line takes the unit off line, but setting the request switch to this position has no effect on the unit — it simply produces an interrupt so that the program can take the unit off line at the appropriate time. When the request switch is used, status flags indicate the state requested. Other switches allow the operator to indicate the type of IBM channel to which the unit is connected, to reset the interface logic when it is off line, and to determine whether control of ac power to the Nova line computer shall be local or remote, *ie* shall be controlled by a power switch on the adapter or by the IBM computer.

### **Instructions**

To set up the IBM interface for operation the program must supply an initial command address (to the 16-bit command address counter) and eight bits for the left half of a status table base address. For each transfer operation the program must supply a command word, and for a data operation it must also supply an initial byte



address (to the 16-bit byte address counter) for data channel access and the 16-bit (twos complement) negative of a byte count.

The interface uses twelve IO transfer instructions with device codes 31 and 32, mnemonics IBM1 and IBM2, and has two busy flags. Associated with code 31 are the standard flags that indicate the state of the device with respect to the Nova line computer; hence these flags are named "Done" and "Busy to Nova computer". They are sensed in the standard fashion by bits 8 and 9 in IO skip instructions with device code 31. The Start function, which can be given only with the DOA, sets Busy to Nova computer but does not affect Done. Associated with code 32 is the Busy to Channel flag, which indicates that the 4025 is occupied with the Nova line computer and is therefore busy to the IBM channel. Busy to Channel sets automatically when Done is set (requesting an interrupt) and can be set by the Nova computer program to make the 4025 busy to the channel while the program is loading registers or reading status; in the latter case, the flag is set by the IO Pulse function (F = 11) with device code 31. With code 32, Start and Pulse have no effect at all. The code-32 skip instructions however use both bits 8 and 9: Busy to Channel is tested in the standard fashion, and the instructions that would ordinarily skip on a done flag test the suppress-out condition on the IBM channel. The Clear function with either code clears Done and both busy flags. Interrupt Disable is controlled by interrupt priority mask bit 13. A second 360/370 interface connected to the IO bus would have device codes 71 and 72. In an interrupt the code returned by INTA is the lower one of the pair (31 or 71).

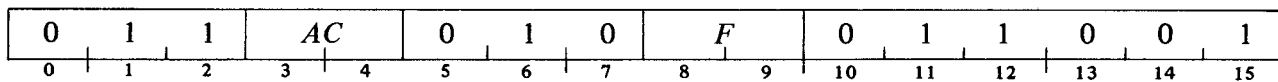
Note: There are conditions vis-a-vis the relationship between the 4025 and the IBM channel that can prevent the Clear, Start or Pulse function from actually affecting the interface flags. Therefore after giving a function, the program should use an IO skip to determine that the desired flag state has been produced, and repeat the function if it has not. *Eg* to start a transfer give

```
DOAS  0,IBM1    ;Start
SKPBN  IBM1     ;Busy to Channel set?
JMP    -2      ;No, try again
:       ;Yes, go ahead
```

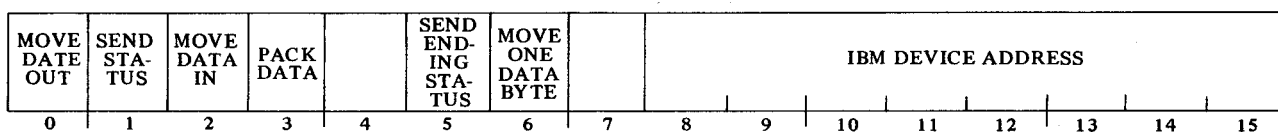
To clear Busy to Channel give

```
NIOC   IBM1
SKPBZ  IMB2
JMP    -2
:
```

### DOA --,IBM1 Data Out A, IBM Interface 1

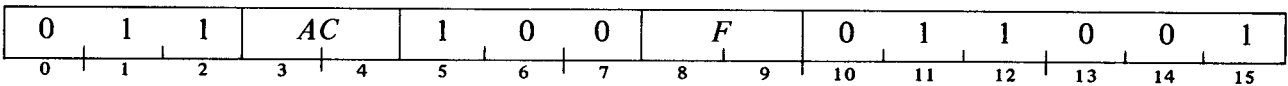


Load the contents of AC into the 4025 command register as shown, and perform the function specified by F.

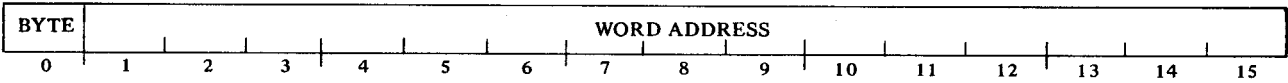


- 0 Perform a data transfer to the IBM channel for the device specified by bits 8–15.
- 1 Send a status byte to the IBM channel from the left half of the memory location for the device specified by bits 8–15 and then clear that location.
- 2 Perform a data transfer from the IBM channel for the device specified by bits 8–15.
- 3 If bit 2 is 1, pack the input data two bytes per word under control of the 16-bit byte address counter; otherwise place one byte in each location addressed by bits 1–15 of the counter in the half specified by bit 16, which is held fixed.
- 5 If bit 1 is 1, the byte sent is ending status.
- 6 If bit 0 or 2 is 1, the data transferred shall consist of a single byte. If this bit is not set in a data transfer, the 4025 assumes burst mode, and a channel malfunction may result if only one byte is transferred.
- 8–15 Operations selected by bits 0–2 shall be performed for the device specified by these bits.

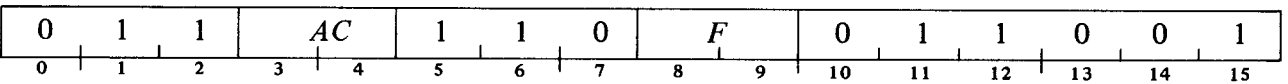
**DOB —,IBM1 Data Out B, IBM Interface 1**



Load the contents of AC into the byte address counter as shown. Perform the function specified by *F* (Clear or Pulse only).

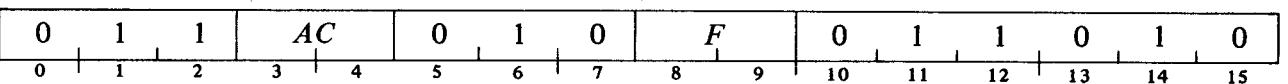


**DOC —,IBM1 Data Out C, IBM Interface 1**

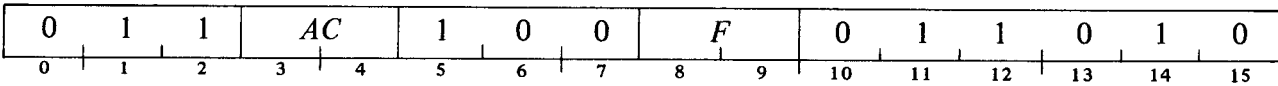


Load the contents of AC into the byte counter, and perform the function specified by *F* (Clear or Pulse only). If only one byte is to be transferred, bit 6 of the command word (sent by DOA —,IMB1) must be set.

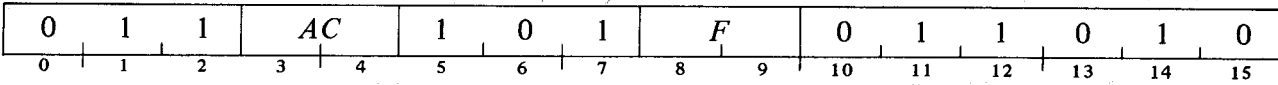
**DOA —,IBM2 Data Out A, IBM Interface 2**



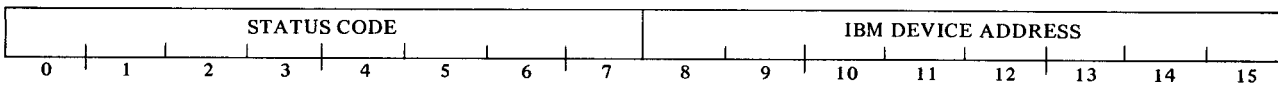
Load the contents of AC into the command address counter (AC bit 0 should be 0), and perform the function specified by *F* (Clear only).

**DOB —,IBM2 Data Out B, IBM Interface 2**

Select the set of 256 locations for the status table according to AC bits 0-7 (AC bit 0 should be 0), and perform the function specified by *F* (Clear only).

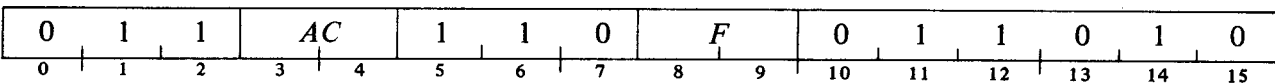
**DIC —,IBM2 Data In C, IBM Interface 2**

Read the status of the 4025 into AC as shown, and perform the function specified by *F* (Clear only).



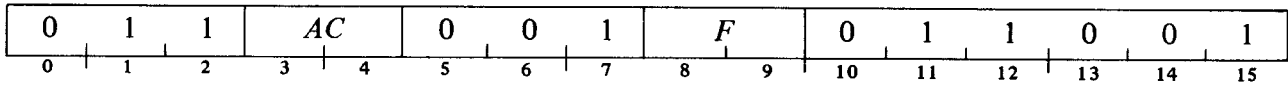
The following lists the bits in the left half that are set to indicate a particular event and the corresponding status code. Note however that events can occur simultaneously, and the codes are then superimposed. Bits 8-15 indicate the device address active or specified by the channel at the time the condition indicated by bits 0-7 occurred.

- |     |          |   |
|-----|----------|---|
| 0,7 | 10000001 | Halt IO — the channel has issued a Halt IO to this device.  |
| 1   | 01000000 | Command Parity Error — a byte placed in the command table had incorrect parity.   |
| 2   | 00100000 | Off Line Request — the operator has requested that the program take the 4025 off line.  |
| 2,7 | 00100001 | Selective Reset — the channel has issued a selective reset to this device.  |
| 3   | 00010000 | Data Parity Error — a data byte placed in Nova memory had incorrect parity.   |
| 4   | 00001000 | Initial Status — an initial status byte sent to the channel was not zero.   |
| 5   | 00000100 | Stop — the IBM computer halted a data transfer before the byte count overflowed.  |
| 5,6 | 00000110 | On Line Request — the operator has used the request switch to place the 4025 on line.   |
| 6   | 00000010 | Request Not Acknowledged — the channel issued a command to the recognized device indicated by bits 8-15 while the 4025 was initiating a transfer requested by the Nova program, and the transfer initiation was terminated. |
| 7   | 00000001 | System Reset — the channel has issued a system reset to all devices.  |

**DOC —,IBM2 Data Out C, IBM Interface 2**

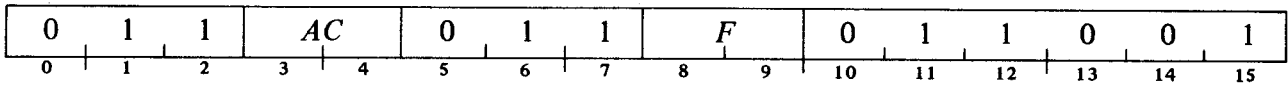
If AC bit 15 is 1, take the 4025 off line. Perform the function specified by *F* (Clear only).

**DIA - ,IBM1 Data In A, IBM Interface 1**

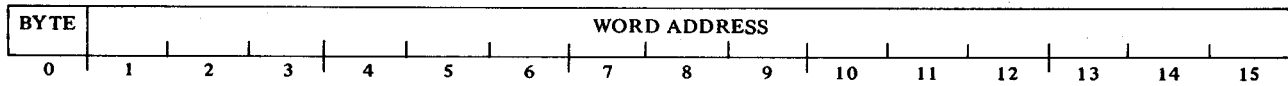


Read the address of the device the 4025 is presently simulating into AC bits 8-15 (Clear AC bits 0-7), and perform the function specified by *F* (Clear or Pulse only).

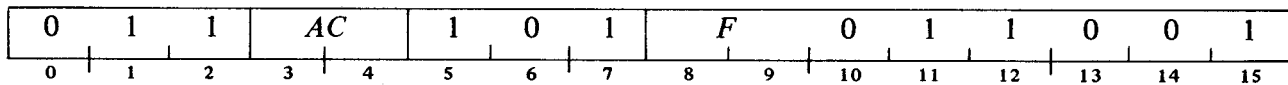
**DIB - ,IBM1 Data In B, IBM Interface 1**



Read the present contents of the byte address counter into AC as shown, and perform the function specified by *F* (Clear or Pulse only).

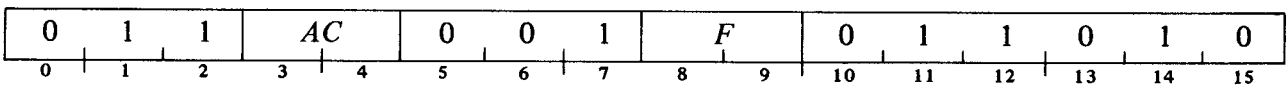


**DIC - ,IBM1 Data In C, IBM Interface 1**



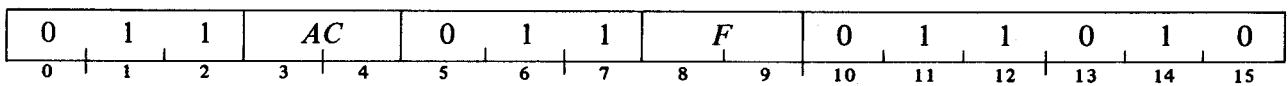
Read the present contents of the byte counter into AC, and perform the function specified by *F* (Clear or Pulse only).

**DIA - ,IBM2 Data In A, IBM Interface 2**



Read the present contents of the command address counter into AC, and perform the function specified by *F* (Clear only).

**DIB - ,IBM2 Data In B, IBM Interface 2**



Read the nonzero (left) half of the status table base address into AC bits 0-7 (clear AC bits 8-15), and perform

the function specified by *F* (Clear only).

**SKPBN IBM1**      **Skip if Busy to Nova is Nonzero**

**SKPBZ IBM1**      **Skip if Busy to Nova is Zero**

**SKPDN IBM1**      **Skip if Done is Nonzero**

**SKPDZ IBM1**      **Skip if Done is Zero**

**SKPBN IBM2**      **Skip if Busy to Channel is Nonzero**

**SKPBZ IBM2**      **Skip if Busy to Channel is Zero**

**SKPDN IBM2**      **Skip if Channel Suppress Out Condition is True**

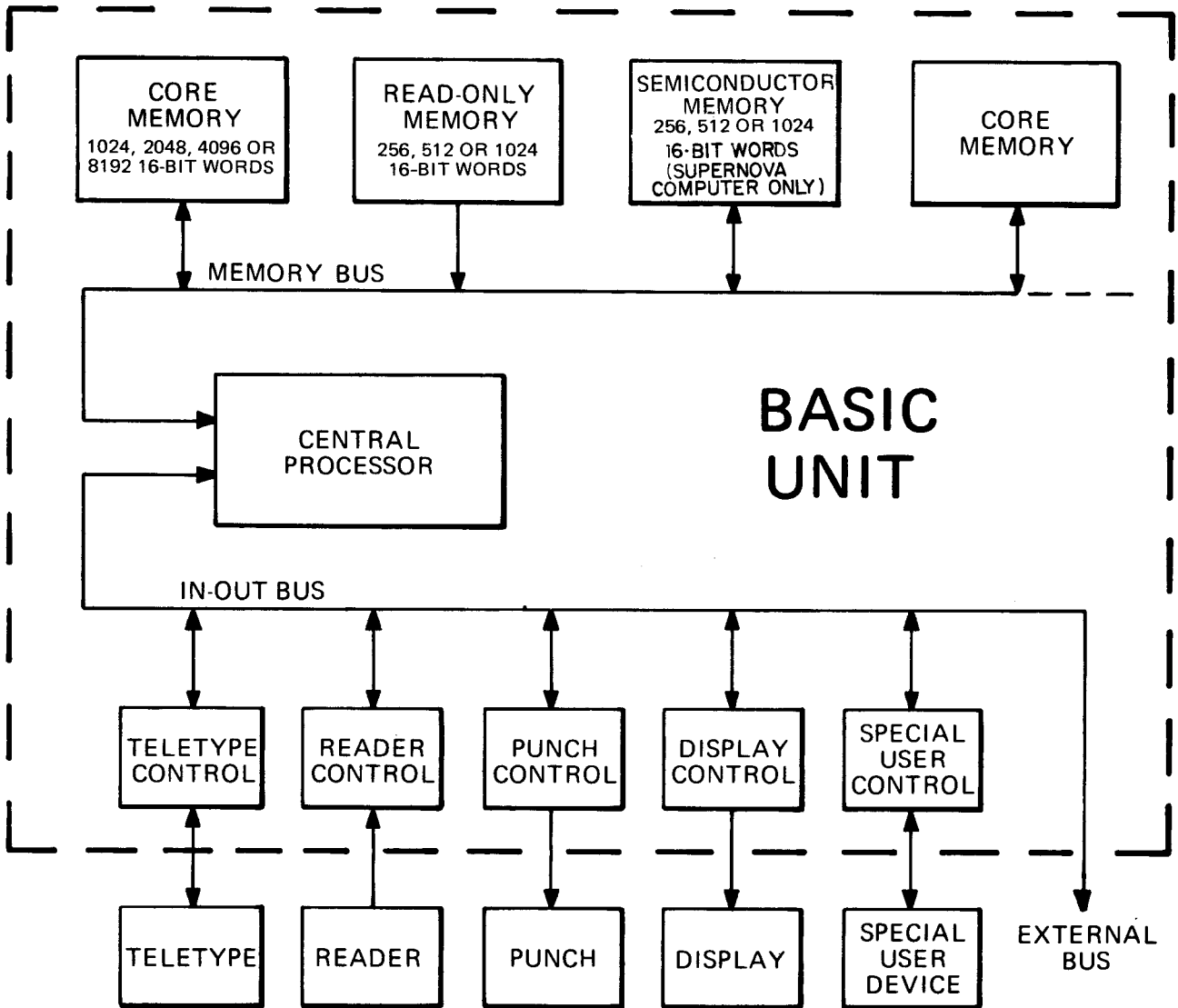
**SKPDZ IBM2**      **Skip if Channel Suppress Out Condition is False**

# Appendices

## APPENDIX A INTERFACING

§§2.3, 2.4 and 2.5 contain a general description of the entire in-out system including the program interrupt and the data channel. These sections explain in-out programming in general terms and indicate the way in which in-out instructions control the various functions involved in moving information between the accumulators and the devices. The reader should be very familiar with the contents of these three sections before he attempts to interface any equipment of his own design.

There are two types of in-out data transfer: the movement of words or characters by the program and the automatic transfer of data via the data channel. The program can handle in-out by sensing Busy or Done or by allowing the device to interrupt when it requires service. If the device is automatic, it can use data channel cycles for the transfer of data and require response by the program only for control purposes (eg when a block transfer is complete or there is some special situation, such as an error, which the program must handle).



TYPICAL NOVA COMPUTER SYSTEM CONFIGURATION

To connect to the in-out bus, every device must have certain fundamental circuit networks. Each device must have a selection net to guarantee that the device will respond when and only when its device code is given by the program, a Busy-Done net to specify the device state and request interrupts, a net to determine the interrupt priority in terms of the device position on the bus, and a net to supply the device code when an interrupt is acknowledged (INTA). If the device is connected to the data channel it must also have a circuit to request access, one to determine priority (identical to the program interrupt priority net), and one to specify the type of data channel cycle required. The standard configurations for these circuits are described in Part III.

The physical layout of the computer allows many standard and customer-designed IO interfaces to be mounted inside the basic unit, which has slots (numbered from the bottom up) for 15 × 15-inch printed circuit boards. The bottom slots are used for the central processor, and the others are wired to the memory and IO buses, except that in the larger units the top slots are wired only to the IO bus. Clearance between slots is 3/8 inch, but the clearance at the top slot is 5/8 inch to accommodate a board using oversized components. A single memory module of any size requires an entire board, but one slot may be used for several interfaces; eg the interfaces for the teletypewriter, real time clock, and high speed paper tape reader and punch are all on one board, which must be mounted in slot 3 (Supernova computer slot 4). The following gives the total number of slots, the slots required for the processor, and the slots wired for memories in the various Nova line computers and expansion chassis available for them. Memories must be mounted in the slots specified; interfaces can be mounted in any slots not used by the processor, except that slot 2 in the 1200 series can be used only for a memory or a 1200 processor option board (eg multiply-divide). Lines connect to the devices via connectors at the back of the unit. If large scale equipment requires too many external lines, the bus itself can go out through one of these connectors.

	<i>Total Slots</i>	<i>Processor Slots</i>	<i>Memory Slots</i>
1210	4	1	2-4
1220	10	1	2-8
820	10	1,2	3-8
1200	7	1	2-7
1200 Jumbo	17	1	2-12
800	7	1,2	3-7
800 Jumbo	17	1,2	3-12
Nova computer	7	1,2	3-7
Supernova computer	7	1-3	4-7
Expansion chassis			
1220, 820	10	—	—
Others	7	—	1-7

## I IN-OUT BUS

The bus consists of sixteen bidirectional data lines, six device selection lines and nineteen control lines from processor to devices, and six control lines from devices to processor. Signals on the control lines from the processor synchronize all transfers on the data lines, start and stop device operations, and control the program interrupt and data channel. Over the control lines to the processor a device can indicate the states of its Busy and Done flags and request a program interrupt or data channel access.



A signal on a control line from the processor not only specifies a particular function but also supplies all timing information needed for the execution of that function. A device control unit usually requires timing circuits for its own internal operations, but no timing functions need be performed by the circuits that connect to the bus – all such timing is supplied by the processor in the signals sent over the bus control lines. Moreover the control lines are set up so that a given device need connect only to those that correspond to the functions the device requires.

Within the basic enclosure the bus is simply printed connections from one subassembly slot to another. If the bus must run out of the basic enclosure, the external bus is in the form of a cable composed of fifty twisted pairs in a single black covering. External bus wires must be terminated at the far end to match the characteristic impedance of the cable; this allows the transmission of high speed digital pulses without reflections or ringing. The cable has very low interpair crosstalk and high surge impedance so individual twisted pairs do not require separate shields. With this system a number of bus drivers can be connected to a single data line, and data may be transmitted and received directly with ICs at distances up to 50 feet (including internal wiring) with good noise margins and low signal delays.

### Bus Signals

The binary signals on the bus have two states, low and high, which correspond respectively to nominal voltage levels of 0 and +2.7 volts. Any level between ground and .4 volt is interpreted as low any level more positive than 2.2 volts is interpreted as high. The level listed for a signal in the following table is the voltage level on the line when the signal represents a 1 or produces the indicated function. A low signal is indicated in the prints by a bar over its name.

Signal	Direction	Level	
DS0 to DS5	To device	Low	Device Selection. The processor places the device code (bits 10–15 of the instruction word) on these lines during the execution of an in-out instruction. The lines select one of 62 devices (codes 01–76) that may be connected to the bus. Only the selected device responds to control signals generated during the instruction.
DATA0 to DATA15	Bidirectional	Low	Data. All data and addresses are transferred between the processor and the devices attached to the bus via these sixteen lines.

For programmed output the processor places the AC specified by the instruction on the data lines and then generates DATOA, DATOB or DATOC to load the data from the lines into the corresponding buffer in the device selected by DS0–5, or generates MSKO to set up the Interrupt Disable flags in all of the devices according to the mask on the data lines. For data channel output the processor places the memory buffer on the data lines and generates DCHO to load the contents of the lines into the data buffer in the device that is being serviced.

For programmed input the processor generates DATIA, DATIB or DATIC to place information from the corresponding buffer in the device selected by DS0–5 on the data lines, or generates INTA to place the code of the nearest device that is requesting an interrupt on lines 10–15. The processor then loads the data from the lines into the AC selected by the instruction. To get an address for data channel access the processor generates DCHA to place a memory address from the nearest device that is requesting access on lines 1–15 and then loads the address into the memory address register. For data channel input the processor generates DCHI to place the data buffer of the device being serviced on the data lines and then loads the contents of the lines into the memory buffer.

DATOA	To device	High	Data Out A. Generated by the processor after AC has been placed on the data lines in a DOA to load the data into the A buffer in the device selected by DS0–5.
DATIA	To device	High	Data In A. Generated by the processor during a DIA to place the A buffer in the device selected by DS0–5 on the data lines.
DATOB	To device	High	Data Out B. Equivalent to DATOA but loads the B buffer.
DATIB	To device	High	Data In B. Equivalent to DATIA but places the B buffer on the data lines.
DATOC	To device	High	Data Out C. Equivalent to DATOA but loads the C buffer.
DATIC	To device	High	Data In C. Equivalent to DATIA but places the C buffer on the data lines.
STRT	To device	High	Start. Generated by the processor in any nonskip IO instruction with an S control function (bits 8–9 = 01) to clear Done, set Busy, and clear the INT REQ flipflop in the device selected by DS0–5.
CLR	To device	High	Clear. Generated by the processor in any nonskip IO instruction with a C control function (bits 8–9 = 10) to clear Busy, Done and the INT REQ flipflop in the device selected by DS0–5.
IOPLS	To device	High	IO Pulse. Generated by the processor in any nonskip IO instruction with a P control function (bits 8–9 = 11) to perform some special function in the device selected by DS0–5 (this signal is for custom applications).
SELB	To processor	Low	Selected Busy. Generated by the device selected by DS0–5 if its Busy flag is set.
SELD	To processor	Low	Selected Done. Generated by the device selected by DS0–5 if its Done flag is set.

RQENB	To device	Low	Request Enable. Generated at the beginning of every memory cycle to allow all devices on the bus to request program interrupts or data channel access. In any device RQENB sets the INT REQ flipflop if Done is set and Interrupt Disable is clear. Otherwise it clears INT REQ. In any device connected to the data channel RQENB sets the DCH REQ flipflop if the DCH SYNC flipflop is set. Otherwise it clears DCH REQ.
INTR	To processor	Low	Interrupt Request. Generated by any device when its INT REQ flipflop is set. This informs the processor that the device is waiting for an interrupt to start.
INTP	To device	Low	Interrupt Priority. Generated by the processor for transmission serially to the devices on the bus. If the INT REQ flipflop in a device is clear when the device receives INTP, the signal is transmitted to the next device.
INTA	To device	High	Interrupt Acknowledge. Generated by the processor during the INTA instruction. If a device receives INTA while it is also receiving INTP and its INT REQ flipflop is set, it places its device code on data lines 10–15.
MSKO	To device	Low	Mask Out. Generated by the processor during the MSKO instruction after AC has been placed on the data lines to set up the Interrupt Disable flags in all devices according to the mask on the lines.
DCHR	To processor	Low	Data Channel Request. Generated by any device when its DCH REQ flipflop is set. This informs the processor that the device is waiting for data channel access.
DCHP	To device	Low	Data Channel Priority. Generated by the processor and transmitted serially to the devices on the bus. If the DCH REQ flipflop in a device is clear when the device receives DCHP, the signal is transmitted to the next device.
DCHA	To device	Low	Data Channel Acknowledge. Generated by the processor at the beginning of a data channel cycle. If a device receives DCHA while it is also receiving DCHP and its DCH REQ flipflop is set, it places the memory address to be used for data channel access on data lines 1–15 and sets its DCH SEL flipflop.
DCHM0	To processor	Low	Data Channel Mode. Generated by a device when its DCH
DCHM1			SEL flipflop is set to inform the processor of the type of data channel cycle desired as follows:

DCHM0	DCHM1	
0 (H)	0 (H)	Data out
0 (H)	1 (L)	Increment memory

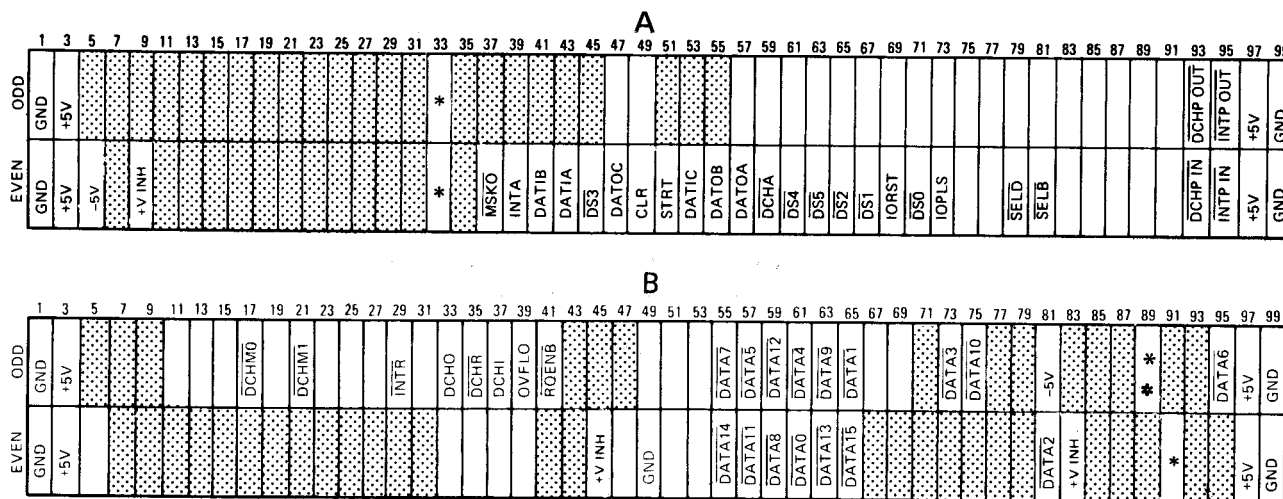
1 (L)      0 (H)      Data in  
 1 (L)      1 (L)      Add to memory

In addition to performing the necessary functions internally, the processor generates DCHI and/or DCHO for the required in-out transfers.

