gri-99

System Reference Manual

gri GRI Computer Corporation
320 NEEDHAM STREET, NEWTON, MASSACHUSETTS 02164

10-50-001C
100-10-75

October 1972

This manual is for informational purposes only.
The technical information is not to be construed
as engineering specifications.

Printed in U.S.A.
08-73-200

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

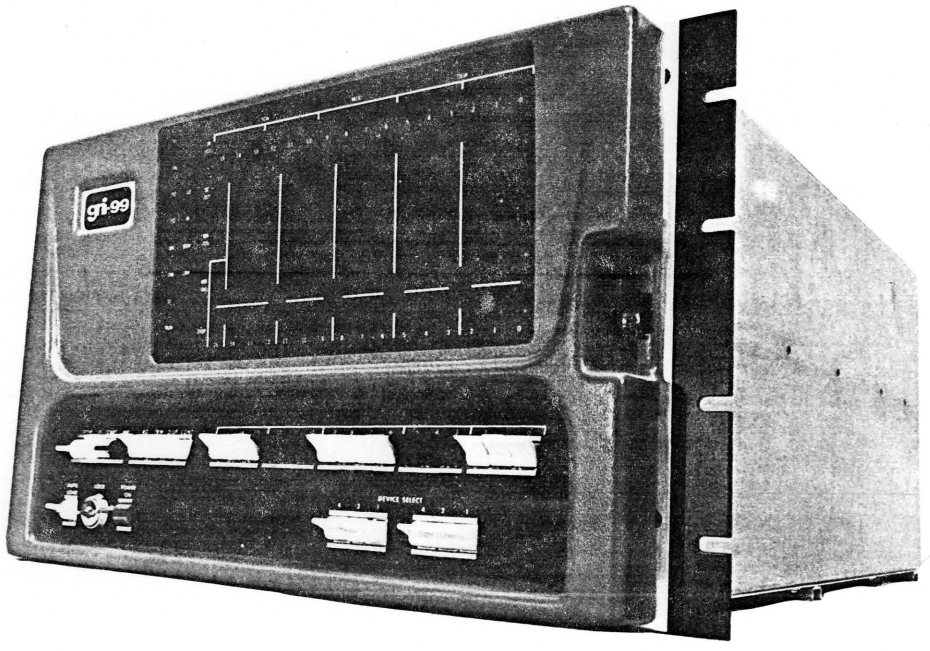
TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF FIGURES

LIST OF TABLES



THE GRI-99

CHAPTER 1
THE GRI-99

The GRI-99 Computer is a compact 16-bit machine that combines direct function processing and the Universal Bus System to provide versatility and reliability unequaled in conventional computers. The GRI-99 is a rugged computer, especially designed to be an economical, powerful part of larger dedicated systems. To meet the needs of the system designer: Interfacing is easy, maintenance is minimal, and data handling is fast and efficient. Programming is directed to performing specific functions, rather than being strictly limited to complex arithmetic or mathematical calculations; the assembly language is easy to understand and highly systems oriented. Section 1.3 lists the specifications for the GRI-99 Computer.

1.1 FEATURES

The GRI-99 is completely flexible to satisfy the changing demands of today's sophisticated systems and is expandable to fulfill the new requirements of future systems.

- *The Universal Bus System* – Each system element can communicate with any other system element, internal or external, using high-speed data buses.
- *Memory* – Maximum of 32,768 words or random-access core memory directly addressable (not page oriented). Minimum core size is 4096 words.
- *Programming* – Simple programming in a very functional language.
- *Software* – All previous GRI software is totally compatible with the new GRI-99. New software packages are also available.
- *Expansion* – Space is available for 32K words of memory, 2 major firmware modules and up to 9 device interface modules in the mainframe.
- *Reliability* – The Universal Bus System uses simple two-sided printed circuit (PC) boards. *Back panel wiring and data cables are eliminated.* Field proven TTL integrated circuits, medium scale integrated circuits, conservative design techniques, and stringent quality control ensure maximum reliability in industrial, electrical, and physical environments.
- *Displays* – All system registers, both internal and external to the computer, can be displayed on the console. Data can be transferred to all system devices using console switches. The displays are illuminated by bright, long-life light emitting diodes (LEDs).
- *Peripherals* – Peripheral options include: mass memory media, I/O devices, communication interfaces, display, and digital system devices.
- *Direct Memory Access* – Any device can transfer data directly to memory. Direct memory access (DMA) channel is available on the same data and control lines as the programmed input/output (I/O) channel (I/O rate: 568,000 16-bit words per second maximum). *No DMA multiplexer is required for multiple DMA devices.*
- *Direct Data Transfer* – Every device in the system is directly addressable by programmed instructions. Data can be transferred between devices without special accumulators or temporary storage; thus, many computer instructions normally required for data manipulation are eliminated.
- *Firmware* – Firmware options are plug-in, hardwired modules to expand the instruction set and provide system flexibility unequaled by more conventional computer designs (e.g., multiple arithmetic units, extended arithmetic options, byte manipulation hardware general-purpose registers).
- *Priority Interrupt System* – Priority interrupt system can be used as a single-channel interrupt or as an automatic hardware interrupt at the option of the system designer. Most standard GRI I/O devices are equipped with full auto-hardware interrupt.
- *Index Register* – An Index Register, compatible with many GRI-99 instructions, makes programming more efficient and easier.
- *Protection* – Memory power fail protection is standard in all models. Also, remote start and stop are available for system interfacing. Automatic restart after power failure is optional.

1.2 DESIGN CONCEPT

The GRI-99 Computer is a *direct function processor* (see Figure 1-1). Direct function processing, a unique concept in computer-controller design, enables each system element to communicate directly with any other systems element using shared high-speed data buses. System elements that utilize this direct access feature include registers internal to the computer or devices external to the computer. Thus, such peripheral devices as a Teletype ® or CRT are connected across the same bus structure as the arithmetic unit and memory of the computer, as well as the other processor elements. Data transfer between external devices and computer devices is accomplished directly, in a single operation, with no temporary storage of data in special I/O registers or accumulators. As a result, data flow between devices increases, and no special complicated command repertoire is necessary to process and translate information transmitted between the computer and the device controlled.

A vital block of logic called the *Bus Modifier* (see Figure 1-1) provides a programmable path between the Source and Destination Buses. The Bus Modifier is designed to accept data from any source device and move the data to any Destination device. Moreover, the Bus Modifier can perform operations "on-the-fly" as the data pass from one device to the other. Operations that can be performed on the data include: left shift, right shift, one's complement, two's complement, and no modification.

The modular design inherent in direct function processing encourages many machine configurations, ranging from highly economical minimal processors (for systems requiring simple data manipulations) to large systems with powerful computing instructions and a variety of peripheral devices. Hardwired firmware devices are available in the form of plug-in modules to augment the current instruction set with virtually thousands of computer instructions. Plug-in modules provide flexibility and expandability not available in conventional computer designs.

The direct function processing technique, developed in the GRI-909 used in the GRI-99, is the culmination of many years of experience in both the design of computers and the use of computers in systems. The GRI-99 is a tested, proven systems control computer. Its flexibility, modularity, and ease of programming provide the original equipment or systems manufacturer with a control center that minimizes many of the problems inherent in conventional computer designs. The modular firmware capability of the GRI-99 offers the system designer the flexibility to meet changing system requirements, the ability to incorporate proprietary and unique control features, and a solid hedge against obsolescence caused by the introduction of new system devices or circuitry.

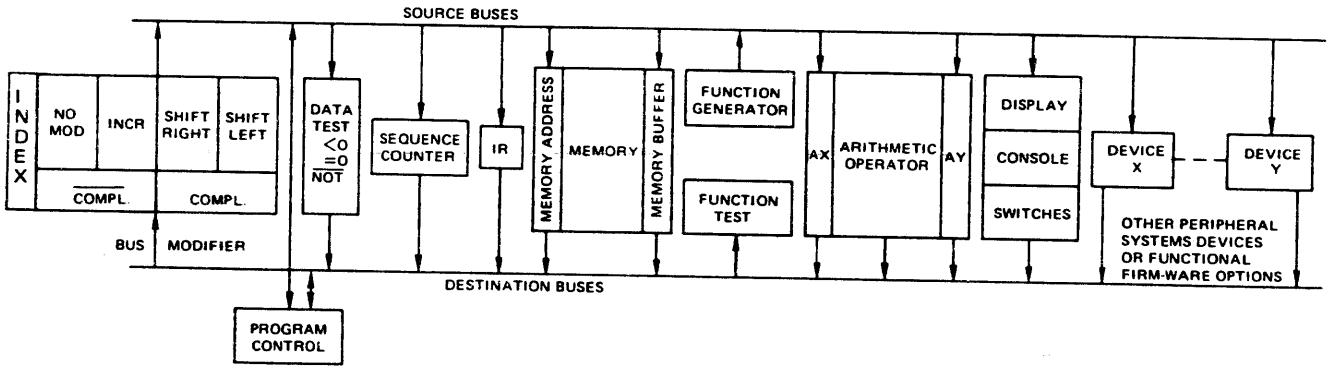


Figure 1-1 Direct Function Processor, Typical Configuration

® Teletype is a registered trademark of Teletype Corp.

1.3 SPECIFICATIONS

1.3.1 Physical

The physical dimensions of the GRI-99 Computer are:

Size: 10-1/2 in. high (26.67 cm), 19 in. wide (48.26 cm), 22 in. deep (55.88 cm)
Weight: 40 lb. (18.18 kg)

The GRI-99 mounts from the front in a standard EIA 19-in. cabinet with provision for rack slides. Space is provided in the basic frame for up to 2 major firmware options and up to 9 firmware or interface modules. Memory capacity of the basic frame is 32K 16-bit words.

1.3.2 Functional

The functional specifications for the GRI-99 Computer are:

Word Length:	16 bits
Core Memory Size:	4096 words, expandable to 32,768 words, direct addressing.
Machine Cycle Time:	1.76 μs when executing instructions from main memory; 880 ns in External Instruction (EI) mode.
Instructions:	The number of machine instructions is modular and depends on the firmware and device options used. A basic processor with arithmetic operator has over 233 instructions.

1.3.3 Electrical

The electrical specifications for the GRI-99 Computer are:

Power:	99-132 Vac or 196-243 Vac, 50Hz $\pm$ 3% or 60 Hz $\pm$ 3%, 1A – 3A, single-phase. (110 Vac – 220 Vac nominal)
Power Dissipation:	150W nominal
I/O Logic Levels:	DTL and TTL compatible.

1.3.4 Environmental

The environmental specifications for the GRI-99 Computer are:

Temperature	0^{o}C to 55^{o}C ambient, based on measurements made at 5 inches below the computer chassis. See cooling for further explanations.
Relative Humidity: (non-condensing)	to 95% (operating)
Cooling:	The frame and board configuration is designed to allow airflow through it. The computer will properly cool itself if suspended in a space which allows at least 3 inches above and below the frame and no additional heat is added by the presence of other equipment. The ultimate criterion for proper cooling must be the temperature of the core stack itself since this is the most temperature sensitive component in the system. The maximum allowable temperature of the stack as measured on the back surface of the core stack board (which allows for a temperature rise across the board material) is 60^{o} (140^{o}F).

CHAPTER 2
GRI-99 SYSTEMS

2.1 GRI-99 MODEL 40

The GRI-99 Model 40 (see Figure 2-1) is the top of the line in the GRI-99 series. In a wide range of applications, the Model 40 clearly demonstrates that it is GRI's most powerful computer in terms of computational ability. The Model 40, with full programmers console, is easy to program; results are available immediately at the five light emitting diode (LED) displays or from a wide choice of peripheral devices.

The GRI-99 Model 40 offers the following features:

a. Low cost and excellent price/performance ratio.

b. Powerful hardware extended arithmetic capability.

c. Extensive systems-oriented, real-time software, simulator, source text editor, math package, floating point interpreter, utility routines, relocatable assembler, and much more.

d. Over 233 instruction types, as well as many variations.

e. Six general-purpose registers.

f. Unlimited priority interrupt levels.

g. Multiply/divide and floating point firmware. Typical floating point execution times are as follows:

Add:	247 μs	Multiply:	376 μs
Add Magnitude:	256 μs	Square:	332 μs
Sine:	3836 μs	Square Root:	543 μs

h. Four I/O slots (expandable to nine slots) for plug-in modules.

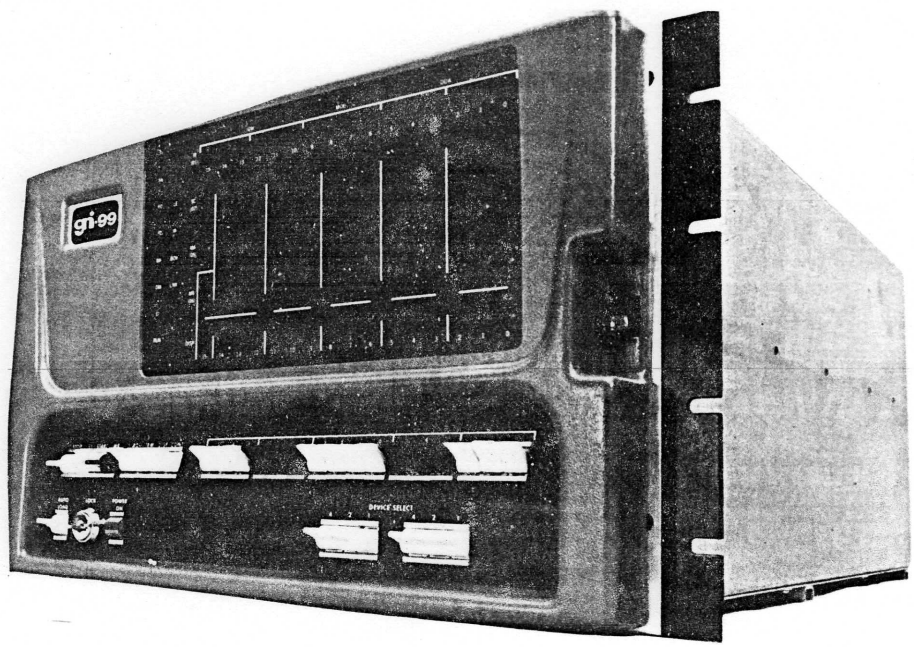


Figure 2-1 GRI-99 Model 40

2.2 GRI-99 MODEL 30

The GRI-99 Model 30 provides many of the features of the Model 40 at a lower price. The front panel of the Model 30 console provides a single set of indicators for data display (see Figure 2-2). The contents of all registers can be displayed in the DISP indicators. The processor cycle indicators are the same as the Model 40 indicators.

The GRI-99 Model 30 combines moderate price with very efficient data manipulation. The Model 30 offers the following features:

a. Excellent price/performance ratio

b. Powerful real-time software, a simulator, source text editor, math package, floating point interpreter, utility routines, relocatable assembler.

c. Over 200 instructions as well as many variations.

d. Unlimited priority interrupt levels.

e. Four I/O slots (expandable to nine slots) for plug-in modules.

2.3 GRI-99 MODEL 10

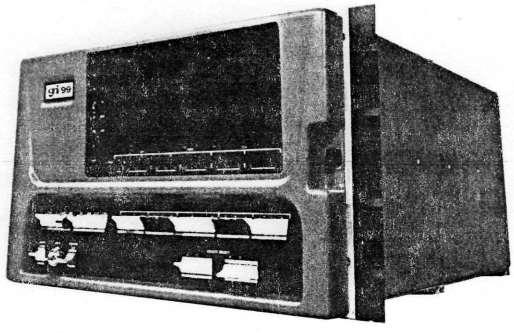
The GRI-99 Model 10 is the most economical of the GRI-99 series. It is particularly effective as a programmable controller in dedicated applications.

A blank panel is provided in the GRI-99 Model 10 (see Figure 2-2). Automatic restart is standard. Programs can be loaded by interchanging the blank front panel with one of the console options or by means of an auto-load for a Grisette II (refer to 2.5.1 and 2.5.2) cassette tape I/O system.

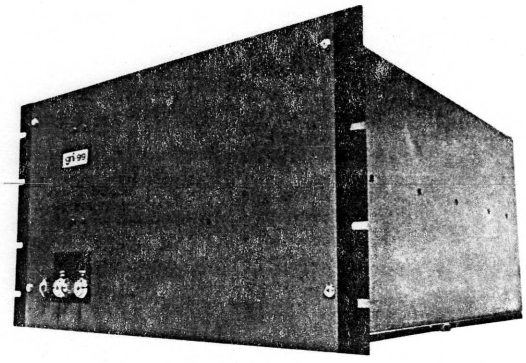
The GRI-99 Model 10 combines many of the key features of the other models in the line with extremely low unit cost. The Model 10 features:

a. Excellent price/performance ratio.

b. Memory expansion from 4K to 32K in the main frame.



GRI-99 Model 30



GRI-99 Model 10

Figure 2-2

c. Blank front panel with power switch and provision for autoload switch mounting (optional).
d. Auto restart.
e. Over 200 instructions as well as many variations.

2.4 PROCESSOR OPTIONS

A wide variety of processor options is available to expand the capabilities of GRI computers. The processor options selected depend on the user's specific application and anticipation of future needs.

The following are common processor options available for all GRI-99 computers:

a. Byte swap/pack operator.
b. Byte comparator.
c. Binary input multiplexer (BIM).
d. Binary output multiplexer (BOM).
e. Gate input card.
f. Pulse input detector.
g. General Output register.
h. Watchdog/Interval timer.
i. Selector channel.

2.4.1 **Byte Swap/Pack** – The Byte Swap/Byte Pack Option is a plug-in firmware card that allows the user to swap or pack two 8-bit bytes. When byte swap (BSW) is specified, a 16-bit word sent to the byte swap operator becomes available with the left half (8-bit byte) and right half (8-bit byte) interchanged. For example, if the following number were sent to BSW:

Two 8-bit Bytes (Binary)	*Octal Value*
00010100 11100101	012345

the result *from* BSW would be:

Swapped Result	*Octal Value*
11100101 00010100	162424

When byte pack (BPK) is specified, the contents of bits 0-7 of BPK are shifted into bits 8-15, and bits 0-7 of the source register are accepted into the low-order eight bits of BPK. Each pair of transfers into BPK packs a pair of bytes from left to right.

The advantage of using hardware byte swap/byte pack lies in the fact that no software subroutines are necessary. Thus, the data operations are faster and more efficient.

2.4.2 **Byte Comparator** – The Byte Comparator Option is in the form of a plug-in firmware card. This option provides the GRI-99 user with the ability to compare two 8-bit bytes, and if the comparison yields a certain result, proceed to an appropriate subroutine or action.

2.4.3 **Binary Input Multiplexer (BIM)** – The Binary Input Multiplexer Option assembles discrete digital inputs into 16-bit words for entry onto the Destination Bus of the processor and subsequent processing by the computer. The primary application for BIM is assembling digital signals from switches or other on/off devices that indicate process states, valve positions, or alarm states. The BIM can be installed in any location on the I/O bus at the rear of the GRI-99 chassis. For a complete discussion of BIM theory of operation, programming, and interfacing, refer to the GRI *BIM/BOM Manual.*

2.4.4 **Binary Output Multiplexer (BOM)** – The BOM is an interface that connects the computer to various output devices. The BOM is used to control the state of on/off devices (e.g., solenoid valves, electric motor starters, lamps, and latching relays). Two 16-bit words are output from the BOM to represent the desired state of the on/off device to be controlled. The BOM is installed in any location in the I/O bus at the rear of the GRI-99 chassis. For a complete discussion of BOM theory of operation, programming, and interfacing, refer to the GRI *BIM/BOM Manual.*

2.4.5 **Gate Input Card** – The Gate Input Card is used to interface most parallel data producing devices to the GRI-99. Function control and testing is provided by the Gate Input Card for such devices, and a complete interrupt system interface is also included on the card. External status inputs are in the form of positive or negative signals. This operator functions as a source of data only. As a source, the Gate Input Card can participate in any instruction that references it as a source device address. An application for the Gate Input Card is in interfacing a digital voltmeter to the GRI-99. An appropriate program for this application can be easily written. In this case, a binary coded decimal (BCD) value is then converted to a binary integer and appropriately scaled. For a complete description of the Gate Input Card, refer to the GRI *Gate Input Card Manual.*

2.4.6 **Pulse Input Detector (PID)** – The Pulse Input Detector provides the capability of detecting the occurrence of pulse events external to the computer via the interrupt system of the GRI-99. Inputs to the PID from external devices are signals that indicate that certain events have occurred. The signals are generally TTL or DTL positive logic signals. The output of the PID specifies a predetermined memory address that is loaded into the GRI-99 Memory Address Register (MA) during the interrupt cycle. The address is also used as a jump address during the interrupt cycle! There are eight predetermined addresses, only one of which is interrupted at a time. The PID is installed in any location on the I/O bus at the rear of the computer chassis. One application for the PID lies in counting external events and incrementing a memory location. If an overflow occurs, a subroutine is entered to accomplish an appropriate action. For a complete description of the PID, refer to the GRI *Pulse Input Detector Manual.*

2.4.7 **General Output Register** – The General Output Register is in the form of a plug-in card that is one of a series of general operators used with the GRI-99 to interface most parallel data output devices. The General Output Register provides control and testing operations for such devices. In addition, a complete interrupt system interface is also contained on the card. The General Output Register accepts external status signals from a device in the form of negative or positive inputs; normally, it is set for negative inputs. This sytem device can function as both a source and destination of data and can participate in any instruction that references it as a source, destination, or both. However, the General Output Register is limited in that data cannot be shifted left or right. For a complete description of the General Output Register, refer to the GRI *General Output Register Manual.*

2.4.8 **Watchdog/Interval Timer** – The combination Watchdog/Interval Timer serves two purposes:

a. The interval timer is used to calibrate the time period between control actions in a process or to measure an elapsed time interval.

b. The Watchdog timer is used to monitor a control program, thereby ensuring that the program is operating properly. If a malfunction occurs (indicated by an overflow), contacts are closed to sound an audible or visual alarm.

Inputs to the Watchdog/Interval timer take the form of timing signals, (available from a variety of sources) and control signals from the computer, as well as power. The output from the interval time is an interrupt that diverts the main program to a subroutine designed for the interval timer. The output from the Watchdog timer is a signal that drives an alarm device and closes a set of contacts. For a complete discussion of the Watchdog/ Interval Timer, refer to the GRI *Interval Timer Manual.*

2.4.9 **Selector Channel** – The selector channel is a digital interface processor option consisting of a set of three P.C. boards that provides auxiliary registers for word count, core address and data buffer for block transfer, word oriented devices. Typical high-speed devices that can be interfaced to the GRI-99 are disk

memories, drum memories, and various A/D devices. The selector channel, as part of the DMA system, enables data words to be transferred into main memory and out of main memory at a rate of one word every 1.76 μs. Access is granted by the processor under program control whenever a request appears on the direct memory request bus of the processor and the current instruction is complete.

2.4.10 Model 30 Processor Options

The Model 30 processor options are the same as the Model 40 options. In addition, the following items are available for the Model 30:

a. Extended Math Board.

b. Six general-purpose registers.

2.4.10.1 Extended Math Board – The Extended Math Board (EMB) may be used on any GRI-99 Models if the six general purpose registers are also installed. The EMB option is contained on a standard, large-size firmware card that can be plugged directly into either of two option slots. The EMB is an inexpensive device that is used to perform complex arithmetic operations (multiply, divide, arithmetic right shift, normalize) when speed and precision are the foremost determining factors. Extended math package is supplied with this option.

2.4.10.2 General-Purpose Registers – This option consists of Six General-Purpose Registers on one printed circuit board. These registers can be used for argument passing or temporary storage of data. Data can be passed through these registers and operations can be performed in the same way as any other processor registers. These general-purpose registers are available for register-to-register transfers or memory-to-register instructions, as well as data test instructions. Extended math package is supplied with this option.

2.5 PERIPHERAL DEVICES

The GRI-99 can be easily interfaced to numerous peripheral devices. The following peripheral equipment is a representative sampling:

a. Teletype Model 33 ASR, Model 35 KSR, and other Teletype models.

b. High-speed synchronous line printers.

c. High-speed and low-speed CRT terminals.

d. Magnetic tape storage systems such as Grisette II.

e. Mass storage disk systems.

f. Card readers.

g. Analog/Digital devices, plotters.

2.5.1 Grisette II – Cassette Tape I/O System

The Grisette II cassette system is designed to be a lower cost replacement for paper tape which exceeds the operating characteristics of high-speed paper tape equipment with greater user convenience. The Grisette II is available in a number of versions namely: Read only; Read/Write, full duplex; Read/Write, Single unit. The units are offered for use as table top equipment or as rack mounted equipment.

The Grisette II is a byte oriented system incorporating standard Philips Type II Cassettes employing certified computer tape (1600 bpi) and a tape speed of 1 - 7/8 ins. per sec. The tape handling units are standard audio recorders with only the electronics modified. Cassettes are available in 50 ft. (83,000 characters per side) and 300 ft. (500,000 characters per side) lengths.

Utilized in conjunction with GRI computers the Grisette II can perform a variety of input/output operations as loading programs or diagnostics. In data acquisition systems a write-only unit can be used to store information quickly and conveniently. Full duplex read/write systems provide a convenient means of assembling and editing programs during software development. A typical configuration combines a GRI-99 Model 30 with an 8K memory and a Grisette II full duplex system which permits source programs to be conveniently loaded into the computer, edited, assembled and final object code stored on a cassette.

Software available for Grisette II users includes cassette-to-cassette tape copy/merge, cassette input/output driver display unit or other similar devices and cassette utility routines. GRI-99 computer software available in cassette form includes a relocatable cassette loader, relocatable library loader, source tape editing system and relocatable assembler. Related computer options include an automatic loader which contains a read-only memory that automatically loads programs from cassette tape into the computer and a console light indicator light to show lowa errors.

2.5.2 Grisette Autoloader

The Grisette Autoloader option is a combination hardware and software system that enables the user to create automatic "load and go" systems on magnetic tape cassettes. On the GRI-99, the autoloader option is activated by the AUTOLOAD switch; a load error is shown by the LE indicator.

The Grisette Autoloader option comprises:

a. Input/output PC card that plugs into any I/O bus slot.
b. A program to create autoload format tapes.

2.6 BASIC SOFTWARE PACKAGE

With the first GRI-99 Computer that is ordered, GRI ships a complete Basic Software Package. Additional software packages may be purchased from GRI. The basic software package comprises:

a. Relocatable assembler.
b. Source Text Editor Package.
c. Utility Package.
d. Diagnostic Packages.
e. Single-Precision Math Package.
f. Double-Precision Math Package.
g. Floating Point Math Package.
h. System Reference Manual.
i. Real-Time Executive Package*.

NOTE

It is important to note that all GRI software is fully compatible with new GRI computers.

2.6.1 Assembler Package

The Assembler Package consists of the following items:

a. The GRI-99 *Assembler Manual.*
b. Line printer conversion program.
c. Assembler program.

The relocatable assembler is required to prepare binary object programs for the GRI-99 Computer. The assembler allows the user to write programs in a terse, comprehensible symbolic language (RAS). The symbolic form of a program, called the source program, is a meaningful sequence of assembly language statements.

*Manual is sent with package tapes by request only.

2.6.2 Source Text Editor Package

The Source Text Editor Package includes the following items:

a. The GRI-99 *Source Text Editor Manual.*
b. Source text editor program.

The source text editor provides a convenient method for generating source text paper tape for input to an assembler; correcting and updating source text tapes from the Teletype; and listing source text tapes on the Teletype. By providing these benefits, the source text editor eliminates many of the problems associated with paper tape as an input medium.

2.6.3 Utility Package

The Utility Package includes the following items:

a. Program loaders for the GRI-99.
b. Debugging aids used in analyzing system programs.
c. A memory dump tape in absolute format.
d. A memory dump tape in bootstrap format.
e. A routine to copy paper tapes.
f. Translator (FAST-to-RELOCATABLE).

For a complete description of program loaders, refer to the GRI-99 *Loaders Manual.*

2.6.4 Diagnostic Package

The Diagnostic Package includes diagnostic programs to test the following GRI-99 elements, as determined by the model that is used:

a. Basic processor.
b. Power fail.
c. Arithmetic operator and indexing.
d. Memory.
e. General-purpose registers (Model 40 only).
f. Extended math operator (Model 40 only).

The diagnostics are an invaluable aid in system troubleshooting. Using the diagnostics and appropriate fault isolation techniques, the system can be tested and corrective measures can be implemented.

2.6.5 Single-Precision Fixed Point Package

The Single-Precision Fixed Point Package includes the following items:

a. Manual.
b. Relocatable object tapes.
c. Specifications and a listing for the functions that can be performed (for example, multiply, divide, arithmetic/logical shift). For a complete description of functions and their usage refer to the GRI-99 *Fixed Point Manual.*

The *Fixed Point Manual* contains very useful reference information regarding number systems, logical operations, and single-precision and double-precision operations, as well as scaling and data conversion routines. The user is directed to this manual, which serves as a valuable tool in understanding fixed point mathematics packages.

2.6.6 Double-Precision Fixed Point Package

The Double-Precision Fixed Point Package includes the following items:

a. Manual.
b. Relocatable object tapes.
c. Specifications and listing for the functions that can be performed.

Again, the user is referred to the *Fixed Point Manual* for complete description of the operations, number systems, and capabilities of software using double-precision arithmetic.

2.6.7 Floating Point Math Package

The Floating Point Math Package includes the following items:

a. Manual.
b. Relocatable object tapes.
c. Relocatable debugging object tapes.

Operations that can be performed with floating point are I/O conversion routines, all arithmetic operations, square root, sine, cosine, exponential, logarithm, and arctangent functions, etc. Refer to the *Floating Point Manual* for a complete description.

The GRI floating point package uses double-precision arithmetic and an exponential concept to greatly extend the range of numerical value over that which is available using fixed point arithmetic.

2.6.8 Real-Time Executive Package

The Real-Time Executive Package includes the following items (furnished by request only):

a. Real-time executive program.
b. Real-time executive manual.
c. Real-time executive listing.

The real-time executive is a supervisory program designed for real-time applications, It monitors shared subroutines, provides for dynamic memory allocation, handles interrupts, program stacking, foreground and background operations, queuing and dequeuing of buffers.

2.6.9 Additional Diagnostics and Manuals

Diagnostics and manuals are available for the options described in Chapter 2 (e.g., real-time clock, BIM/BOM, pulse input detector, interval timer, etc.). For information concerning software for the options or peripherals, write to GRI.

CHAPTER 3
GRI-99
ARCHITECTURE

Chapter 3 describes the architecture of the GRI-99 Computer. The level of treatment is purposely simplified to acquaint the user with basic GRI-99 elements and associated functions. The basic computer elements and the power supply are described individually, as well as in relation to the total system. For an in-depth treatment of GRI-99 theory of operation, refer to the *GRI-99 Maintenance Manual.*

3.1 PHYSICAL ARCHITECTURE

The physical organization of the basic GRI-99 is shown in Figure 3-1. The framework is of stiffened sheet metal, perforated for ventilation, and serves as a grounding cage. The guides for the printed circuit cards are plastic. The console, including all console switches and indicators with associated driving and sensing circuits, is built into the hinged door on the front of the cabinet.

All back panel wiring inside the frame is contained on printed circuit (PC) cards. Card sockets are soldered onto each PC card; as a result, one card is plugged directly to another card. *No wire wrapping or point-to-point wiring is used anywhere in the system.* The processor PC card (bus) has connectors for seven 9-in. x 13-in. plug-in cards, two of which are available for large firmware devices, such as the Extended Math Board. The I/O bus has 9 positions for 9-in. x 4-in. cards, which are the smaller firmware devices. The memory bus is used for core memory modules.

3.2 BUS SYSTEM

To achieve functional modularity, the machine architecture is designed around a dual bus arrangement (see Figure 3-2). All functional operators in the system are interfaced to these buses, which provide communication and control paths from one device to another. As shown in Figure 3-2, when a 16-bit data word is to be transmitted from Device *X* to Device *Y*, Device *X* places the data onto the lower bus (Destination Bus). These data are then passed to the upper bus (Source Bus), through a short logic path between the buses, and then sent to Device *Y*. A 16-bit data transmission as just described is the most basic operation performed in the machine and is referred to as a *microinstruction.* Data are transmitted by the stored program and also by the built-in machine control to perform various bookkeeping tasks. Each of these transfers requires 440 ns.

The Source Bus and the Destination Bus each contain 44 conductors that can be grouped into three major categories:

a. 16 lines are used for the data paths.
b. 6 lines for selection of the device.
c. 22 lines provide control and power.

The format of the 16-bit data transmission instruction word is also shown in Figure 3-2. The left-most six bits of this instruction word select a Source Device Address (SDA). These six bits are impressed upon the six address lines of the Destination Bus to select one device as a data source. The right-most six bits of the instruction word appear on the six address lines of the Source Bus to select the Destination Device Address (DDA).

3.2.1 Busing Scheme

The busing scheme and connections internal to the main chassis are shown in Figure 3-3. All PC cards shown in the lower row are of the larger size (9-in. x 13-in.). The processor is contained on four large cards, labeled PC1, PC2, PC3, and AO. The processor Source and Destination Buses extend to the console on ribbon cable. The processor Source and Destination buses are connected to the I/O Source and Destination buses (as well as memory) also by ribbon cable. Core memory cards are the largest cards used and contain the core planes, and all read, write, inhibit and sense circuitry also timing, MA, MB, and address decoding. Two connectors are available in the processor bus for the addition of firmware operators.

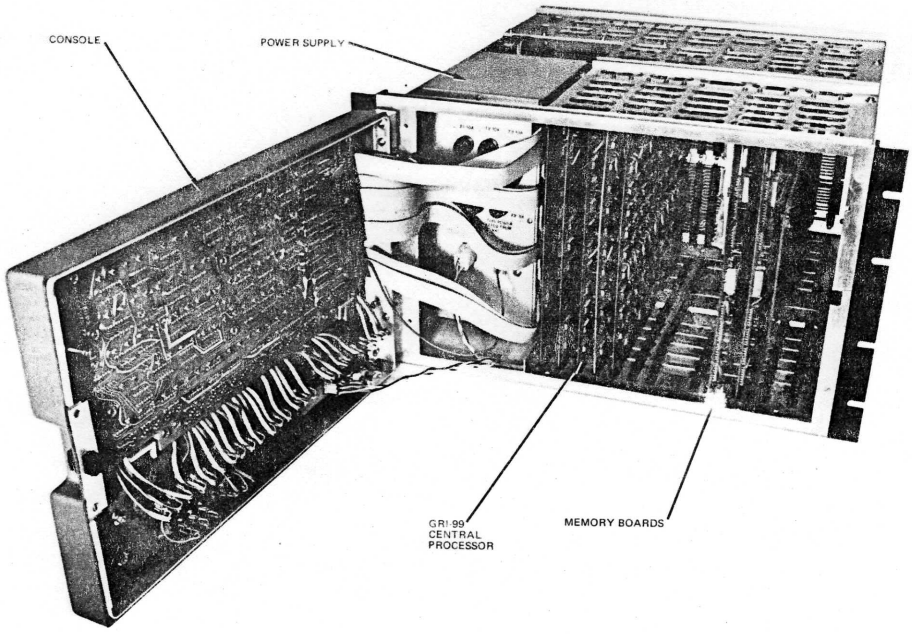


Figure 3-1 Physical Organization of the Basic GRI-99 Package

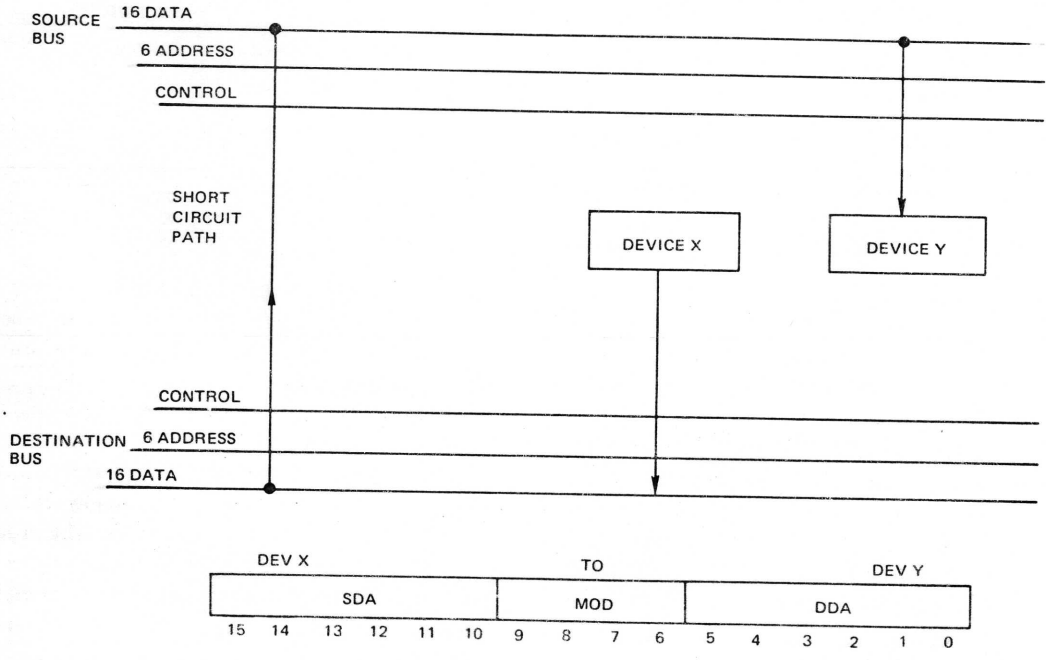


Figure 3-2 Basic Machine Microinstruction

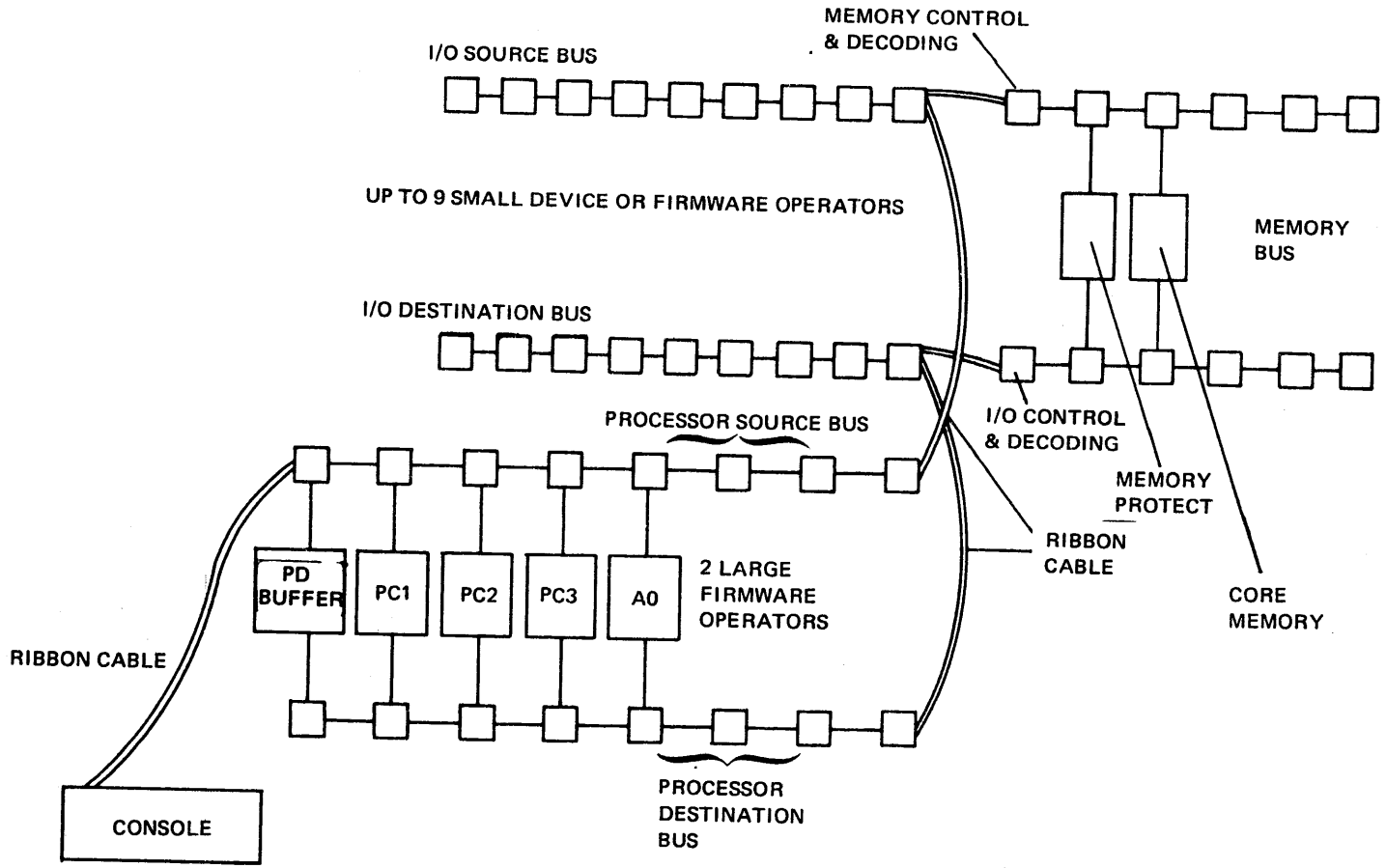


Figure 3-3 System Busing

Note the memory bus buffers and decoding circuitry that are present between the memory, the processor and I/O buses. The memory write protect feature is optionally contained on the memory bus.