DCHI	To device	High	Data Channel In. Generated by the processor for data channel input (DCHM0 = 1) to place the data register of the device selected by DCHA on the data lines.
DCHO	To device	High	Data Channel Out. Generated by the processor for data channel output (DCHM0-1 ≠ 10) after the word from memory or the arithmetic result has been placed on the data lines to load the contents of the lines into the data register of the device selected by DCHA.
OVFLO	To device	High	Overflow. Generated by the processor during a data channel cycle that increments memory or adds to memory (DCHM1 = 1) when the result exceeds $2^{16} - 1$ .
IORST	To device	High	IO Reset. Generated by the processor in the IORST instruction or when the console reset switch is pressed to clear the control flipflops in all interfaces connected to the bus. This signal is also generated during power turnon.

### Interface Connectors

The back panel of the frame that holds the printed circuit boards has two 100-pin connectors for each slot. The bottom slots are wired for the central processor. The back panel connectors of the slots that can be used for memories (as indicated in the table on page A2) are identical and are wired to the



### BACK PANEL CONNECTOR LAYOUT

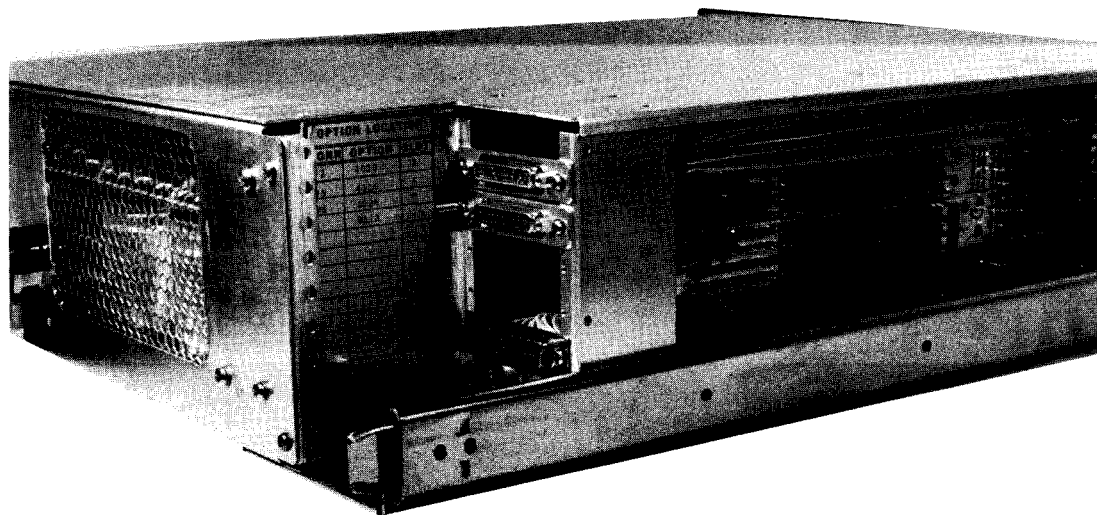
\*Memory pins on original Nova computer, but ground on all subsequent Nova line computers.

\*\*Ground on Nova 2 computer.

memory and in-out buses as shown here (viewed from the back the A connector is on the left). The named unshaded positions carry the in-out bus signals; shaded positions are used by the memory bus; blank positions indicate pins available for connections from interface to device. At slots above those for memories the shaded unlabeled positions are available for customer use, but any interface that uses them is incompatible with the standard back panel configuration and cannot be plugged into one of the slots in which those pins are not free.

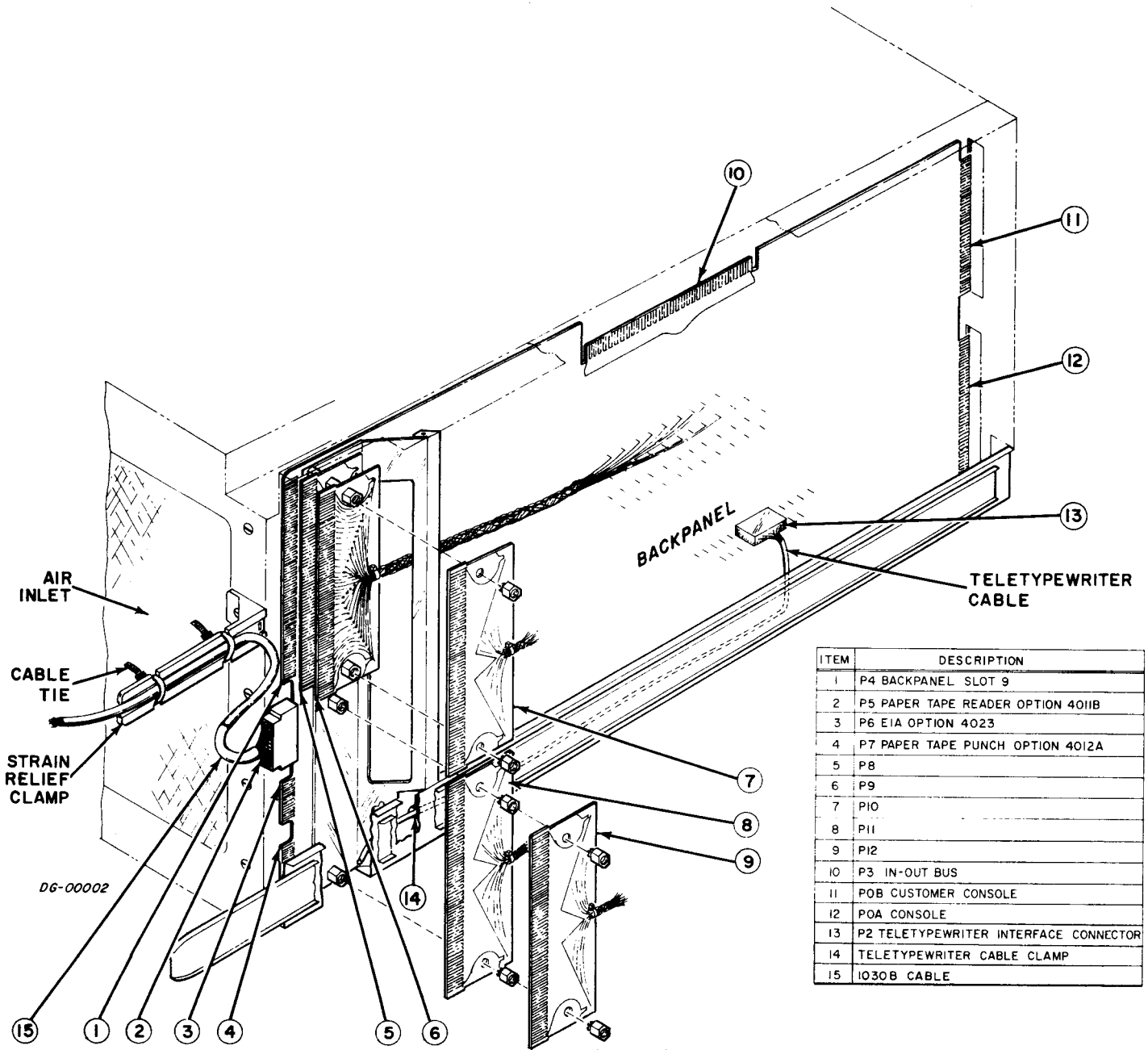
Cabling from the back panel to external equipment (other than the console teletypewriter) is made via connectors at the back of the unit. The drawings on the next two pages show the external connector layouts of the Nova 1210, 1220 and 820 computers. In these computers the teletypewriter cable is plugged directly into the back panel, and all other external cables use edge connectors. The male connectors shown as part of the back panel are wired to the slot connectors by etched tracks on the panel; additional external cables as needed match to paddle boards that are mounted parallel to the back panel at the rear and are wired to the slot connectors by internal cables. Only the connectors for teletypewriter, in-out bus, reader, punch, and EIA option 4023 are unique. All other connections from interfaces to devices are made by dual 50-pin paddle boards that are wired to the standard free connector positions at the appropriate back panel slot and have the common option number 4192. In the 1210 the rear end of the back panel is the IO bus connector (item 1); item 5 is added for connection to a reader, EIA option and/or punch (with the top and bottom groups of pins wired to slot 3); and a single 4192 (item 6) can be added as well. Built into the rear of the 1220 and 820 back panels are the connectors for the IO bus, reader, EIA option and punch. The connector on the left side of the panel (item 1) is a 4192 that is etch wired to slot 9. Up to five other 4192s can be added at the rear as shown.

The external connector layout for the Nova 1200 and 800 computers is illustrated below. The second, third and fourth holes from the left at the bottom are assigned respectively to teletypewriter, punch and reader. The teletypewriter socket is prewired to the back panel; internal cables for the



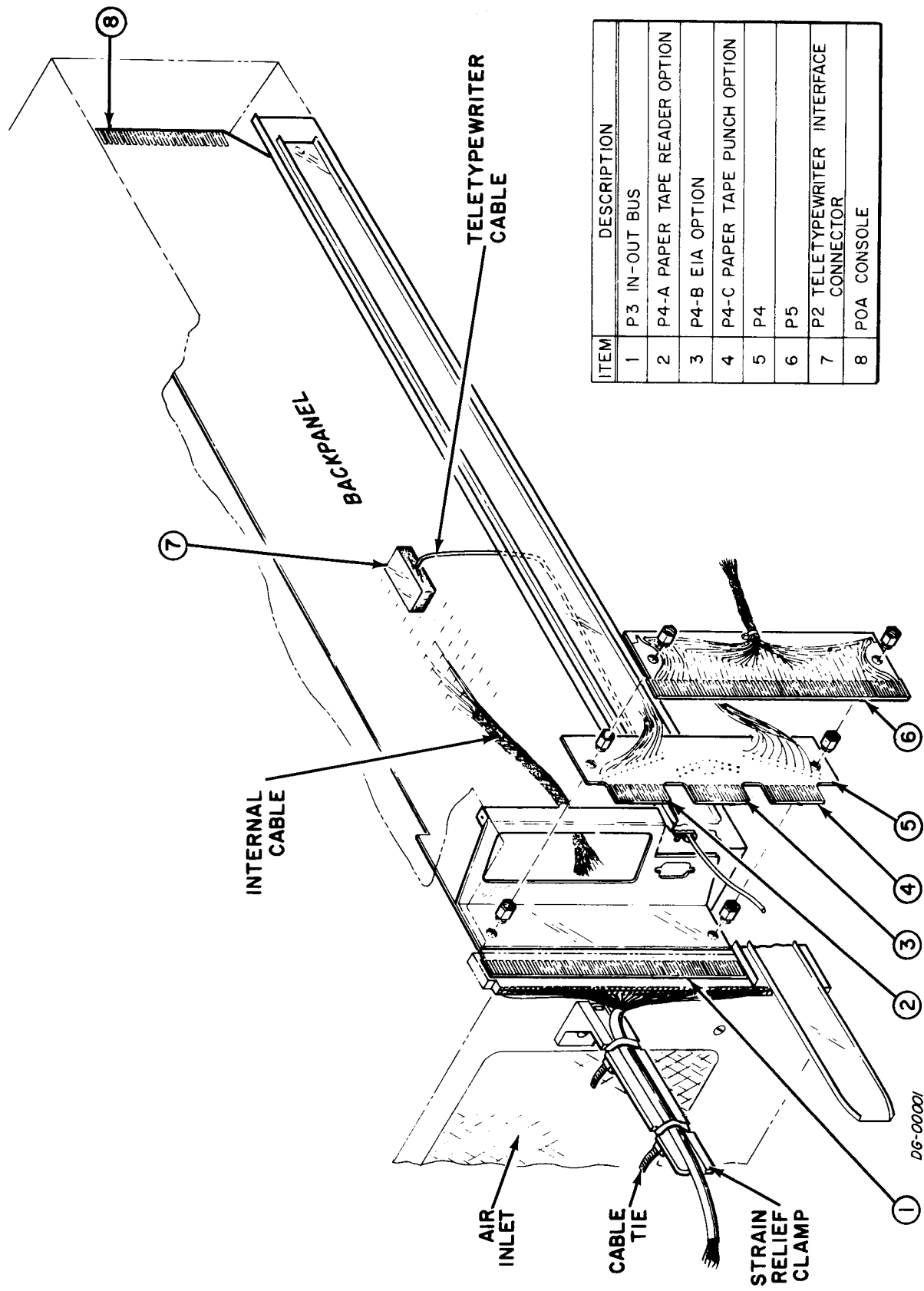
Typical 1200 and 800 External Connector Layout

reader, punch and IO bus can be connected from the external sockets to sockets mounted at the bottom of the back panel (there are three sockets for the bus, one each for the devices). To connect any other interface, run the wires from a slot connector across the back panel to the appropriate output connector (in a Jumbo chassis, sockets for fixed head disc and magnetic tape are mounted on the back panel and are etch wired respectively to slots 15 and 16).



ITEM	DESCRIPTION
1	P4 BACKPANEL SLOT 9
2	P5 PAPER TAPE READER OPTION 4011B
3	P6 EIA OPTION 4023
4	P7 PAPER TAPE PUNCH OPTION 4012A
5	P8
6	P9
7	P10
8	P11
9	P12
10	P3 IN-OUT BUS
11	POB CUSTOMER CONSOLE
12	POA CONSOLE
13	P2 TELETYPEWRITER INTERFACE CONNECTOR
14	TELETYPEWRITER CABLE CLAMP
15	1030B CABLE

NOVA 1220 AND 820 EXTERNAL CONNECTOR LAYOUT



ITEM	DESCRIPTION
1	P3 IN-OUT BUS
2	P4-A PAPER TAPE READER OPTION
3	P4-B EIA OPTION
4	P4-C PAPER TAPE PUNCH OPTION
5	P4
6	P5
7	P2 TELETYPEWRITER INTERFACE CONNECTOR
8	POA CONSOLE

NOVA 1210 EXTERNAL CONNECTOR LAYOUT

External sockets on the Nova and Supernova computers are at the rear (the teletypewriter uses the 9-pin socket in the middle). To connect the IO bus externally or to connect an interface to a device, run the wires from a slot connector down the back panel, underneath the chassis, and through the power supply to the socket at the back (wires going through the power supply must be shielded).

Order numbers and part numbers for connectors, connector parts and associated tools for all computers are listed in the table at the end of this appendix.

### External Signal Connections

On the next page is a complete IO bus signal summary. For each signal the table lists the level, direction, back panel pin and external bus connector pin. In a 1210, 1220 or 820 the signals are at pins on the lettered side, and the other wires in the twisted pairs are connected to ground on the numbered side. In all other computers this arrangement is reversed: the signals are on the numbered side, ground wires are on the lettered side.

The correspondence of free back panel pins to the pins in the standard device connector 4192 (1210, 1220 and 820 only) is as follows.

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

\*Pin 50 is connected to +5V thru a fuse.

### Bus Circuits

Signal levels on the bus are nominally 0 and +2.7 volts. Every line that a device must drive should be driven toward ground by an NPN transistor collector that is capable of sinking 90 mA and maintaining a maximum saturated output voltage of .4 volt. The DGC#100-000117 integrated circuit is especially suited to these requirements.



## IN-OUT BUS SIGNAL CONNECTIONS

H High  
L Low

B Bidirectional  
D From processor to device

P From device to processor

Signal	Level	Direction	Panel Pin	External Bus Pin	Signal	Level	Direction	Panel Pin	External Bus Pin
CLR†	H	D	A50	K 2	DCHM0	L	P	B17	d 27
DATA0	L	B	B62	w 3	DCHM1	L	P	B21	e 28
DATA1	L	B	B65	z 4	DCHO†	H	D	B33	h 29
DATA2	L	B	B82	AD 5	DCHP IN	L	D*	A94 } A93 }	b 30
DATA3	L	B	B73	AB 6	DCHP OUT	L	D*		
DATA4	L	B	B61	v 7	DCHR	L	P	B35	j 31
DATA5	L	B	B57	r 8	DS0	L	D	A72	X 32
DATA6	L	B	B95	AE 9	DS1	L	D	A68	V 33
DATA7	L	B	B55	n 10	DS2	L	D	A66	U 34
DATA8	L	B	B60	u 11	DS3	L	D	A46	H 35
DATA9	L	B	B63	x 12	DS4	L	D	A62	S 36
DATA10	L	B	B75	AC 13	DS5	L	D	A64	T 37
DATA11	L	B	B58	s 14	INTA†	H	D	A40	D 38
DATA12	L	B	B59	t 15	INTP IN	L	D*	A96 } A95 }	c 39
DATA13	L	B	B64	y 16	INTP OUT	L	D*		
DATA14	L	B	B56	p 17	INTR	L	P	B29	f 40
DATA15	L	B	B66	AA 18	IOPLS†	H	D	A74	Y 41
DATIA†	H	D	A44	F 19	IORST†	H	D	A70	W 42
DATIB†	H	D	A42	E 20	MSKO	L	D	A38	C 43
DATIC†	H	D	A54	M 21	OVFLO†	H	D	B39	l 44
DATOA†	H	D	A58	P 22	RQENB†	L	D	B41	m 45
DATOB†	H	D	A56	N 23	SELB	L	P	A82	a 46
DATOC†	H	D	A48	J 24	SELD	L	P	A80	Z 47
DCHA†	L	D	A60	R 25	STR†	H	D	A52	L 48
DCHI†	H	D	B37	k 26	Power on	+5	D		B 49

\*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.

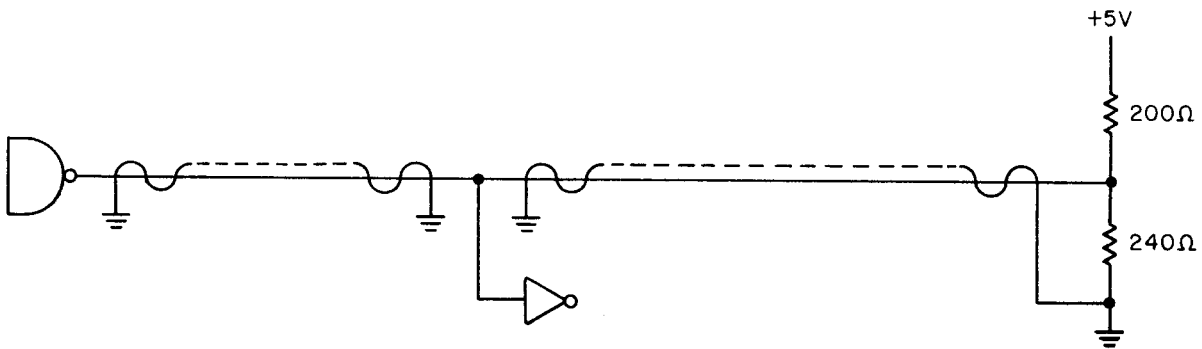
†Use filters as described in text [page A-13].

On the external bus for a 1210, 1220 or 820, pins A and AF are grounded, and the ground wires of all twisted pairs on the numbered side are connected to them. With the other computers, pins 1 thru 50 are grounded because the power on +5V is fused, it cannot be used to supply power to any external device; it is available only for picking up relays for remote power turn-on.

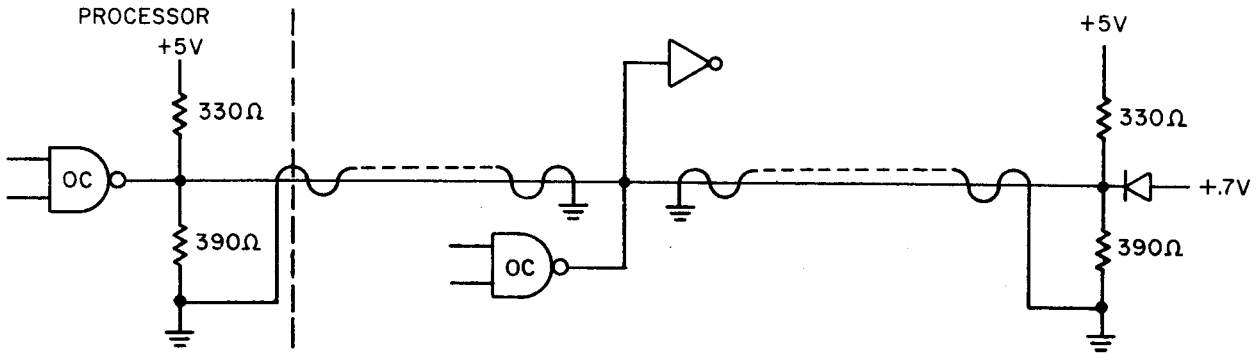
It is DGC practice to draw only one load from any signal on a single board regardless of the number of interfaces or options on that board, ie all interfaces on a board share a common set of receivers. It is strongly recommended that the customer designing his own interface equipment use the DGC#100-000117 IC for drivers and follow the DGC practice of using only one receiver for any signal on a single board.

The maximum current draw of the receivers on each line should not exceed 16 mA.

When an external bus is connected, signals that originate in the processor and drive devices must be terminated in this manner.



Bidirectional signals and signals from device to processor must be terminated this way.

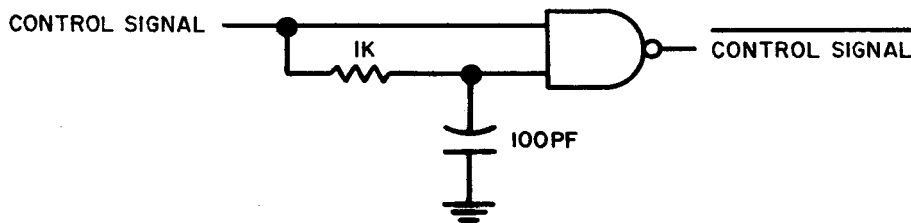


The +5 volts supplied to the external terminators should be decoupled to pins 1 and 50 and should be capable of supplying 900 ma. Use of the power-on line (pin 49) for this purpose is prohibited.

Proper termination for all forty-seven bus signals is available in DGC 1013. This part has an IO cable plug for plugging into the last device on the bus in place of the cable to another device.

The bus system is designed for a maximum length of 50 feet including signal path length within devices and inside the processor. A bus line within a device may be a single wire if it runs less than 9 inches from the IO connector. For greater distances it is good practice to run twisted pairs from the input connector to the receiver circuits and from there to the output connector.

For interfaces for any computer except the original Nova, noise margins can be improved substantially by using this filter circuit at the input to the board on the control signals indicated by a dagger (†) in the table on page A11.



The bus will drive ten of these circuits if the gate input current is 1.6 ma. Such filters are used in DGC 4040 series options [see Part IV].

## II INTERFACE TIMING

Three classes of operations take place over the in-out bus: programmed transfers (or more generally the execution of in-out instructions), events associated with requesting and acknowledging a program interrupt, and data channel transfers. Detailed relationships among the various bus signals involved in these operations are shown in a series of timing diagrams accompanying this section. In the diagrams each signal or group of signals is represented by a horizontal line with a raised section. In the case of a control signal that is generated at a specific time to control some particular function, the raised section represents the time that the function is true. For signals that carry binary information, such as the data and device selection signals, the raised section indicates the time during which that information is held on the bus. The level of a line in the diagram has no connection with the voltage level of the signal: the time that a control signal is true is represented by the raised part of the line no matter whether the signal is true when high or low. All times are in nanoseconds.

## Programmed Transfers

Throughout the duration of any in-out instruction, the processor holds the device code on the device selection lines (DS0–5) for decoding by the device.

**Data In.** The processor generates DATIA, DATIB or DATIC to place the corresponding buffer on the data lines in the device selected by DS0–5. At the end of the DATI level the processor strobes the data into the AC selected by the instruction. Following the transfer the processor generates the pulse for an S, C or P control function if called for by the instruction. In an NIO the timing of the control pulse is the same but there is no data transfer.

The acknowledgement of an interrupt is the same as data input except that INTA (which replaces the DATI level) places on the data lines the device code of the nearest device that is requesting an interrupt.

**Data Out.** While the processor places the AC selected by the instruction on the data lines, it generates DATOA, DATOB or DATOC to load the data from the lines into the corresponding buffer in the device selected by DS0–5. When the data is dropped, the processor generates the pulse for an S, C or P control function if called for by the instruction.

When using a mask to set device priorities for the program interrupt, the processor executes the same sequence as for data output but generates MSKO (in place of a DATO pulse) to set up the Interrupt Disable flags in all devices according to the information on the data lines.

**Skip.** To allow the processor to sense the state of a device, every device places its Busy and Done flags on the SELB and SELD lines whenever it recognizes its code on the device selection lines.

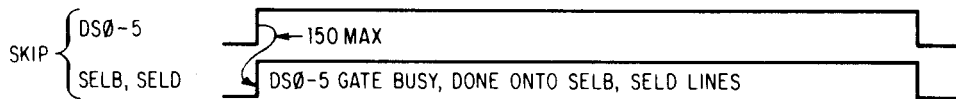
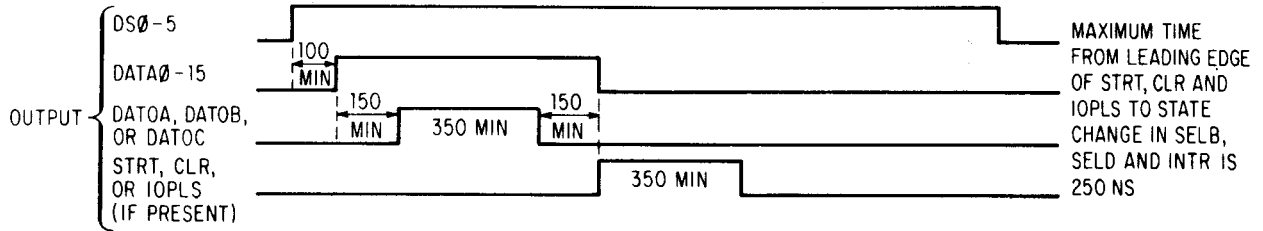
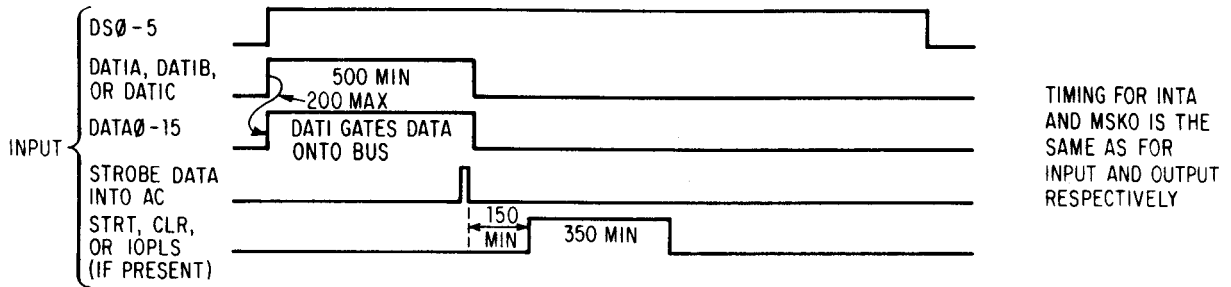
## Program Interrupt

Of the events associated with a program interrupt, many are internal to the processor and hence do not affect the bus; and others are simply straightforward applications of operations already discussed, such as sensing Busy or Done or masking out lower priority devices. There are however three sequences that must be discussed here: the interrupt request, interrupt acknowledgement (device recognition), and flag clearing.

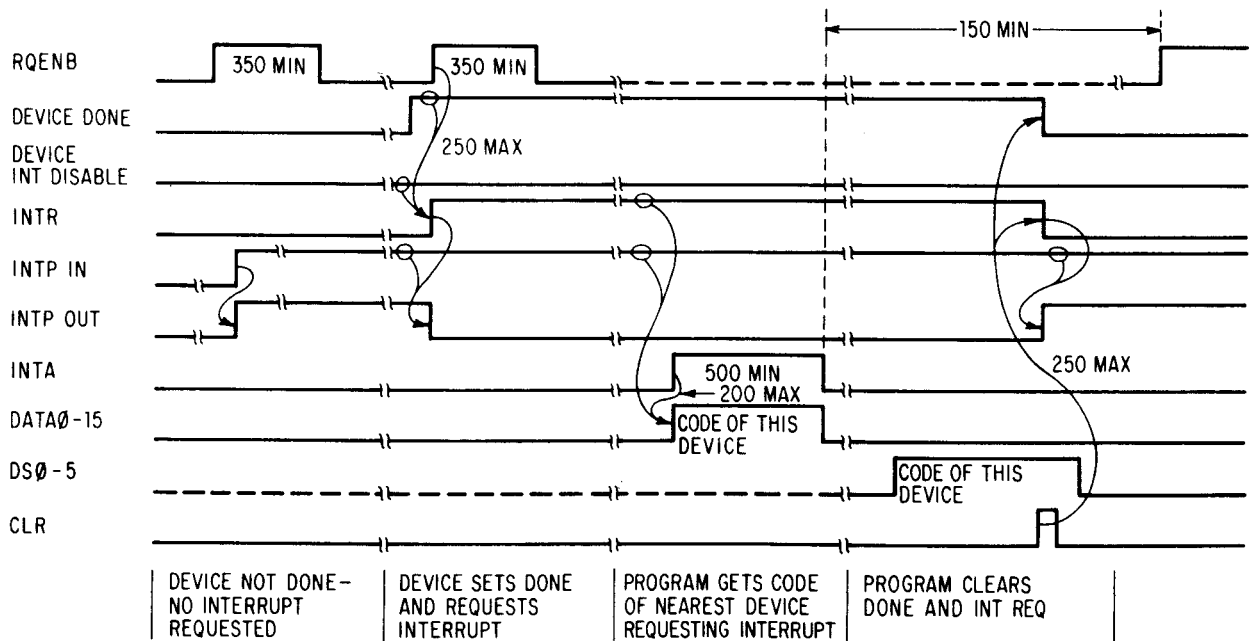
When a device completes an operation it sets Done. In every cycle the processor generates RQENB, which places the interrupt request signal INTR on the bus from a given device (*ie* sets its INT REQ flipflop) if its Done flag is set and its Interrupt Disable flag is clear. (In a complex device there may be other flags besides Done that can request an interrupt.) The leading edge of RQENB must be used to set INT REQ to ensure sufficient time for the serial INTP function to settle down before the processor attempts to discover which device has priority. A given device receives INTP IN only if there is no INT REQ flipflop set in a device closer to the processor on the bus; the INTP signal terminates at the first device whose INT REQ flipflop is set.