Both the processor bus and the I/O bus are actually each two buses: Source Bus and Destination Bus. In general, the Destination Bus is associated with *output* from system devices, whereas the Source Bus is the source of data and control information for *input* to system devices.

All connections between the buses and the console are made with ribbon cable that runs between the processor bus PC and the console PC. The expansion chassis to extend the I/O bus is of the same type of construction as the mainframe but is half as deep. This chassis is placed either above or below. Ribbon cable connects the mainframe buses to the expansion bus where the bus signals are buffered.

3.2.2 Power and Signal Distribution

Figure 3-4 shows a simplified version of power and signal distribution for the GRI-99. As noted in the illustration, the heavy line indicates system ground. The grounding for the I/O extensions is shown. Note the presence of the +5V voltages and the switching connection for the POWER, CONSOLE LOCK, and AUTOLOAD switches.

3.2.3 Bus Schematic

Figure 3-5 is a bus schematic of a GRI-99 configuration. The illustration shows the origin and routing of signals and signal groups. The termination points of wired OR gates are also shown. Note the routing of the interrupt and DMA signals from PC3 into the firmware options. Interrupt priority signals go to the memory bus and, subsequently, to the I/O bus and the extension bus.

Some of the signals and signal groups shown in the schematic are private communications signals between the three major control assemblies (PC1A, PC2A and PC3A) that make up the basic processor. The power supply delivers voltages and power status and control signals to the bus via two connectors that go to the main processor bus. From PC3 through the last slot on the main processor bus, the signals become the system signals that are distributed in parallel through the entire system, including the I/O bus section in the rear of the machine. Note that the majority of signals that arrive at the back panel bus of the processor are still identical to the processor signals used internally and, therefore, facilitate the use of internal options plugged into these slots, as well as I/O options.

3.3 PROCESSOR ELEMENT DESCRIPTION

3.3.1 Bus Modifier

The four center bits of the instruction word called *MOD bits*, are used to specify data modification as data are passed from the Destination Bus to the Source Bus (see Figure 3-6). For a complete description of the effects of the MOD bits, refer to Chapter 4. This short logic path contains an arithmetic unit called the Bus Modifier. During data transmission, the data can be complemented and then optionally incremented, shifted left 1 bit, or shifted right 1 bit. An overflow (BOV) flip-flop stores a binary carry out of the sixteenth bit caused by the incrementing of a data word that became positive because of the increment or the combined effect of indexing and incrementing. Both right and left shifts are done in a circular fashion through the Link (L) flip-flop. Following a shift during data transmission, one bit of the source data is retained in the Link, and the bit initially in the Link is passed on as part of the data word to the destination device.

3.3.2 Sequence Counter (SC)

The SC is a 15-bit register that is provided to keep track of the program instructions. A Sequence Counter (or program counter) is common to all computers and indicates the address of the next instruction to be executed.

In the GRI-99, the Sequence Counter is connected across the buses, as are all other elements in the system, providing direct access from device to device. The device address of the SC is 07.

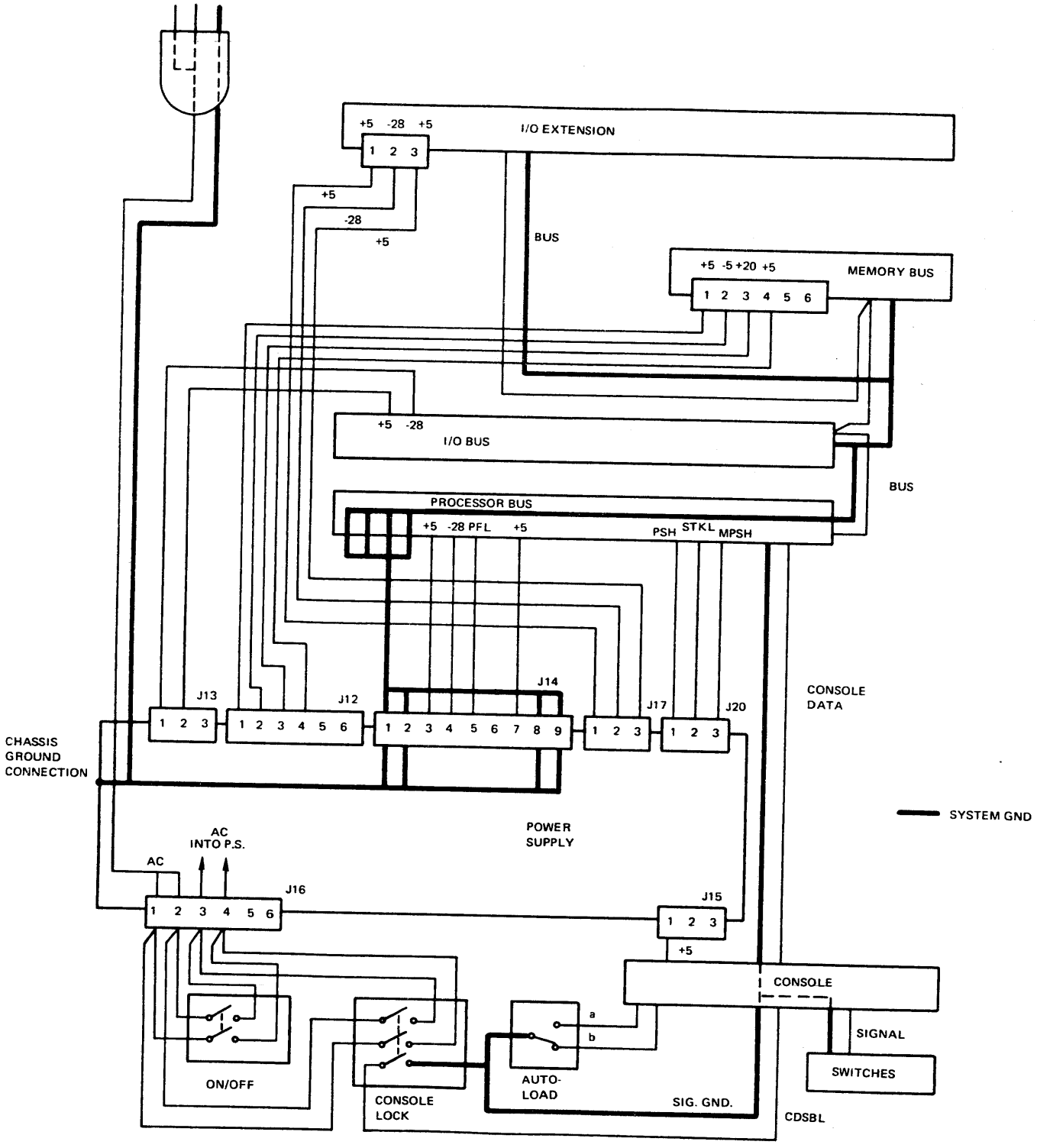
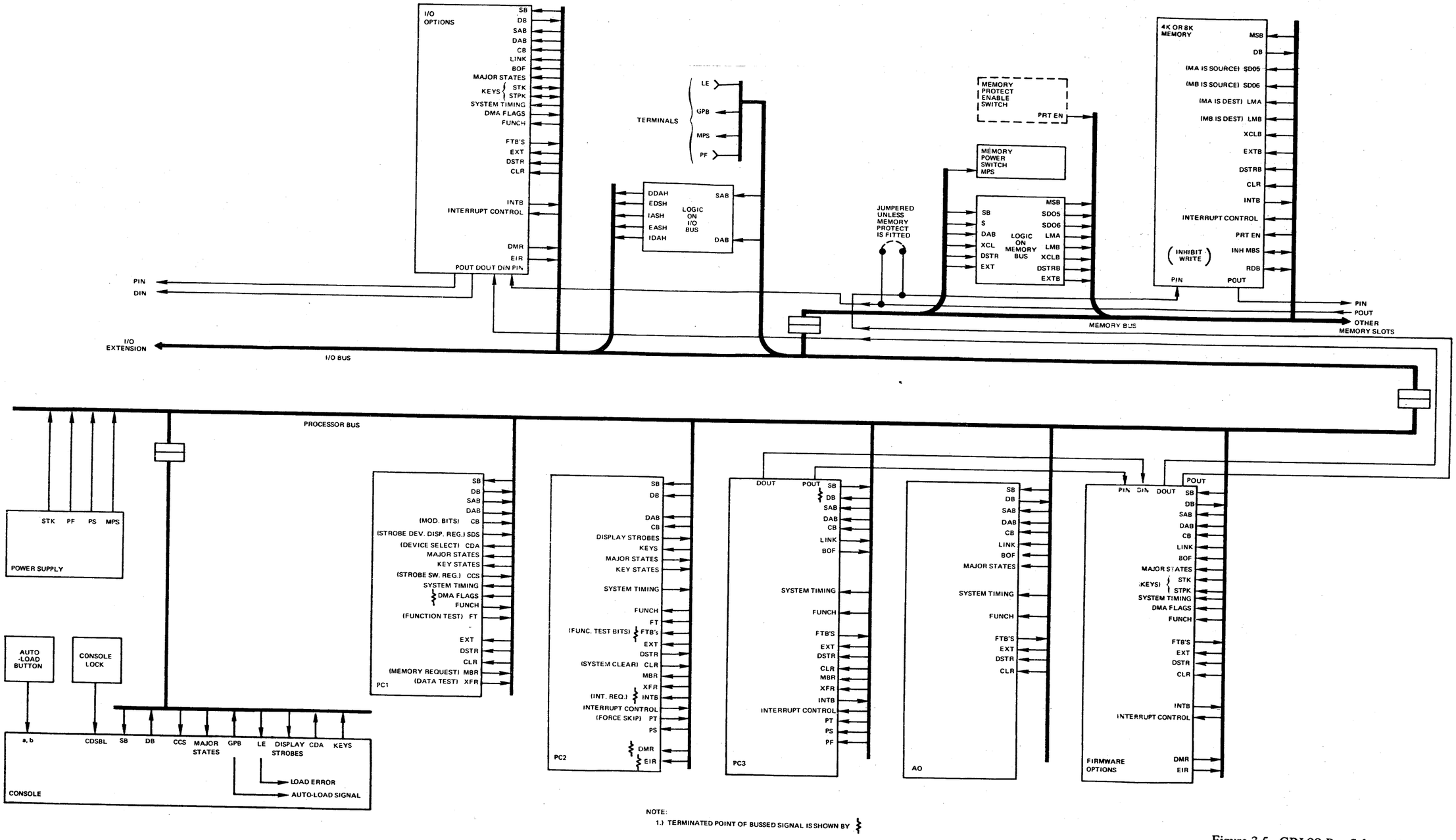


Figure 3-4 GRI-99 Power and Signal Distribution



NOTE:
1.) TERMINATED POINT OF BUSSED SIGNAL IS SHOWN BY ≷

Figure 3-5 GRI-99 Bus Schematic

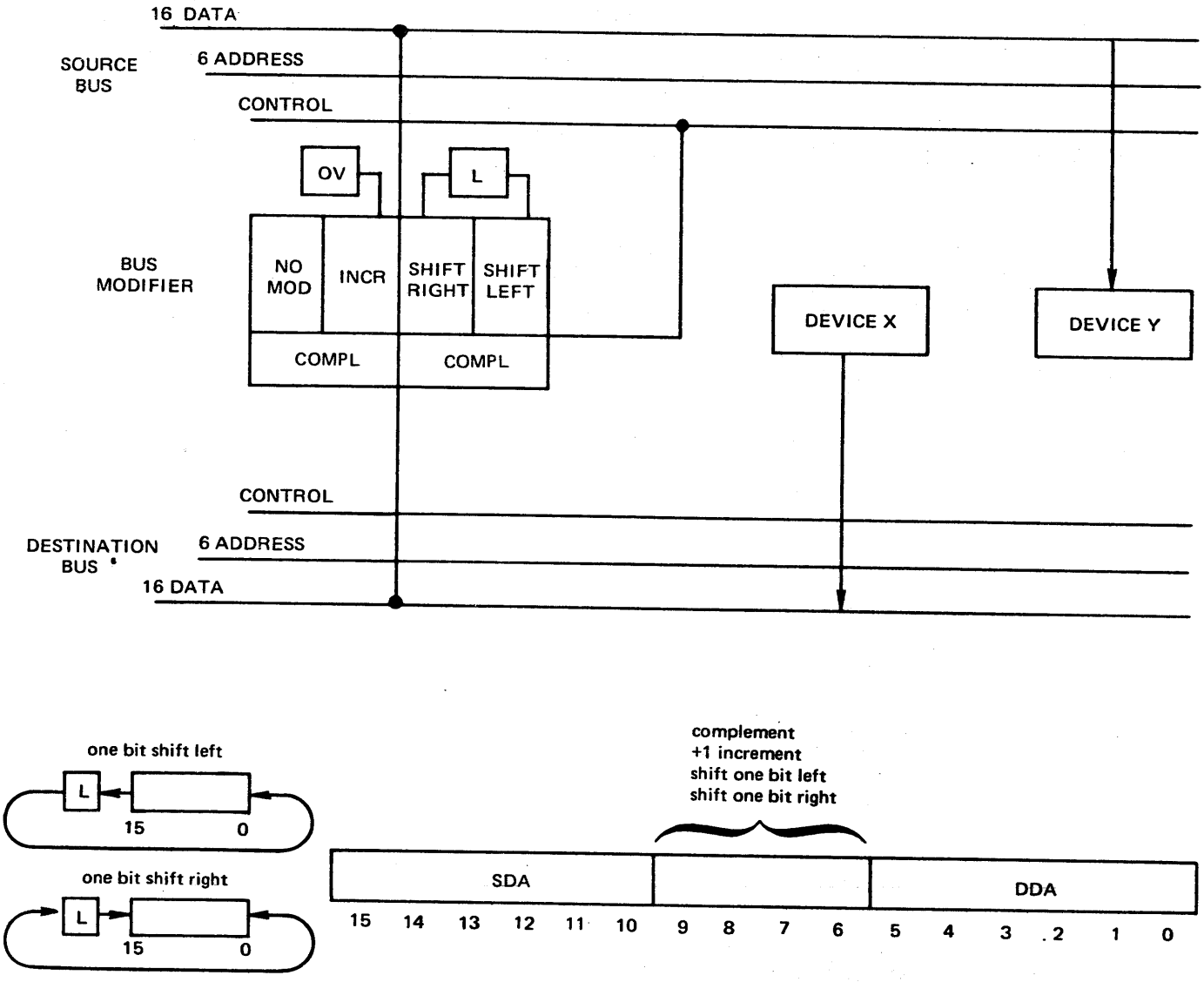


Figure 3-6 Data Modification During a Microinstruction

3.3.3 Instruction Register (IR)

The Instruction Register contains the current instruction in the computer to be executed. As the other elements in this system organization, the IR is connected across the Source and Destination Buses. The device address of the IR is 01.

3.3.4 Data Tester

A computer must decide on the paths that it will follow, based on the value of the data that it receives. In the GRI-99, the Data Tester uses the modifier code bits to determine the value of the information it receives, i.e., less than zero, equal to zero, or any combination thereof. This tester is connected between the Source and Destination Buses and is programmed to accept data directly from any source. A positive response to a data test results in a jump instruction. The contents of the Sequence Counter are automatically stored in a Trap Register associated with the Data Tester when a jump is executed. The device address of the Data Tester is 03. The Trap Register associated with the Data Tester can be utilized as a general-purpose register and carries the device address of 23 in that case.

3.3.5 Function Tester

Peripheral devices produce status signals that indicate particular conditions to the computer. The GRI-99 contains a Function Test Operator that compares status signals to the modifier bits and acts on the result.

Three function test lines are provided for this purpose, and they may be tested for their logical assertion or negation via the Function Tester. A positive response by the Function Tester to the sense lines results in a skip instruction. The device address of the Function Tester is 02.

3.3.6 Function Generator

The Function Generator is used by the program to perform specific tasks in the destination device. The function generation process is accomplished by pulsing certain control lines selected by bits 6 through 9 of the instruction. An example of function generation is an instruction that causes the high-speed reader to read the next character on tape. The source device address of the Function Generator is 02.

3.3.7 Arithmetic Operator (AO)

The AO is that part of the basic processor that performs arithmetic and logical operations. The functions that can be performed in the functional Arithmetic Operator are ADD, AND, OR, and EXCLUSIVE OR (XOR). The AO of the GRI-99 operates somewhat differently from that of a typical computer. In the GRI-99, no instructions are issued that say "ADD". A conventional computer ADD instruction translates as, "one number is in the accumulator and the other number is in memory. Pull the number out of memory, add the number in the accumulator to the number in memory, and put the sum back into the accumulator."

In the GRI-99, the Function Generator is used to generate the ADD function. The instruction can be shown as:

Function Output ADD ⟶ AO

This element always performs the ADD function between the current value of the AX and AY registers until the user issues another command changing the state of the AO.

When either of those registers is changed, a new sum appears, immediately available for transfer to any point in the system. New values can be presented from a system register, and a new result can be obtained in a single cycle time, 1.76 μs. The result, contained in a separate accumulator register (AO), always reflects the instantaneous output generated by the contents of the AX and AY registers as controlled by the function selected. It can be stored in memory by a single instruction. The introduction of new values to one register does not alter the contents of the other register.

3.3.8 Machine Status Register (MSR)

Certain flag and control flip-flops in the computer are connected to the Source and Destination Buses in such a way that their states can be saved and then restored as though they were bits in a full-word register. The register is referred to as the MSR, which can be addressed as device address 17.

3.3.9 Interrupt Status Register

Priorities of interrupts are determined by a combination of the program-controlled Interrupt Status Register (ISR), which is a mask register, and hardware position on the bus.

Each device with interrupt capability contains an Interrupt status flip-flop, which can be set and cleared under program control. When this flip-flop is set, the device can interrupt, if desired. However, if the Interrupt Status flip-flop is clear, the device cannot interrupt under any circumstances. These Interrupt Status flip-flops, although distributed throughout the system, are collectively called the Interrupt Status Register. The device address is 04.

3.3.10 Memory Address (MA)

The MA is a *15-bit* register that holds the address of the word being read from or written into memory. Unlike the SC, the MA is used as a pointer during the different modes of addressing. The device address of the MA is 05.

3.3.11 Memory Buffer

The MB holds the data being written into and read out of memory. Data from the MB are subsequently stored in a memory location, or transmitted to a device in the system. The device address of the MB is 06.

3.3.12 Index Register (XR)

The XR is a 16-bit register that is used to index memory addresses. The XR can also be used as a general-purpose register. When indexing is specified, the XR contains a value that is added to an address, thereby generating an effective address which is then used to address memory. The device address of the XR is 22.

In the GRI-99, any value that is used as an address can be indexed. For memory reference instructions, Jump instructions, or any value sent to the SC, if bit 15 of the address word is set, the value will be indexed.

3.4 PROCESSOR TIMING

The basic clock frequency in the GRI-99 is provided by a 9.09 MHz crystal clock (Y1). This clock produces a signal (XCLL) that is distributed throughout the system on source bus pin L. The origin of the clock is on PC2. The basic clock period is 110 ns. This clock frequency is divided by four to produce the 440 ns micro-cycle timing (T0, T1, T2, T3). The timing chart demonstrating this relationship is shown in Figure 3-9. The four microcycles within the basic memory cycle are shown with a 110 ns strobe occurring during the last 110 ns of each microcycle, except T0 and T1, in which it occurs 110 ns earlier. All clocking of data transfers is done with these strobe pulses. The balance of the time period between the trailing edge of one strobe and the leading edge of the next is used for propagation and settling time through gates.

3.5 MEMORY

The GRI-99 memory is a random-access, ferrite core memory. Standard memory size is 4K (4096) words of storage. Storage capacity can be increased to a maximum of 32K (32,768) words. In terms of addressing, the entire memory is a set of contiguous locations that range from zero to a maximum number that is dependent on the capacity of the particular system. The highest location for a 4K machine is 07777_8. Data can be read from memory or written into memory in 16-bit words by using the appropriate console switches.

3.6 REGISTER INTERACTION

Chapter 4 discusses the instructions used by programmers to perform a task within a program. Those instructions are referred to as *macroinstructions.* In the GRI-99, a group of *microinstructions* is used to implement

a macroinstruction. The microinstructions are in the same format as macroinstructions; the only difference between the two instructions is the source of the macroinstruction and the speed at which it is executed.

This section describes the GRI-99 major states and the relation of time states (T0 - T3) to the various operations performed. An example of the execution of single cycle instructions is provided to show the interaction of the various registers. For a complete description of instruction execution for all instructions, refer to Chapter 6.

The GRI-99 macroinstruction set includes the instructions that require 1, 2, 3 or 4 major states to complete. In addition, the interrupt, direct memory access, and external instruction states are three more major states. When power is applied to the processor, it is in the FI state. During each state, one memory cycle is executed. Each state is given a two-letter designator, as follows:

FI	Instruction cycle 1
FA	Instruction cycle 2
FO	Instruction cycle 3
FD	Instruction cycle 4
BK	Break cycle
DM	Direct memory access cycle
EI	External instruction cycle

Figure 3-7 is a state flow diagram showing priorities and all possible paths between machine cycles. The End State, Halt State and Console Stop State are not separate machine states but merely conditions that exist within the processor, while the processor is in one of the above states. Note that these states represent stop states for the machine, and no new memory cycle is initiated.

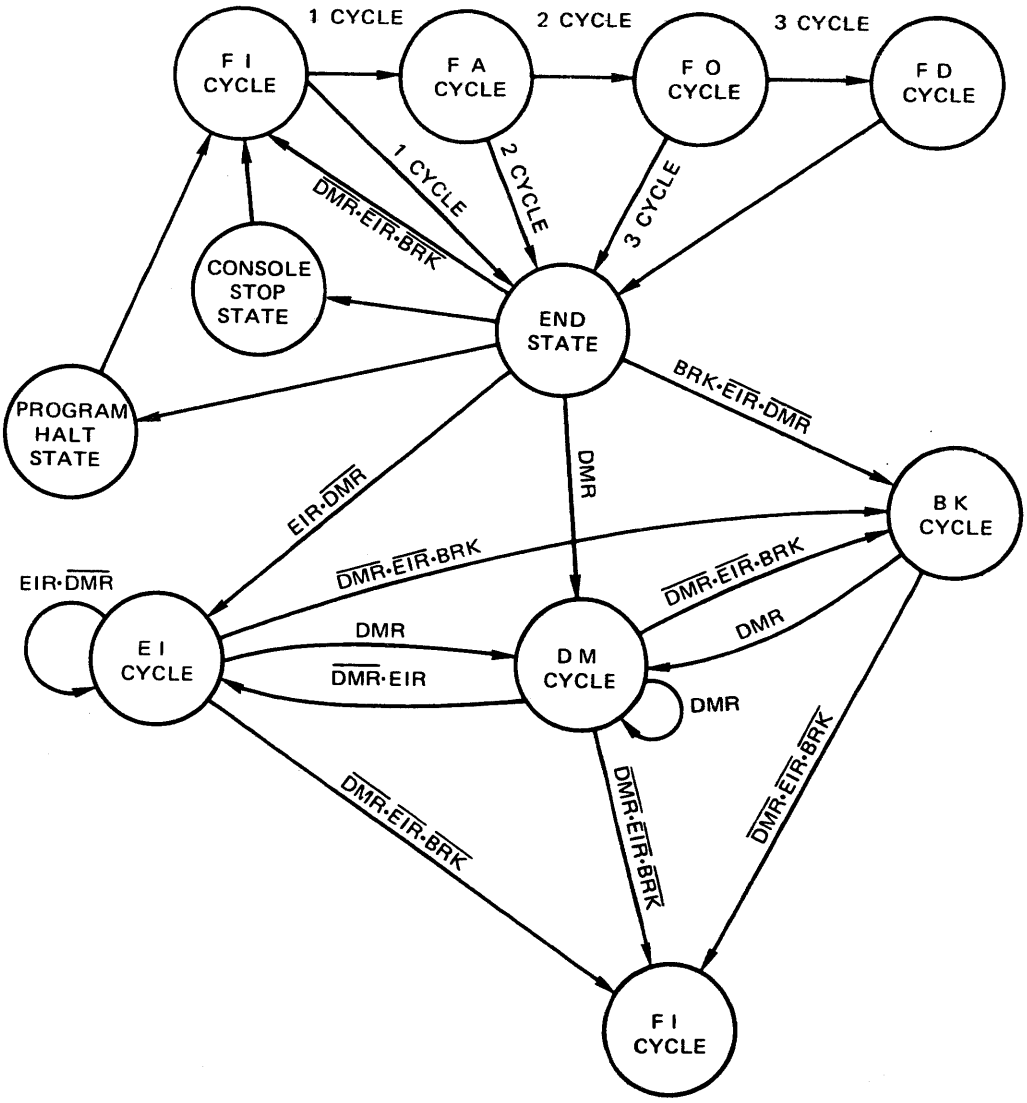
During each memory cycle up to four microinstructions can be performed using either the Instruction Register (IR) or the processor's ROM as a microinstruction source. The technique of transferring 16 bits of data from a source to a destination is discussed in detail in Chapter 4.

For example, consider a single-cycle instruction. At the start of the memory cycle (T0), the instruction word is in memory and the control logic has no indication of what the macroinstruction will be. A data transfer from the SC to the memory address (MA) register is made to read the macroinstruction from memory. During the next period (T1), the instruction word read from memory into the MB is transferred through the bus modifier unaltered to the instruction register (IR). At the end of T1, the control logic receives the macroinstruction the programmer wishes to perform (e.g., a data transfer from SDA to DDA). During T2, the transfer is performed. To prepare for the next instruction, the SC must be incremented. During T3, the SC is transferred to itself and incremented as it passes through the bus modifier.

Every macroinstruction is implemented in this manner with 1, 2, 3, or 4 memory cycles and various microinstruction sequences. These sequences are stored in a read only memory (ROM), which is part of the basic machine control logic. The arrangement shown in Figure 3-8 is used to derive microinstructions from either the IR or the ROM. In the sequence previously discussed, during T0, T1, and T3, microinstructions were taken from the ROM. During T2, the IR supplied the instruction to be executed.

NOTE

When the programmer is manipulating data within the control registers of the machine (such as the trap or SC), it is quite important to know the event sequence of microinstructions relative to macroinstruction execution. Complete event sequences for all instructions classes are presented in the *GRI-99 Maintenance Manual* and in Chapter 6 of this manual.



PRIORITIES

1. DM DIRECT MEMORY ACCESS
2. EI EXTERNAL INSTRUCTION
3. BK INTERRUPT (BREAK)
4. MACHINE CYCLES (FI, FA, FO, FD)

NOTE:

FI CYCLE IS DRAWN TWICE FOR SIMPLICITY.

Figure 3-7 State Flow Diagram

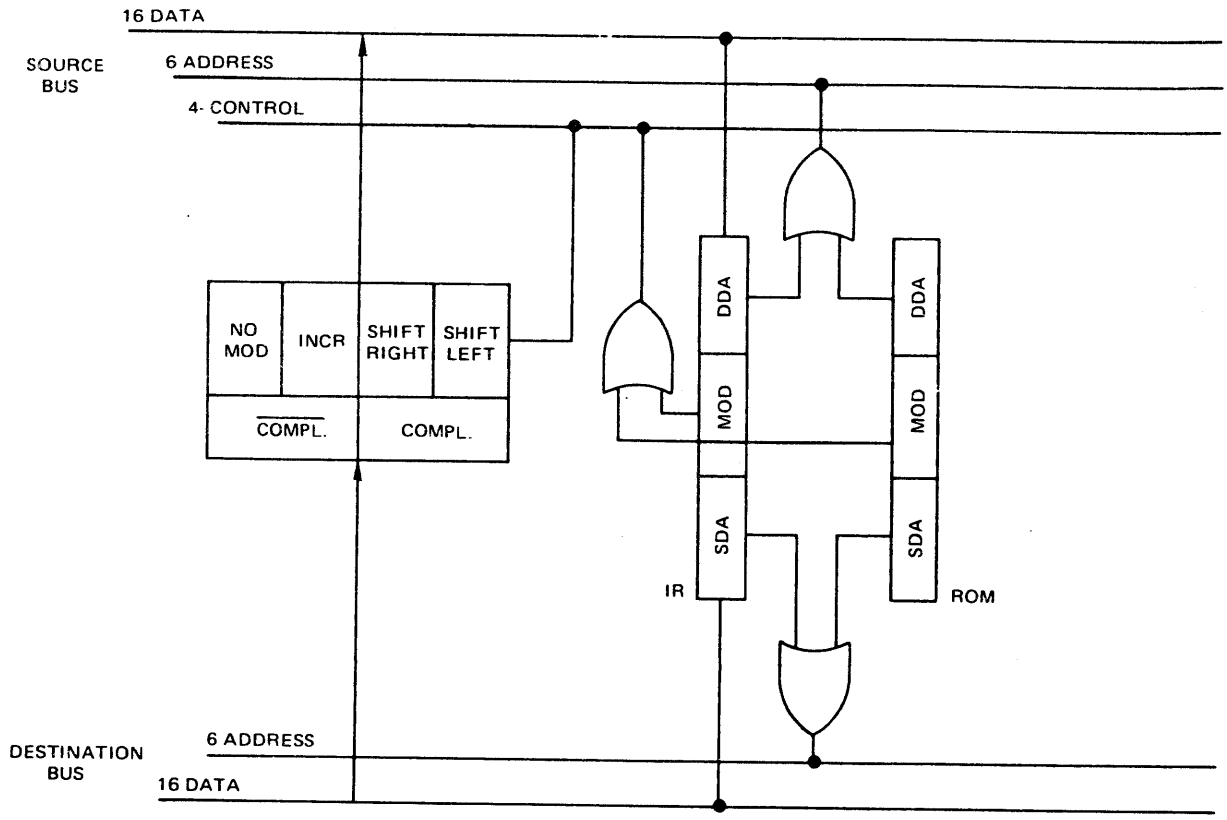


Figure 3-8 Derivation of Addresses and Control Signals from Either the IR or ROM

Figure 3-9 describes all single-cycle instructions. The hexagonal boxes indicate a transfer of informations to or from memory. Tables 3-1 and 3-2 are provided to supplement the figures with exact sequence information.

3.7 POWER SUPPLY

The GRI-99 power supply furnishes the multiple power requirements for the computer. The power supply protects the computer from both hardware and software damage during power up, power down, and power fault conditions (such as ac line failure). Internal circuitry provides ac line sensing and protection and signaling of power status, through the power supply sense and control electronics. When ac power goes down, the power supply sense and control board initiates an interrupt and an appropriate memory shutdown sequence. On power up, if the autorestart option is installed, the SC is set to location 6 and execution starts automatically.

3.7.1 Input Specification

The input specifications for the GRI-99 power supply are as follows:

Voltage (1 phase):	99-132 or 196-243 Vac
Voltage Surge (3 cycles):	360V
Voltage Spike:	60V - ms, 600V max.
Current at Full Load:	5A max.
Frequency:	50 ± 3% Hz, 60 ± 3% Hz
Storage at full load (Without power down):	20 ms

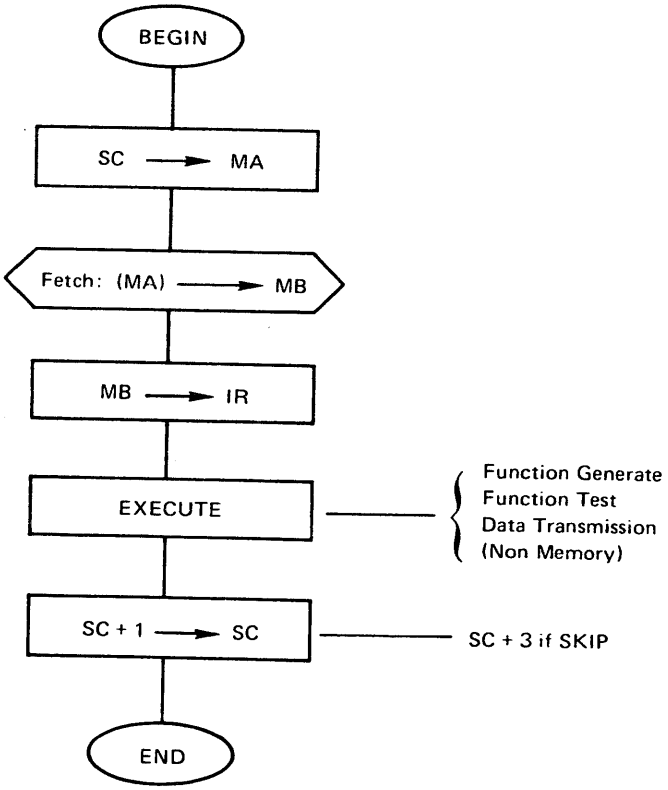


Figure 3-9 Event Sequence for Single-Cycle Instructions

Table 3-1

FUNCTION GENERATE (NON-MEMORY DATA TRANSMISSION)

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begins memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Macroinstruction now executed as a microinstruction.
	T3	SC + 1 to SC	The SC now points to next instruction in memory.

Table 3-2

FUNCTION TEST (SKIP)

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begins memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Macroinstruction executed as micro.
	T3	SC + 1 to SC	Test condition not true.
		SC + 3 to SC	Test condition true.

3.7.2 Output Specifications

The Output voltages provided by the power supply are as follows:

a. +5 Vdc (+6% - 4%), 20A max., 6.5A min.
b. -5 Vdc (+6% - 4%), 0.8A max., 0.1A min.
c. + 21 Vdc (+6%), 4A max., 0.5A min.
d. −28 Vdc (+ 4V), 0.5A max., 0A min.

The +5Vdc is used to supply power to the GRI-99 logic. The 20 Vdc is memory power. The -28 Vdc is provided for peripheral devices.

3.7.3 AC Line Failure Protection

The power supply sense and control electronics issue the necessary signals to protect memory data in the event of ac failure in the basic machine. The signal PFL goes low when there is insufficient drive in the power supply to maintain the dc voltages at their actual load, and generates an interrupt to location 0. About 3/4 ms later, PSH goes high and turns off the "clear" state after the current instruction ceases. 0 μs later MPSH goes high and turns off the 20V supply at the memory. Thus an orderly shutdown is assured.

3.7.4 Overcurrent Protection

The GRI-99 power supply is electronically protected against continuous overload.

3.7.5 Overvoltage Protection

The power supply is protected against overvoltage by crowbar circuits. The various outputs are crowbarred as follows:

a. +5V: between 5.5 V and 6.25V.
b. −5V: between −5.5 V and −6.25V.
c. +20V: between 25V and 29V.
d. −28V: between −34V and −38V.

The GRI-99 power supply is set at the factory for optimum performance.

CHAPTER 4
GRI-99
OPERATION AND PROGRAMMING

Chapter 4 describes GRI-99 console controls, indicators, and turn-on and shutdown procedures. The internal processing command repertoire and I/O commands are also fully described.

4.1 CONSOLE DESCRIPTION

Figure 4-1 shows the GRI-99 console. The Operating Keys (STOP, SS, START, WR etc.) on the lower left of the console and Data Switch Register (SWR) toggle switches, in conjunction with the DEVICE SELECT switches, are used manually to operate the computer. The indicators on the upper left (FI, FA, FO, etc.) display internal computer conditions. The rows of indicators on the right display the contents of the various processor registers, Instruction Register (IR), Sequence Counter (SC), Memory Address (MA), Memory Buffer (MB), and Data Display (DISP). When any indicator is lit, the associated flip-flop is in the 1 state (set). When an indicator is out, the associated flip-flop is in the 0 state (clear).

4.1.1 Controls

4.1.1.1 POWER – The POWER switch on the lower left-hand side of the console supplies primary power to the computer. In the up position, the switch is ON; in the down position, the switch is OFF.

4.1.1.2 KEY DISABLE – The LOCK switch on the lower left-hand side of the console is an optical switch that disables the Operating Keys to prevent unauthorized tampering with the operation of the processor or turning off ac power. The key is inserted and turned clockwise to disable the Operating Keys. If the LOCK switch is turned clockwise, the SWR and DEVICE SELECT are the only switches that are operational.

4.1.1.3 AUTOLOAD – The AUTOLOAD switch is an optional pushbutton on the bottom left of the console that enables the user to load and start an entire software system automatically from a magnetic tape cassette (GRI-sette). In the ON position, the AUTOLOAD switch initializes the GRI-99 and initiates an EIR, DMA sequence which brings in and starts the bootstrap loader. The bootstrap loader, in turn, brings in the absolute loader, which is used to load and start the system program. If a checksum error occurs, the LE indicator lights and the machine halts.

4.1.1.4 Operating Keys – The Operating Keys are located at the lower left of the console. All keys (except SS and STOP) are momentary contact switches. Table 4-1 lists the keys and their associated functions.

Table 4-1

GRI-99 OPERATING KEYS AND FUNCTIONS

Key	Function
STOP	Stops at the completion of the current instruction with the IR (01) indicators displaying the instruction and SC pointing to the next instruction. This key is an alternate-action key; the operator can run a program one instruction at a time by leaving STOP on, and pressing CONT.
SS	Allows the operator to run diagnostic routines or other programs one major state at a time for maintenance purposes. This key is an alternate-action key: while the key is down, the processor stops at the end of every cycle it executes. The operator can run a program one cycle at a time by leaving SS down and pressing CONT.

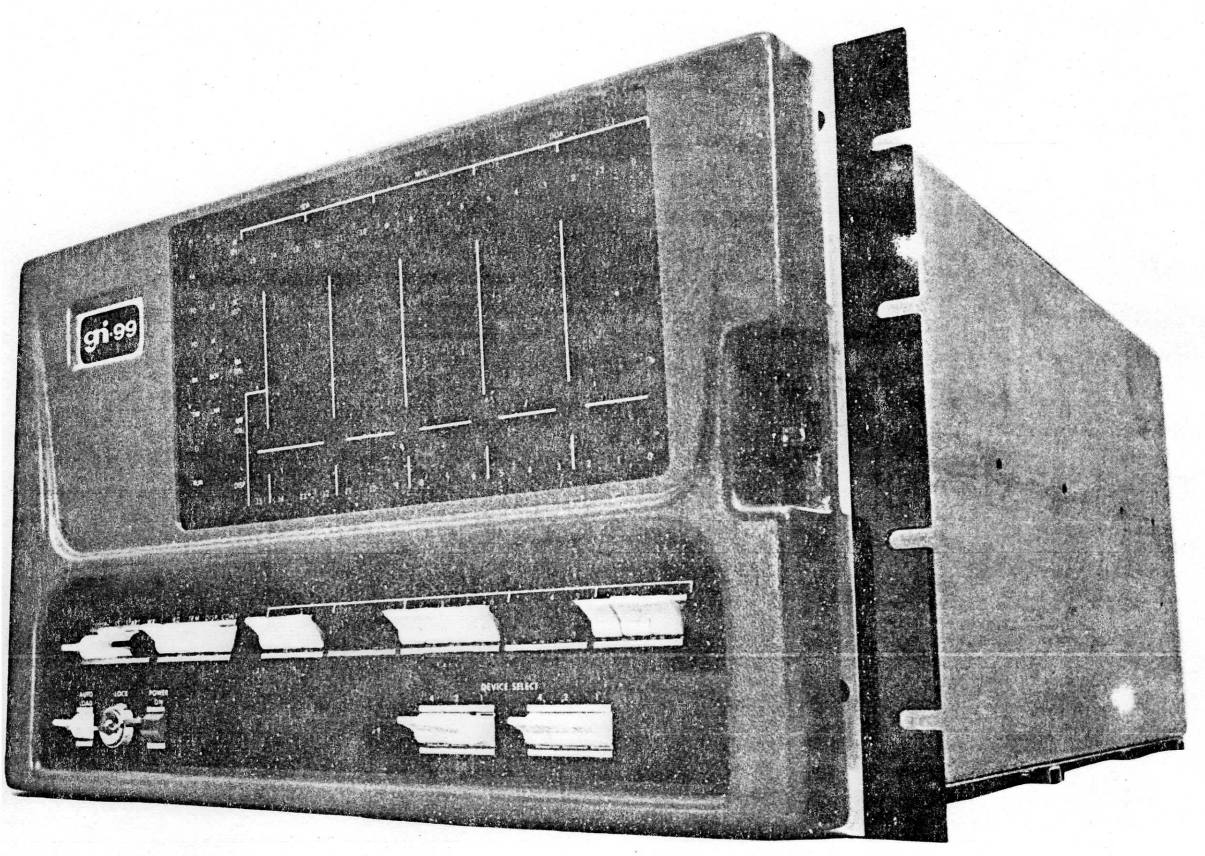


Figure 4-1 GRI-99 Model 40 Front Panel with Programmer's Console

Table 4-1 (Cont.)