After an interrupt has started, the program can determine who needs service by simply sensing Busy or Done, or it may give an INTA to read the code of the nearest device that is requesting service. For the latter procedure the processor generates INTA, which places the device code on data lines 10–15 in that device that is both receiving INTP IN and generating INTR. As discussed previously, the processor strobes the data into the specified AC at the end of the INTA level.

If the program is to use the same device again, it must clear Done so the device will not immediately request an interrupt when the interrupt system is turned back on and interrupt Disable is cleared. Clearing Done also clears INT REQ, disabling INTR.



PROGRAMMED TRANSFERS (IN-OUT INSTRUCTIONS)



PROGRAM INTERRUPT

## Standard Data Channel Transfers

Timing diagrams for the four types of data channel transfer at standard speed are shown together on pages A17 and A18. Before considering the individual signals involved in these cycles it is instructive to investigate their overall structure, noting their similarities and differences. Not all events associated with a data channel transfer actually occur in the processor cycle devoted to it: there is overlap so preliminary events occur in the preceding cycle, which may be the final cycle of an instruction or another data channel cycle. In all cases a memory address is sent into the processor in the preceding cycle. For both data in and add to memory the preceding cycle is extended while the data is sent in. Transfer operations within the processor cycle officially designated as the data channel cycle for the given access occur only if a word is sent out, but this happens in data out, increment memory and add to memory.

The events associated with a data channel request are similar to those of an interrupt request. A device must have a DCH SYNC flipflop, which corresponds to the Done flag. It must also have a DCH REQ flipflop and a net for transmitting the serial priority signal to the next device, *ie* if the device receives DCHP IN and its own DCH REQ flipflop is clear, it generates DCHP OUT. The DCHP signal terminates at the first device whose DCH REQ flipflop is set. When a device requires access it sets DCH SYNC. Once this flipflop is set the next RQENB from the processor places the data channel request signal DCHR on the bus by setting DCH REQ. Synchronization must be on the leading edge of RQENB to ensure sufficient time for the serial DCHP function to settle down.

If a device is waiting for access, then after RQENB terminates in the final cycle of an instruction, the processor turns on DCHA, whose leading edge sets the DCH SEL flipflop in the nearest device that is requesting service, *ie* in that device that is receiving DCHP IN and whose DCH REQ flipflop is set. The same priority conditions place a memory address from this device on the bus for the duration of DCHA. When DCHA terminates, the processor strobes the address into its memory address register. In data in or data out the address would usually be supplied by an address counter in the device so that access is made to consecutive locations.

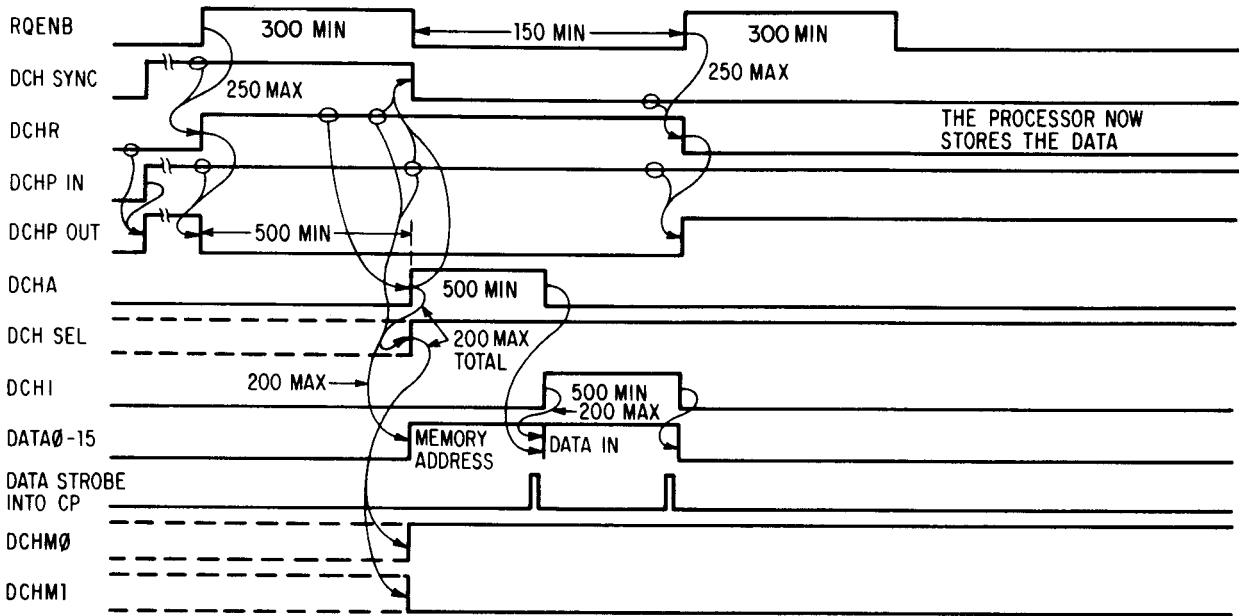
The 1 state of DCH SEL places the appropriate configuration of DCHM0 and DCHM1 signals on the bus to select the transfer mode. These signals remain on the bus as long as the flipflop remains set; but there is no conflict with other cycles, for when DCHA sets DCH SEL in one device, it clears those in all others.

The leading edge of DCHA also clears DCH SYNC. Then while the leading edge of the next RQENB is setting request flipflops in other devices, it will clear DCH REQ in this device unless this device has again set DCH SYNC and is therefore requesting access at the maximum rate.

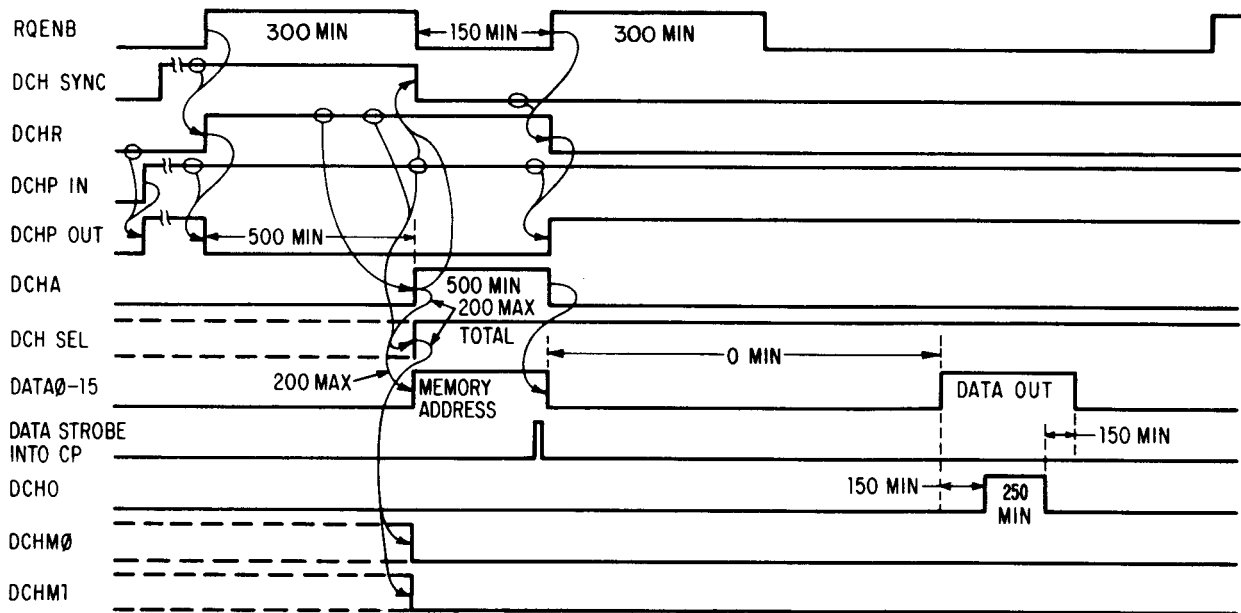
The remaining functions associated with data channel access depend on the type of transfer being made (we will first consider a single isolated request of each type).

**Data In.** As DCHA ends, the processor turns on DCHI and the final instruction cycle is extended while DCHI holds the contents of the device data register on the bus. At the end of DCHI the processor strobes the data into the memory buffer and begins the next processor cycle by generating RQENB, which turns off DCHR. During the actual data channel cycle, the processor simply stores the data in the addressed memory location.

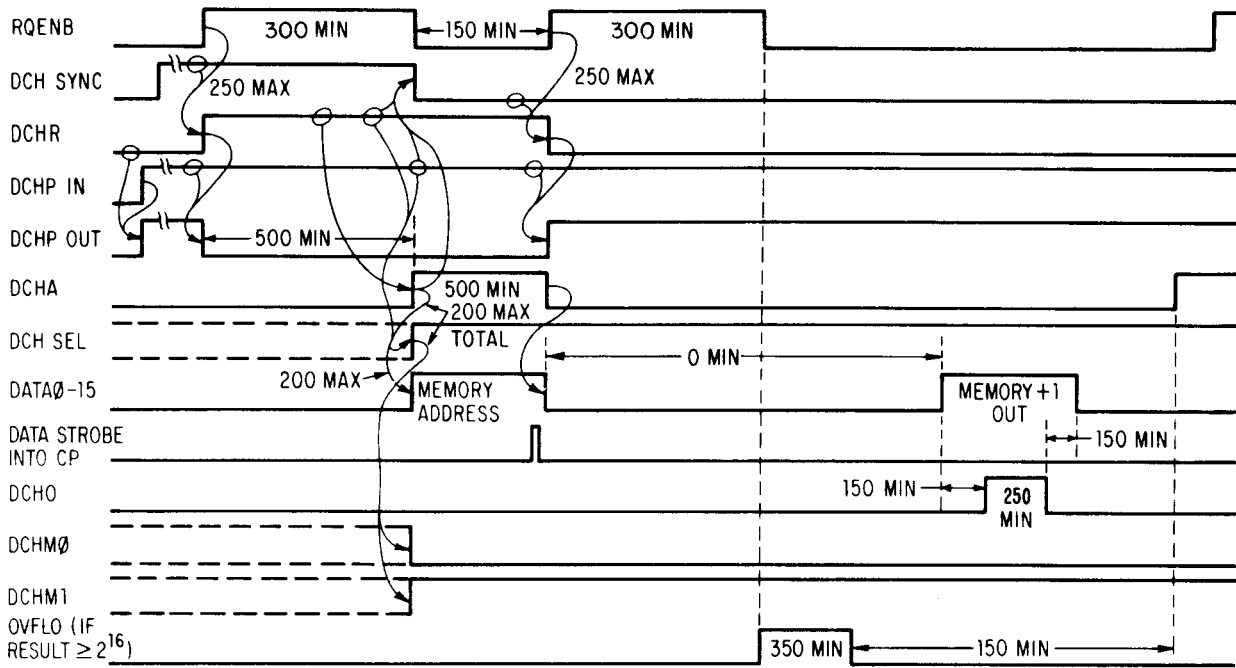
**Data Out.** At the end of DCHA the processor begins the next cycle by generating RQENB, which turns off DCHR. During this cycle the processor retrieves a word from the addressed memory location and brings it into the memory buffer. It completes the cycle by placing the contents of the memory buffer on the data lines and generating DCHO to load the word into the device data register.



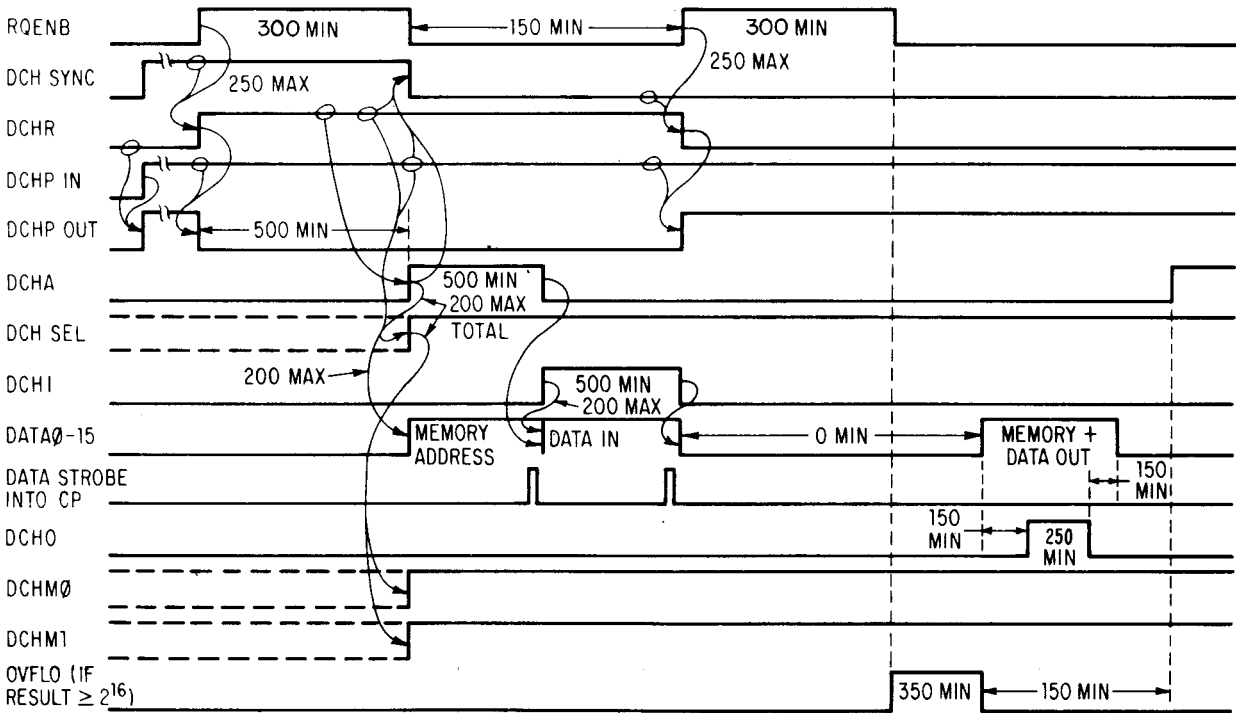
STANDARD DATA CHANNEL CYCLE: DATA IN



STANDARD DATA CHANNEL CYCLE: DATA OUT



STANDARD DATA CHANNEL CYCLE: INCREMENT MEMORY



STANDARD DATA CHANNEL CYCLE: ADD TO MEMORY



**Increment Memory.** The processor performs exactly the same operations as for data out with two exceptions: after retrieving a word from the addressed memory location, instead of writing the same word back into memory, the processor adds 1 to the word and writes the result back in memory; and if that result is greater than or equal to  $2^{16}$ , the processor sends an overflow pulse to the device at the trailing edge of RQENB.

**Add to Memory.** The processor completes the preceding cycle by performing exactly the same operations as for data in. Then during the data channel cycle it performs exactly the same operations as for data out with two exceptions: after retrieving a word from the addressed memory location, instead of writing the same word back into memory, the processor adds the data word brought in from the device to the word taken from memory and stores the result; and if that result is greater than or equal to  $2^{16}$ , the processor sends an overflow pulse to the device at the trailing edge of RQENB.

**Multiple Requests.** If several devices are requesting access simultaneously or a single device is requesting access at the maximum rate, the processor will execute a number of data channel cycles consecutively before going on to an interrupt or the next instruction. When this occurs adjacent cycles overlap in the same way that a single cycle overlaps the final cycle of the instruction preceding it. The two timing diagrams on the next page show the sequence of events in a pair of consecutive data in cycles and a pair of consecutive data out cycles. In both cases the events that occur within the final instruction cycle for the first data channel cycle also occur within the first data channel cycle for the second.

If the DCH SYNC flipflop in the device that is being serviced is clear at the leading edge of RQENB in the data channel cycle, then RQENB clears DCH REQ in that device. But if DCH SYNC is already set again, DCH REQ simply stays set, making a second request. In either case RQENB sets the request flipflops in any other devices that require service.

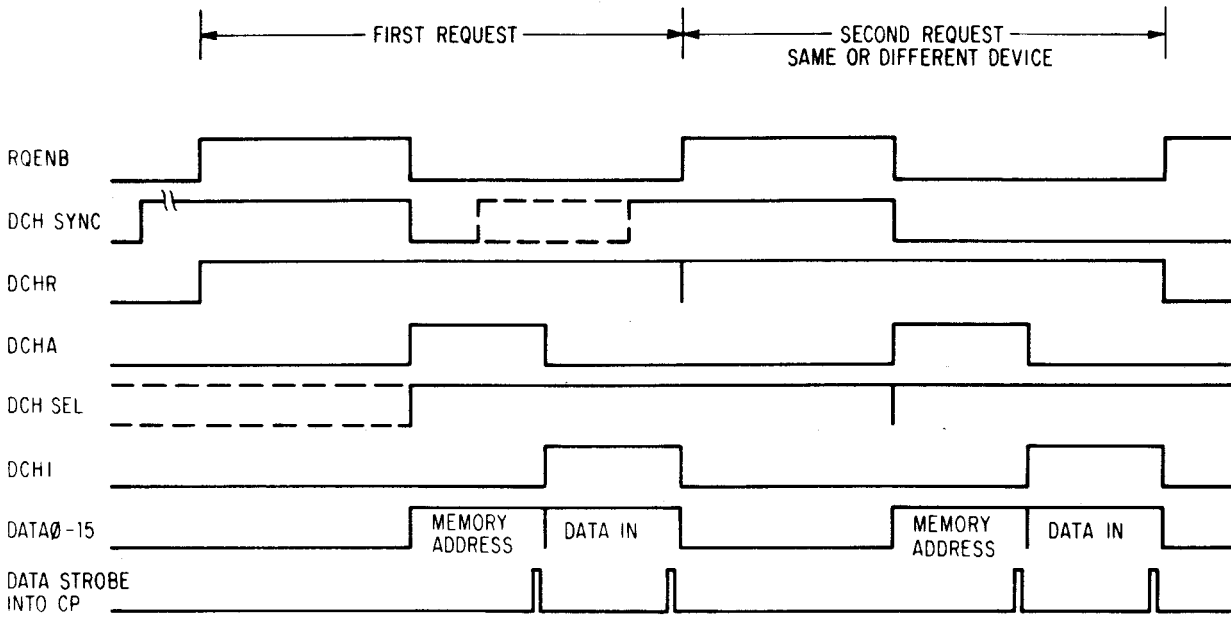
If there is a second request from any source, the processor generates a second DCHA after completing whatever operations are necessary for the first access. DCHA thus occurs at the end of RQENB for data in, but following the output of data for any other mode. This second DCHA sets the DCH SEL flipflop in the device that now has priority (clearing all others) and initiates whatever other operations are necessary to prepare for the second transfer.

### High Speed Data Channel Transfers

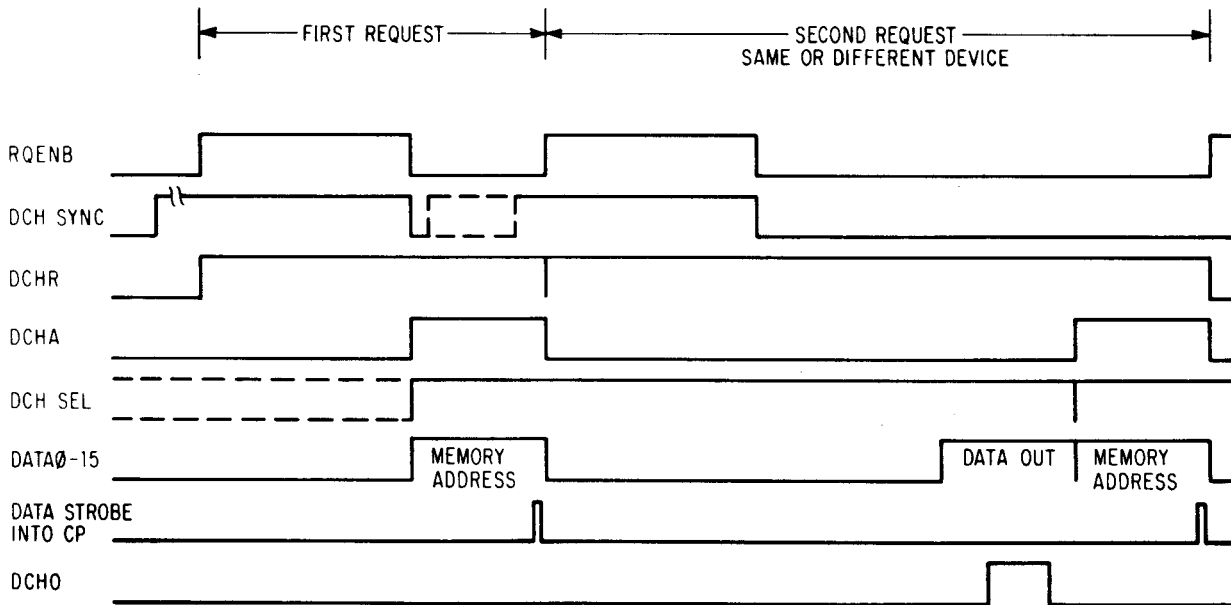
On page A21 are timing diagrams of the high speed data channel cycles for data in and data out. Note that the sequence of IO bus operations is the same as for standard transfers, but signal durations and required device response times are generally shorter. Many of the high speed times are given as typical, and these are approximately the times the designer should assume and use. Times listed as maximal are especially critical; *eg* once RQENB goes on, DCHR must be returned within 75 ns for a transfer to be executed at the high speed. Timing differentials between standard and high speed for the other types of data channel cycles are the same as for those shown except as noted in the diagrams.

All device interfaces that use the high speed capability must be mounted in the main frame or an expansion chassis and must be grouped at the processor end of the bus. The DCHP OUT signal out of the last high speed interface must be connected not only to the next device on the bus but also to pin 1A95 in the 800 series, 5A12 in the Supernova computer. This connection defines the two classes of interfaces: all interfaces on the bus before the return point operate at high speed, all beyond it at standard speed.

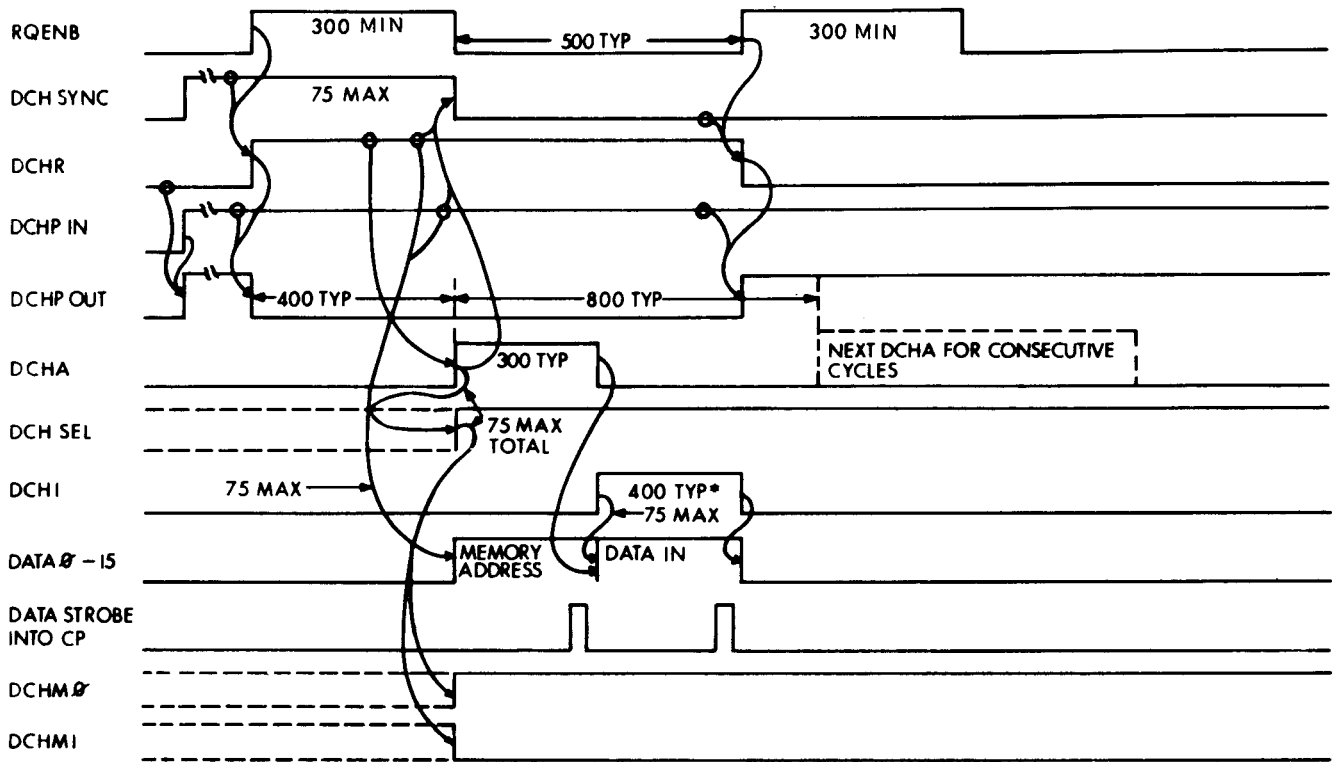
A device having a data rate slightly lower than the maximum can be synchronized to the high speed channel. Each time RQENB is generated, the device must respond by returning DCHR and simultaneously grounding the WAIT signal in the high speed logic (pin 1A90 in the 800 series, 5A63 in



CONSECUTIVE DATA CHANNEL CYCLES: DATA IN

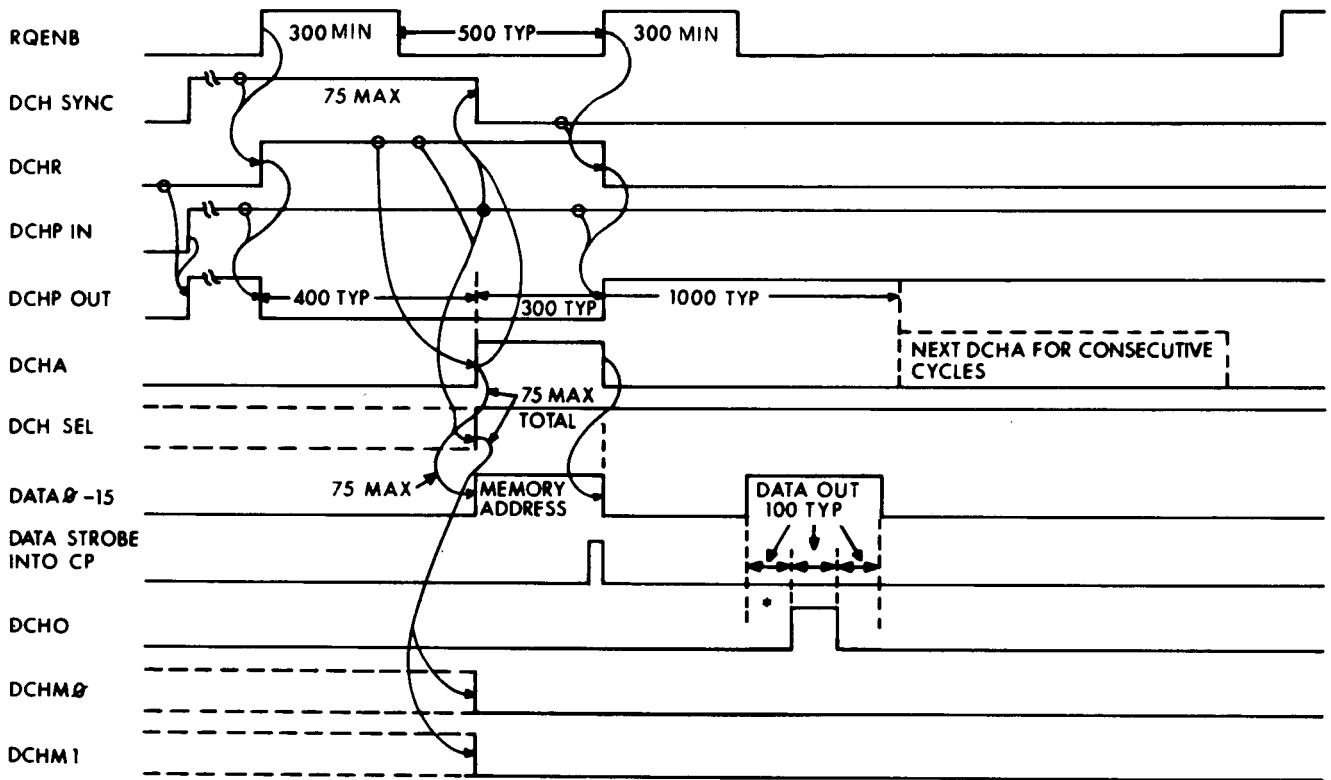


CONSECUTIVE DATA CHANNEL CYCLES: DATA OUT



\*600 TYPICAL FOR ADD TO MEMORY

### HIGH SPEED DATA CHANNEL CYCLE: DATA IN



\*300 TYPICAL FOR INCREMENT AND ADD TO MEMORY

### HIGH SPEED DATA CHANNEL CYCLE: DATA OUT

the Supernova computer). Then the device actually initiates the transfer by returning WAIT to the high state. Through this procedure the device effectively takes complete control of the processor, timing and executing transfers by controlling WAIT. Although such operation completely shuts out both the program and other channel service, it eliminates the need for multiple buffering and is particularly useful for handling small bursts of words at high speed.