Key	GRI-99 OPERATING KEYS AND FUNCTIONS Function
START	Sets all Output Ready flags, clears all other flags and control flip-flops, and begins normal operation by executing the instruction at the location specified by SC (07). Specifically, START resets the machine, i.e., Output flags are set ready, input flags are set not ready; LNK and BOV are cleared, the AO is set to the ADD state, IA is turned off, and ISR is set to all 0s.
WR	Stores the contents of the Data Switch Register (SWR) in the memory location specified by SC. Then adds 1 to SC. At completion, the MB indicators display the word stored; and MA displays the address where the word is stored.
RD	Displays the contents of the memory location addressed by SC in the MB (06) indicators. Then adds 1 to SC. MA displays the address of the word shown in MB.
TRM	Transmits the contents of the SWR to the destination register specified by the DEVICE SELECT switches. At completion, DISP shows the word transmitted.
DISP	Displays the contents of the source register addressed by the DEVICE SELECT switches in the DISPLAY indicators.
CONT	Begins normal operation in the state shown by the indicators, without sending a system clear pulse.

4.1.1.5 Data Switch Register (SWR) – The SWR is located beside the Operating Keys. The computer operator manipulates these switches to supply addresses, instructions, or data to the Destination Bus of the computer. A switch that is depressed (down) represents a binary 1; a switch that is not depressed represents a binary 0. The SWR can be used as follows:

a. As sense switches under program control by specifying source address 10.

b. To write into memory in conjunction with the WR key.

c. To transmit data to a device selected by the DEVICE SELECT switches.

4.1.1.6 DEVICE SELECT Switches – Six toggle switches (to represent two octal digits) are located at the bottom right of the console. The DEVICE SELECT switches are used to select a device address to which data can be transmitted (using the TRM key) from the SWR or the contents of the device can be displayed using the DISP key. To use the TRM or DISP key, the GRI-99 must be stopped.

NOTE

All of the operating keys except STOP and SS will not function when the machine is running (RUN light on).

4.1.2 Indicators

Most indicators on the GRI-99 console change state too frequently or too quickly to display useful, readable information when the processor is running. For this reason, the description of many of the indicators is limited to information displayed when the processor is not running.

NOTE

The top four rows of indicators at the right of the GRI-99 console are installed only on models that have the programmer's console.

4.1.2.1 Processor Register Indicators – The indicators that display the contents and status of the various processor registers are listed in Table 4-2, accompanied by a brief description of the function of each display.

4.1.2.2 **Cycle Indicators** – These indicators are located to the left of the processor registers on GRI-99 console. Table 4-3 lists the indicators and gives a brief description of the function of each indicator.

Table 4-2
PROCESSOR REGISTER INDICATORS

Display	Function
Instruction Register (IR)	Displays the instruction being executed or the last instruction completed.
Sequence Counter (SC)	Displays, in the low-order 15 bits, the address in memory of the next instruction to be executed. (Except in the case of Read (RD) and Write (WR) operating keys, where the SC points to the next location to be read from or written into.)
Memory Address (MA)	Displays the address to which the last memory access was made, also 15 bits (as SC).
Memory Buffer (MB)	Displays the last data transmitted to memory or read from memory.
Data Display (DISP)	DISP always displays the last value sent to the device selected by the DEVICE SELECT switches if the DEVICE SELECT switches were set to that device at the time the data were sent.

Table 4-3
PROCESSOR CYCLE INDICATORS

Indicator	Function
FI	Indicates that the next processor cycle will be used to fetch an instruction from memory.
FA	Indicates that the next processor cycle will be used to fetch the address or to process an immediate operand in a memory reference instruction.
FO	Indicates that the next processor cycle will be used to process the operand in a memory reference data transmission instruction, to fetch the second address in a deferred memory reference instruction of any type, or to process the operand in an immediate deferred memory reference instruction.
FD	Indicates that the next processor cycle will be used to process the operand in a deferred memory reference data transmission instruction.
BK	Indicates that the next processor cycle will be used to start an interrupt (BRK).
DM	Indicates that the next processor cycle will be used for direct memory access (DMA).
EI	Indicates that the next processor cycle will be used to execute a pair of external instructions.
RUN	Indicates that the processor is in normal operation with one instruction following another. When the computer stops, this indicator goes out.
LE	Indicates that a load error occurred in loading the system program using the Autoload option. A checksum error halts the loading process and lights LE.
IA	Indicates that the interrupt control is active (on).
BOV	Displays the state of the 1-bit Bus Overflow Register.
LNK	Displays the state of the 1-bit Link Register.

4.1.3 Start-Up Procedure

When ac power is applied to the computer, it is important to note the following conditions:

a. The register indicators do *not* necessarily represent the actual contents of the registers until the operator initializes them by performing an operation.

b. The machine is reset as if START had been depressed.

c. If the autorestart switch (located on the power supply) is *off* (or not fitted), the SC is set to location 6 and the computer is stopped.

d. If the autorestart switch is on, the processor begins normal operation at location 6, provided the STOP and SS switches are in the up position.

e. If the autorestart switch is on, it is recommended that the operator depress the STOP switch before applying ac power. The computer then performs one instruction and stops, thereby initializing the register indicators. The contents of the SC depends on the instruction executed.

The following procedure is used to start the GRI-99 Computer:

Step	Procedure
1.	Place the STOP and SS switches in the desired position (refer to par. 4.1.3) and place the POWER switch in the ON position.
2.	Set the DEVICE SELECT switches to 07 (SC).
3.	Set the starting address of the program using the Data Switch Register (SWR).
4.	Depress the TRM key to load the starting address into the SC.
5.	Place STOP and SS switches in the up position.
6.	Depress the START key or CONT to start the program at the SC value. The RUN indicator then lights.

To continue operation in the current computer state, but at *any* desired location, proceed as follows:

Step	Procedure
1.	Depress the STOP switch.
2.	Set the DEVICE SELECT switches to 07 (SC).
3.	Set the new address using the SWR switches.
4.	Depress the TRM key to transmit the new address to the SC.
5.	Place the STOP and SS switches in the up position.
6.	Depress CONT to continue the program or START if it is desired to reset the machine state before continuing.

NOTE

Use of the TRM or DISP key affects only the register selected by the DEVICE SELECT switches and, in some cases, a device connected to that register. For example, transmitting the contents of the SWR to the teleprinter causes the 8-bit character in switches 0-7 to be printed, thus affecting the Output Ready flag and possibly generating an interrupt request if the Teletype interrupt status bit is set.

4.1.4 Examining and Altering Registers

To examine the contents of any register, proceed as follows:

Step	Procedure
1.	Depress the STOP key.
2.	Set the DEVICE SELECT switches to the device address of the desired register. (Appendix B lists the device addresses for GRI-99 devices.)
3.	Depress the DISP key.
4.	Read the contents of the selected register in the DISP indicators.

NOTE

Use of the DISP key does not affect the machine state. However, the DISP key can affect the state of an output device that may share the same system address as an input device. For example, TTI and TTO share 77; displaying 77 also cause the contents of TTI to echo on TTO.

To alter the contents of any register, proceed as follows:

Step	Procedure
1.	Depress the STOP key.
2.	Set the DEVICE SELECT switches to the address of the desired register.
3.	Set the SWR to the new value.
4.	Depress the TRM key.
5.	The DISP key may be depressed to verify that the correct value was loaded.

4.1.5 Reading and Writing in Memory

To read the contents of a memory location, proceed as follows:

Step	Procedure
1.	Depress the STOP key.
2.	Set the DEVICE SELECT switches to 07 (SC).
3.	Set the SWR to the address to be read.
4.	Depress TRM.
5.	If an operator's console is used, set the DEVICE SELECT switches to 06. If a programmer's console is used, this step is unnecessary.
6.	Depress RD.
7.	If the programmer's console is used, the contents of the memory location is displayed in MB. If an operator's console is used, the contents of the location is displayed in the DISP indicators.
8.	Depress the RD switch successively to read successive locations.

To write a value into a memory location, proceed as follows:

Step	Procedure
1.	Depress the STOP key.
2.	Set the DEVICE SELECT switches to 07 (SC).
3.	Set the SWR to the location into which data is to be written.
4.	Depress TRM.
5.	Set the SWR to the value that is to be written into the location.
6.	Lift the WR key.
7.	To write into successive locations, repeat Steps 5 and 6.

4.1.6 Shut-Down Procedure

The following procedure is used to shut down the GRI-99 Computer.

Step	Procedure
1.	Depress the STOP key except in case of Autorestart (Just turn power off).
2.	Turn the POWER switch to OFF.

NOTE

When ac power is turned off, it is important to note the the following conditions:

a. The processor shuts down after approximately 20 ms.

b. The previous contents of memory are unaltered.

4.2 PROGRAMMING

As a prerequisite to a description of programming, it is important to understand the elements of the basic processor and the instruction format. The basic processor (see Figure 4-2) comprises:

a. Back panel bus with bus modifier

b. Controller

c. Four functional operators connected to the bus.

A core memory operator to hold a program and data is also shown in Figure 4-2. The Sequence Counter (SC), Instruction Register (IR), Data Test Operator, and Function Generate/Function Test Operator are required devices using fixed addresses in the bus address scheme. In conjunction with the control logic, these devices select instructions from the program stored in memory, interpret the instruction type as being one of four classes of instructions, and then perform the decoded instruction.

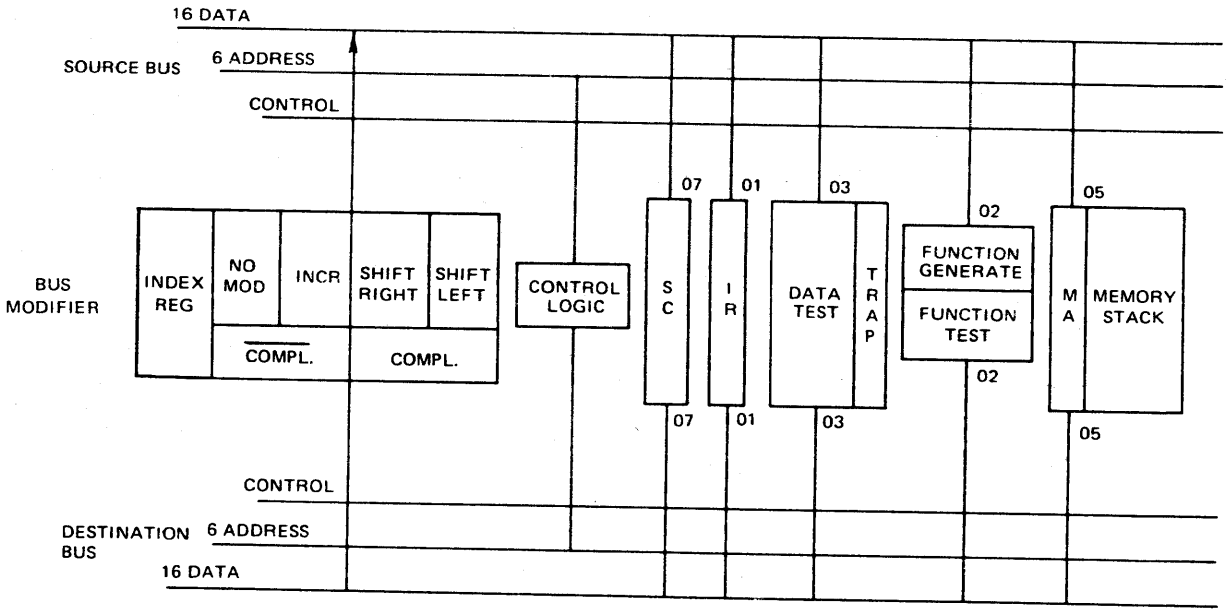
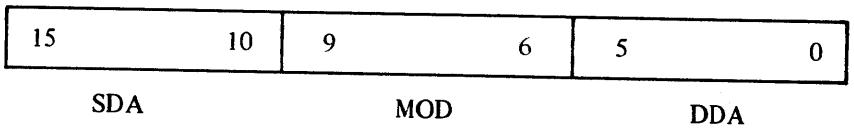


Figure 4-2 Basic Processor

4.2.1 Instruction Format

A GRI-99 command consists of either a one-word instruction or a one-word instruction followed by a second word that contains an address or data. In either case, the one-word instruction is of the general form *DEVICE X TO DEVICE Y* and is described by the format:



where:

SDA = Source Device Address, the source of information transfer.
MOD = Modifier bits for control or functional information.
DDA = Destination Device Address, the destination of information transfer.

4.2.2 Instruction Classes

There are four general classes of GRI-99 instructions:

a. *Function Generation* – Control pulses specified by MOD are transmitted to the named destination device; the manner in which the Destination Device (specified by DDA) interprets the MOD bits defines the function to be performed.

b. *Function Testing* – Status indicators associated with the named source device are sensed, and program flow is altered if the test specified by MOD is true; flow alteration, if any, consists of a skip over the next two memory words.

c. *Data Testing* – Data in the named source device register are tested, and program flow is altered if the test specified by MOD is true; flow alteration, if any, consists of an absolute transfer (jump) to some new location specified by the second word of this instruction.

d. *Data Transmission* – Data are transmitted from the named source device to the named destination device; binary modifications to data in transit and, for memory reference instructions, addressing modes are specified by MOD.

Tables 4-5 through 4-7 in the back of this chapter contain reference data pertaining to the four classes of instructions.

If the elements named as SDA and DDA are system registers that can supply or receive data (i.e., not memory), then the instruction is simply for data transmission and the MOD bits specify the way in which the word taken from the source is modified before being sent to the destination. In some cases, the actual transmission may cause the destination device to perform some function; e.g., sending a character to the tape punch causes that device to punch the character, but the instruction is still classed as data transmission. If the instruction is for data transmission, but the device specified as source or destination is memory, the contents of the next consecutive location following the instruction are used either as the data word transmitted or as an address to locate that word. The MOD bits in the instruction specify not only the modification of the word in transit, but also the way in which the word following the instruction is to be used.

Other types of instructions are produced by specifying an SDA or DDA that is not simply a register to hold data. Naming the Function Generator as SDA supplies up to four control signals to the named DDA, such signals being selected by the MOD bits of the instruction. Specifying the Function Tester as the DDA causes it to test signals from the source device for a skip: bits 6-9 (MOD) select the signals and the conditions they must satisfy. Naming the Data Tester as the DDA causes it to test a word supplied by the source for the conditions specified by the modifier bits; if the condition is true, the processor jumps to a location determined from the word following the instruction in memory.

Thus, although all instructions are of the same form, the repertoire includes a variety of operation types depending upon the properties of the devices addressed. These devices range from basic core memory storage to peripheral devices (such as line printer and magnetic tape), to functional operators for performing arithmetic calculations, to purely control devices that require function control or supply functions for testing.

4.2.3 Device Addresses

The instruction format allows 64_{10} device address codes. Table 4-4 lists the octal codes and mnemonics for the basic device addresses that are part of every system. A complete list of device addresses is included in Appendix B. Sometimes a single code may name two devices, one when referenced as a source, another when referenced as a destination. Thus, the code 02 as a source specifies the Function Generator, but as a destination it specifies the Function Tester. In some cases, a code cannot be used with all instruction types; every code generally has different meanings with different instruction types. For example, the code 00 specifies the control logic when used with Function Generation or Function Testing, but it is a null code when used with data transmission (i.e., as a source, it supplies a zero word, and as a destination it can receive no data). Similarly, the code 04 represents interrupt control elements when used with Function Generation or Function Testing, but represents the Interrupt Status Register when used for a data transfer.

Consider the arithmetic operator (AO), which has three device addresses, two for the registers (AX and AY) and one for the device itself. The registers can be addressed as source or destination of data, but no functions can be generated for them nor do they supply any functions for testing. The AO, on the other hand, can receive functions (specifically the arithmetic or logical operation to be performed), and it has two flags that can be tested. The AO can also be a source of

data because it supplies the result of the arithmetic operation, but it can receive no data as destination. Similarly all I/O devices use both types of function instructions, but an output device generally cannot supply data as a source, and an input device generally cannot receive data as a destination. Specifying a source device that cannot supply data, or a destination device that cannot receive data, is operationally equivalent to specifying the null device address (00). However, pairs of I/O devices sometimes share a common device address; examples are Teletype input and output, and paper-tape reader and punch. In each case, the device address specifies an input buffer as source in data transmission or data testing, an output buffer as destination in data transmission, and the control logic for both devices as source or destination in Function Generation or Function Testing.

Some device addresses are used only for the internal operations of the processor and cannot be used by the program. An attempt by the program to send data to the Instruction Register (IR), device address 01; Memory Address Register (MA), device address 05; results simply in a no-operation (NOP).

Table 4-4

ASSIGNED DEVICE ADDRESSES*

Device Address	Device	Device Address	Device
00	Machine (Null)	14	External Instruction Register
01	Instruction Register	17	Machine Status Register
02	Function Generator	22	Index Register
03	Data Test	23	Trap Register (General-Purpose Register)
04	Interrupt Status Register	24	Byte Swap
05	Memory Address	25	Byte Pack
06	Memory Buffer	30-35	General Registers
07	Sequence Counter	76	High-Speed Reader/Punch
10	Console Switch Register	77	Teletype I/O
11	AX Register		
12	AY Register		
13	Arithmetic Operator		

*For a complete list of device addresses refer to Appendix B.

4.2.4 Effective Address

An instruction that specifies memory as source or destination or specifies the Data Tester as destination requires two words, i.e., it uses the next consecutive memory location following the instruction. Data can be stored in the second location, or its contents can be used as an address.

For any instruction of these types, the processor must determine the *effective address*, which is the actual memory address used to fetch or store the operand or alter program flow. A data transmission instruction that references memory can specify the following addressing modes: direct, deferred (also called indirect and auto-increment), immediate, or immediate and deferred. A jump instruction (i.e., one that names the Data Tester as destination) can specify only direct or deferred addressing. With direct addressing bits 14-0 of the location following the instruction are used as the effective address (i.e., the address of the location to be used for retrieval or storage of data or retrieval of the next instruction). Of course, the latter case holds in a conditional jump only if the condition is satisfied. For deferred addressing, the second word of the instruction specifies a location, the *contents* of which are incremented and replaced. The incremented value is the effective address.

Immediate mode addressing can be used only by the data transmission instructions that reference memory. In this case, the effective address is simply one greater than the address of the instruction; in other words, the contents of the location following the instruction is used as the data or that location is the destination for the data. If addressing is both immediate and deferred, the processor takes the word from the location following the instruction, increments it by one, places the incremented word back in the same location and uses it as the effective address.

4.2.5 Order of Presentation

The description of GRI-99 instructions is presented in three parts:

a. A diagram of the actual bit patterns with appropriate notation as to the interaction of the various bits.

b. Machine language examples in octal and binary, accompanied by the assembly language equivalents and a brief description.

c. The assembly language format of the instruction in symbolic form. Each symbol is explained in detail.

Detailed descriptions and flowcharts of actual instruction execution are included in Chapter 6.

4.2.6 Programming Conventions

The GRI-99 assembly language is called *RAS.* RAS consists of terse, symbolic statements of efficiently writing system programs. The assembler that translates RAS statements into machine language instructions is %RASX. For a complete description of RAS, refer to the GRI-99 *Relocatable Assembler Manual.*

Although all instructions have essentially the same format, the assembler distinguishes four classes: data transmission, data testing, function generating, and function testing. The assembler also distinguishes data transmission instructions that reference memory from those that do not.

The relocatable assembler recognizes a number of mnemonics and other initial symbols that facilitate constructing complete instruction words and organizing them into a program. In particular, there are two-letter mnemonics for the basic instruction types. Letters can be added to these to specify the type of addressing, to select the complement of the source in data transmission, and to specify particular devices for function generating and testing. The basic form for assembling any instruction word is

T *SDA, MOD, DDA*

Where *T* is the mnemonic for the instruction type, *SDA* and *DDA* are the source and destination device address, and *MOD* represents the modifier bits in the middle of the instruction word. For all one-word instructions (those that do not reference memory), *SDA* and *DDA* are two-digit octal device addresses or symbolic names for them. For memory reference instructions, the device address is implied by *T*, and a memory address then replaces the device address in the *SDA* or *DDA* position.

Consider the instruction that takes the word in register AX, shifts the word right one place, and deposits the shifted result in register AY. In mnemonic form this instruction is:

RR AX, R1, AY

which assembles as 11 1100 12. To move a word from memory location 317 to register AX, the mnemonic form is:

MR 317, 11

or

MR 317, AX

either of which assembles as two consecutive words, 06 0000 11 and 000317.

NOTE

Numbers representing instruction words always have a pair of octal digits at each end for the source and destination device address and four binary digits between them for the MOD bits. All numbers representing codes, addresses, and register contents (except instruction words) are always octal, and any numbers appearing in program examples are octal unless otherwise specified. Computer words (other than instructions) are represented by six octal digits, where the left digit is always 0 or 1 representing the value of bit 15.