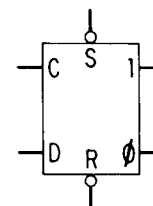
### III DESIGN OF INTERFACE EQUIPMENT

The logical and physical organization of the Nova computers with their in-out buses makes the design and installation of interfaces for user equipment especially simple and convenient.

#### Basic Interface Networks

The networks discussed here are for use at a relatively basic level; eg the data channel request networks only for isolated transfers – it cannot gain consecutive cycles.

Control flipflops used in device interfaces have a clock input, a synchronous data input, asynchronous set and reset inputs, and complementary outputs. A positive transition at C sets the flipflop if D is high, clears it if D is low. In the set state the flipflop 1 output is high, the 0 output is low. In general the D input must reach a steady state some given time (typically 20 ns) before the positive transition at C. The outputs reflect the new state typically 30 ns later.\* A ground level at S or Rsets or clears the flipflop respectively, and these inputs take precedence over the clock input. A small circle drawn at the D input means the flipflop is set when D is low, cleared when D is high. A typical control flipflop suitable for device interfaces is DGC#100-000017.



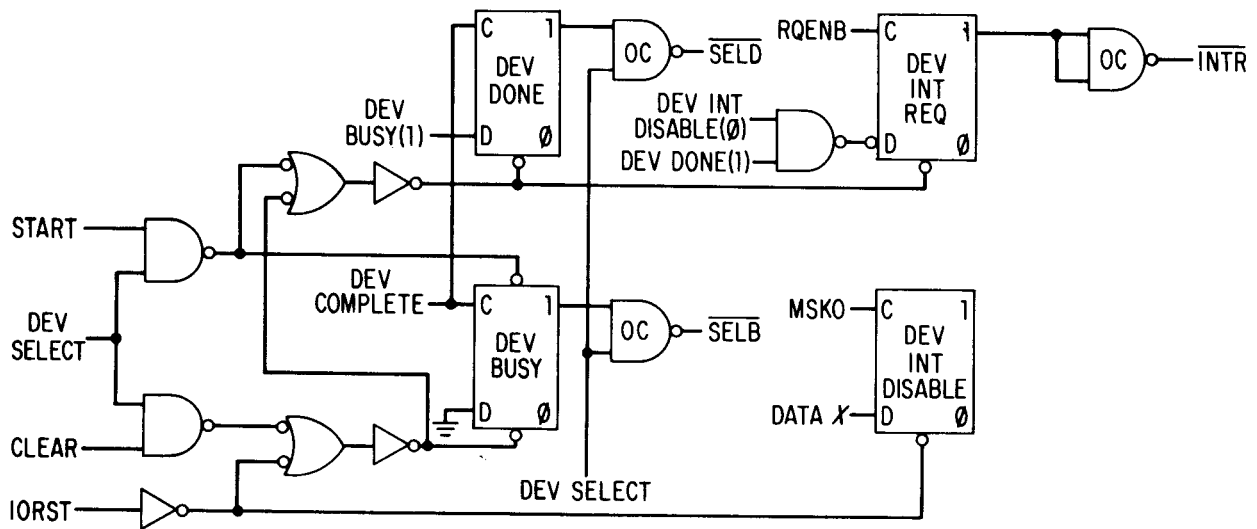
Every device must decode the device selection lines to generate a select level that ensures that only the single addressed device responds to the program. Decoding is performed by a simple NAND gate, but since the device selection lines provide only one polarity, the inputs to the gate must be inverted for all device code bits that are 1s.

The network that specifies the state of the device and requests interrupts contains four flipflops, BUSY, DONE, INT DISABLE and INT REQ. The IO reset for all devices clears all of these flipflops directly. With exception of the general reset, INT DISABLE is controlled exclusively by a particular bit of the mask in an MSKO. Signals generated by the control function part of an IO instruction affect the flipflops only if the device has recognized its device code on the selection lines. The clear pulse clears all but INT DISABLE; the start pulse clears DONE and INT REQ, but sets BUSY to place the device in operation. Whenever this device is selected, the states of BUSY and DONE are placed on the SELB and SELD lines.

When the device completes its operation it generates a completion signal that clears BUSY and sets DONE. The signal need not act on both flipflops directly; it can just as well clear BUSY whose state change sets DONE, or set DONE whose state change clears BUSY. Note that the completion signal is guaranteed to set DONE only if its data input is independent of any logic signal, eg if D is held at +3 volts. In the configuration shown here the input is BUSY(1), so the completion signal will not set DONE if the program has cleared BUSY.

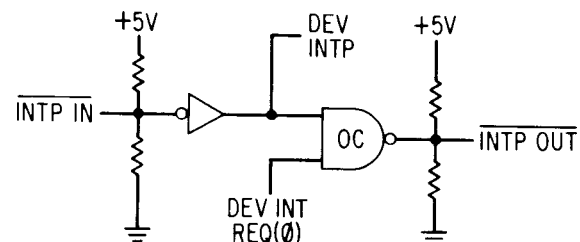
---

\*Of course, any interface must be designed for the worst case of the components being used.



Once DONE has been set, and provided INT DISABLE is clear, the leading edge of the next RQENB signal from the processor sets INT REQ, whose 1 state puts the INTR request signal on the bus. RQENB is generated in every processor cycle, and as soon as either INT DISABLE is set or DONE is cleared, the next RQENB clears INT REQ, dropping the request. In designing an interface do not attempt to do without any of these flipflops if the device is connected to the interrupt. There is no redundancy between DONE and INT REQ. Because of the serial nature of the priority-determining signal on the bus, it is essential that request signals be synchronized by the processor. Hence DONE must not generate INTR directly. Moreover INT REQ must change state only on the leading edge of RQENB in order to ensure sufficient time for request acknowledgement to work properly. Remember that although INTA may occur several cycles later than the request signal that causes an interrupt, nonetheless the timing is still critical because RQENB occurs in every cycle, and the device that has priority when INTA is given may not be the same device that caused the interrupt initially.

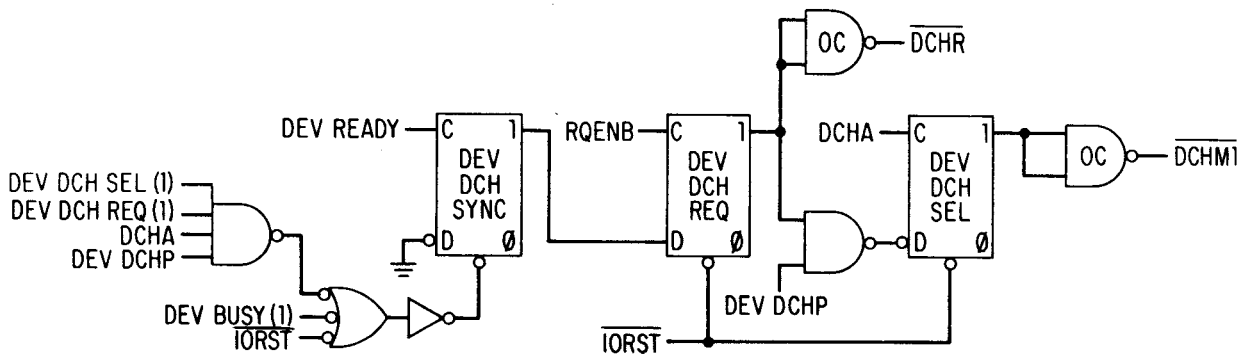
If the INTP line into a device is low, the device generates a high INTP signal for its own internal use; if its INT REQ flipflop is clear it also transmits a low INTP signal out to the next device. The terminators shown here are necessary only where the signal goes in and out of the board. Usually several interfaces are on a single board, and the single stage shown here is replaced by a chain with terminators at each end. If the system contains a large number of interfaces, timing becomes critical and the chain on a board should be replaced by two circuits, one of which establishes the priority among the devices on the board, while the other quickly passes the signal along the bus if no device on the board is requesting an interrupt. Note that if a board is removed, the input and output pins must be jumpered to maintain the continuity of the signal on the bus.



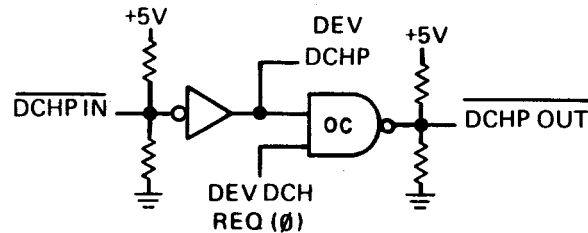
Associated with the priority circuit is a logic net that places the device code on data lines 10-15 when INTA is true, INT REQ is 1, and INTP IN is true at this device. Drivers need be used only to place 1s on the bus; a data line is automatically 0 (high) if no driver is attached.

For handling data channel requests a device must have a network containing three flipflops, DCH SYNC, DCH REQ and DCH SEL. When the device requires access it sets DCH SYNC, whose 1 state allows the leading edge of the next RQENB to set DCH REQ. As in the case of an interrupt request,

every device connected to the data channel must contain both of these flipflops as no asynchronous requests can be allowed; DCH REQ must change state only on the leading edge of RQENB. Associated with these flipflops is a priority circuit which is identical to the interrupt priority circuit, but which



passes DCHP if DCH REQ is 0. When this device has priority (*ie* when DCHP terminates here) the 1 state of DCH REQ allows the next DCHA to set DCH SEL. The 1 state of this flipflop generates the mode signals to inform the processor of the type of transfer desired. Note that since drivers are required only for 1s on the mode lines, two drivers are necessary only for add to memory, and none are required for data out (the circuit illustrated above requests access to increment memory).

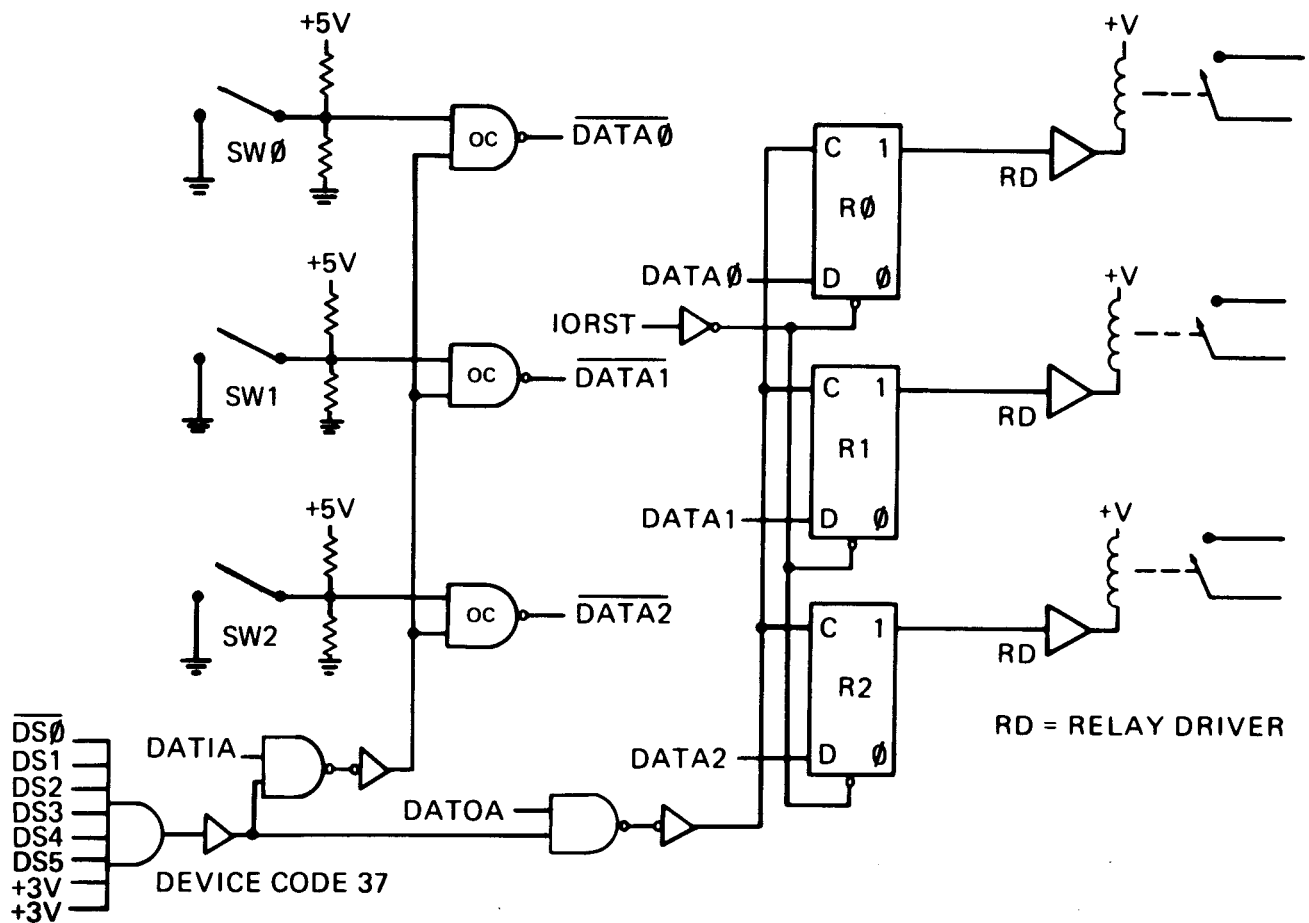


During the cycle in which this device is being serviced, DCHA also clears DCH SYNC so it is available in case the device wishes to set it again to request another transfer. Note that clearing BUSY also clears DCH SYNC so that no device can belatedly gain access after the program has turned it off.

### Design Examples

Consider first a very simple device that is connected to neither the program interrupt nor the data channel and thus needs no flags at all. Such a device is the one illustrated here, which allows the program to read three switches and to control three relays through a buffer. Of the basic circuits discussed above, the only one this device has is the NAND gate to decode the device selection lines. Giving a DIA with device code 37 loads the contents of the switches into the left three bits of the selected AC (an open switch is read as a 1). 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. Giving a DOA with device code 37 loads the left three bits of the selected AC into the relay buffer to control the three relays (a 1 from AC closes the relay contacts). Initially the contacts are open as the buffer is cleared by IORST. If the device contained only the switch register or the relay buffer, we would need only a single input gate, as the data transfer signal from the processor could replace one of the constant inputs to the device selection decoder.

Note that 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 that draws only one load from any given bus line. In other words all the interfaces on the board share the same set of receivers. 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.

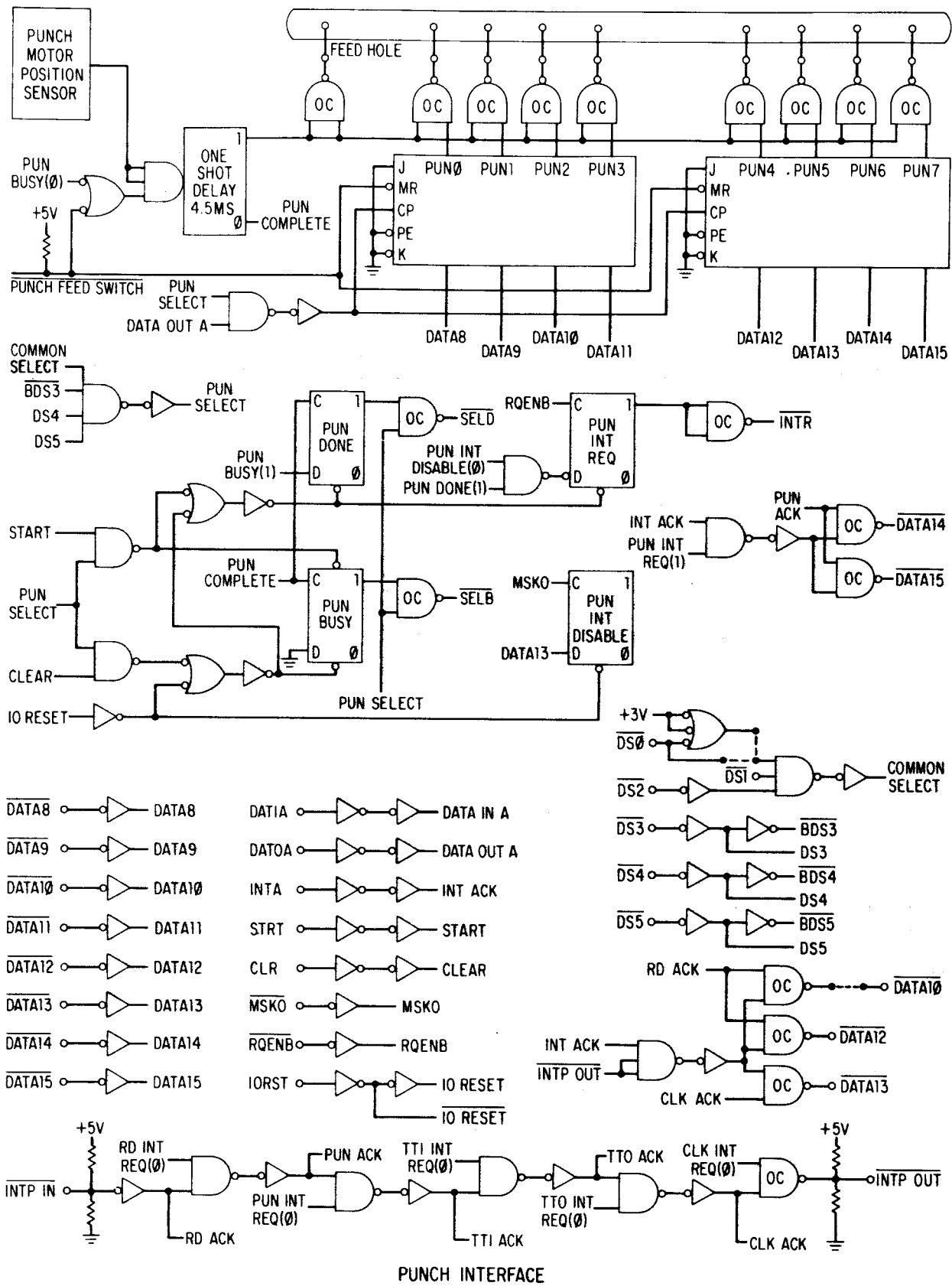


SWITCH REGISTER, RELAY BUFFER

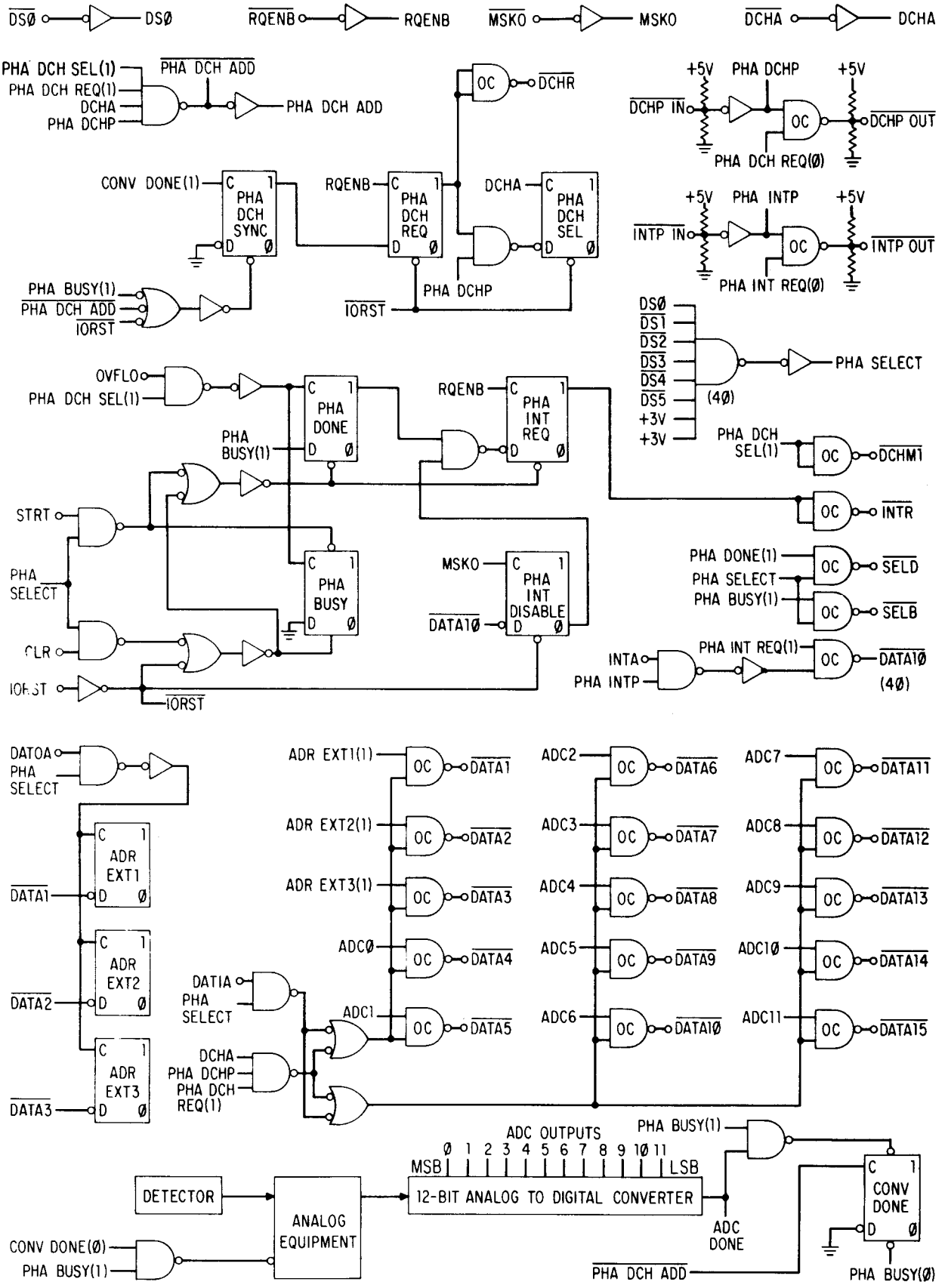
**Example: Punch.** The interface for the high speed paper tape punch, which is illustrated on the next page, shares a single board with the interfaces for the teletype, tape reader and real time clock. The lower half of the drawing contains circuits for functions common to all the interfaces. At the left are the receivers for the data lines and other signals. At the right are nets that generate a common select signal by decoding DS0–2 and generate common device code digits for INTA. The codes are 10–14 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 50–54 instead.) Across the bottom is a chain that receives INTP IN, generates an individual acknowledgment signal for each device, and passes the priority signal along the bus only if no device on the board has an INT REQ flipflop 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 DS3–5 gated by the common select level. At the right is the net that places bits 14 and 15 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 above (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 DOA that selects the punch (eg DOA AC,PTP) loads the buffer from bits 8–15 of an AC. If BUSY is set, the advent of the proper position in the punch operating cycle triggers a one-shot that allows 1s in the punch buffer to drive the lines to the punch for 4.5 ms. Note that the leftmost driver always goes on – it punches the feed hole. The termination of the delay generates a completion signal that clears BUSY and sets DONE.







PULSE HEIGHT ANALYZER INTERFACE

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 (*ie* tape with only feed holes punched.)

**Example: Pulse Height Analyzer.** The interface shown on page A27 uses the data channel as well as programmed transfers. Its function is to increment the word in the memory location whose address is equal to 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 net, the single driver required for the data channel mode lines, and the net that supplies the device code for an interrupt acknowledgement. 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 supplies 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 gives a 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 leading edge of the next RQENB sets DCH REQ to generate DCHR. When DCHA turns on and this device has priority, DCH SEL is set, generating 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 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 a DIAS AC,40 which reads and restarts), and it can stop the process at any time by giving an NIOC to clear BUSY.

#### IV CONSTRUCTION OF INTERFACE EQUIPMENT

Data General supplies various types of hardware to facilitate the connection of customer designed and built interfaces to the IO bus. The most extensive is the general purpose interface, which contains all of the ordinary circuitry needed for any interface (this is described below). Other items are more primitive and allow for greater flexibility in organization and arrangement of components.

The illustration on the next page shows the layout and dimensions of a standard 15 X 15 circuit board. Following that is an illustration of a standard pc board with a hole pattern for mounting 155 integrated circuits of the dual inline type with 14 or 16 pins. The middle three rows have wider spacing to accommodate twenty-three 24- or 36-pin packages (each pair of the larger packages replaces A28

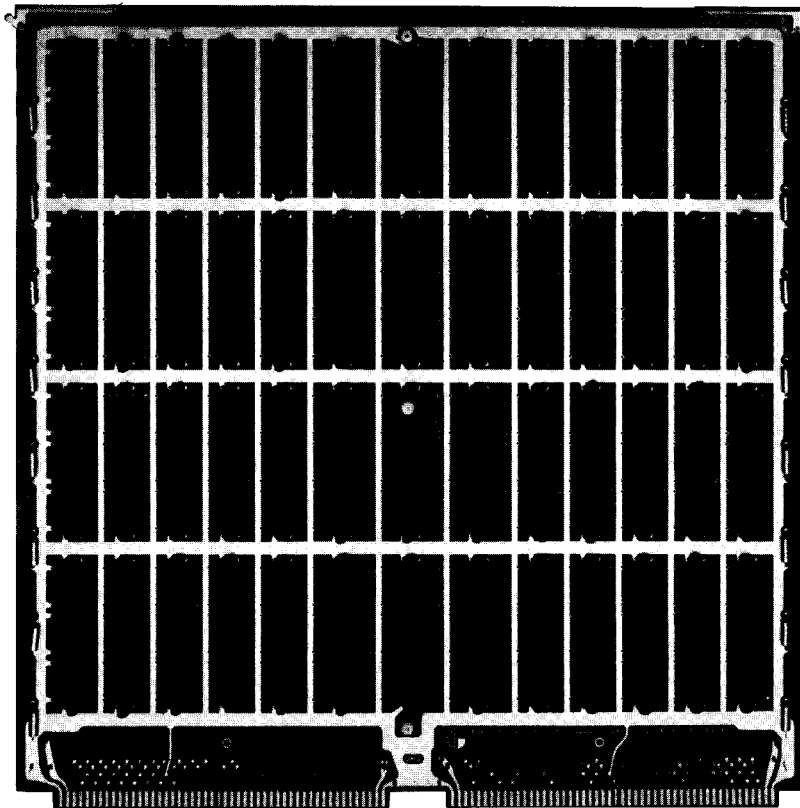
three of the smaller). Next to each hole for an IC pin is a pin for wire wrap connected to it by printed circuit wiring. The board is also available with sockets for 14- and 16-pin ICs or with neither sockets nor pins.

<i>General purpose wiring item</i>	<i>DGC order number</i>
Blank board	1021
Board with pins only	1022
Board with pins and 155 sockets	1023
Protective cover	1024

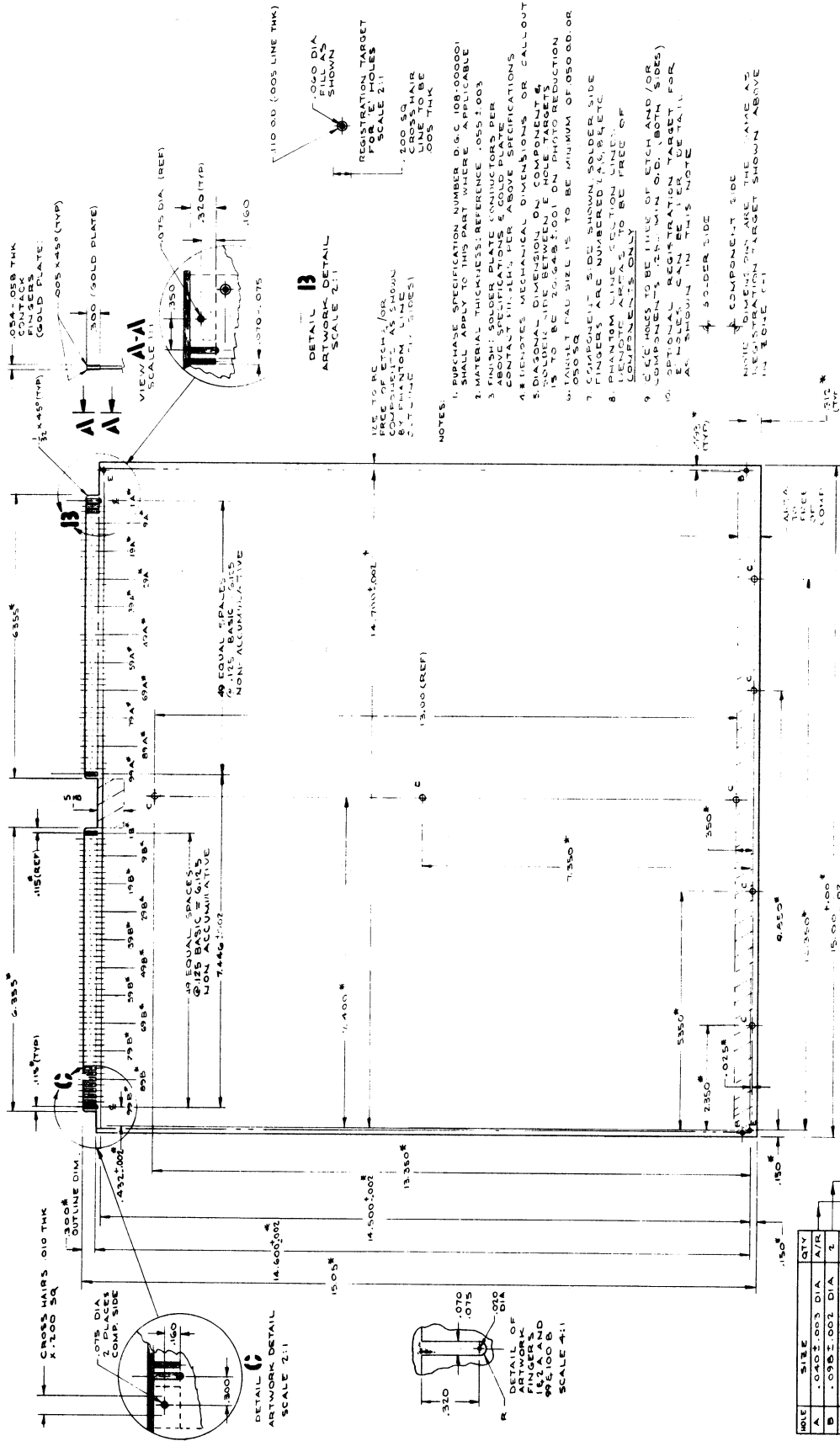
To construct and install an interface simply insert the ICs into the board, wire wrap the board pins to each other and to the connector pins as required, and insert the board in a vacant slot. Blank boards must be used for 24- and 36-pin ICs.

The sockets for the board are DGC#111-000030. Wire wrap pins are available on a reel, DGC#111-000031; DGC#109-000036 30 AWG wire wrap wire is recommended. The following items are also available:

128-000070	Wire wrap gun
128-000061	Wire unwrap tool, 30 AWG
128-000066	Wire wrap 30 AWG bit
128-000065	Wire wrap sleeve, 30 AWG



1021 BLANK BOARD



**CIRCUIT BOARD SPECIFICATIONS**

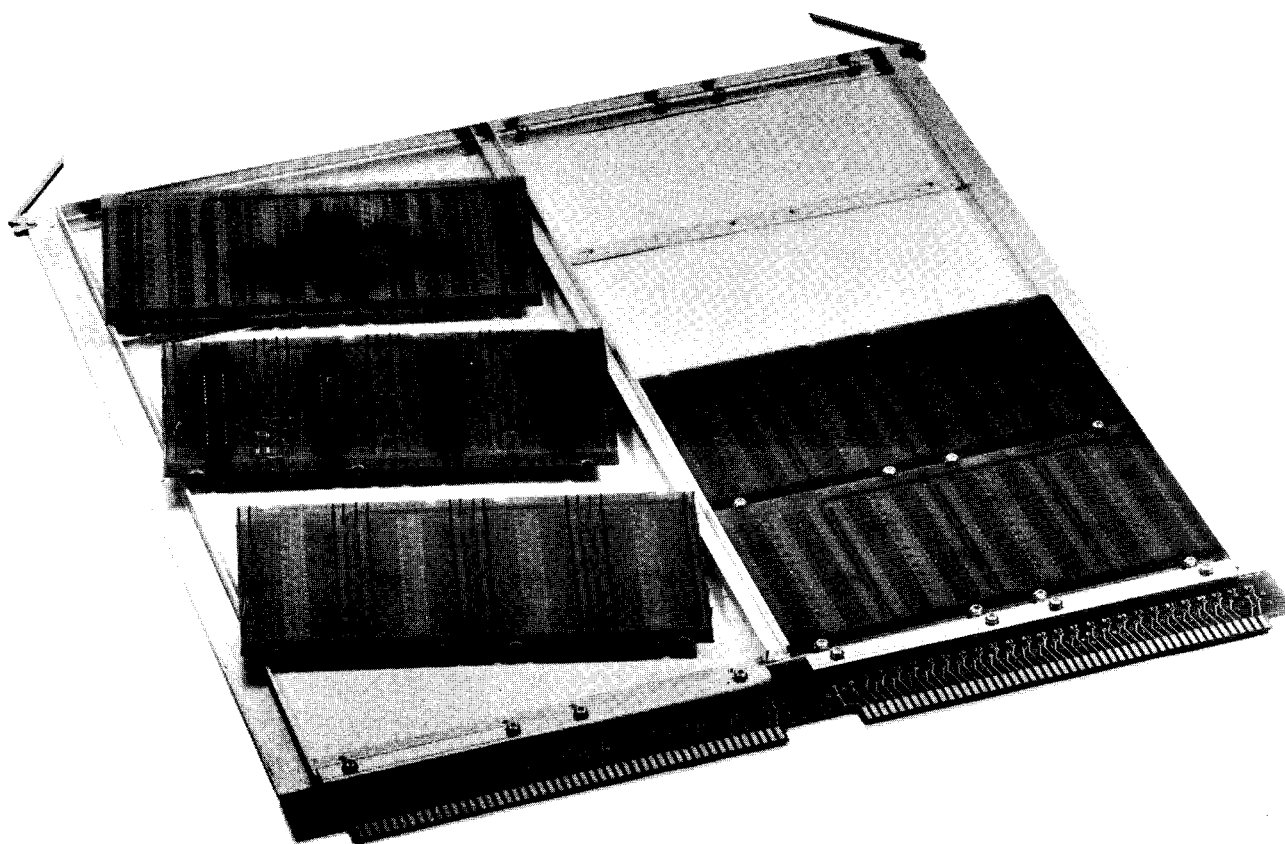
MAXIMUM HEIGHT OF COMPONENTS ABOVE THE BOARD = .31 INCHES, MAXIMUM PROTRUSION BELOW THE BOARD = .05 INCHES.

HOLES NOT PLATED THRU COMPONENT HOLES LOCATED BY ARTWORK

Also available is a special subassembly frame with two 100-pin connectors on one edge. This frame occupies one slot, and eight  $6\frac{1}{2} \times 3\frac{1}{4}$ -inch boards can be mounted on it. These small boards are available in the same configurations as the large boards discussed above. Each can hold twelve ICs with 14 or 16 pins or eight with 24 pins.

<i>Subassembly item</i>	<i>DGC order number</i>
Frame	1001
Blank board	1002
Board with pins only	1003
Board with pins and 12 sockets	1004
Protective cover (for entire frame)	1014

To use the subassembly accessories for an interface insert the ICs into the small boards, attach the boards to the frame (four screws per board), wire wrap the board pins to each other and to the connector pins as required, and insert the frame in a vacant slot. Blank boards must be used for 24-pin ICs.

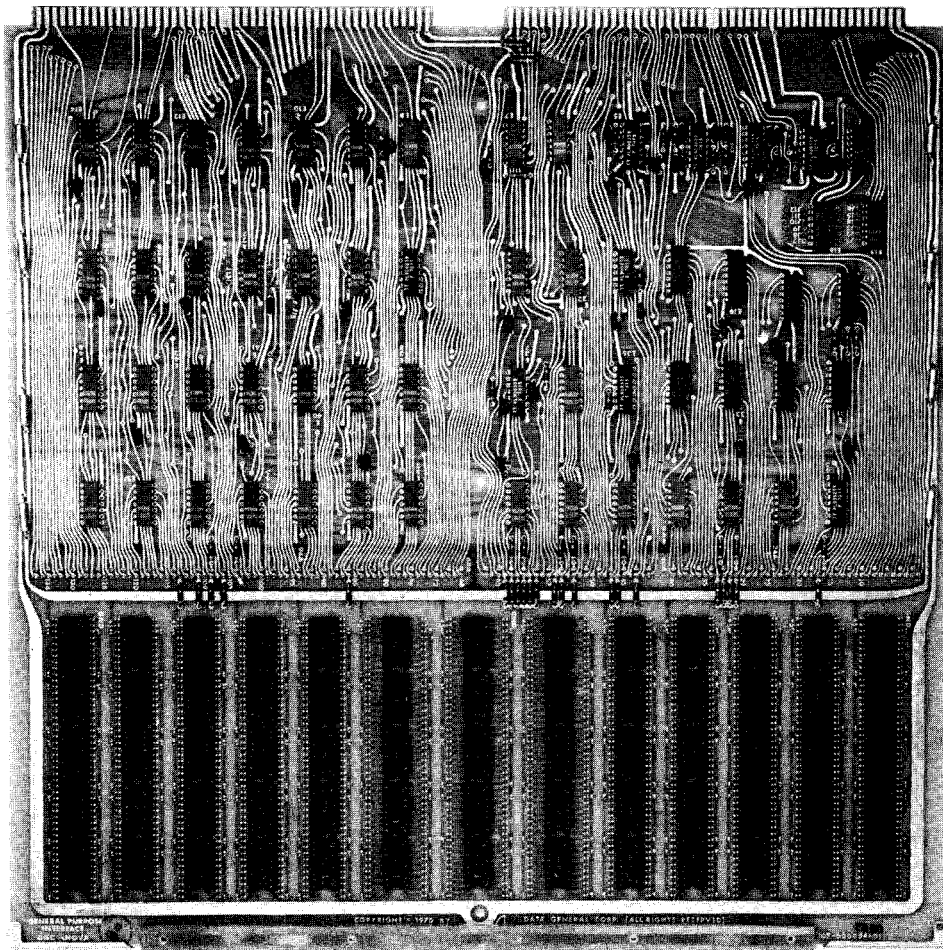


Subassembly Accessories

## General Purpose Interface

To further facilitate the addition of special peripheral equipment to the system, DGC has available a general purpose interface that includes all of the ordinary circuitry needed to connect a device to the IO bus. This interface is mounted on a standard 15 × 15-inch board that has two 100-pin connectors along one edge and is divided into two parts by several rows of wire wrap pins. These pins number some 200, of which forty-eight are wired to edge connector pins corresponding to unused positions on the computer back panel. One part of the board is reserved for customer logic, and is configured for mounting sixty-five 14- or 16-pin ICs (sockets and extra wire wrap pins are also available). The other part contains the DGC logic with various points connected to the wire wrap pins or edge connector pins as appropriate.

Drawings on pages A34–A38 show the logic in the interface (wire wrap pins are indicated by squares, connector pins by circles). The basic interface, option 4040, includes the board as described above and those basic interface networks described in Part II that are necessary for handling almost any device. Two other options are also available, either or both of which can be mounted on the same board. 4041 is a pair of data registers; 4042 is the logic necessary for connecting to the data channel (the control parts of this logic are also described in Part II). Tables referred to in the text are grouped at the end of this section.



General Purpose Interface Board

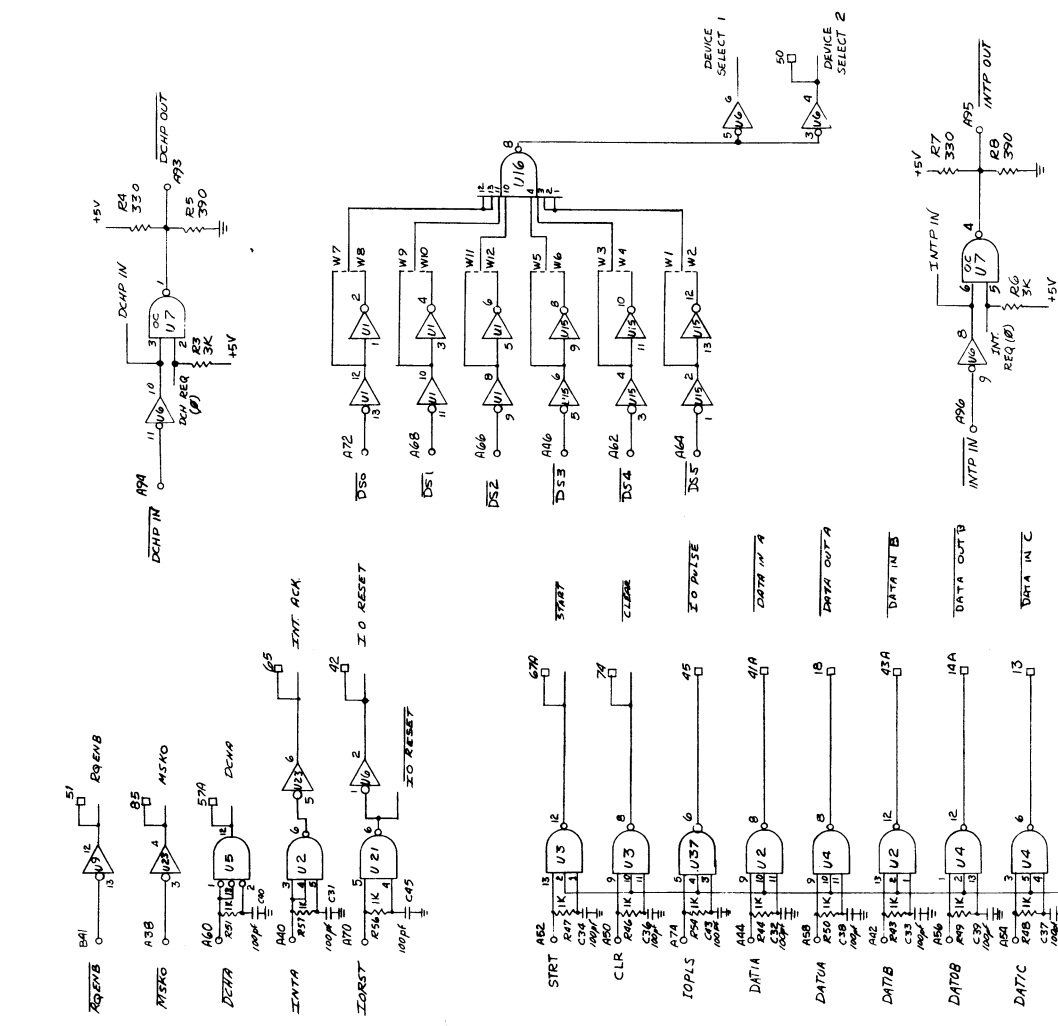
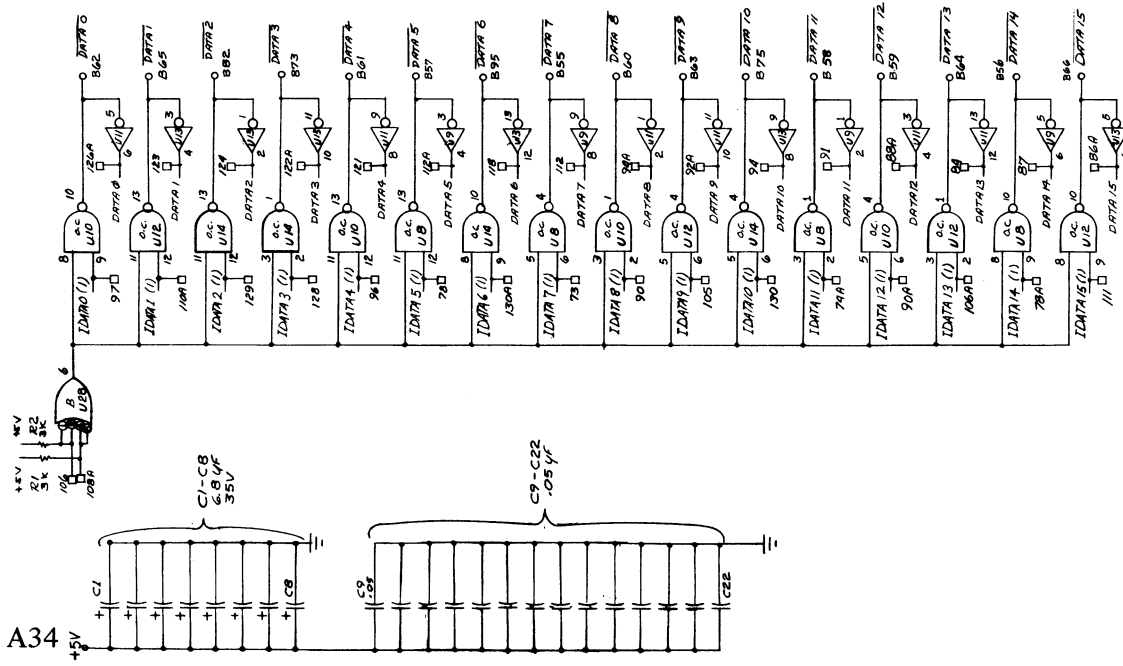
Pins used for connecting the board to the device are those that are available at any slot. In the 1210, 1220 and 820, the rear connector is the standard paddle board and the wiring configuration to it from the slot is the same as for any other device. Other machines use a 52-pin socket connector externally and the correspondence of back panel signal pins to connector pins is given at the bottom of page A34.

**Basic Interface 4040.** Mounted on every board is the logic illustrated on pages A34 and A35. The first drawing shows the circuitry that connects the interface to the data and control lines on the bus, networks for passing the interrupt and data channel priority signals along the bus, and a device selection net that allows a choice of any device code by putting in the appropriate jumpers. Note that the output of this last net gates those control signals that should be received only by the selected device. Connected to the data line drivers is an OR gate that is capable of driving all sixteen of them for the transfer of a full word. The second drawing shows the Busy, Done and interrupt logic, including a net that allows selection of any device code for reading by an INTA.

Table I lists all wire wrap pins associated with the basic interface logic and gives the fanout or load factor for each.

**Data Registers 4041.** This option consists simply of the two 16-bit shift registers illustrated on page A36. Each register can be cleared at MR and can receive serial data at J-K for right shifting under control of clock input CP fed through an OR gate. The output register can receive parallel data, enabled at PE through an OR gate, from the receivers for the IO bus data lines; its outputs are available to the customer logic. The input register receives parallel data from the customer logic, and its outputs are connected internally to the data line drivers in the 4040 (where they are available at wire wrap pins). Table II lists the pins and fanout/load data for the 4041.

**Data Channel Logic 4042.** The final two drawings [*pages A37 and A38*] show the logic supplied for connecting a device to the data channel. The first shows the standard flipflops and nets that handle the data channel control signals on the bus. The second shows two 16-bit counters for keeping track of the number of words processed and the current location for direct memory access. Each counter has an OR-gated count input at CL1, an OR-gated data strobe input for receipt of parallel data from the bus, and a full register clear input. The outputs of both registers are available to the customer logic, and the current address is supplied to data line drivers enabled through an OR gate. Pins and fanout/load data are given in Table III.



- WIRE WRAP PIN
- CONNECTOR PIN

GENERAL PURPOSE INTERFACE: BUS SIGNALS AND DEVICE SELECTION (4040)

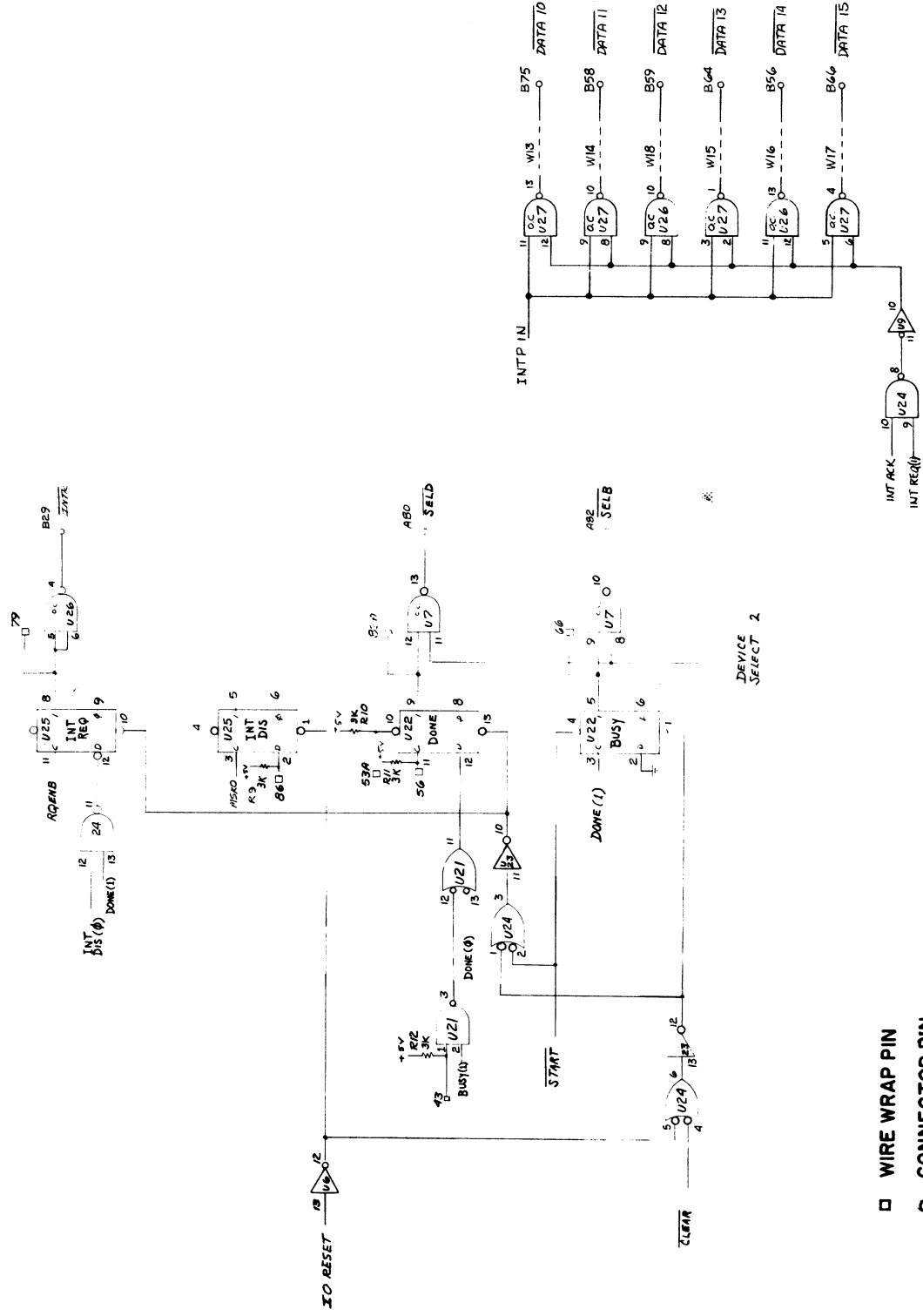
A34

A35



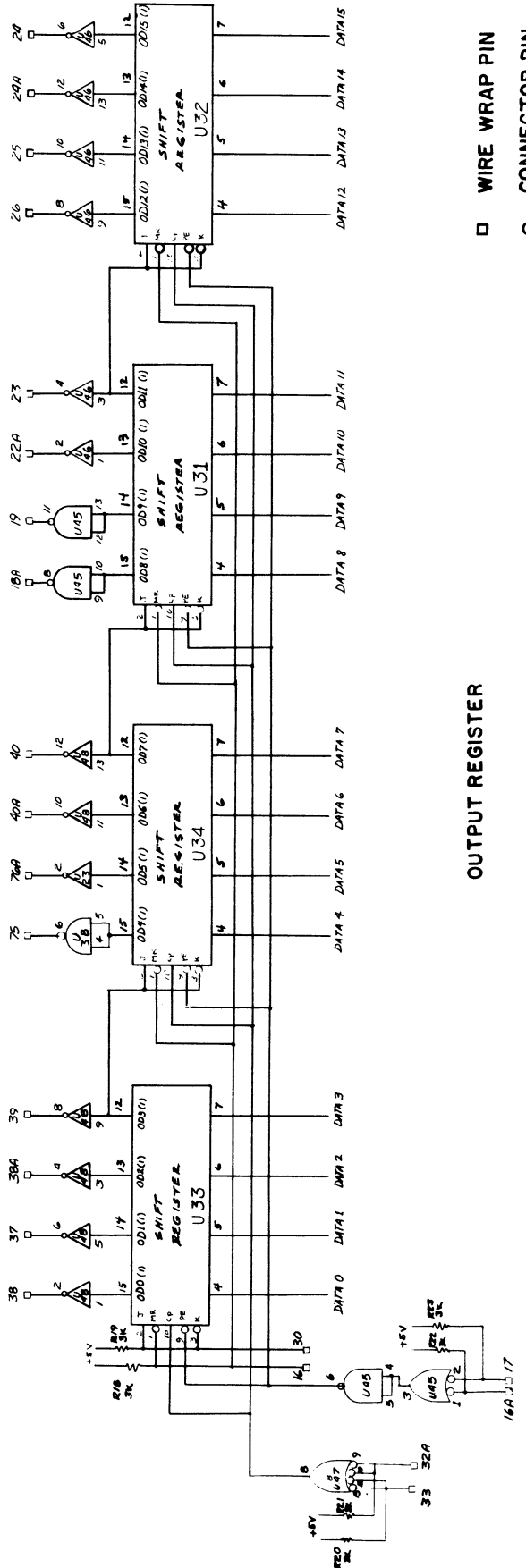
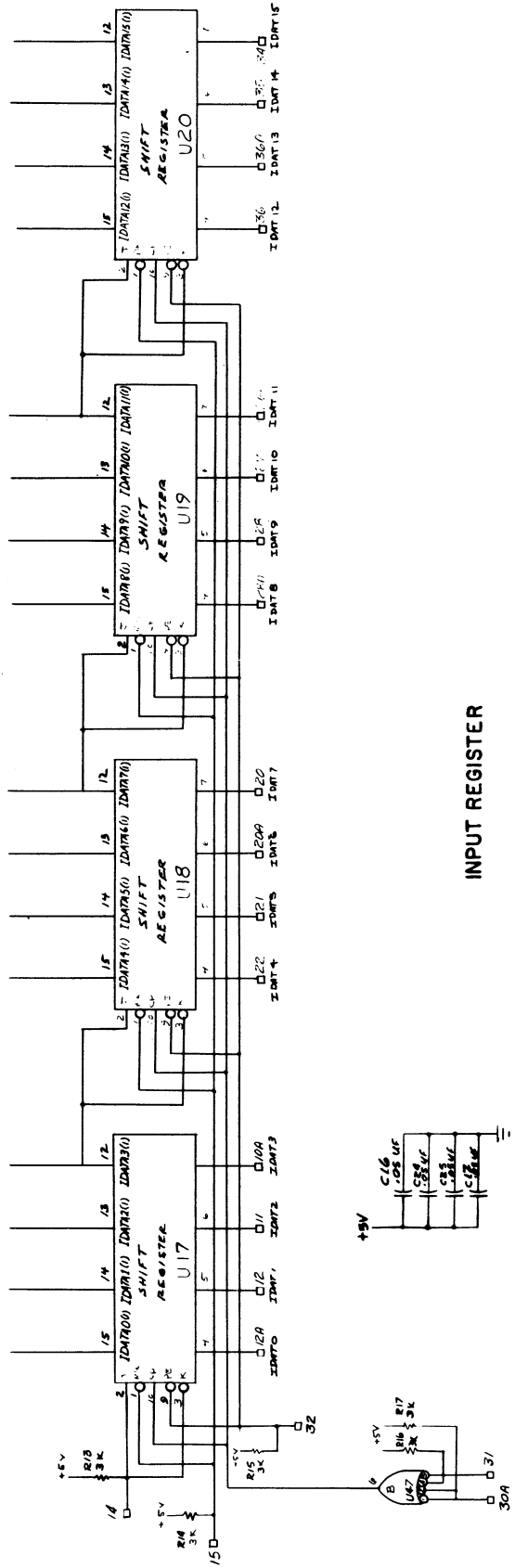
PINS AVAILABLE FOR INTERFACE

- A47 ○ □ 10
- A49 ○ □ 9
- A57 ○ □ 8A
- A59 ○ □ 8
- A61 ○ □ 34A
- A63 ○ □ 29
- A65 ○ □ 7
- A67 ○ □ 41
- A69 ○ □ 6A
- A71 ○ □ 48
- A73 ○ □ 49
- A75 ○ □ 49A
- A76 ○ □ 1
- A77 ○ □ 6
- A78 ○ □ 2
- A79 ○ □ 57
- A81 ○ □ 67
- A83 ○ □ 68A
- A84 ○ □ 2A
- A85 ○ □ 68
- A86 ○ □ 3
- A87 ○ □ 69
- A88 ○ □ 4
- A89 ○ □ 70A
- A90 ○ □ 4A
- A91 ○ □ 70
- A92 ○ □ 5
- B6 ○ □ 71
- B11 ○ □ 72A
- B13 ○ □ 72
- B15 ○ □ 76
- B19 ○ □ 85
- B23 ○ □ 84A
- B25 ○ □ 89
- B27 ○ □ 93
- B31 ○ □ 82
- B34 ○ □ 131
- B36 ○ □ 132A
- B38 ○ □ 132
- B40 ○ □ 133
- B48 ○ □ 134A
- B49 ○ □ 96A
- B51 ○ □ 104
- B52 ○ □ 134
- B53 ○ □ 135
- B54 ○ □ 136A
- B67 ○ □ 125
- B69 ○ □ 136