The programming examples in this manual use the following addressing conventions:

a. A *colon* following a symbol indicates that it is a symbolic location name. For example,

```
A:      RR      10, 04
```

indicates that the location that contains RR 10, 04 may be addressed symbolically as A.

b. A *period* denotes the current address. For example,

```
A:      RR      10, .+4
```

is equivalent to

```
A:      RM      10, A+4
```

c. Anything written at the right of a *semicolon* is commentary that explains the program but is not part of it. For example,

```
        RM      10, .+4         ; store contents of
                                  register 10
```

4.3 INSTRUCTIONS

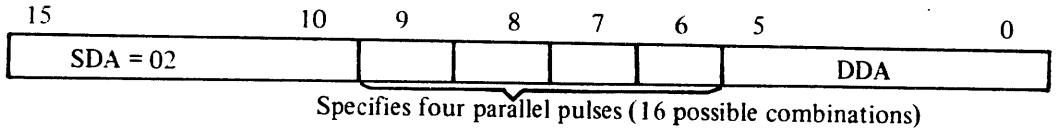
The following sections describe the four general classes of GRI-99 instructions. In each description, a bit map shows the way in which the various bits of the instruction word are placed to produce the desired result. Machine language examples are presented for each instruction type, and assembly language (RAS) statements are described.

4.3.1 Function Generation

The Function Generator, specified by a Source Device Address of 02 (represented as SDA = 02), causes control signals (rather than data) to be transmitted to the device at DDA. The instruction format is:

FO
1 Cycle

Function Output
1.76 μs



Specifies four parallel pulses (16 possible combinations)

The unique combination of MOD and DDA defines the function to be performed. Pulses are transmitted in parallel; therefore, up to 16 unique commands can be issued to the device at DDA. The receiving device must, of course, be capable of decoding the pulse pattern.

Refer to Tables 4-5 through 4-7 for an instruction summary, instruction times, and examples.

4.3.1.1 Machine Language Format – The following examples show machine instructions and RAS statements for some specific Function Generation instructions:

Machine Instruction	RAS	Description
02 0001 76	FO STRT, HSR	Start the high-speed reader.
02 1100 13	FO OR, AO	Set the arithmetic operator to the OR state.
02 1001 77	FO CLIF STRT, HSR	Clear input flag and start the Teletype reader.
02 0001 00	FO CLL, 0	Clear the Link.
02 0010 00	FO STL, 0	Set the Link.
02 0011 00	FO CML, 0	Complement the Link.

4.3.1.2 Assembly Language Format – The SDA is implied by the mnemonic FO; thus, assembly statements for the function generating instructions are of the form:

FO *P, DDA*

where:

FO = Instruction mnemonic (Function Output)
P = Pulse, defines the MOD bits
DDA = Destination device

The assembler recognizes special mnemonics that imply the addresses for the more common destination devices. When the special mnemonics are used, there is no need to specify DDA. For example:

Long Form	Short Form	Description
FO *P*, 0	FOM *P*	Function output to the machine.
FO *P*, ISR	FOI *P*	Function output to ISR.
FO *P*, AO	FOA *P*	Function output to AO.

Also, there are mnemonics for programming the pulse (*P*) bits. Function bits can be combined (ORed) simply by giving the appropriate mnemonics separated by spaces; for example:

FO CLIF STRT, TTI

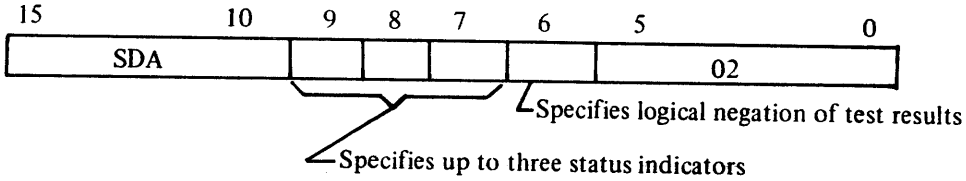
The assembler will also recognize abbreviated forms of the more common mnemonics, such as:

OR AND XOR CLL STL CML

4.3.2 Function Testing

The Function Tester, specified by Destination Device Address 02 (represented as DDA=02), senses status indicators associated with the device at SDA. If the indicators *satisfy* the test specified by MOD, the control logic of the machine causes a skip of the next two locations. In other words, the next instruction to be executed is in the third location following the function testing instruction. If the indicators *do not satisfy* the test specified by MOD, then the instruction in the location following the function test is executed next. The instruction format is:

SF
1 Cycle
Sense Function
1.76 μs



Any bit in MOD (9-7)=1 selects the testing of the corresponding status indicator at SDA. If MOD (6)=0, the test is true if *any* of the selected indicators is on (true). If MOD (6)=1, the test is true only if *all* of the selected indicators are off (false).

Refer to Tables 4-5 through 4-7 for an instruction summary, instruction times, and examples.

4.3.2.1 Machine Language Format – The following examples show machine instructions and RAS statements for some specific Function Testing instructions:

Machine Instruction	RAS	Description
76 1000 02	SF HSR, IRDY	Skip if high-speed reader flag on
76 1001 02	SF HSR, NOT IRDY	Skip if high-speed reader flag not on.
00 0100 02	SF 0, LNK	Skip if Link set (=1).
00 0110 02	SF 0, BOV LNK	Skip if Link or bus overflow set.
00 0111 02	SF 0, NOT BOV LNK	Skip if *neither* Link *nor* bus overflow set.

4.3.2.2 Assembly Language Format – The DDA is implied by the mnemonic SF; as a result, assembly statements for the Function Testing instructions are of the form:

SF *SDA, S*

where

SF = Instruction mnemonic (Sense Function)
SDA = Source device address
S = Status, defines the MOD bits.

NOTE

Note that a function testing instruction skips the next two locations. Hence, if the instruction to be skipped uses only one location, the programmer must fill in with a no-op. The standard no-op is simply 00 0000 00, for which the assembler recognizes the mnemonic NOP.

The assembler recognizes special mnemonics that include the codes for the more common source devices. When the special mnemonics are used, there is no need to specify SDA. For example:

Long Form	*Short Form*	*Description*
SF 0, S	SFM *S*	Sense status of machine
SF AO, *S*	SFA *S*	Sense status of AO.

Also there are mnemonics for programming the status (S) bits. Functions can be combined (ORed) simply by giving the appropriate mnemonics separated by spaces. For example:

SFM BOV LNK

assembles as 00 0110 02. This instruction skips the next two locations if *either* the Bus Overflow *or* Link is set.

The mnemonic NOT preceding the status mnemonics inverts the test, i.e., it places a 1 in bit 6. For example:

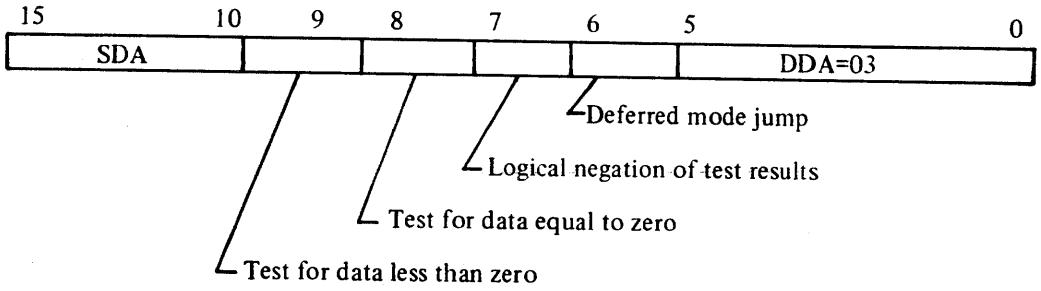
SFM NOT BOV LNK

assembles as 00 0111 02. This instruction skips the next two locations if *neither* Bus Overflow *nor* Link is set.

4.3.3 Data Testing (Jump)

The Data Tester, specified by DDA = 03, tests data at SDA relative to zero. The SDA must be a non-memory register (≠06). If the data test specified by MOD is true, program control is transferred to the address specified in the next location. If the test is not true, the control logic of the machine skips this next location. The instruction format is:

JC Jump Conditional



The bits in MOD (9-7) specify the precise test that the data at SDA must satisfy in order for a jump to occur. The combinations are:

MOD (9-7)	Mnemonic	Test
100	LTZ	Less than zero.
101	GEZ	Not less than zero (greater than or equal to zero).
010	-ETZ	Equal to zero.
011	NEZ	Not equal to zero.
110	LEZ	Less than or equal to zero.
111	GTZ	Not less than *and* not equal to zero (greater than zero).

A jump instruction is a two-word instruction. If a jump occurs, the address of the second word of the Data Test instruction is placed in the Trap Register, 23 (also addressed as 03 for compatibility purposes with GRI-909 software) before the jump address is transmitted to the SC. This procedure provides a link back to the calling program if the jump was to a subroutine.

NOTE

When a Data Test instruction is true and a jump occurs, the previous contents of the Trap Register (23) will be destroyed. If no jump occurs, the Trap Register is not affected.

In the direct mode jump (MOD 6=0), the contents of the second word of the instruction is transmitted to the SC. In the deferred mode jump (MOD 6=1), the contents of the second word is used as a memory address to fetch the jump address. This jump address is incremented, replaced in memory, and the incremented value is transmitted to the SC.

Refer to Tables 4-5 through 4-7 for an instruction summary, instruction times, and examples.

4.3.3.1 Machine Language Format – The following examples show machine instructions and RAS statements for some specific Data Testing instructions.

Machine Instruction	RAS	Description
11 1000 03 000125	JC AX, LTZ, 125	If AX less than zero, jump to location 125.
77 0101 03 000200	JCD TTI, ETZ, 200	If Teletype input equal to zero, jump deferred through location 200.
23 1110 03 004150	JC TRP, GTZ, 4150	If the Trap Register is greater than zero (i.e., not less than nor equal to zero), jump to location 4150.
00 0100 03 000500	JU 500	Jump to location 500 (00 as source is a zero word; thus, the equal to zero test is always true).

4.3.3.2 Assembly Language Format – The DDA is implied by the mnemonic JC; as a result assembly statements for data testing are of the form:

JC *SDA, T, Address*

where:

JC = Instruction mnemonic (Jump Conditional)
SDA = Source device containing data to be tested
T = Test to be performed, as defined in the MOD bits, as follows:

Bit 9-7	Mnemonic	Jump Function
000	–	Never jump
001	–	Always jump
010	ETZ	Jump if Equal to Zero
011	NEZ	Jump if not equal to zero
100	LTZ	Jump if less than Zero
101	GEZ	Jump if Greater than or Equal to Zero
110	LEZ	Jump if less than or Equal to Zero
111	GTZ	Jump if Greater than Zero

address = The address to which control is transferred if the test is true. This address is called the *jump address*. The *address* is assembled into the second word of the two-word data test instruction.

Jump Conditional Deferred (JCD) – A deferred jump instruction takes the following form in the assembly language:

JCD *SDA, T, address*

where

JCD = Instruction mnemonic that specifies the setting of bit 6 (Jump Conditional Deferred).
SDA = Source device containing data to be tested.
T = Test to be performed, as defined in the MOD bits (refer to JC)
address = The address of the *jump address* – 1. When a deferred jump occurs, the contents of the location referenced by *address* is auto-incremented and control is transferred to the location specified by the incremented result.

Jump Unconditional (JU) – The assembler recognizes a special mnemonic (JU) that specifies a condition that is *always* satisfied, i.e., the mnemonic always causes a jump to occur. When using JU, it is not necessary to specify the SDA or the test to be performed, because the condition implied by JU is that the SDA is the null register and the test performed is ETZ. For example,

Long Form	Short Form	Description
JC 0, ETZ, 100	JU 100	Jump unconditional to location 100.

As in the case of JC, the JU instruction can specify deferred mode addressing. For example,

Long Form	Short Form	Description
JCD 0, ETZ, 100	JUD 100	Jump unconditional deferred *through* location 100.

4.3.3.3. **Indexing** – Indexing can be used with Jump instructions. When indexing is specified the contents of the Index Register is added to the address that is fetched to produce the jump address. To index an instruction, bit 15 of the word that is fetched must be set.

JC with Indexing – The assembly language format for a JC instruction with indexing is:

JC *SDA, T, #address*

where:

JC = Instruction mnemonic (Jump Conditional)
SDA = Source device containing data to be tested.
T = Test to be performed, as defined in the MOD bits (refer to JC).
# = The special symbol to specify indexing. (Sets bit 15 of *address* to 1).
address = If *T* is true, the address to which the contents of the XR are added to produce the jump address. Address is assembled into bits 14-0 of the second word of the two-word instruction.

JCD with Indexing – The assembly language format for a JCD instruction with indexing is:

JCD *SDA, T, #address*

where

JCD = Instruction mnemonic (Jump Conditional Deferred)
# = The symbol to specify indexing.
SDA = Source device containing data to be tested.
T = Test to be performed, as defined in the MOD bits (refer to JC).

address = The XR is added to *address* to produce another address, the contents of which is incremented and replaced. Program control is then transferred to the address specified by the incremented value.

JU with Indexing – The JU instruction, which specifies a condition that is always satisfied, can also be used with indexing. That is, a jump will always occur to the address formed by the sum of *address* and the Index Register value. The assembly language format is:

```
JU      # address
```

4.3.3.4 Using the Jump Instruction – The jump instruction can be used to call a subroutine, because when a jump occurs the address of second word of the jump instruction is stored in the Trap Register. When the subroutine is finished, and if the subroutine *has not modified* the contents of the Trap Register, it can return control to the calling program by executing the one-cycle instruction:

```
RR      TRP, SC
```

which causes a return to the contents of TRP + 1 (refer to Section 4.3.9). If the contents of the Trap register is likely to be affected by the subroutine itself, then the first instruction of the subroutine must save the return address. For example,

```
SUB:    RMI TRP, 0
```

The subroutine returns control to the calling program in either of the following ways:

```
JUD SUB + 1
    or
JCD     SDA, T, SUB + 1
```

For a complete description of subroutine linkage and argument passing, refer to the GRI-99 *Relocatable Assembler Manual.*

4.3.4 Data Transmission

A Data Transmission Instruction causes the transmission of data from a source device, through the bus modifier, to a destination device. There are two types of Data Transmission instruction:

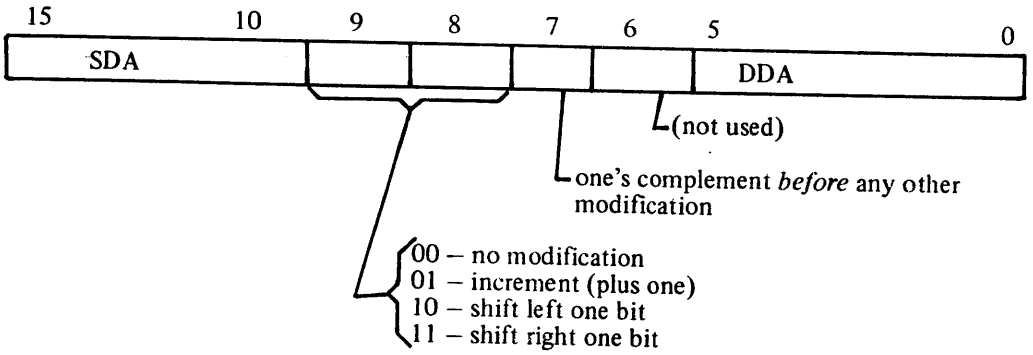
a. Non-memory reference – instructions that transmit data from one non-memory register to another.

b. Memory Reference – instructions that specify memory as the source or destination of data.

4.3.4.1 Non-Memory Reference Transmission – These instructions enable the transmission of data directly between system devices. The instruction format is:

RR

Register to Register



NOTE

It is important to note that for RR transfers, if bit 15 of the source data is set and DDA=07 (SC) the source data are automatically indexed and placed in the SC.

Refer to Tables 4-5 through 4-7 for an instruction summary, instruction times, and examples.

Machine Language Format – The following examples show machine instructions and RAS statements for some specific non-memory reference instructions:

Machine Instruction	RAS	Description
11 0000 76	RR AX, HSP	Transmit contents of register AX to high-speed punch.
77 0000 77	RR TTI, TTO	Transmit Teletype input to Teletype output (echo function).
11 0100 11	RS AX, P1	Increment register AX.
00 0000 12	ZR AY	Clear register AY (00 is source of zeros).
76 0110 12	RRC HSR, P1, AY	Two's complement of high-speed reader to register AY (one's complement followed by increment.
11 1000 11	RS AX, L1	Shift AX left one bit (rotate left one bit through link)
11 1100 11	RS AX, R1	Shift AX right one bit (rotate right one bit through link)

Assembly Language Format – The basic assembly language format for instructions that transfer data from one non-memory register to another is:

RR [C] *SDA, MOD, DDA*

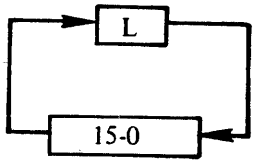
where

RR = Instruction mnemonic (Register to Register).

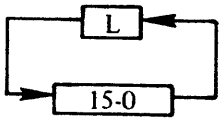
SDA = Source device that originates the data.

MOD = The modification to data, as defined in the MOD bits (9 and 8). The combinations of bits 9 and 8 and the associated mnemonics are:

Bits 9-8	Mnemonic	Modification
00	–	None
01	P1	Add +1. If the result is 2^{16}, set Bus Overflow, otherwise clear it.
10	L1	Rotate left one place. Bit 15 is shifted into the link, the link into bit 0.



11	R1	Rotate right one place. Bit 0 is shifted into the link, the link into bit 15



DDA = Destination device to receive the data

Also, the 1's complement of the data at SDA can be taken, modified or not modified by the user, and the result transferred to DDA. The assembly format is:

RRC *SDA, MOD, DDA*

To transfer zero to a register, the mnemonic is ZR. To transfer the 1's complement of zero to a register, the mnemonic is a ZRC. When using ZR or ZRC the SDA is unnecessary:

Long Form	Short Form	Description
RR 0, P1, AX	ZR P1, AX	Transfer 0 plus 1 to the AX.
RRC 0, L1, AX	ZRC L1, AX	Transfer 1's complement of 0 rotated left one bit, to the AX.

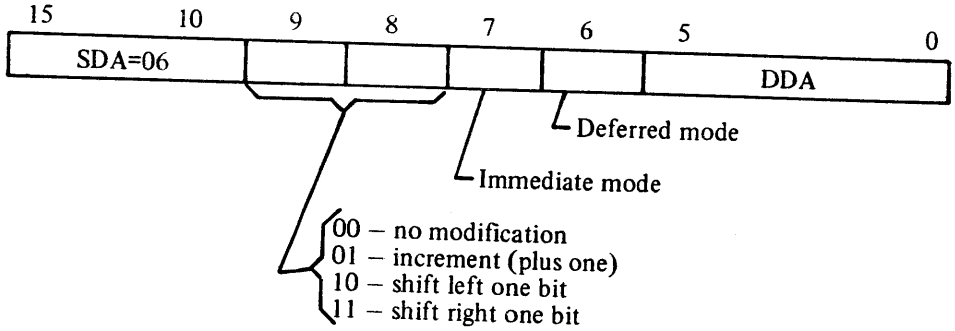
A register can also be transferred to itself (RS) or complemented and transferred to itself (RSC). When using the mnemonics RS or RSC, the DDA is unnecessary:

Long Form	Short Form	Description
RR AX, P1, AX	RS AX, P1	Transfer the contents of AX plus 1 to the AX.
RRC AX, P1, AX	RSC AX, P1	Transfer the 1.s complement of AX plus 1 to the AX. This instruction produces the 2's complement of the source datum.

4.3.4.2 **Memory Reference Transmission** – These instructions enable the transmission of data to and from memory. If SDA=06, data are transmitted from memory. If DDA=06, data are transmitted to memory. If SDA=DDA=06, data are transmitted from a memory location back to itself. Memory reference instructions require 2-4 cycles for execution. The instruction formats are:

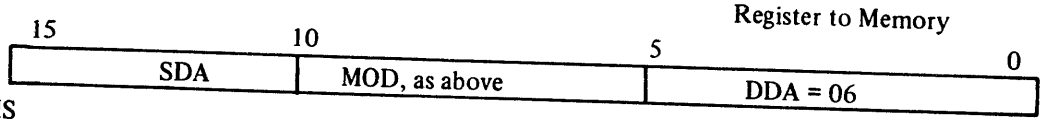
MR

Memory to Register



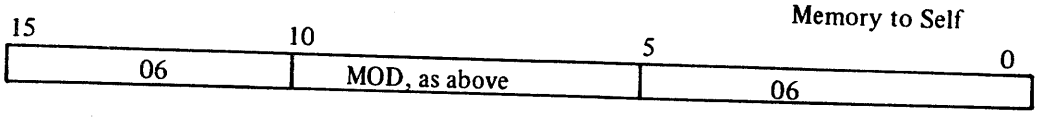
RM

Register to Memory



MS

Memory to Self



All Memory Reference Data Transmission Instructions are two words in length. The first word is the actual instruction. The second word is either:

a. The source of data.
b. The destination of data.
c. An address of data.
d. An address that points to another memory address (a deferred or indirect address).