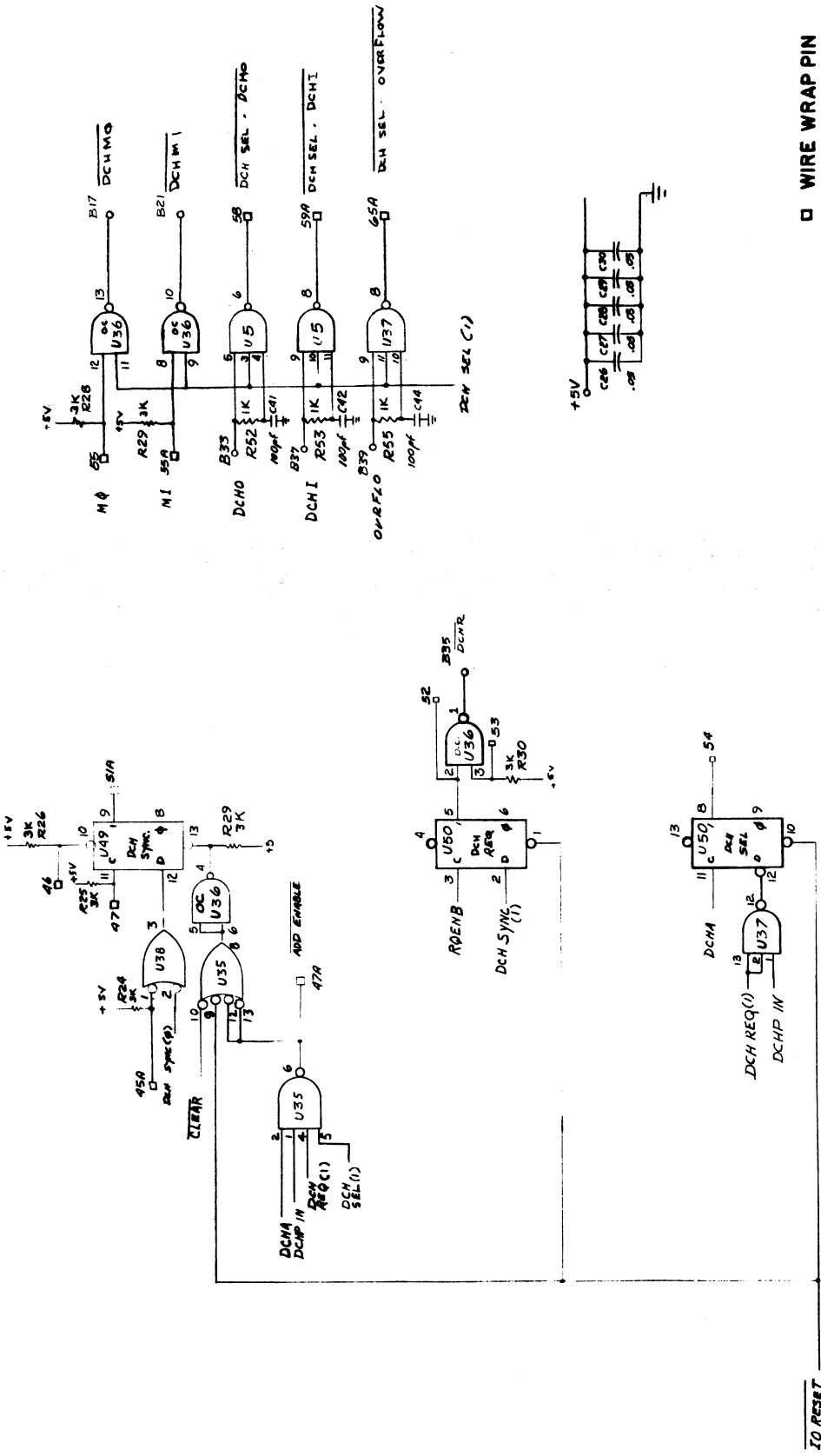
- WIRE WRAP PIN
- CONNECTOR PIN

GENERAL PURPOSE INTERFACE: BUSY, DONE, INTERRUPT (4040)



GENERAL PURPOSE INTERFACE: DATA REGISTERS (4041)

- WIRE WRAP PIN
- CONNECTOR PIN



GENERAL PURPOSE INTERFACE: DATA CHANNEL CONTROL (4042)



### EXTERNAL PIN CONFIGURATION

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

\*Pin 50 is grounded unless used for a special purpose, such as 60Hz signal

TABLE I BASIC INTERFACE 4040

Signal	Pin	Fanout	Load
DATA0–DATA15	<i>See below</i>	10*	
IDATA0(1)–IDATA15(1)	<i>See below</i>		1
OB1	106		4
OB2	108A		4
RQENB	51	6	
MSKO	85	8	
DCHA	57A	7	
INT ACK	65	9	
IO RESET	42	5	
<u>START</u>	67A	7	
<u>CLEAR</u>	74	8	
<u>IO PULSE</u>	45	10	
<u>DATA IN A</u>	41A	10	
<u>DATA OUT A</u>	18	10	
<u>DATA IN B</u>	43A	10	
<u>DATA OUT B</u>	14A	10	
<u>DATA IN C</u>	13	10	
<u>DATA OUT C</u>	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*

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	112	15	86A

*IDATA0(1)–IDATA15(1) pins*

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 Table II].

TABLE II DATA REGISTERS 4041

Signal	Pin	Fanout	Load
<i>Input register</i>			
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	
<i>Output register</i>			
OCP1	32A		4
OCP2	33		4
OPE1	16A		1
OPE2	17		1
OMR	16		4
OJ-K	30		2
$\overline{OD}0(1)–\overline{OD}15(1)$	See below	10	
DATA0–DATA15	See below	9*	

*IDAT0–IDAT15 pins*

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*

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*

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*

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	112	15	86A

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

TABLE III 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
<u>ADD ENABLE</u>	47A	9	
DCH REQ(1)	52	9	
DCH SEL(1)	64	5	
<u>DCHO</u>	58	10	
<u>DCHI</u>	59A	10	
<u>OVERFLOW</u>	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
<u>WC0(1)–WC15(1)</u>	See below	3	

*CA0(1)–CA15(1) pins*

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*

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



## CONNECTORS AND CONNECTOR PARTS

### Nova 1210, 1220, 820

Complete Connectors	DGC Order Number
Paddle board (male)	4192
Plug (female)	1039B

Connector Parts	DGC Part Number
50 dual-position connector	111-000117
Key	111-000116
Contacts (twin leaf)	111-000115
Contact spring	111-000123

Tools needed are available from AMP: crimper, 90268-1; extractor, 91073-1.

### Nova 1200, 800; Nova, Supernova

Complete Connectors	DGC Order Number	
	<i>Spare Parts List</i> No. — Socket	<i>Spare Parts List</i> No. — Plug
100 pin	2100	2101
52 pin	2102	2103
25 pin	2104	2105
19 pin	2106	2107
9 pin	2108	2109
50 pin I/O	2110	2111

Connector Parts	DGC Part Number	
	<i>Plug</i>	<i>Socket</i>
<i>Plugs and sockets</i>		
9 pin	111-000001	111-000002
19 pin	111-000007	111-000008
25 pin	111-000003	111-000004
50 pin (IO)	111-000005	111-000006
52 pin	111-000009	111-000010
100 pin	111-000011	111-000012
<i>Junction shells</i>		
9/19 pin	111-000019	
25/52 pin	111-000020	
50/100 pin	111-000021	

*Screw lock assembly, male*

9/19 pin 111-000023

25/52 pin 111-000024

50/100 pin 111-000025

*Screw lock assembly,  
female*

111-000022

All tools needed are available in a tool kit, order number 1112.

## APPENDIX B INSTALLATION

Every Nova line computer mounts in a standard 19-inch rack, has cooling fans at the rear, and contains slots for 15 X 15-inch printed circuit boards or DGC subassembly frames. The slots are numbered from the bottom up, and boards are inserted and removed from the right side. The table on page A2 lists the number of slots and their use for the various computers. In the 1210, 1220 and 820, the power supplies are beneath the space for the circuit boards; in other models the power supplies are at the rear.

The first drawing of the group at the end of this appendix shows the way in which a Nova line computer is packed; unpack the computer with care, and should reshipping ever be necessary, pack it in exactly the same way (in particular cardboard spacers must be put between the circuit boards). The second drawing shows in detail the mounting of the draw slides (the unit is shipped with the movable parts of the slides attached). The remaining drawings show the physical layout and dimensions of the various computers. Several inches should be left open at the back of the rack for cabling. The console protrudes 1-9/16 inches at the front of the rack (Nova and Supernova computers, 1 3/4 inches), and the entire unit slides out clear of the rack. In the following table the figures in parentheses are the depth with the console.

	<i>Height (inches)</i>	<i>Width (inches)</i>	<i>Depth (inches)</i>	<i>Weight (pounds)</i>
Nova 1210	5 1/4	19	19 (20-9/16)	43
Nova 1220, 820	10 1/2	19	19 (20-9/16)	56
Nova 1200, 800	5 1/4	19	21-3/8 (22-15/16)	50
Jumbo (1200, 800)	10 1/2	19	21-3/8 (22-15/16)	100
Nova, Supernova computers	5 1/4	19	20-1/4 (22)	60
Expansion chassis				
1220, 820	10 1/2	19	19	50
1200, 800	5 1/4	19	21-3/8	40
Nova, Supernova computers	5 1/4	19	20-1/4	40
Teletype ASR33	45	22	19	56

It is recommended that the ambient temperature at the installation be maintained between 20° and 30°C, but the temperature can vary from 0° to 55° without adverse effect (the equipment can be stored in temperatures as high as 70°). The relative humidity can be as high as 90% noncondensating. (Although all exposed surfaces are treated to prevent corrosion, exposure to extreme humidity for long periods of time should be avoided.)

The computer uses 47 to 63 Hz single phase line power, generally either 115 or 230 vac with a tolerance of ±20% (Nova and Supernova computers, ±10%); other frequencies and voltages are available on special order. The power source should be capable of supplying 15 amperes; the power cable has a standard 3-wire plug and should be plugged into a receptacle rated at 15 amperes. The minimum configuration of a computer is a processor, teletype interface, console and 4K of memory; the maximum configuration is the same but with 32K of memory.

	<i>Line Current (115 Vac, amperes)</i>	<i>Dissipation (watts)</i>	<i>Remaining +5Vdc capacity (Amps)</i>
Nova 1200			
Minimum	1.5	175	5.3
Maximum	2.4	275	3.6*

<b>Nova 1210</b>			
Minimum	1.5	175	3.3
Maximum	2.4	275	1.6*
<b>Nova 1220</b>			
Minimum	2.0	230	13.3
Maximum	3.5	400	11.6*
<b>Nova 800</b>			
Minimum	2.2	250	3.3
Maximum	3.1	350	
<b>Nova 820</b>			
Minimum	2.2	250	11.3
Maximum	3.7	425	6.2**
<b>Nova computer</b>			
Minimum	2.2	250	6.5
Maximum	3.5	400	2.7
<b>Supernova computer</b>			
Minimum	2.2	250	4.5
Maximum	3.1	350	0.7
<b>Supernova SC computer</b>			
Minimum	2.6	300	4.5
Maximum	5.2	600	0.7***
<b>Nova 2/4</b>			
Minimum			4.0
Maximum	2.6	300	2.5
<b>Nova 2/10</b>			
Minimum			20.0
Maximum	6.3	725	18.5

\* (2) 16K Memory boards      \*\* (4) 8K Memory boards

\*\*\* 28K (The SC memories are packaged separately and draw 8.7 amps/1K module.)

Each memory module, regardless of size, requires about 1.7 amperes at +5 vdc. The +5 output of the 1210 power supply can deliver 10 amperes, the 1220-820 power supply can deliver 20 amperes, and the +5 output of any other power supply can deliver 12 amperes. The 1220-820 expansion chassis has a dual +5 supply with two 12-ampere outputs, each of which supplies five slots. The 1200-800 Jumbo chassis has two power supplies, one standard and one dual +5; the basic supply is for slots 1-7, and the two dual outputs are for slots 8-12 and 13-17. In any chassis with the power supply at the rear, the power supply end of the back panel has pins for wire wrapping that carry an unregulated -15 vdc for customer use; this source, which is separate from the slot connectors, supplies 2 amperes maximum with a voltage tolerance of  $\pm 20\%$  and a maximum ripple of 1 volt.

Complete assembly instructions for the teletype are given in Section 574-100-201 of Bulletin 273B, Volume 1, *Technical Manual, 32 and 33 Teletypewriter Sets*. In particular, Part 6 of that section describes the installation of the power pack, which is mounted inside the stand as shown in the illustration on page 14. Plug the power pack cable into the pack.

All connections to the processor are made at the back. Plug the teletype power cable into the convenience outlet, or with the 1210, into a wall outlet. Connect the signal cable as shown in the appropriate installation drawing at the end of this appendix. With a 1210, 1220 or 820, the signal cable is plugged directly into the back panel and there is strain relief hardware at the rear of the chassis. With the other computers the signal cable plugs into a socket at the rear (the signal plug has a pair of captive bolts that should be screwed into the holes on either side of the socket). The

configuration of the connectors at the rear depends on what other equipment is included in the system, and for this, special installation information is provided. Complete information on the type and arrangement of connectors is given in Part I of Appendix A.

### Peripheral Equipment

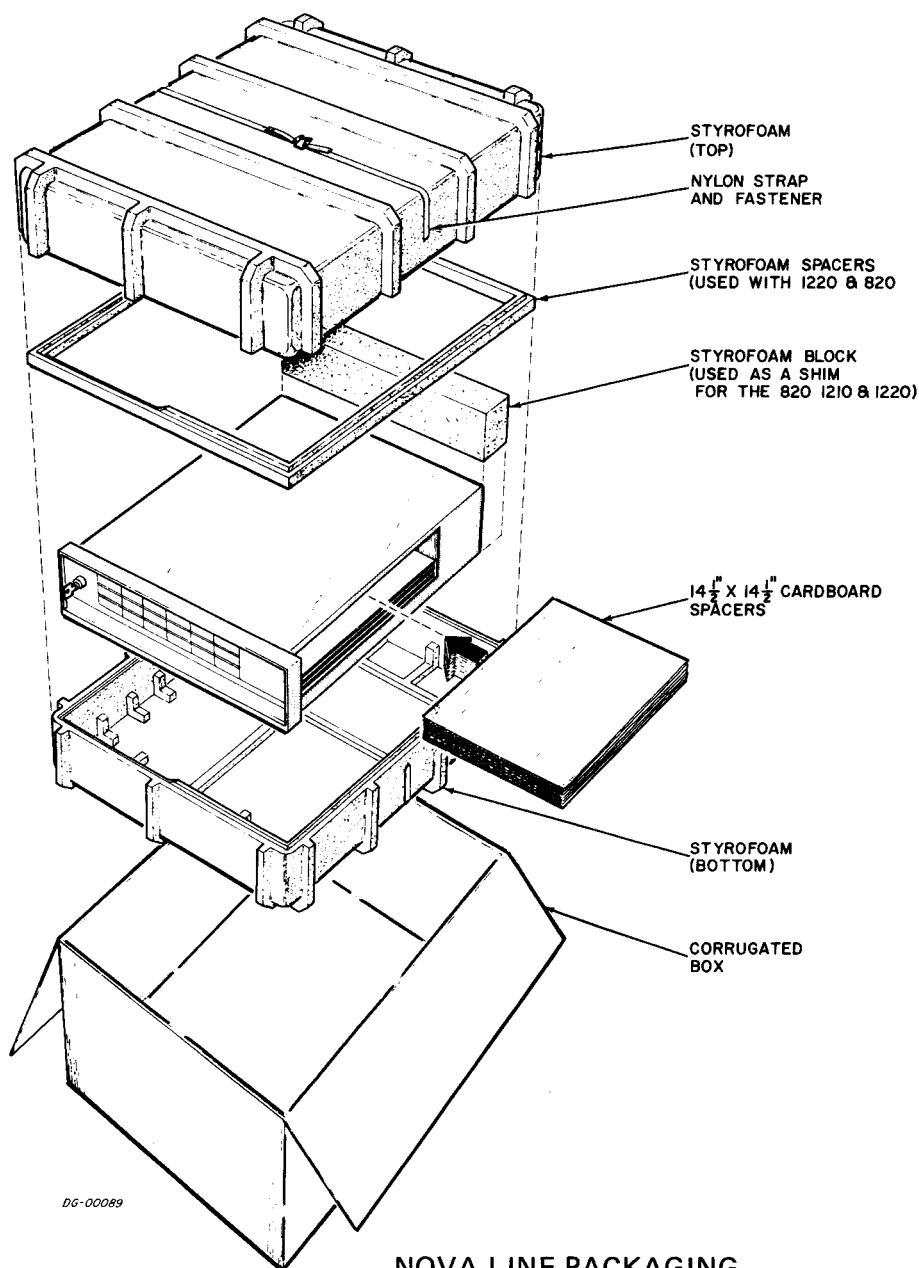
In general the environmental requirements (temperature, humidity) of the peripheral equipment are the same as those for the processor; for any special considerations refer to the option bulletin or the manufacturers manual. The physical dimensions and power requirements (at 115 vac) of the peripheral equipment are as follows:

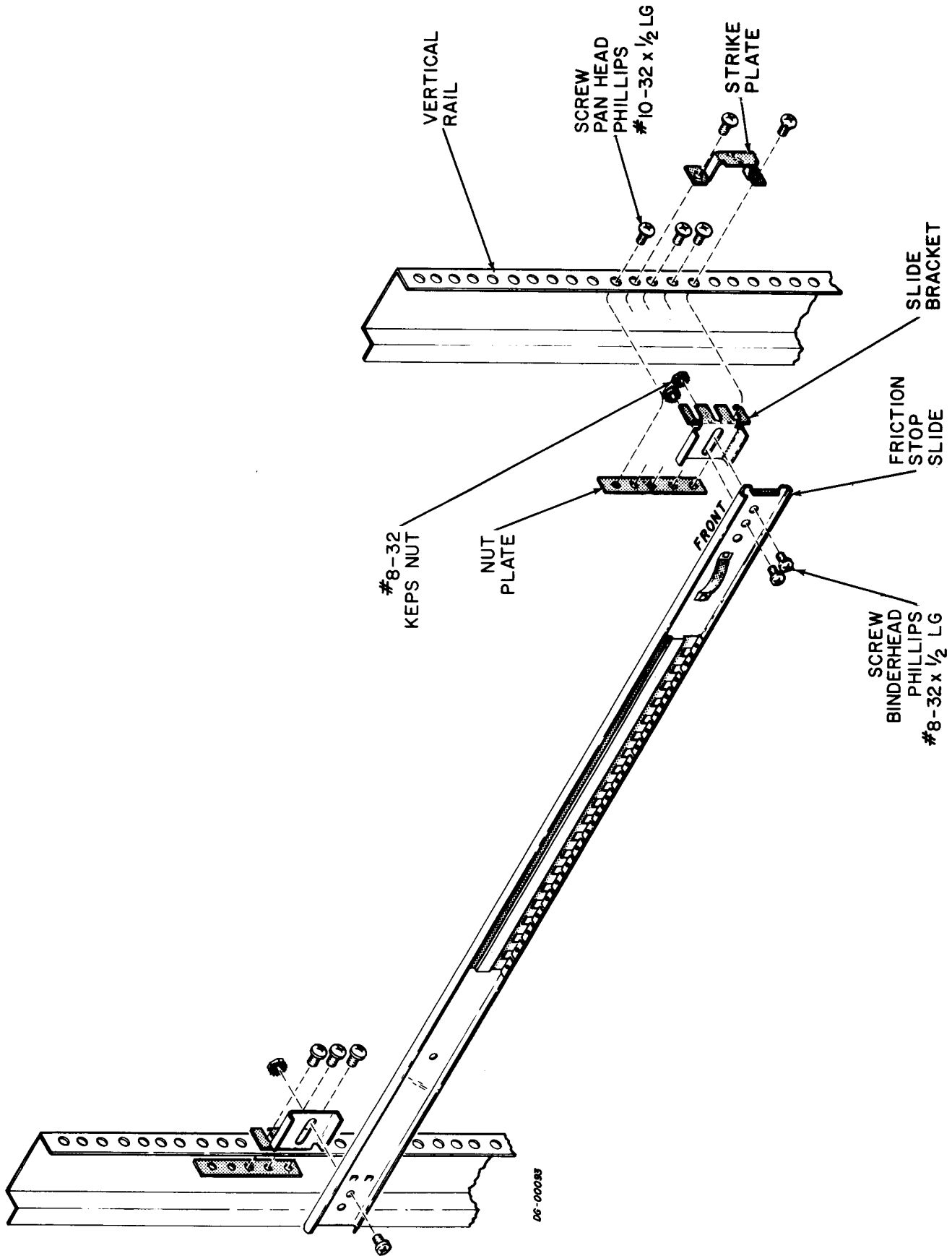
	<i>Height (inches)</i>	<i>Width (inches)</i>	<i>Depth (inches)</i>	<i>Weight (pounds)</i>	<i>Power (watts)</i>
<b>Teletypes</b>					
4010A—33ASR	8 $\frac{3}{8}$	22	18 $\frac{1}{2}$	44	128
4010B—33KSR	8 $\frac{3}{8}$	18 $\frac{5}{8}$	18 $\frac{1}{2}$	40	128
4010C—35KSR <sup>C</sup>	38 $\frac{1}{2}$	20	24	136	250
4023A—37ASR <sup>D</sup>	36 $\frac{1}{4}$	44 $\frac{1}{2}$	27 $\frac{1}{2}$	340	550
4023B—37KSR <sup>D</sup>	36 $\frac{1}{4}$	32 $\frac{1}{2}$	27 $\frac{1}{2}$	221	550
Paper Tape Reader	7	19	9	19	230
Paper Tape Punch	14	19	20	60	65
<b>Line Printers</b>					
4034A	22 $\frac{3}{4}$	23 $\frac{1}{2}$	22	185	330
4034B	46	48 $\frac{1}{2}$	24 $\frac{1}{2}$	575	500
4034C, 4034D	11 $\frac{1}{2}$	27 $\frac{3}{4}$	20	118	300
<b>Plotters</b>					
4017A	10	18	15 $\frac{1}{4}$	33	175
4017B <sup>A</sup>	12 $\frac{1}{2}$	19	12	35	175
4017C	10	39 $\frac{1}{2}$	15 $\frac{1}{4}$	53	175
4017D	41	46	45 $\frac{1}{2}$	250	290
4017E	8 $\frac{1}{2}$	17 $\frac{1}{4}$	14 or 37 $\frac{1}{2}$ <sup>B</sup>	40	200
<b>Card Readers</b>					
4016A, 4016B	13	23	12 $\frac{1}{2}$	68	400
4016C, 4016H	8 $\frac{1}{2}$	11	18 $\frac{1}{4}$	30	165
4016D, 4016I	11	19 $\frac{1}{4}$	14	60	950 start, 400 run
4016E, 4016F, G, J, K, L	16 $\frac{1}{4}$	23 $\frac{1}{16}$	18	77	1600 start, 600 run
<b>Tape Transports, less adapter</b>					
4030I and 4030J	24	19	11	100	475
<b>Magnetic Tape Adapter<sup>E</sup></b>					
4030A, B, C, D, E, F, G, H	5 $\frac{3}{4}$	19	8 $\frac{1}{2}$	15	15
Novadisc	12 $\frac{1}{4}$	19	27	130	770
Cassette Tape (3-unit)	7	19	18	58	450
Fixed Head Disc	12 $\frac{1}{4}$	19	18 $\frac{1}{2}$	70	200
Disc Cartridge	6 $\frac{1}{2}$	17 $\frac{1}{2}$	22 $\frac{7}{8}$	43	70 <sup>F</sup>
Disc Pack	40	30	24	350	580 <sup>G</sup>
Moving Head Disc Adapter	7	19	21		20

A-D and D-A systems

4055A	3½	17	17	16	20
4055B	3½	17	17	20	40
4055O	3½	17	17	20	25
4055P	3½	17	17	20	50
4056H	3½	17	17	20	55
IBM Interface Adapter	5¼	19	12		20

- A Rack model
- B Trays fully extended
- C Pedestal mounted
- D Table mounted
- E Mounts in separate rack with models 4030E and 4030F; otherwise below transport
- F Uses dc power from adapter
- G Requires 208 or 230 vac, ±10%, 3 phase; turnon surge, 15 amperes for 5-7 seconds

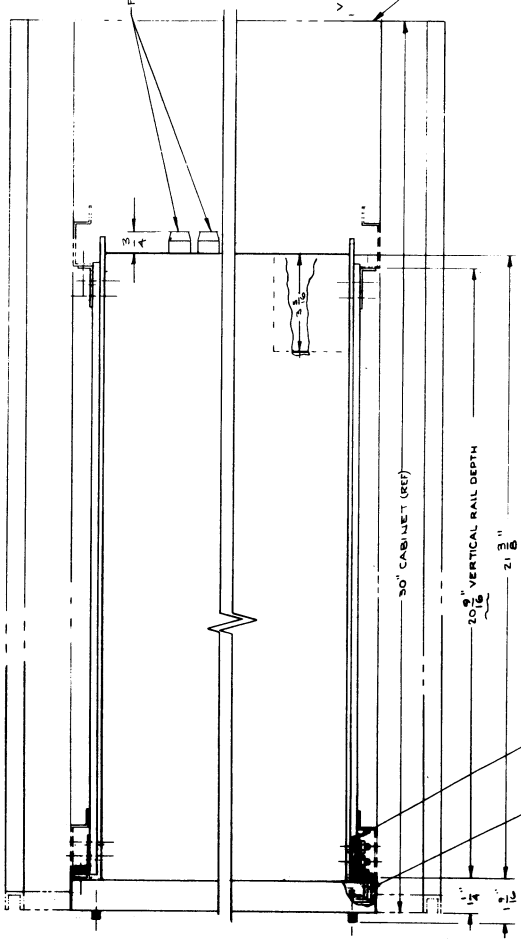
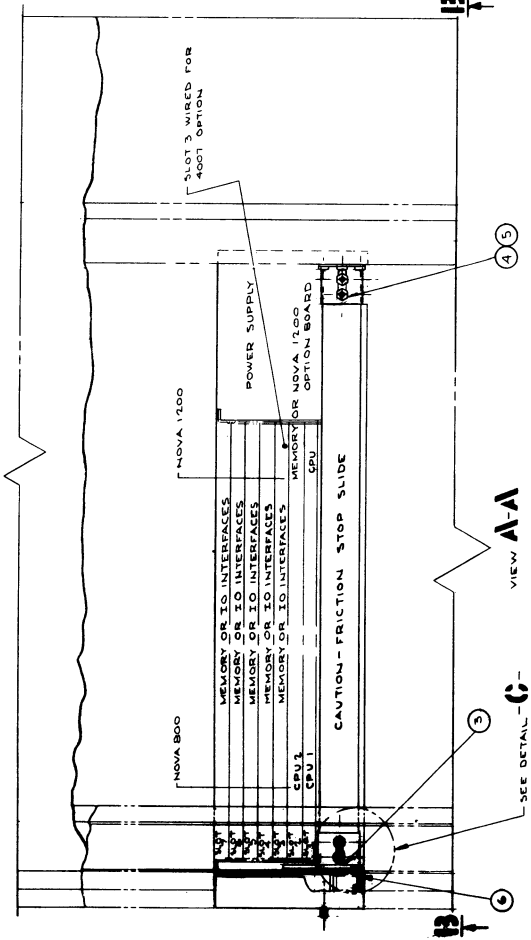
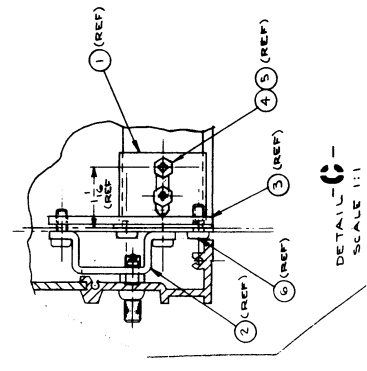
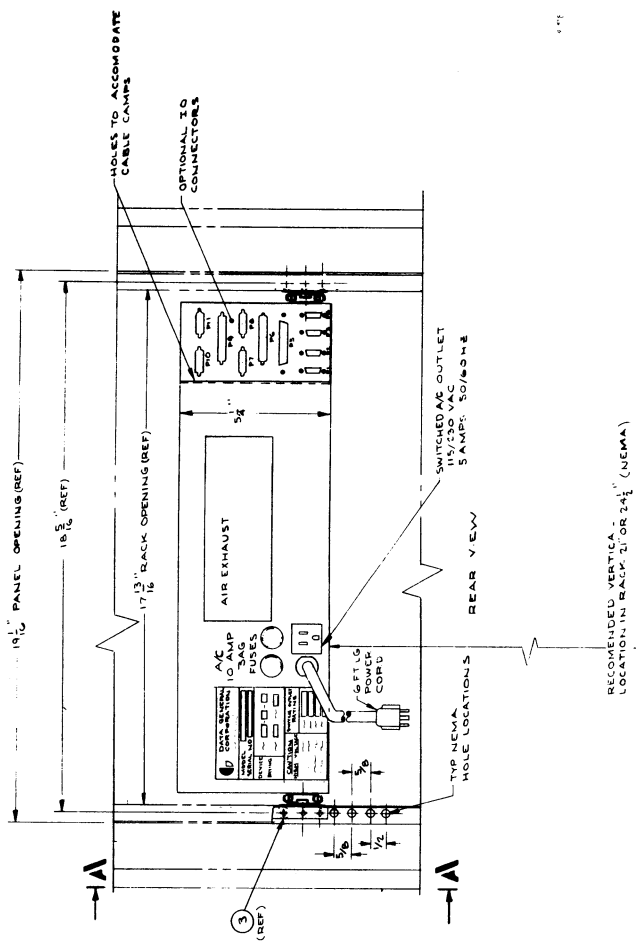




NOVA LINE SLIDE MOUNTING





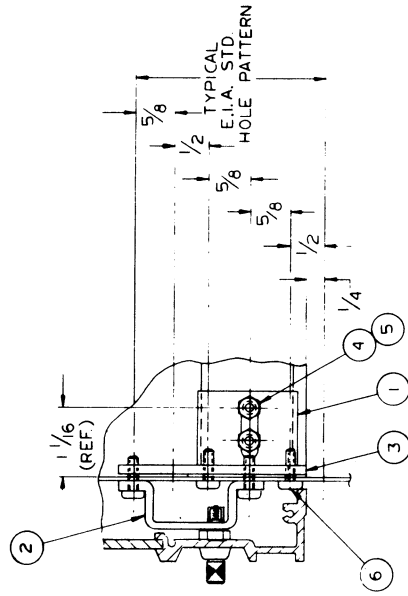
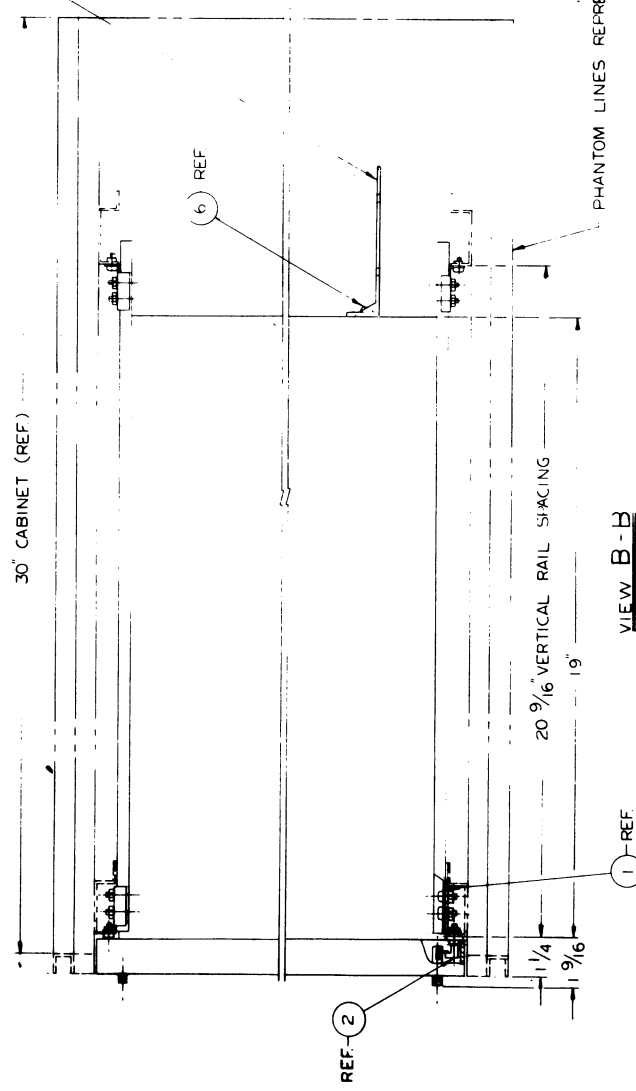
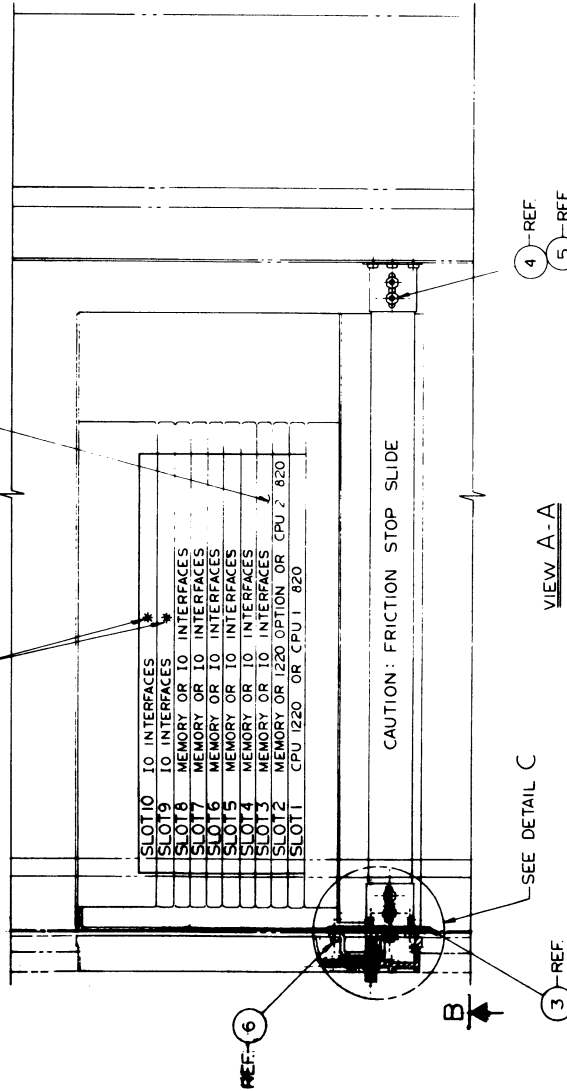


ITEM NO.	QTY	DESCRIPTION	PART NO.
6	14	SCREW, PAN HEAD, PHILIPS	
5	8	KEPS NUT, 8-32	
4	8	WASHER, 8-32, PHILIPS	
3	4	NUT PLATE (10-32)	A002-00030B
2	2	STRIKE PLATE	B002-000132
1	4	BRACKET, SLIDE	B002-000270
REF		NOVA 800 OR NOVA 1200	

INSTALLATION PROCEDURE:  
 REMOVE SIDE PANELS, REASSEMBLE ITEM TO OUTER SLIDE USING ITEMS 4 & 5 AS INDICATED. NOTE: SEE DETAIL 6 FOR REAR VIEW AND ACCESSIBLE AS SHOWN.

INSTALLATION: NOVA 800, NOVA 1200

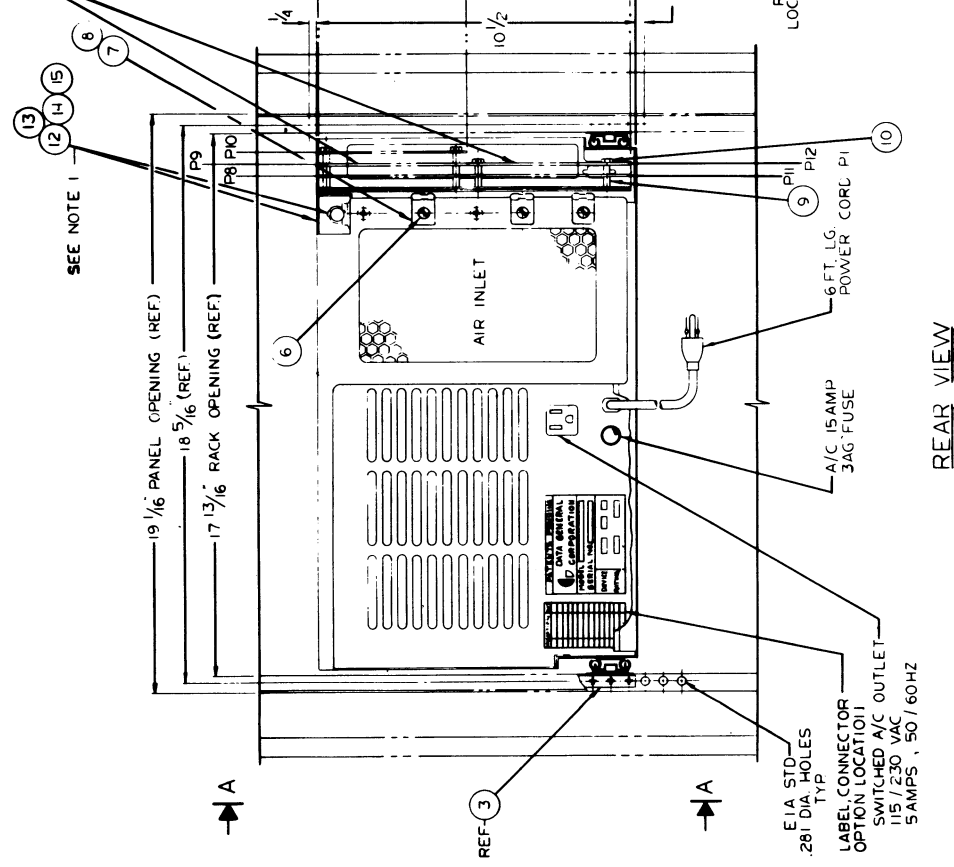
SEE NOTE 2  
 IF TTY, PTP, PTR OPTIONS ARE ORDERED  
 THESE MUST BE CONFIGURED IN SLOT 3



INSTALLATION PROCEDURE:  
 REMOVE SIDE PANELS, PREASSEMBLE  
 ITEM 1 TO OUTER SLIDE USING ITEMS 4 & 5  
 AS INDICATED. MOUNT OUTER SLIDE ASSEMBLY  
 AS INDICATED. NOTE: SEE DETAIL C FOR REAR  
 VIEW AND ASSEMBLY AS SHOWN

ITEM NO.	QTY.	DESCRIPTION	PART NO.
15	2	KEPS NUT 4-40	
14	2	SCREW PAN HD. PHILIPS 4-40 X 3/8 LG.	
13	1	CABLE TIE MOUNT	123-000067
12	1	COVER	002 000537
11	1	3/16 DIA. CABLE CLAMP	123-000073
10	4	SCREW HEX HD-R 3/8 X 1/4 LG. (SS)	---
9	A/R	HEX STAND-OFF	B002-000538
8	A/R	STRAIN RELIEF SUPPORT	123-000053
7	A/R	CABLE TIE	123-000054
6	17	SCREW PAN HD. PHILIPS #10-32 X 1 1/2 LG.	---
5	8	KEPS NUT #8-32	---
4	8	SCREW BINDER HD PHILIPS #8-32 X 1/2 LG. (SS)	---
3	4	NUT PLATE (#10-32)	002-000308
2	2	STRIKE PLATE	002-000232
1	4	BRACKET, SLIDE	002-000270
REF		NOVA 1220	#

ADDITIONAL INTERFACES ARE PROVIDED USING FINGER-FRINTEL CIRCUIT BOARDS (DGC 207-000131 OR 207-000132) WHICH MATE WITH EXTERNAL CABLES TERMINATED WITH DGC CONNECTORS III-000117 & III-000119 RESPECTIVELY. IF REQUIRED OTHER TYPES OF CONNECTORS CAN BE INTERFACED IN THE SPACE PROVIDED AS SHOWN IN VIEW F-F ON SHEET 3

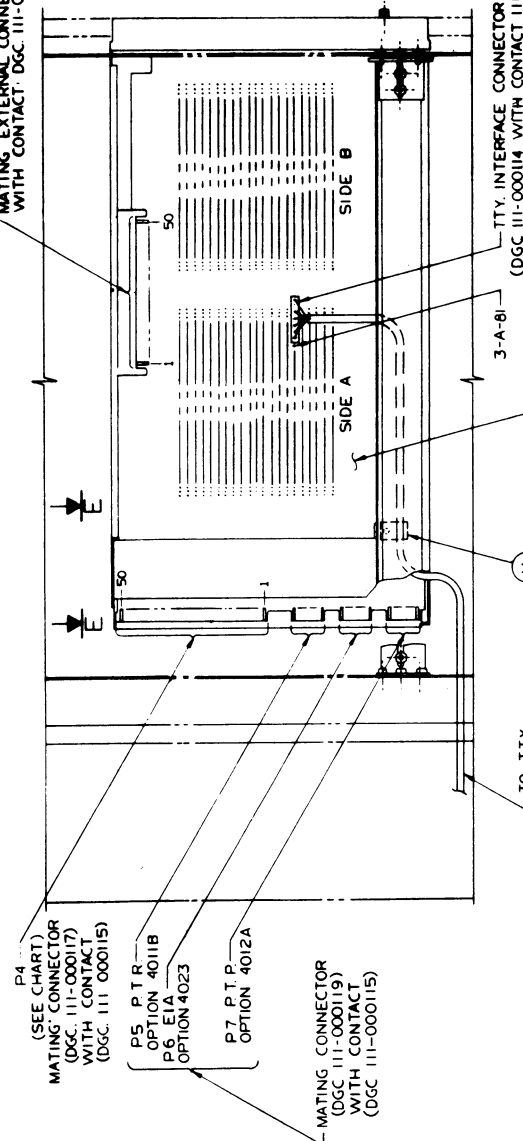


- NOTES:
1. THESE PARTS ARE USED WITH EXTERNAL I/O CABLE INTERFACE
  2. SLOTS 9 & 10 ARE CONFIGURED TO ALLOW THE CUSTOMER A LABEL NUMBER OF I/O PINS (THAT IS 77/51/31 EXTRA)
  3. P8 THRU P12 CAN BE EITHER 3X20 SIGNAL INTERFACE (DGC 207-000132) OR 50 DUAL SIGNAL INTERFACE (DGC 207-000131)
  4. CONNECTOR/OPTION ASSIGNMENT IS SPECIFIED ON LABEL, CONT. OPTION LOCATION
- DGC 002-000590 (SEE ABOVE)

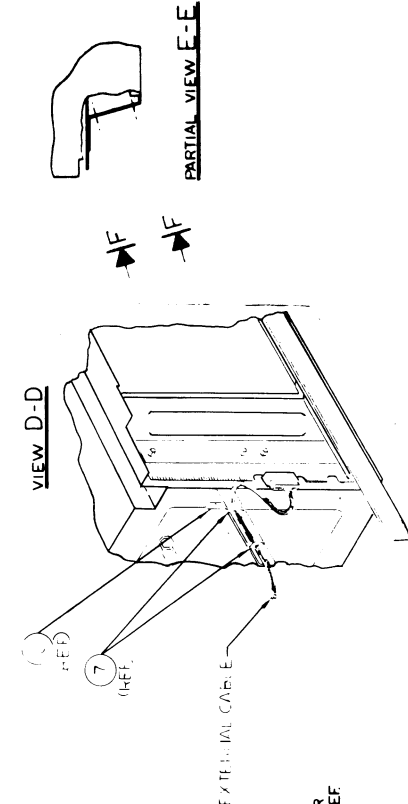
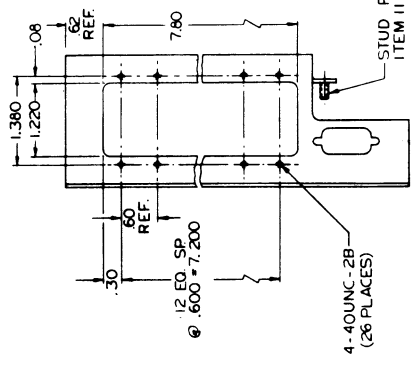
P3 IS EXTERNAL I/O BUSS AND HAS THE FOLLOWING SIGNAL ASSIGNMENT

LETTER SITE	NUMBER SIDE	SIGNAL NAME
A	1 THRU 50	GND
B		GND
C		PWR ON (+5V)
D		MSKO
E		INTA
F		DATAB
G		DATIB
H		DS3
J		DATA
K		DATA
L		DATA
M		DATA
N		DATA
P		DATA
R		DATA
S		DATA
T		DATA
U		DATA
V		DATA
W		DATA
X		DATA
Y		DATA
Z		DATA
a		DATA
b		DATA
c		DATA
d		DATA
e		DATA
f		DATA
g		DATA
h		DATA
i		DATA
j		DATA
k		DATA
l		DATA
m		DATA
n		DATA
o		DATA
p		DATA
q		DATA
r		DATA
s		DATA
t		DATA
u		DATA
v		DATA
w		DATA
x		DATA
y		DATA
z		DATA
aa		DATA
ab		DATA
ac		DATA
ad		DATA
ae		DATA
af		DATA
ag		DATA
ah		DATA
ai		DATA

P3 (NUMBER SIDE OF CONNECTOR SHOWN)  
 (SEE CHART ON SHEET 2 FOR SIGNAL ASSIGNMENT)  
 MATING EXTERNAL CONNECTOR (DGC 111-000117  
 WITH CONTACT DGC 111-000115)



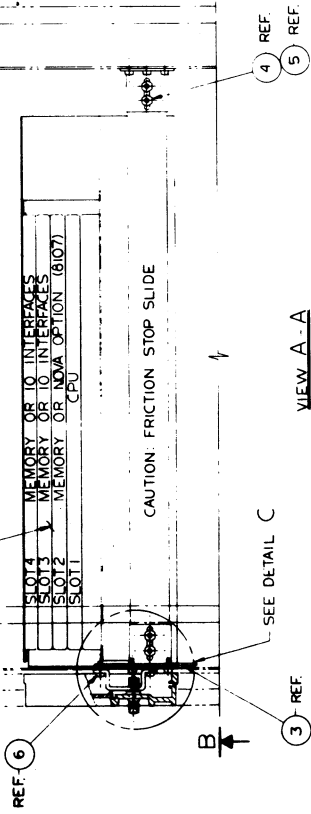
NUMBER SIDE	LETTER SIDE	BACK PANEL	
		SLOT SIDE	PIN NO.
1	A-THRU-AF	GND	
2	A	9-A-92	
3	A	9-A-91	
4	A	9-A-78	
5	A	9-A-77	
6	A	9-A-76	
7	A	9-A-75	
8	A	9-A-73	
9	A	9-A-71	
10	A	9-A-69	
11	A	9-A-67	
12	A	9-A-65	
13	A	9-A-63	
14	A	9-A-61	
15	A	9-A-59	
16	A	9-A-47	
17	A	9-A-46	
18	A	9-A-40	
19	A	9-A-39	
20	A	9-A-81	
21	A	9-A-84	
22	A	9-A-83	
23	A	9-A-86	
24	A	9-A-85	
25	A	9-A-88	
26	A	9-A-87	
27	A	9-A-89	
28	A	9-A-90	
29	B	9-B-6	
30	B	9-B-11	
31	B	9-B-13	
32	B	9-B-12	
33	B	9-B-24	
34	B	9-B-25	
35	B	9-B-27	
36	B	9-B-31	
37	B	9-B-34	
38	B	9-B-36	
39	B	9-B-38	
40	B	9-B-40	
41	B	9-B-48	
42	B	9-B-49	
43	B	9-B-51	
44	B	9-B-52	
45	B	9-B-53	
46	B	9-B-34	
47	B	9-B-34	
48	B	9-B-67	
49	B	9-B-69	
50		RESERVED	



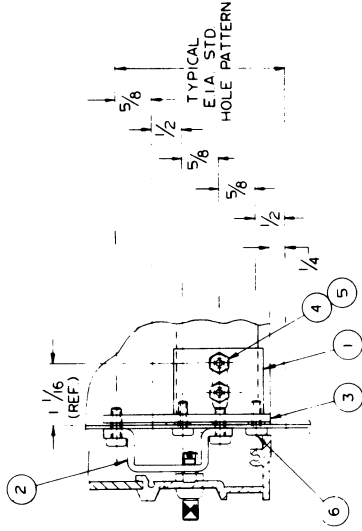
VIEW F-F  
 SCALE: FULL SIZE

SEE SHEET 1 FOR VIEW D-D TO  
 DETERMINE SIGNAL ASSIGNMENT. SEE SHEET 2 FOR  
 SIGNAL ASSIGNMENT CHART.

BACKPANEL CONNECTOR FOR TTY. REQUIRES TTY INTERFACE (I/O) TO BE PLUGGED INTO SLOT 3

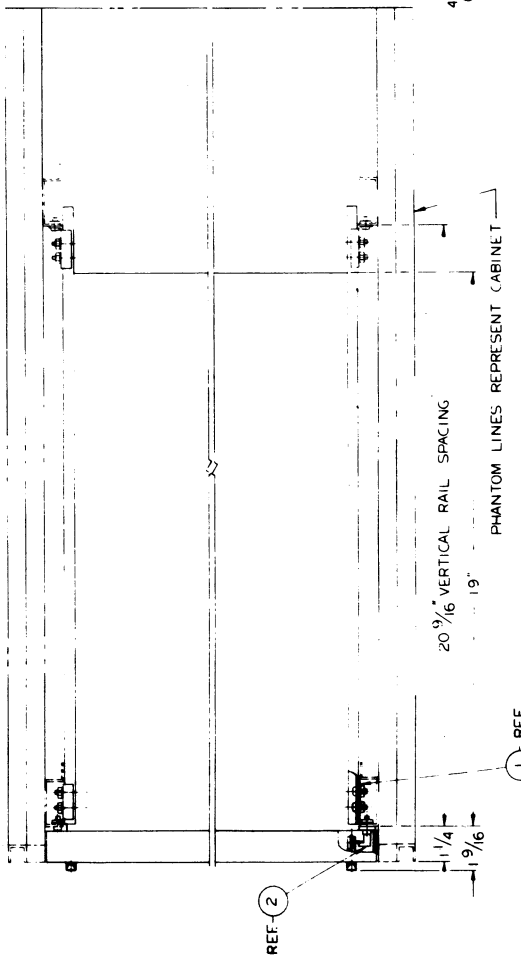


VIEW A-A



DETAIL C  
SCALE: FULL SIZE  
2 PLACES

NOTE: DETAIL C ALSO TYPICAL FOR TURNKEY CONSOLE.

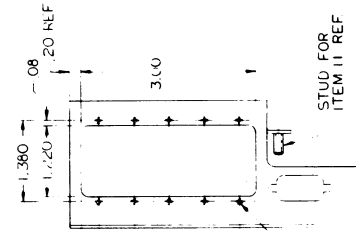


VIEW B-B

INSTALLATION PROCEDURE:

REMOVE SIDE PANELS. PRE-ASSEMBLE ITEM 1 TO OUTER SLIDE USING ITEMS 4 & 5 AS INDICATED. MOUNT OUTER SLIDE ASSEMBLY AS INDICATED. NOTE: SEE DETAIL C FOR REAR VIEW AND ASSEMBLY AS SHOWN

ITEM QTY	DESCRIPTION	PART NO
11 1	3/16 DIA CABLE CLAMP	123-000073
10 2	SCREW HEX HD 8-32 X 1/4 LG. (SST)	
9 3	AIR HEX STAND-OFF	B002-000538
8 8	AIR STRAIN RELIEF SUPPORT	123-000053
7 7	AIR CABLE TIE	123-000054
6 16	SCREW PAN HD PHILIPS #10-32 X 1/2 LG.	
5 8	KEPS NUT #8 .32	
4 8	SCREW BINDER HD PHILIPS #8-32 X 1/2 LG (SST)	
3 4	NUT PLATE (10-32)	A002-000308
2 2	STRIKE PLATE	B002-000232
1 4	BRACKET, SLIDE	B002-000270
REF	NOVA 1210	



VIEW F-F (SEE SHEET 2)  
SCALE: FULL SIZE



## APPENDIX D

### INSTRUCTION MNEMONICS AND TIMING

The table beginning on the next page lists the instruction mnemonics in numerical order. Following that is a listing in alphabetical order that gives the octal value, a short description of the instruction, and the number of the page on which the full description appears in Chapter 2. Instruction execution times in microseconds are listed on page D12.

The derivation of the instruction mnemonics is as follows.

Load } Store }	} Accumulator			
Increment } Decrement }	} and Skip if Zero			
<b>Jump</b>				
Jump to SubRoutine				
Complement Negate Move Increment Add Complement Subtract Add AND	} for carry bit base value use	current carry Zero One Complement of current carry	} { ~ shift Left shift Right Swap bytes	} { ~ #
Skip Skip	{ on Zero { om Nonzero { if Either is Zero { if Both are Nonzero	{ Carry { Result		
No IO transfer Data { In } { Out }	{ A } { B } { C }	buffer — and	{ ~ Start Clear special Pulse	
Skip if { Busy } { Done }	is { Nonzero } { Zero }			
<b>READ Switches</b>				
<b>IO ReSeT</b>				
<b>HALT</b>				
<b>INTerrupt Acknowledge</b>				
<b>MaSK Out</b>				
<b>INTerrupt ENable</b>				
<b>INTerrupt DiSable</b>				
<b>MULtiply</b>				
<b>DIVide</b>				

# INSTRUCTION MNEMONICS

## NUMERIC LISTING

000000	JMP	062677	IORST	100350	COMOS#
000001	SKP	062700	DICP	100360	COMCS
000002	SZC	063000	DOC	100370	COMCS#
000003	SNC	063077	HALT	100400	NEG
000004	SZR	063100	DOCS	100410	NEG#
000005	SNR	063200	DOCC	100420	NEGZ
000006	SEZ	063300	DOCP	100430	NEGZ#
000007	SBN	063400	SKPBN	100440	NEGO
000010	#	063500	SKPBZ	100450	NEGO#
002000	@	063600	SKPDN	100460	NEGC
004000	JSR	063700	SKPDZ	100470	NEGC#
010000	ISZ	073101	DIV	100500	NEGL
014000	DSZ	073301	MUL	100510	NEGL#
020000	LDA	100000	@	100520	NEGZL
040000	STA	100000	COM	100530	NEGZL#
060000	NIO	100010	COM#	100540	NEGOL
060100	NIOS	100020	COMZ	100550	NEGOL#
060177	INTEN	100030	COMZ#	100560	NEGCL
060200	NIOC	100040	COMO	100570	NEGCL#
060277	INTDS	100050	COMO#	100600	NEGR
060300	NIOP	100060	COMC	100610	NEGR#
060400	DIA	100070	COMC#	100620	NEGZR
060477	READS	100100	COML	100630	NEGZR#
060500	DIAS	100110	COML#	100640	NEGOR
060600	DIAC	100120	COMZL	100650	NEGOR#
060700	DIAP	100130	COMZL#	100660	NEGCR
061000	DOA	100140	COMOL	100670	NEGCR#
061100	DOAS	100150	COMOL#	100700	NEGS
061200	DOAC	100160	COMCL	100710	NEGS#
061300	DOAP	100170	COMCL#	100720	NEGZS
061400	DIB	100200	COMR	100730	NEGZS#
061477	INTA	100210	COMR#	100740	NEGOS
061500	DIBS	100220	COMZR	100750	NEGOS#
061600	DIBC	100230	COMZR#	100760	NEGCS
061700	DIBP	100240	COMOR	100770	NEGCS#
062000	DOB	100250	COMOR#	101000	MOV
062077	MSKO	100260	COMCR	101010	MOV#
062100	DOBS	100270	COMCR#	101020	MOVZ
062200	DOBC	100300	COMS	101030	MOVZ#
062300	DOBP	100310	COMS#	101040	MOV0
062400	DIC	100320	COMZS	101050	MOV0#
062500	DICS	100330	COMZS#	101060	MOVC
062600	DICC	100340	COMOS	101070	MOVC#



101100	MOVL	101660	INCCR	102440	SUBO
101110	MOVL#	101670	INCCR#	102450	SUBO#
101120	MOVZL	101700	INCS	102460	SUBC
101130	MOVZL#	101710	INCS#	102470	SUBC#
101140	MOVOL	101720	INCZS	102500	SUBL
101150	MOVOL#	101730	INCZS#	102510	SUBL#
101160	MOVCL	101740	INCOS	102520	SUBZL
101170	MOVCL#	101750	INCOS#	102530	SUBZL#
101200	MOVR	101760	INCCS	102540	SUBOL
101210	MOVR#	101770	INCCS#	102550	SUBOL#
101220	MOVZR	102000	ADC	102560	SUBCL
101230	MOVZR#	102010	ADC#	102570	SUBCL#
101240	MOVOR	102020	ADCZ	102600	SUBR
101250	MOVOR#	102030	ADCZ#	102610	SUBR#
101260	MOVCR	102040	ADCO	102620	SUBZR
101270	MOVCR#	102050	ADCO#	102630	SUBZR#
101300	MOVS	102060	ADCC	102640	SUBOR
101310	MOVS#	102070	ADCC#	102650	SUBOR#
101320	MOVZS	102100	ADCL	102660	SUBCR
101330	MOVZS#	102110	ADCL#	102670	SUBCR#
101340	MOVOS	102120	ADCZL	102700	SUBS
101350	MOVOS#	102130	ADCZL#	102710	SUBS#
101360	MOVCS	102140	ADCOL	102720	SUBZS
101370	MOVCS#	102150	ADCOL#	102730	SUBZS#
101400	INC	102160	ADCCL	102740	SUBOS
101410	INC#	102170	ADCCL#	102750	SUBOS#
101420	INCZ	102200	ADCR	102760	SUBCS
101430	INCZ#	102210	ADCR#	102770	SUBCS#
101440	INCO	102220	ADCZR	103000	ADD
101450	INCO#	102230	ADCZR#	103010	ADD#
101460	INCC	102240	ADCOR	103020	ADDZ
101470	INCC#	102250	ADCOR#	103030	ADDZ#
101500	INCL	102260	ADCCR	103040	ADDO
101510	INCL#	102270	ADCCR#	103050	ADDO#
101520	INCZL	102300	ADCS	103060	ADDC
101530	INCZL#	102310	ADCS#	103070	ADDC#
101540	INCOL	102320	ADCZS	103100	ADDL
101550	INCOL#	102330	ADCZS#	103110	ADDL#
101560	INCLL	102340	ADCOS	103120	ADDZL
101570	INCLL#	102350	ADCOS#	103130	ADDZL#
101600	INCR	102360	ADCCS	103140	ADDOL
101610	INCR#	102370	ADCCS#	103150	ADDOL#
101620	INCZR	102400	SUB	103160	ADDCL
101630	INCZR#	102410	SUB#	103170	ADDCL#
101640	INCOR	102420	SUBZ	103200	ADDR
101650	INCOR#	102430	SUBZ#	103210	ADDR#