The address of the memory location into which data are transmitted or from which data are fetched is called the *effective address.* The effective address is determined by the addressing mode as specified by MOD (7-6):

Bits 7-6	Mnemonic	Description
00	–	Direct mode – the contents of the second word is the effective address.
01	D	Deferred mode – the contents of the second word is used to fetch another address which is incremented and replaced. The incremented value is the effective address.
10	I	Immediate mode – the effective address is the *address* of the second word of the instruction. Thus, the contents of the second word of the instruction is data.
11	ID	Immediate and deferred mode – the contents of the second word is incremented and replaced. The incremented value is the effective address.

Refer to Tables 4-5 through 4-7 for an instruction summary, instruction times, and examples.

Machine Language Format – The following examples show the machine instructions and RAS statements for some specific memory reference instructions (assume location 500 contains 1000):

Machine Instruction	RAS	Description
11 0000 06 000200	RM AX, 200	Transmit contents of AX to location 200.
06 0010 23 177776	MRI -2, TRP	Transmit contents of second word (-2) to the Trap Register.
23 0010 06 000000	RMI TRP, 0	Transmit contents of Trap Register to second word – sometimes used to save subroutine linkage.
06 0110 06 000100	MSI 100, P1	Increment second word of this instruction.
06 0001 11 00500	MRD 500, AX	Transmit contents of location 1001 to register AX.
13 0011 06 001000	RMID AO, 1000	Transmit contents of AO to location 1001.

The deferred mode is sometimes called *one level indirect with auto-indexing.* The term *indirect* in this sense means that the address in the instruction is not the effective address but is the *address* of the effective address-1. The term *auto-indexing* means that the effective address is automatically incremented before it is used.

Assembly Language Format – There are four basic types of memory reference transfers:

a. Memory to register (MR).
b. Register to memory (RM)'
c. Memory to self (MS).
d. Zero to memory (ZM).

These four transfers can be accomplished in four addressing modes: direct, deferred, immediate, and immediate and deferred. Also, the indexing feature can be used with the instructions.

4.3.4.3 Indexing – Indexing is a valuable tool for expanding the capability of many memory reference instructions, as well as Jump instructions (Data Testing). To effectively use indexing, it is essential that the programmer understand the method of specifying indexing and the way in which indexing interacts with the various instructions.

In the GRI-99 any value that is used as an address can be indexed. For memory reference instructions, Jump instructions, or any value sent to the SC, if bit 15 is set, the value will be indexed.

When a value is indexed, the 15 bit address quantity is added to the 16 bit XR to form a *15-bit* effective address. (The high-order bit of the sum is dropped.)

Example 1

If the XR contains 100 and the instruction

```
MR      # 3, AX
```

is executed, the quantity # 3, which is internally 100003, is added to the XR (000100) to produce the sum 100103. The high-order bit is then dropped, yielding the effective address 103. (The contents of location 103 is then loaded into AX.)

Example 2

If the XR contains 100 and the instruction

```
MR      # -3, AX
```

is executed, the effective address is the low-order 15 bits of the sum of 100 and -3 (177775), which is location 75.

Example 3

Similarly, if the XR contains -3 and the instruction

```
MR      # 100, AX
```

is executed, the effective address is the low-order 15 bits of the sum of the XR (177775) and the quantity # 100 (100100). The actual sum is 100075; however, the high-order bit is dropped to produce the effective address (75).

Example 4

If the XR contains 100 and the instruction

```
MRI     # 3, SC
```

is executed, the low-order 15 bits of the sum of the XR value and the quantity # 3 is incremented (refer to Section 4.3.9) and sent to the SC. Thus, the program resumes execution at location 104.

Example 5

If the instruction:

```
MRI     # 3, AX                    (or any register
                                    other than SC)
```

is executed, the value 100003 is deposited in the AX (or the indicated register).

Memory to Register (MR) Direct Mode – The basic assembly language format for MR instructions in direct mode is as follows:

```
MR      address, MOD, DDA
```

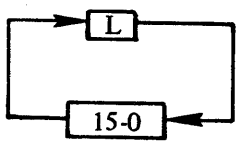
where

MR = Instruction mnemonic

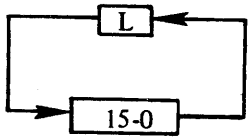
address = The address in memory from which data is fetched. *Address* is stored in the second word of the instruction.

MOD = Modification to data, as defined in the MOD bits (9 and 8). The combinations of bits 9 and 8 and associated mnemonics are:

Bits 9-8	Mnemonic	Modification
00	–	None
01	P1	Add +1. If the result is 2^{16}, set Bus Overflow, otherwise clear it.
10	L1	Rotate left one place. Bit 15 is shifted into the link, the link into bit 0.



11	R1	Rotate right one place. Bit 0 is shifted into the link, the link into bit 15.



DDA = Destination device that receives the data.

MR Direct with Indexing – The assembly language format for MR instructions in direct mode using indexing is as follows:

MR *# address, MOD, DDA*

where:

= The special mnemonic to specify indexing (Sets bit 15).

address = The address to which the contents of the index register is added, yielding the address from which data are to be transferred (effective address).

MR Deferred (MRD) – The basic assembly language format MRD instructions is as follows:

MRD *address, MOD, DDA*

where:

MRD = Instruction mnemonic for deferred addressing

address = *Address* is stored in the second word of the instruction and points to another location which contains the *effective address* –1. During execution, the contents of this (latter) location is auto-incremented to produce the effective address. Data are then fetched from the effective address.

MRD with Indexing – The assembly statement for MRD instructions with indexing is as follows:

MRD *# address, MOD, DDA*

where

= the special mnemonic to specify indexing

address = The Index Register value is added to *address* to form a new address. The contents of the new address is incremented and replaced and the incremented value is the effective address.

MR Immediate Mode (MRI) – The basic assembly language format for MRI instructions is:

where: MRI *data, MOD, DDA*

MRI = Instruction mnemonic for immediate mode addressing

data = The contents of the second word of the instruction. DDA is loaded with this value.

NOTE

If DDA = 07 (SC) for an MRI instruction, the SC is loaded with *data + 1*. If *data* has bit 15 set (indexing), then the sum of XR + *data* + 1 is loaded into the SC.

MR Immediate and Deferred (MRID) – The basic assembly language statement for MRID instruction is:

MRID *address –1, MOD DDA*

where

MRID = Instruction mnemonic for immediate and deferred mode addressing.

Address –1 = This is the second word of the instruction. It is incremented and replaced to produce the effective address. Data are then transferred from the effective address to *DDA*.

MRID with Indexing – The basic assembly language statement for MRID instructions with indexing is:

MRID *# address –1, MOD, DDA*

where:

= The special mnemonic to specify indexing

address –1 = This is the second word of the instruction. It is incremented and replaced to produce *address*. The XR value is then added to *address* (if bit 15 remains set after incrementing), thereby producing the effective address. Data are then transferred to *DDA* from the effective address.

Register to Memory (RM) Direct Mode – The basic assembly language format for RM instructions in direct mode is:

RM SDA, *MOD, address* (DDA = 06 is implied by the mnemonic)

where

RM = Instruction mnemonic

SDA = The source device that supplies the data

MOD = Modification to data, as defined in the MOD bits (9 and 8). For the combinations of bits 9 and 8 and associated mnemonics, refer to MR direct mode.

address = The address in memory to receive the data. *Address* is stored in the second word of the instruction.

RM Direct and Indexing – The basic assembly language format for RM instructions in direct mode using indexing is:

RM *SDA, MOD, # address*

where:

= The special mnemonic to specify indexing

address = The address to which the contents of the Index Register is added, yielding the address to which data are transferred (effective address).

RM Deferred (RMD) – The basic assembly language format for RMD instructions is:

RMD *SDA, MOD, address*

where

RMD = Instruction mnemonic for deferred mode addressing.

address = *Address* is stored in the second word of the instruction and points to a location which contains the effective address –1. During execution, the contents of this location is autoincremented to produce the effective address, to which data are then transferred.

RMD with Indexing – The assembly statement for RMD instructions with indexing is:

RMD *SDA, MOD, # address*

where

= The special mnemonic to specify indexing.

address = The Index Register value is added to address to yield an address the contents of which is autoincremented to form the effective address.

RM *Immediate Mode (RMI)* – The basic assembly language format for RMI instructions is:

RMI *SDA, MOD, X*

where

RMI = Instruction mnemonic for immediate addressing.

X = A dummy expression. The effective address (where data are stored) is actually the location following the instruction (i.e., SC + 1), thereby overwriting the value *X*.

RM Immediate and Deferred (RMID) – The basic assembly language statement for RMID instructions is:

RMID *SDA, MOD, address –1*

where

RMID = Instruction mnemonic for immediate and deferred mode addressing.

address –1 = This is the second word of the instruction. It is autoincremented to produce the effective address, to which data are then transferred.

RMID with Indexing – The basic assembly language statement for RMID instructions using indexing is:

RMID *SDA, MOD # address –1*

where

= The special mnemonic to specify indexing.

address –1 = *Address –1* is incremented to produce *address* and, if the incremented value has bit 15 set, the contents of the Index Register is added to *address*, yielding the effective address.

Zero to Memory (ZM) – The ZM instruction is a special case of register to memory (RM). The source of a ZM instruction is the null device (00); therefore, the source word is 0, and the data sent to memory depends on the operations performed in the bus modifier. The basic assembly language format for ZM instructions is:

```
ZM      MOD, address
```

where

ZM = Instruction mnemonic

MOD = Modification to data, as defined in the MOD bits (9 and 8). The combinations of bits 9 and 8 and associated mnemonics are shown under MR.

address = The effective address that receives data from the bus modifier.

The four addressing modes can be used with ZM in the same manner as RM. Also, ZM instructions can be indexed.

Memory to Self (MS) – The MS instruction specifies the MB as both the source and destination of data; thus, the contents of a single memory location can be modified directly. The basic assembly language format for MS instructions is:

```
MS      address, MOD
```

where

MS = Instruction mnemonic

address = The address in memory to be modified

MOD = The modification to data, as defined in the MOD bits (9 and 8). The combinations of bits 9 and 8 and associated mnemonics are described under MR.

The four addressing modes can be used with MS. Also, MS instructions can be indexed.

ZM and MS Examples – For convenience the assembler recognizes special mnemonics for clearing a location and transferring a location into itself. Letting *W* be the word given for the second location.

ZM *M*, W	is equivalent to	RM 0, *M*, *W*
MS *W*, *M*	is equivalent to	MR *W*, *M*, 6

In both cases the letters *D*, *I* and *ID* can be appended to the basic mnemonic to select deferred, immediate, and immediate-deferred addressing. To simply keep a count of thirty in the location following an instruction we could give

```
MSI     -36, P1         ;30(10) = 36(8)
```

which assembles as 06 0110 06 and 177742. The thirtieth iteration of this instruction sets the Bus Overflow flag.

Addressing Examples: Suppose AX contains 000132 and the following locations contain the numbers listed:

Location	Contents
321	001742
1742	005360
1743	134267
5361	000023

Also assume that the link contains a 0 prior to instruction execution. Executing each of the following instructions in location 320 produces the effects given.

NOTE

Each of the instructions is a two-word instruction; thus, the second word is in location 321.

Instruction		Effect
MR	1742, AX	Load 5360 in AX.
MRD	1742, AX	Change location 1742 to 5361 and load 23 in AX.
MRI	1742, AX	Load 1742 in AX.
MRID	1742, AX	Change location 321 to 1743 and load 134267 in AX.
RM	AX, 1742	Store 132 in location 1742.
RMD	AX, 1742	Change contents of location 1742 to 5361 and store 132 in location 5361.
RMI	AX, 1742	Store 132 in location 321.
RMID	AX, 1742	Change contents of location 321 to 1743 and store 132 in location 1743.
MS	1742, R1	Change contents of location 1742 to 2570 (5360 divided by 2) and clear link.
MSD	1742, R1	Change contents of location 1742 to 5361 and change contents of location 5361 to 11 (23 divided by 2) and set link.
MSI	1742, R1	Change contents of location 321 to 761 (1742 divided by 2) and clear link.
MSID	1742, R1	Change contents of location 321 to 1743 and change contents of location 1743 to 56133 (134267 divided by 2) and set link.
ZM	P1, 1742	Store 1 in location 1742 and clear BOV.
ZMD	P1, 1742	Change contents of location 1742 to 5361 and store 1 in location 5361 and clear BOV.
ZMI	P1, 1742	Store 1 in location 321 and clear BOV.
ZMID	P1, 1742	Change contents of location 321 to 1743 and store 1 in location 1743 and clear BOV.

Suppose further that:

XR = 100

Executing each of the following instructions in location 320 *with indexing* produces:

Instruction		Effect
MR	# 1642, AX	Load 5360 into AX.
MRD	# 1642, AX	Change location 1742 to 5361 and load 23 into AX.
MRI	# 1742, AX	Load 101742 into AX.
MRID	# 1642, AX	Change location 321 to 101643 and load 134267 in AX.
RM AX,	# 1642	Store 132 in location 1742.
RMD AX,	# 1642	Change contents of location 1742 to 5361 and store 132 in location 5361.
RMI AX,	# 1742	Store 132 in location 321.
RMID AX,	# 1642	Change contents of location 321 to 101643 and store 132 in location 1743.
MS	# 1642, R1	Change contents of location 1742 to 2570 and clear link.
MSD	# 1642, R1	Change contents of location 1742 to 5361 and change contents of location 5361 to 11 and set link.
MSI	# 1742, R1	Change contents of location 321 to 40761 (101742 divided by 2) and clear link.
MSID	# 1642, R1	Change contents of location 321 to 101643 and change contents of location 1743 to 56133 and set link.

Indexing can also occur at *two* levels when deferred mode addressing is specified. Suppose further that:

location 1100 = 101642

Instruction		Effect
MRD	# 1000, AX	Change contents of location 1100 to 101643 and load 134267 into AX.
RMD	AX, #1000	Change contents of location 1100 to 101643 and the store 132 in location 1742.
MSD	# 1000, R1	Change contents of location 1100 to 101643 and change contents of location 1743 to 56133.

4.3.5 Program Interrupt

Most Input/Output (I/O) devices must be serviced infrequently relative to the processor speed. Only a small amount of processor time is required to service these devices, but they must be serviced within a short time after they request service. Failure to service within the specified time (varies for different 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, i.e., the use of interrupts in the current program sequence facilitates concurrent operation of the main program and a number of peripheral devices. The hardware also allows a power failure to signal the program by requesting an interrupt.

4.3.5.1 Interrupt Requests – Interrupt requests by a device are governed by its Ready and Interrupt Status bits. When a device is ready to send or receive data, it sets Ready, and this action requests a program interrupt if its Interrupt Status bit has been set by the program; if the Interrupt Status bit for a particular device is clear, the device cannot cause an interrupt.

At the beginning of every cycle the processor synchronizes any requests that are then being made. After a request has been synchronized, the device that made the request must wait for an interrupt to start. The request signal is a level; thus, after being synchronized it remains on the bus until the program clears the Ready or Interrupt Status bit. In other words, clearing either bit in a device disables any request the device has already made and that had been synchronized; the device can no longer interrupt. If the Interrupt Status bit was cleared, but Ready remained set, then subsequently setting the Interrupt Status bit again restores the request for that device.

4.3.5.2 Starting an Interrupt – The processor starts the interrupt sequence if all four of the following conditions are met:

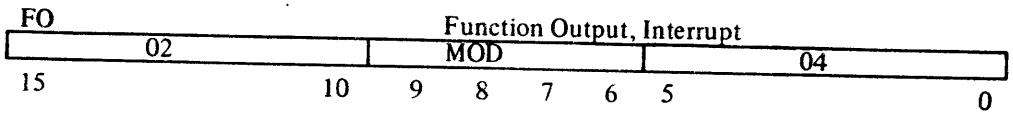
a. The processor has just completed an instruction or a direct memory access. Insofar as interrupts are concerned, an entire sequence of external instructions is equivalent to a single instruction in the program: when a sequence is started, the processor does not handle any interrupts until the sequence is finished.

b. At least one device is waiting for an interrupt to start (i.e., it was requesting an interrupt at the beginning of the last cycle).

c. The interrupt control is on (via an FOI ICO instruction).

d. No device is waiting for direct memory access; i.e., there are no requests for such access that the processor has synchronized but not yet fulfilled. *The direct memory channel has priority over program interrupts.*

When the processor finishes an instruction, it takes care of all direct memory requests before it starts an interrupt; this order includes any additional direct memory requests that are synchronized while access is occurring. When no more devices are waiting for access, the processor starts an interrupt if the interrupt control is on and a device was requesting an interrupt at the beginning of the last access. The program governs the interrupt operator through the following instruction:



Perform the functions specified by 1s in MOD as follows.

Bit	Mnemonic	Function
6	ICF	Turn interrupt control off.
7	ICO	Turn interrupt control on.

When an interrupt sequence occurs, the processor automatically turns off the interrupt control to prevent further interrupts, saves SC (which points to the next instruction) in a location the address of which is supplied by the device, and loads SC with an address one greater than that supplied. The processor then proceeds to the instruction in the location now addressed by SC and continues sequential operation from there. In general, three locations are allocated for each interrupt, as follows:

a. The first location (address is supplied by the device) receives the current contents of SC.

b. The second location should contain the instruction 06 0010 07 i.e., an MRI *XXX*, SC.

c. The third location should contain an address one less than the first location in the service routine for the device.

The memory locations allocated to the basic I/O devices are as follows:

Location	Device
11-13	Teletype output
14-16	Teletype input
17-21	High-speed punch
22-24	High-speed reader

Other GRI supplied devices generally are allocated locations in lower core. (Appendix B lists the interrupt locations for all GRI-supplied devices).

A breakpoint (refer to paragraph 4.3.5.6) or a power failure causes an interrupt to location 0. The first six locations could be set up for these two combined channels in the following manner:

Location	Device
0	SC stored here.
1	SFM POK (skip if power OK)
2, 3	MRI PFAIL-1, SC (power fail routine)
4, 5	MRI BRKPT -1, SC (breakpoint routine)

In a large system, it may be necessary to have two or more devices sharing a single channel; in such a case, the third location must contain an address for a common routine for all of them. The hardwired address in any device can be disabled so that it interrupts to location 0. If some device (but not all) interrupt to 0, then the instruction in location 4 should take the processor to a service routine for those device (plus breakpoint). If all devices interrupt to 0, the service routine can begin right in location 1.

A device may be hardwired to interrupt directly to its service routine. In this case, SC is stored in the first location of the routine, and the second location contains the first instruction of the routine.

4.3.5.3 Servicing an Interrupt – If more than one device is connected to a single channel, (generates the same interrupt address) the service routine must first determine the device that requires service. A series of SF instructions is used for this purpose. After the device has been identified, the routine should clear the Ready flag and save the contents of any registers or flags that are to be used in the routine or may be affected by it. Consequently, the routine should save the Machine Status Register if there is to be any modification in the bus modifier, as Bus Overflow or the Link can be affected by such operations. Similarly TRP should be saved if the routine contains a jump or uses it as a general-purpose register. Then the program services the device. While doing so it can simply leave the interrupt off, or it can re-enable the interrupt and establish a priority structure that allows higher priority devices to interrupt the current device service routine. This priority is determined by controlling the states of the individual Interrupt Status bits for the various devices.

4.3.5.4 Device Priority – There are several ways in which priorities are assigned to devices:

a. An elementary priority is established by the hardware for devices that are requesting interrupts simultaneously in that the processor brings in a channel address from one *and only one* device: among those that are waiting it takes the address from the device that is physically closest to the processor on the bus. This situation, however, applies only to those devices that are waiting at the time an interrupt is started.

b. Using Sense Functions (SF's) to determine which device to service establishes a priority by the order in which the devices are tested, but again this situation applies only to those devices that are waiting at the time.

c. The most significant method is by controlling the Interrupt Status bits to specify which devices can interrupt a service routine currently in progress. These flags are each connected to a particular data line in the Source and Destination Buses. Collectively they constitute the Interrupt Status Register. By addressing this register as device address 04, mnemonic ISR, the program can save the current priority structure, establish a new one, or restore a previous one. There is no established priority, as the program can set up any ISR configuration.

All devices whose Interrupt Status bits are clear cannot cause an Interrupt to start and, therefore, are regarded by the program as being of lower priority. Those devices in which Interrupt Status is set can interrupt the current routine and, therefore, are regarded by the program as being of higher priority.

The following lists the devices assigned to the bits in ISR. (Complete information on all devices is given in Appendix B.)

ISR Bit	Device
0	Teletype output
1	Teletype input
2	High speed punch
3	High speed reader
4	
5	
6	
7	
8	
9	Other Devices
10	
11	
12	
13	
14	
15	

By means of ISR the program can establish any priority structure with one limitation: *two or more devices whose flags are the same bit in ISR* (i.e., are connected to the same data line) *are all at the same priority level.* When an interrupt is in progress for a device, the rest of the devices assigned to the same bit must be regarded as all of higher priority or all of lower priority depending on whether they are enabled or not.