103220	ADDZR	103420	ANDZ	103620	ANDZR
103230	ADDZR#	103430	ANDZ#	103630	ANDZR#
103240	ADDOR	103440	ANDO	103640	ANDOR
103250	ADDOR#	103450	ANDO#	103650	ANDOR#
103260	ADDCR	103460	ANDC	103660	ANDCR
103270	ADDCR#	103470	ANDC#	103670	ANDCR#
103300	ADDS	103500	ANDL	103700	ANDS
103310	ADDS#	103510	ANDL#	103710	ANDS#
103320	ADDZS	103520	ANDZL	103720	ANDZS
103330	ADDZS#	103530	ANDZL#	103730	ANDZS#
103340	ADDOS	103540	ANDOL	103740	ANDOS
103350	ADDOS#	103550	ANDOL#	103750	ANDOS#
103360	ADDCS	103560	ANDCL	103760	ANDCS
103370	ADDCS#	103570	ANDCL#	103770	ANDCS#
103400	AND	103600	ANDR		
103410	AND#	103610	ANDR#		

## INSTRUCTION MNEMONICS

### ALPHABETIC LISTING

			<i>Page</i>
ADC	102000	Add the complement of ACS to ACD; use Carry as base for carry bit.	2-16
ADCC	102060	Add the complement of ACS to ACD; use complement of Carry as base for carry bit.	2-16
ADCCL	102160	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; rotate left.	2-16
ADCCR	102260	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; rotate right.	2-16
ADCCS	102360	Add the complement of ACS to ACD; use complement of Carry as base for carry bit; swap halves of result.	2-16
ADCL	102100	Add the complement of ACS to ACD; use Carry as base for carry bit; rotate left.	2-16
ADCO	102040	Add the complement of ACS to ACD; use 1 as base for carry bit.	2-16
ADCOL	102140	Add the complement of ACS to ACD; use 1 as base for carry bit; rotate left.	2-16
ADCOR	102240	Add the complement of ACS to ACD; use 1 as base for carry bit; rotate right.	2-16
ADCOS	102340	Add the complement of ACS to ACD; use 1 as base for carry bit; swap halves of result.	2-16
ADCR	102200	Add the complement of ACS to ACD; use Carry as base for carry bit; rotate right.	2-16
ADCS	102300	Add the complement of ACS to ACD; use Carry as base for carry bit; swap halves of result.	2-16
ADCZ	102020	Add the complement of ACS to ACD; use 0 as base for carry bit.	2-16
ADCZL	102120	Add the complement of ACS to ACD; use 0 as base for carry bit; rotate left.	2-16
ADCZR	102220	Add the complement of ACS to ACD; use 0 as base for carry bit; rotate right.	2-16
ADCZS	102320	Add the complement of ACS to ACD; use 0 as base for carry bit; swap halves of result.	2-16
ADD	103000	Add ACS to ACD; use Carry as base for carry bit.	2-15
ADDC	103060	Add ACS to ACD; use complement of Carry as base for carry bit.	2-15
ADDCL	103160	Add ACS to ACD; use complement of Carry as base for carry bit; rotate left.	2-15
ADDCR	103260	Add ACS to ACD; use complement of Carry as base for carry bit; rotate right.	2-15
ADDCS	103360	Add ACS to ACD; use complement of Carry as base for carry bit; swap halves of result.	2-15
ADDL	103100	Add ACS to ACD; use Carry as base for carry bit; rotate left.	2-15
ADDO	103040	Add ACS to ACD; use 1 as base for carry bit.	2-15
ADDOL	103140	Add ACS to ACD; use 1 as base for carry bit; rotate left.	2-15

			<i>Page</i>
ADDOR	103240	Add ACS to ACD; use 1 as base for carry bit; rotate right.	2-15
ADDOS	103340	Add ACS to ACD; use 1 as base for carry bit; swap halves of result.	2-15
ADDR	103200	Add ACS to ACD; use Carry as base for carry bit; rotate right.	2-15
ADDS	103300	Add ACS to ACD; use Carry as base for carry bit; swap halves of result.	2-15
ADDZ	103020	Add ACS to ACD; use 0 as base for carry bit.	2-15
ADDZL	103120	Add ACS to ACD; use 0 as base for carry bit; rotate left.	2-15
ADDZR	103220	Add ACS to ACD; use 0 as base for carry bit; rotate right.	2-15
ADDZS	103320	Add ACS to ACD; use 0 as base for carry bit; swap halves of result.	2-15
AND	103400	And ACS with ACD; use Carry as carry bit.	2-16
ANDC	103460	And ACS with ACD; use complement of Carry as carry bit.	2-16
ANDCL	103560	And ACS with ACD; use complement of Carry as carry bit; rotate left.	2-16
ANDCR	103660	And ACS with ACD; use complement of Carry as carry bit; rotate right.	2-16
ANDCS	103760	And ACS with ACD; use complement of Carry as carry bit; swap halves of result.	2-16
ANDL	103500	And ACS with ACD; use Carry as carry bit; rotate left.	2-16
ANDO	103440	And ACS with ACD; use 1 as carry bit.	2-16
ANDOL	103540	And ACS with ACD; use 1 as carry bit; rotate left.	2-16
ANDOR	103640	And ACS with ACD; use 1 as carry bit; rotate right.	2-16
ANDOS	103740	And ACS with ACD; use 1 as carry bit; swap halves of result.	2-16
ANDR	103600	And ACS with ACD; use Carry as carry bit; rotate right.	2-16
ANDS	103700	And ACS with ACD; use Carry as carry bit; swap halves of result.	2-16
ANDZ	103420	And ACS with ACD; use 0 as carry bit.	2-16
ANDZL	103520	And ACS with ACD; use 0 as carry bit; rotate left.	2-16
ANDZR	103620	And ACS with ACD; use 0 as carry bit; rotate right.	2-16
ANDZS	103720	And ACS with ACD; use 0 as carry bit; swap halves of result.	2-16
COM	100000	Place the complement of ACS in ACD; use Carry as carry bit.	2-14
COMC	100060	Place the complement of ACS in ACD; use complement of Carry as carry bit.	2-14
COMCL	100160	Place the complement of ACS in ACD; use complement of Carry as carry bit; rotate left.	2-14
COMCR	100260	Place the complement of ACS in ACD; use complement of Carry as carry bit; rotate right.	2-14
COMCS	100360	Place the complement of ACS in ACD; use complement of Carry as carry bit; swap halves of result.	2-14
COML	100100	Place the complement of ACS in ACD; use Carry as carry bit; rotate left.	2-14
COMO	100040	Place the complement of ACS in ACD; use 1 as carry bit.	2-14
COMOL	100140	Place the complement of ACS in ACD; use 1 as carry bit; rotate left.	2-14

			<i>Page</i>
COMOR	100240	Place the complement of ACS in ACD; use 1 as carry bit; rotate right.	2-14
COMOS	100340	Place the complement of ACS in ACD; use 1 as carry bit; swap halves of result.	2-14
COMR	100200	Place the complement of ACS in ACD; use Carry as carry bit; rotate right.	2-14
COMS	100300	Place the complement of ACS in ACD; use Carry as carry bit; swap halves of result.	2-14
COMZ	100020	Place the complement of ACS in ACD; use 0 as carry bit.	2-14
COMZL	100120	Place the complement of ACS in ACD; use 0 as carry bit; rotate left.	2-14
COMZR	100220	Place the complement of ACS in ACD; use 0 as carry bit; rotate right.	2-14
COMZS	100320	Place the complement of ACS in ACD; use 0 as carry bit; swap halves of result.	2-14
DIA	060400	Data in, A buffer to AC.	2-24
DIAC	060600	Data in, A buffer to AC; clear device.	2-24
DIAP	060700	Data in, A buffer to AC; send special pulse to device.	2-24
DIAS	060500	Data in, A buffer to AC; start device.	2-24
DIB	061400	Data in, B buffer to AC.	2-24
DIBC	061600	Data in, B buffer to AC; clear device.	2-24
DIBP	061700	Data in, B buffer to AC; send special pulse to device.	2-24
DIBS	061500	Data in, B buffer to AC; start device.	2-24
DIC	062400	Data in, C buffer to AC.	2-25
DICC	062600	Data in, C buffer to AC; clear device.	2-25
DICP	062700	Data in, C buffer to AC; send special pulse to device.	2-25
DICS	062500	Data in, C buffer to AC; start device.	2-25
DIV	073101	If overflow, set Carry. Otherwise divide AC0-AC1 by AC2. Put quotient in AC1, remainder in AC0.	2-41
DOA	061000	Data out, AC to A buffer.	2-24
DOAC	061200	Data out, AC to A buffer; clear device.	2-24
DOAP	061300	Data out, AC to A buffer; send special pulse to device.	2-24
DOAS	061100	Data out, AC to A buffer; start device.	2-24
DOB	062000	Data out, AC to B buffer.	2-25
DOBC	062200	Data out, AC to B buffer; clear device.	2-25
DOBP	062300	Data out, AC to B buffer; send special pulse to device.	2-25
DOBS	062100	Data out, AC to B buffer; start device.	2-25
DOC	063000	Data out, AC to C buffer.	2-25
DOCC	063200	Data out, AC to C buffer; clear device.	2-25
DOCP	063300	Data out, AC to C buffer; send special pulse to device.	2-25
DOCS	063100	Data out, AC to C buffer; start device.	2-25
DSZ	014000	Decrement location <i>E</i> by 1 and skip if result is zero.	2-6

			<i>Page</i>
HALT	063077	Halt the processor (= DOC 0,CPU).	2-28
INC	101400	Place ACS + 1 in ACD; use Carry as base for carry bit.	2-15
INCC	101460	Place ACS + 1 in ACD; use complement of Carry as base for carry bit.	2-15
INCL	101560	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; rotate left.	2-15
INCCR	101660	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; rotate right.	2-15
INCCS	101760	Place ACS + 1 in ACD; use complement of Carry as base for carry bit; swap halves of result.	2-15
INCL	101500	Place ACS + 1 in ACD; use Carry as base for carry bit; rotate left.	2-15
INCO	101440	Place ACS + 1 in ACD; use 1 as base for carry bit.	2-15
INCOL	101540	Place ACS + 1 in ACD; use 1 as base for carry bit; rotate left.	2-15
INCOR	101640	Place ACS + 1 in ACD; use 1 as base for carry bit; rotate right.	2-15
INCOS	101740	Place ACS + 1 in ACD; use 1 as base for carry bit; swap halves of result.	2-15
INCR	101600	Place ACS + 1 in ACD; use Carry as base for carry bit; rotate right.	2-15
INCS	101700	Place ACS + 1 in ACD; use Carry as base for carry bit; swap halves of result.	2-15
INCZ	101420	Place ACS + 1 in ACD; use 0 as base for carry bit.	2-15
INCZL	101520	Place ACS + 1 in ACD; use 0 as base for carry bit; rotate left.	2-15
INCZR	101620	Place ACS + 1 in ACD; use 0 as base for carry bit; rotate right.	2-15
INCZS	101720	Place ACS + 1 in ACD; use 0 as base for carry bit; swap halves of result.	2-15
INTA	061477	Acknowledge interrupt by loading code of nearest device that is requesting an interrupt into AC bits 10–15 (= DIB –,CPU).	2-33
INTDS	060277	Disable interrupt by clearing Interrupt On (= NIOC CPU).	2-32
INTEN	060177	Enable interrupt by setting Interrupt On (= NIOS CPU).	2-32
IORST	062677	Clear all IO devices, clear Interrupt On, reset clock to line frequency (= DICC 0,CPU).	2-28
ISZ	010000	Increment location <i>E</i> by 1 and skip if result is zero.	2-6
JMP	000000	Jump to location <i>E</i> (put <i>E</i> in PC).	2-7
JSR	004000	Load PC + 1 in AC3 and jump to subroutine at location <i>E</i> (put <i>E</i> in PC).	2-7
LDA	020000	Load contents of location <i>E</i> into AC.	2-5
MOV	101000	Move ACS to ACD; use Carry as carry bit.	2-15
MOV C	101060	Move ACS to ACD; use complement of Carry as carry bit.	2-15
MOVCL	101160	Move ACS to ACD; use complement of Carry as carry bit; rotate left.	2-15
MOVCR	101260	Move ACS to ACD; use complement of Carry as carry bit; rotate right.	2-15
MOVCS	101360	Move ACS to ACD; use complement of Carry as carry bit; swap halves of result.	2-15
MOVL	101100	Move ACS to ACD; use Carry as carry bit; rotate left.	2-15

			<i>Page</i>
MOV0	101040	Move ACS to ACD; use 1 as carry bit.	2-15
MOVOL	101140	Move ACS to ACD; use 1 as carry bit; rotate left.	2-15
MOVOR	101240	Move ACS to ACD; use 1 as carry bit; rotate right.	2-15
MOVOS	101340	Move ACS to ACD; use 1 as carry bit; swap halves of result.	2-15
MOVR	101200	Move ACS to ACD; use Carry as carry bit; rotate right.	2-15
MOVS	101300	Move ACS to ACD; use Carry as carry bit; swap halves of result.	2-15
MOVZ	101020	Move ACS to ACD; use 0 as carry bit.	2-15
MOVZL	101120	Move ACS to ACD; use 0 as carry bit; rotate left.	2-15
MOVZR	101220	Move ACS to ACD; use 0 as carry bit; rotate right.	2-15
MOVZS	101320	Move ACS to ACD; use 0 as carry bit; swap halves of result.	2-15
MSKO	062077	Set up Interrupt Disable flags according to mask in AC (= DOB -, CPU).	2-33
MUL	073301	Multiply AC1 by AC2, add product to AC0, put result in AC0-AC1.	2-41
NEG	100400	Place negative of ACS in ACD; use Carry as base for carry bit.	2-15
NEGC	100460	Place negative of ACS in ACD; use complement of Carry as base for carry bit.	2-15
NEGCL	100560	Place negative of ACS in ACD; use complement of Carry as base for carry bit; rotate left.	2-15
NEGCR	100660	Place negative of ACS in ACD; use complement of Carry as base for carry bit; rotate right.	2-15
NEGCS	100760	Place negative of ACS in ACD; use complement of Carry as base for carry bit; swap halves of result.	2-15
NEGL	100500	Place negative of ACS in ACD; use Carry as base for carry bit; rotate left.	2-15
NEGO	100440	Place negative of ACS in ACD; use 1 as base for carry bit.	2-15
NEGOL	100540	Place negative of ACS in ACD; use 1 as base for carry bit; rotate left.	2-15
NEGOR	100640	Place negative of ACS in ACD; use 1 as base for carry bit; rotate right.	2-15
NEGOS	100740	Place negative of ACS in ACD; use 1 as base for carry bit; swap halves of result.	2-15
NEGR	100600	Place negative of ACS in ACD; use Carry as carry bit; rotate right.	2-15
NEGS	100700	Place negative of ACS in ACD; use Carry as carry bit; swap halves of result.	2-15
NEGZ	100420	Place negative of ACS in ACD; use 0 as base for carry bit.	2-15
NEGZL	100520	Place negative of ACS in ACD; use 0 as base for carry bit; rotate left.	2-15
NEGZR	100620	Place negative of ACS in ACD; use 0 as base for carry bit; rotate right.	2-15
NEGZS	100720	Place negative of ACS in ACD; use 0 as base for carry bit; swap halves of result.	2-15
NIO	060000	No operation.	2-23
NIOC	060200	Clear device.	2-23
NIOP	060300	Send special pulse to device.	2-23

			<i>Page</i>
NIOS	060100	Start device.	2-23
READS	060477	Read console data switches into AC (= DIA -,CPU).	2-27
SBN	000007	Skip if both carry and result are nonzero (skip function in an arithmetic or logical instruction).	2-13
SEZ	000006	Skip if either carry or result is zero (skip function in an arithmetic or logical instruction).	2-13
SKP	000001	Skip (skip function in an arithmetic or logical instruction).	2-13
SKPBN	063400	Skip if Busy is 1.	2-23
SKPBZ	063500	Skip if Busy is 0.	2-23
SKPDN	063600	Skip if Done is 1.	2-23
SKPDZ	063700	Skip if Done is 0.	2-23
SNC	000003	Skip if carry bit is 1 (skip function in an arithmetic or logical instruction).	2-13
SNR	000005	Skip if result is nonzero (skip function in an arithmetic or logical instruction).	2-13
STA	040000	Store AC in location <i>E</i> .	2-5
SUB	102400	Subtract ACS from ACD; use Carry as base for carry bit.	2-16
SUBC	102460	Subtract ACS from ACD; use complement of Carry as base for carry bit.	2-16
SUBCL	102560	Subtract ACS from ACD; use complement of Carry as base for carry bit; rotate left.	2-16
SUBCR	102660	Subtract ACS from ACD; use complement of Carry as base for carry bit; rotate right.	2-16
SUBCS	102760	Subtract ACS from ACD; use complement of Carry as base for carry bit; swap halves of result.	2-16
SUBL	102500	Subtract ACS from ACD; use Carry as base for carry bit; rotate left.	2-16
SUBO	102440	Subtract ACS from ACD; use 1 as base for carry bit.	2-16
SUBOL	102540	Subtract ACS from ACD; use 1 as base for carry bit; rotate left.	2-16
SUBOR	102640	Subtract ACS from ACD; use 1 as base for carry bit; rotate right.	2-16
SUBOS	102740	Subtract ACS from ACD; use 1 as base for carry bit; swap halves of result.	2-16
SUBR	102600	Subtract ACS from ACD; use Carry as base for carry bit; rotate right.	2-16
SUBS	102700	Subtract ACS from ACD; use Carry as base for carry bit; swap halves of result.	2-16
SUBZ	102420	Subtract ACS from ACS; use 0 as base for carry bit.	2-16
SUBZL	102520	Subtract ACS from ACD; use 0 as base for carry bit; rotate left.	2-16
SUBZR	102620	Subtract ACS from ACD; use 0 as base for carry bit; rotate right.	2-16
SUBZS	102720	Subtract ACS from ACD; use 0 as base for carry bit; swap halves of result.	2-16
SZC	000002	Skip if carry is 0 (skip function in an arithmetic or logical instruction).	2-13
SZR	000004	Skip if result is zero (skip function in an arithmetic or logical instruction).	2-13
D10			



			<i>Page</i>
@	002000	When this character appears in an instruction, the assembler places a 1 in bit 5 to produce indirect addressing.	2-3
@	100000	When this character appears with a 15-bit address, the assembler places a 1 in bit 0, making the address indirect.	2-3
#	000010	Appending this character to the mnemonic for an arithmetic or logical instruction places a 1 in bit 12 to prevent the processor from loading the 17-bit result in Carry and ACD. Thus the result of an instruction can be tested for a skip without affecting Carry or the accumulators.	2-13

## INSTRUCTION EXECUTION TIMES

Supernova read-only time equals semiconductor time, except add .2 for LDA, STA, ISZ, DSZ if reference is to core. Nova times are for core; for read-only subtract .2 except subtract .4 for LDA, STA, ISZ, DSZ if reference is to read-only memory.

When two numbers are given, the one at the left of the slash is the time for an isolated transfer, the one at the right is the minimum time between consecutive transfers. All times in microseconds.

	Supernova				
	SC	Core	800 Series	1200 Series	Nova
LDA	1.2	1.6	1.6	2.55	5.2
STA	1.2	1.6	1.6	2.55	5.5
ISZ, DSZ	1.4	1.8	1.8	3.15*	5.2
JMP	.6	.8	.8	1.35	2.6
JSR	1.2	1.4	.8	1.35	3.5
Indirect addressing add	.6	.8	.8	1.2	2.6
Base register addressing add	0	0	0	0	.3
Autoindexing add	.2	.2	.2	.6	0
COM, NEG, MOV, INC	.3*	.8*	.8*	1.35*	5.6
ADC, SUB, ADD, AND	.3*	.8*	.8*	1.35*	5.9
*If skip occurs add	‡	.8	.2	1.35	
IO input (except INTA)	2.8	2.9	2.2†	2.55	4.4
NIO	3.2	3.3	2.2†	3.15	4.4
IO output	3.2	3.3	2.2†	3.15	4.7
†S, C or P add			.6		
IO skips	2.8	2.9	1.4*	2.55	4.4
INTA	3.6	3.7	2.2	2.55	4.4
MUL			8.8	3.75	11.1
Average	3.7	3.8			
Maximum	5.3	5.4			
DIV	6.8	6.9	8.8	4.05	11.9
Unsuccessful	1.5	1.6	1.6	2.55	
Interrupt	1.8	2.2	1.6	3.0	5.2
Latency				7	12
With multiply-divide	9	9	10.6		
Without multiply-divide	5	5	4.6		
Data Channel					
Input	2.3	2.3	2.0	1.2	3.5
Output	2.8	2.8	2.0	1.2/1.8	4.4
Increment	2.8	2.8	2.2	1.8/2.4	4.4
Add	2.8	2.8			5.3
Latency			3.6	7	12
With multiply-divide	9	9			
Without multiply-divide	5	5			
High speed channel					
Input	.8	.8	.8		
Output	.8/1.0	.8/1.0	.8/1.0		
Increment	1.0/1.2	1.0/1.2	1.0/1.2		
Add	1.0/1.2	1.0/1.2			
Latency					
With IO	4.5	4.5	3.6		
Without IO	2.5	2.5	2.0		

‡ Add .3 if arithmetic or logical instruction is skipped, otherwise add .6.

## APPENDIX E

### IN-OUT CODES

The table on the next two pages lists the in-out devices, their octal codes, mnemonics and DGC option numbers. 4000 series options are for all machines or the Nova computer only, 8100 series options are for the 1200 series (but occasionally for the 1200 and 800 series), 8200 for the 800 series only, and 8000 for the Supernova computer only. Codes 40 and above are used in pairs (40-42, 42-43, . . .) for receiver-transmitter sets in the high speed communications controller. The table beginning on page E4 lists the complete teletype code. The lower case character set (codes 140-176) is not available on the Model 33 or 35, but giving one of these codes causes the teletypewriter to print the corresponding upper case character. Other differences between the 33-35 and the 37 are mentioned in the table. The definitions of the control codes are those given by ASCII. Most control codes, however, have no effect on the computer teletypewriter, and the definitions bear no necessary relation to the use of the codes in conjunction with the software.

## IN-OUT DEVICES

Device Code (Octal)	Mnemonic	Priority Mask Bit	Device
00			
01	MDV	—	Multiply/Divide Option
02			
03			
04			
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	Multi-Line Asynchronous controller
35	MX2 }		(MAC)
36			
37			
40+	SCR	8	Synchronous communications receiver
41•	SCT	8	Synchronous communications transmitter
42	DIO	7	Digital I/O
43	DIOT	6	Digital I/O timer
44	MXM	12	Modem control for MAC
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

Device Code (Octal)	Mnemonic	Priority Mask Bit	Device
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			
62	MTA1	10	Second magnetic tape
63			
64*	FPU1	5	} Alternative location for floating point
65	FPU2	5	
66	FPU	5	
67			
70	QTY1	14	Second asynchronous hardware multiplexor
71* }			
72 }		13	Second IBM 360/370 interface
73	DKP1	7	Second moving head disc
74	CAS1	10	Second cassette tape
74*	FPU1	5	} Floating point option
75	FPU2	5	
76	FPU	5	
77	CPU	—	Central processor and console functions

\* code returned by INTA

+ 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

## TELETYPE CODE

Even Parity Bit	7-Bit Octal Code	Character	Remarks
0	000	NUL	Null, tape feed. Repeats on Model 37. Control shift P on Model 33 and 35.
1	001	SOH	Start of heading; also SOM, start of message. Control A.
1	002	STX	Start of text; also EOA, end of address. Control B.
0	003	ETX	End of text; also EOM, end of message. Control C.
1	004	EOT	End of transmission (END); shuts off TWX machines. Control D.
0	005	ENQ	Enquiry (ENQRY); also WRU, "Who are you?" Triggers identification ("Here is. . .") at remote station if so equipped. Control E.
0	006	ACK	Acknowledge; also RU, "Are you. . .?" Control F.
1	007	BEL	Rings the bell. Control G.
1	010	BS	Backspace; also FEO, format effector. Backspaces some machines. Repeats on Model 37. Control H on Model 33 and 35.
0	011	HT	Horizontal tab. Control I on Model 33 and 35.
0	012	LF	Line feed or line space (NEW LINE); advances paper to next line. Repeats on Model 37. Duplicated by control J on Model 33 and 35.
1	013	VT	Vertical tab (VTAB). Control K on Model 33 and 35.
0	014	FF	Form feed to top of next page (PAGE). Control L.
1	015	CR	Carriage return to beginning of line. Control M on Model 33 and 35.
1	016	SO	Shift out; changes ribbon color to red. Control N.
0	017	SI	Shift in; changes ribbon color to black. Control O.
1	020	DLE	Data link escape. Control P (DC0).
0	021	DC1	Device control 1, turns transmitter (reader) on. Control Q (X ON).
0	022	DC2	Device control 2, turns punch or auxiliary on. Control R (TAPE, AUX ON).
1	023	DC3	Device control 3, turns transmitter (reader) off. Control S (X OFF).
0	024	DC4	Device control 4, turns punch or auxiliary off. Control T (AUX OFF).
1	025	NAK	Negative acknowledge; also ERR, error. Control U.
1	026	SYN	Synchronous idle (SYNC). Control V.
0	027	ETB	End of transmission block; also LEM, logical end of medium. Control W.
0	030	CAN	Cancel (CANCL). Control X.
	031	EM	End of medium. Control Y.
1	032	SUB	Substitute. Control Z.
0	033	ESC	Escape, prefix. This code is also generated by control shift K on Model 33 and 35.
1	034	FS	File separator, Control shift L on Model 33 and 35.
0	035	GS	Group separator. Control shift M on Model 33 and 35.
0	036	RS	Record separator. Control shift N on Model 33 and 35.
1	037	US	Unit separator. Control shift O on Model 33 and 35.
1	040	SP	Space.
0	041	!	
0	042	"	

Even Parity Bit	7-Bit Octal Code	Character	Remarks
1	043	#	
0	044	\$	
1	045	%	
1	046	&	
0	047	'	Accent acute or apostrophe.
0	050	(	
1	051	)	
1	052	*	Repeats on Model 37.
0	053	+	
1	054	,	
0	055	-	Repeats on Model 37.
0	056	.	Repeats on Model 37.
1	057	/	
0	060	Ø	
1	061	1	
1	062	2	
0	063	3	
1	064	4	
0	065	5	
0	066	6	
1	067	7	
1	070	8	
0	071	9	
0	072	:	
1	073	;	
0	074	<	
1	075	=	Repeats on Model 37.
1	076	>	
0	077	?	
1	100	@	
0	101	A	
0	102	B	
1	103	C	
0	104	D	
1	105	E	
1	106	F	
0	107	G	
0	110	H	
1	111	I	
1	112	J	
0	113	K	
1	114	L	
0	115	M	

Even Parity Bit	7-Bit Octal Code	Character	Remarks
0	116	N	
1	117	O	
0	120	P	
1	121	Q	
1	122	R	
0	123	S	
1	124	T	
0	125	U	
0	126	V	
1	127	W	
1	130	X	Repeats on Model 37.
0	131	Y	
0	132	Z	
1	133	[	Shift K on Model 33 and 35.
0	134	\	Shift L on Model 33 and 35.
1	135	]	Shift M on Model 33 and 35.
1	136	↑	
0	137	←	Repeats on Model 37.
0	140	`	Accent grave.
1	141	a	
1	142	b	
0	143	c	
1	144	d	
0	145	e	
0	146	f	
1	147	g	
1	150	h	
0	151	i	
0	152	j	
1	153	k	
0	154	l	
1	155	m	
1	156	n	
0	157	o	
1	160	p	
0	161	q	
0	162	r	
1	163	s	
0	164	t	
1	165	u	
1	166	v	
0	167	w	
0	170	x	Repeats on Model 37.



Even Parity Bit	7-Bit Octal Code	Character	Remarks
1	171	y	
1	172	z	
0	173	{	
1	174		
0	175	}	
0	176	~	{ On early versions of the Model 33 and 35, either of these codes may be generated by either the ALT MODE or ESC key. Delete, rub out. Repeats on Model 37.
1	177	DEL	

#### Keys That Generate No Codes

REPT	Model 33 and 35 only: causes any other key that is struck to repeat continuously until REPT is released.
PAPER ADVANCE	Model 37 local line feed.
LOCAL RETURN	Model 37 local carriage return.
LOC LF	Model 33 and 35 local line feed.
LOC CR	Model 33 and 35 local carriage return.
INTERRUPT, BREAK	Opens the line (machine sends a continuous string of null characters).
PROCEED, BRK RLS	Break release (not applicable).
HERE IS	Transmits predetermined 20-character message.