4.3.5.5 Dismissing an Interrupt – After servicing a device, the routine should restore the pre-interrupt states of any devices affected by the routine (MSR, TRP, general registers, ISR), turn on the interrupt control (FOI ICO), and return to the interrupted program. The instruction that turns the interrupt on again has no effect until the next instruction is completed. Thus, after that instruction the processor always executes one more instruction (assumed to be the return to the interrupted program) before another interrupt can start. It is recommended that this return be implemented by, thus guaranteeing that the TRP is not disturbed by the return after interrupt.

```
MR      INTLC, SC
```

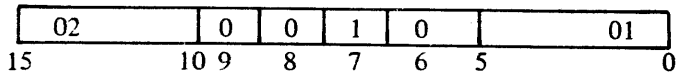
where:

INTLC = location where SC was stored on interrupt.

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 re-enable 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.

4.3.5.6 Breakpoint – The program can generate special interrupt at any time by the use of the following instruction.

FO 2,1 Function Output, Breakpoint



After executing this instruction, the processor first handles any direct memory requests that are waiting, but as soon as the last DMA is completed, a break cycle occurs. The sequence counter is stored at 0, and execution begins at location 1. The ISR should be 0, but the Interrupt control may be on or off. Starting the interrupt automatically removes the breakpoint request, but as previously indicated by the instructions recommended for locations 1-5, it should be assumed that a breakpoint has occurred if there is no power failure.

Obviously breakpoints are not used in any normal programming situation. They are used almost exclusively for debugging purposes.

4.3.5.7 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 direct memory channel is in use. A single device shuts out all others of lower priority if every time its service routine dismisses the interrupt, it is already waiting with another request. The direct memory channel can also preempt all processor time. If the channel is not in use, the highest priority 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. Maximum waiting time for ordinary instructions including synchronization is, therefore, about 9 μs. However, the maximum possible time can be much longer if the program uses optional functional operators that stop the stored program (External Instruction devices). For example, multiplication can delay an interrupt by 56 μs when done by external instructions.

The time the processor takes to start an interrupt and jump to a service routine is three cycles, about 5 μs. An interrupt directly to a routine takes only the single cycle for storing SC.

4.3.5.8 When to Use the Interrupt – If the program has little computing to do and is using only one or two fast I/O 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 an ISR priority scheme, are: what the program is doing in addition to I/O and the time required by the service routines.

Example 1 Suppose the program is doing nothing but processing data using reader, punch and Teletype, and further suppose that no service routine requires more than half a millisecond. In these circumstances, the program could dispense with the interrupt and test all the devices (in the order just stated) by a loop containing SF and jump instructions, where the reader service routine returns to the punch test and all others return to the beginning of the loop. The fastest device, the reader, is still capable of running at full speed with no loss of data.

Example 2 Suppose the program has a significant amount of computing to do. Then the interrupt must be used, but priority is a very real consideration. If input-output service for the Teletype requires .8 ms and punch service requires .5 ms, then the reader service is never delayed too much if the interrupt is simply turned off while servicing each device. But if Teletype service requires 20 ms per character, then neither reader nor punch is able to run at full speed unless ISR is used to set up priority levels.

4.3.6 Direct Memory Access (DMA)

A DMA is a machine cycle granted to a requesting device for the purpose of sending a 16-bit word to memory or receiving a 16-bit word from memory or incrementing a memory location. Devices using the DMA channel require special logic for synchronizing with the channel.

The maximum rate for data transfers between external devices and core memory can be no greater than 80,000 word per second if the transfers are executed under program control. To allow rates up to 568,000 words per second, the processor contains a direct memory channel through which data can be transferred automatically using only one processor cycle per 16-bit word. At lower rates the channel also frees processor time to allow execution of a program concurrently with data transfers for a device.

In addition to the straightforward transfer of a word between memory and a device in either direction, the channel also allows a device to increment by one a word already in memory. The direct memory channel is used by devices requiring very high data transfer rates, such as magnetic tape or disk and by devices that utilize the memory increment feature, such as the real-time clock.

The program cannot affect the direct memory channel because there are no instructions for this purpose; instead the program sets up the device to use the channel. When the device requires data service, it requests direct memory access. At the beginning of every cycle the processor synchronizes any requests that are then being made. As soon as the processor completes an instruction in the program or a cycle of two instructions in an external instruction sequence, it takes care of all requests that have been synchronized or are synchronized while it is handling transfers. If several devices are waiting for service simultaneously, the first to receive service is the device that is physically closest to the processor on the bus. After taking care of all direct memory requests, the processor returns to an external instruction sequence if one is in progress; starts an interrupt if a device is waiting for one; or executes the instruction pointed to by the Sequence Counter.

The DMA cycle is actually a stolen cycle from the operating program. Each DMA cycle occurs between instructions (at the completion of the current instruction) and has a higher priority than an interrupt cycle or an external instruction cycle.

4.3.6.1 Timing – The time a device must wait for access depends on when its request is made and how many devices of higher priority are also requesting access. After the processor starts handling requests, a given device must wait until all devices closer than that device on the bus have been serviced: 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 wait no longer than the time required for the processor to finish the instruction that is being performed when the request is synchronized. Maximum waiting time including synchronization is, therefore, approximately 9 μs for the first word in a block. After that word, every memory cycle can be taken, and the full 568,000 words per second attained.

4.3.7 External Instructions

Operations in some functional devices are limited to single data transfers or state changes; thus, these operations take place entirely within the instructions that cause them. But, in many cases, an FO instruction for a device can trigger an operational sequence that delays execution of the stored program until it is finished. This extended sequence actually a macro-function (eg. multiply, divide, etc.) and the entire sequence is considered as one stored program function.

The extended instruction takes control of the processor to execute a sequence of external instructions which are retrieved from its own built in ROM and set to the IR.

In either case, the sequence is normally started by an FO instruction that addresses the device as destination. A sequence of external instructions takes control of the processor in order to use other devices, such as the arithmetic operator, or at least the data transfer paths. Such a sequence can be regarded as an extension of the FO that triggered it, for SC remains constant, and program interrupts are shut out until the sequence is finished. During EI execution, the processor can pause for direct memory access. Thus, high-speed I/O operations are not endangered except to the extent that a program interrupt may be delayed.

4.3.8 Input/Output (I/O)

With direct function processing, an I/O instruction is simply one that addresses an I/O device, i.e., a peripheral device. A table in Appendix B lists all devices for which device addresses have been assigned, as well as their mnemonics. In addition to the logic for decoding source and destination addresses, every device is assigned a Ready flag and an Interrupt Status bit. The first of these denotes the state of the device. At power turnon, all Output Ready flags are set, all Input Ready flags and Interrupt Status bits are clear. Placing a device in operation clears Ready. If the device is to be used for input, the program generally places it in operation by giving an FO instruction. A complex device to be used for output may require an FO, but a simple output device is usually started automatically by giving a data transmission instruction that sends a 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 bus system to the peripheral equipment; *input* refers to the transfer in the opposite direction.)

When the device has processed a unit of data, it sets Ready 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 responds with a transmission instruction to send more data; in the latter case, the program responds with a transmission instruction to bring in the data, followed by an FO to restart the device. If the program has set the Interrupt Status bit and turned the interrupt control on the setting of Ready notifies the program by causing an interrupt; if Interrupt Status is clear, then the program must keep testing Ready to determine when the device is available.

A device may require no data transfers, such as a real-time clock that uses only an FO to turn it on and off. All of the simpler data handling devices have only one buffer (e.g., to hold a single character for the Teletype; tape reader and tape punch, or to receive incremental plotting data for a single point in the plotter). A high-speed device, such as magnetic tape or disk, may use data transmission instructions only for control information with data moving between the device and memory via direct memory access. Control information that the program must supply to a tape system includes a transport address and an actual command the tape device is to perform; input information includes error flags and transport status levels.

4.3.9 Program Control

The present section discusses the use of the various instruction types to control the program sequence and the basic state of the computer. The use of the jump instructions for handling subroutines is described in Section 4.3.3 A jump always causes the next instruction to be taken from the address loaded into SC, but this case is not always true when a data transmission instruction loads SC. The data transfer always occurs in the final cycle of the instruction, and SC is incremented in the first and second cycles. Hence, if the instruction takes only one or two cycles (i.e., an RR or MRI), SC is incremented after it is loaded. In general, this condition simplifies the return because the address saved in TRP must be incremented in order to point to the correct return location. Thus, the instruction RR TRP, SC causes a correct return (refer to Section 4.3.3.4). This incrementing can also be handled by using a deferred jump, but the program must increment in the bus modifier when using a data transmission instruction of three or four cycles to return with an address originally saved in TRP. Refer to the GRI-99 *Assembler Manual* for further information.

Sending a word to a device that cannot receive information or that does not exist is a no-op, which is effectively a program delay. The basic no-op (NOP) is a one-cycle null transfer. The length of the delay is equal to the number of cycles the instruction takes. Provided no modification is called for, many transmission instructions have no effect on the computer at all except for the regular SC incrementing to go to the next instruction; but the programmer must remember that deferred addressing does affect some memory location.

Certain flags and control flip-flops in the computer are connected to the source and destination buses in such a way that their states can be saved and then restored as though they constituted a register. These elements are referred to collectively as the Machine Status Register, which can be addressed as device address 17, mnemonic MSR. The elements that make up this register are the following:

BOV	LNK					AO STATE								SOV	AOV
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

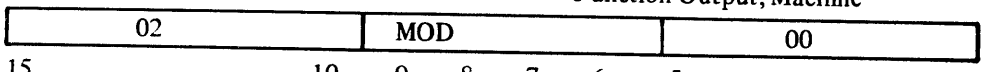
Saving and restoring the machine state is a procedure used primarily in program interrupts, but the programmer can use it anytime; he can even set up the machine state in any way he likes by loading a word of his own construction into MSR. (AOV and SOV, however, are states that can only be affected by the arithmetic operator and the contents of AX and AY.) Having the Bus Overflow (BOV) and Arithmetic Overflow (AOV) flags at the ends of the register is especially convenient because either of them can be moved to the link in only one cycle without affecting the machine state. This is done by

```
RR      MSR, L1, 0      (BOV)
          or
RR      MSR, R1, 0      (AOV)
```

respectively.

BOV is the sign bit of the machine status register and therefore may also be tested by a jump conditional instructional (see section 4.33). There are also Function Generating and Function Testing instructions for the processor control logic.

FOM — Function Output, Machine

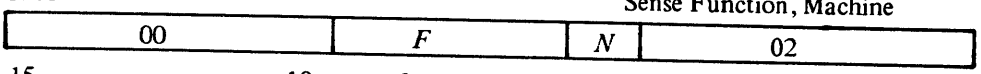


Perform the functions specified by 1s in MOD as follows:

Bit	Mnemonic	Function
6	CLL	Clear Link.
7	STL	Set Link.
8	HLT	Halt the processor.

Programming 1s in bits 6 and 7 (mnemonic CML) complements the link.

SFM — Sense Function, Machine



Perform a function test on the conditions selected by 1s in *F* as follows:

Bit	Mnemonic	Condition
7	BOV	Bus Overflow set.
8	LNK	Link set.
9	POK	Power ok.

The letter *N* in bit 6 specifies logical negation of test results. The following instruction pair is used to determine whether location *A* contains all 1s:

```
MR      A, P1, 0      ; Add 1, throw away
                      ; result
SFM     BOV           ; Skip if overflow
                      ; occurred
```

Suppose bit 0 of location *A* is to be used as a program flag. Bit 0 is set with this instruction:

```
ZM      P1, A
```

and tested by giving this sequence:

```
MR      A, R1, 0      ; Put bit 0 in link
SFM     LNK           ; Skip if Link set
```

4.4 USING THE AO

The Arithmetic Operator (AO) contains two registers, AX and AY, both of which can be addressed as source and destination for data. At all times, the AO is in some specific functional state such that the AO output, which is addressable as a data source, is the given function of the contents of the two registers. For example, turning on system power or starting the processor from the console places AO in the ADD state, making the output continuously equal to the sum of AX

and AY. Changing the contents of either register changes the output to a new sum. When the AO is in a given state, it retains that state until changed by the program or by the operator pressing the START switch.

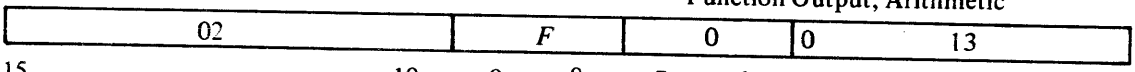
The AO output is actually 17 bits, wherein the extra bit is a carry (AOV, bit 0 of MSR), or equivalently an extra magnitude bit in a sum. This extra bit is the overflow of the *unsigned* addition of AX and AY, regardless of the functional state of AO (even if the low-order 16 bits of the AO output are a logical function). The overflow value can be determined only by Function Testing or reading machine status.

The circumstances that generate an AOV are obvious when dealing with unsigned numbers. An addition with result greater than $2^{16}-1$ overflows. For subtraction, the condition is the same in terms of adding the two's complement; in terms of the original operands the subtraction $A - B$, which is executed by adding A and the two's complement of B, produces a carry if $A \geqslant B$ (unless both A and B are zero).

The statement of the overflow conditions for signed arithmetic is as follows: if two number of the *same sign* are added and the answer is of the opposite sign, an overflow (Sum Overflow – SOV bit 1 of MSR) has occurred. It should be noted that AOV and SOV are *not* the same thing.

The device addresses of AX and AY are 11 and 12, mnemonics AX and AY respectively. Device address 13, mnemonic AO, addresses the arithmetic operator, both for its output and for Function Generating and Testing. The functional state of the AO is available to the program as bits 9 and 8 of the Machine Status Register (in the same configuration as given by the following FO instruction).

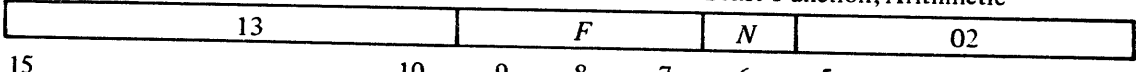
FOA — Function Output, Arithmetic



This instruction sets AO to the state specified by *F*, as follows:

Bits 9-8	Mnemonic	Function
0	ADD	Addition
1	AND	AND
2	XOR	Exclusive OR
3	OR	OR

SFA — Sense Function, Arithmetic



This instruction performs a function test (as previously described) on the carry if bit 7 in *F* is 1. The mnemonic AOV places a 1 in bit 7. *N* specifies logical negation of test results. The instruction performs a Function Test on the signed arithmetic overflow if bit 8 is F is 1. The mnemonic SOV places a 1 in bit 8.

One way to determine whether the contents of AX and AY are identical is as follows:

```
FOA     XOR         ; Exclusive or
JC      AO, ETZ, YES ; Jump to YES if
                     ; AX = AY
```

The following computes the number ten times that contained in location *Z* (Assume *Z* now has a number less than 2^{16} divided by 10):

```
FOA     ADD         ; Add
FOM     CLL         ; Clear Link
MR      Z, L1, AX   ; AX = 2Z
RR      AX, L1, AY  ; AY = 4Z
RR      AO, AX      ; AX = 6Z, AO = 10Z
```

NOTE

The first two instructions above could be replaced by the single instruction ZR MSR which clears the LNK, sets AO to the ADD state, but also clears BOV.

Suppose we wish to use the word in location *Z* with its bytes swapped. Example 1 can be used for this purpose.

Example 1

```
        MR      Z, AX
        RR      AX, AY          ; Put word in AX and AY
        MRI     -10, TRP        ; Set up count for eight shifts
        RM      TRP, M2+1
M1:     RS      AY, L1          ; Shift AY into link
        RS      AX, L1          ; Shift link into AX
M2:     MSI     0, P1           ; Count step
        SFM     BOV             : Done?
        JU      M1              ; No shift again
        .                       ; Yes, AX has word with bytes swapped
        .
        .
```

Note that the example does not use the functional properties of AO; the shifting could just as well be done in a pair of general-purpose registers, or in a pair of core locations if AO were not available (the latter would be longer in both space and time). The count could be kept in the XR instead of in core; this alternative eliminates the memory reference in the sixth line (inside the loop) and eliminates the fourth line altogether, because the third line could read:

```
MRI     - 10, XR
```

4.5 OTHER ARITHMETIC OPERATIONS

Examples of using fixed point arithmetic for addition, subtraction, shifting, multiplication, division, and scaled arithmetic operations, are contained in the GRI-99 *Fixed Point Manual.* The GRI-99 *Floating Point Manual* contains valuable programming information for performing operations using floating point arithmetic.

Table 4-5

GRI-99 INSTRUCTION SUMMARY

Class	SDA	MOD Bits 9	8	7	6	DDA	Effect
Function Generation (control signals)	02					00	
		0	0	0	1		CLL (Clear Link)
		0	0	1	0		STL (Set Link)
		0	1	0	0		HLT
		0	0	1	1		CML (Complement Link)
	02					13	
		0	0	0	0		ADD
		0	1	0	0		AND
		1	0	0	0		XOR
		1	1	0	0		OR
	02					14	
		0	0	0	1		Multiply
		0	0	1	0		Divide
		0	0	1	1		Arith. Rt. Shift
		0	1	0	0		Normalize
	02					04	
		0	0	0	1		ICF (Int. Control Off)
		0	0	1	0		ICO (Int. Control On)
	02					76, 77	
		0	0	0	1		STRT
		1	0	0	0		CLIF (Clear IRDY)
		0	0	1	0		CLOF (Clear ORDY)
	02	–	–	–	–	Any	Depends on DDA.
Function Test (Skip)	00	0	0	0	1	02	Always Skip
		0	0	1	–		BOV (Bus Overflow)
		0	1	0	–		LNK
		1	0	0	–		POK (Power OK)
	13					02	
		0	0	1	–		AOV (Arith. Overflow)
		0	1	0	–		SOV (Sum Overflow)
	76,77					02	
		1	0	0	–		IRDY
		0	0	1	–		ORDY
	Any					02	
		–	–	–	–		Depends on Device
		–	–	–	0		Skip if Any Tests True
		–	–	–	1		Skip if None Test True
Data Test* (Jump)	Any Except 06					03	
		0	0	1	–		Always Jump
		0	1	0	–		ETZ
		0	1	1	–		NEZ
		1	0	0	–		LTZ
		1	0	1	–		GEZ
		1	1	0	–		LEZ
		1	1	1	–		GTZ
		–	–	–	1		Deferred
Data Transmission Non-Memory Reference	Any Except 06					Any Except 06	
		0	0	0	0		None
		0	1	–	0		Increment
		1	0	–	0		Left One Bit
		1	1	–	0		Right One Bit
		–	–	1	0		1's Compl. Before Bits 8, 9

Table 4-5 (Cont.)

GRI-99 INSTRUCTION SUMMARY

Class	SDA	MOD Bits 9 8 7 6	DDA	Effect
Memory Reference*	06		Any	
	Any		06	
	06		06	
		Same 0 0		Direct
		as 1 0		Immediate
		Non-Memory 0 1		Deferred
		Reference 1 1		Immediate-Deferred

*These instructions can be used with indexing; refer to the appropriate description in this chapter.

NOTE: Data cannot be complemented when transferred into or out of memory.

Table 4-6

GRI-99 INSTRUCTION TIMES

Class	Length (words)	Memory Cycles	Time (μs)
Function Generation	1	1	1.76
Function Test			
Skip	1	1	1.76
No Skip	1	1	1.76
Data Test			
No jump	2	1	1.76
Jump direct	2	2	3.52
Jump deferred*	2	3	5.28
Data Transmission			
Non-Memory reference	1	1	1.76
Memory reference			
direct	2	3	5.28
immediate	2	2	3.52
deferred*	2	4	7.04
immediate – deferred*	2	3	5.28

*The deferred mode selects one level of indirect addressing with auto-indexing; the indirect address is incremented prior to instruction execution.

NOTE: Indexing requires no additional execution time.

Table 4-7

GRI-99 INSTRUCTION EXAMPLES

Class	Sample Machine Instruction	Description
Function Generate	02 0001 77	Start Teletype paper-tape input
	02 1100 13	Select Arithmetic Operator OR function.
Function Test	77 0011 02	Skip if Teletype output not ready.
	13 0010 02	Skip if arithmetic operation caused overflow (AOV)
Data Test	77 0100 03 jump address DONE	If Teletype input equal to zero go to DONE
	13 1110 03 jump address ALARM	If arithmetic result greater than zero, go to ALARM
Data Transmission		
Non-memory Reference	35 0100 35	Increment the general purpose register 35.
	77 0000 76	Transmit Teletype input character to the high-speed punch.
	65 0110 12	Transmit the two's complement of the converter register to the AY register (for a comparison with a limit value in the AX register, for example).
Memory Reference	06 0000 11 address UPLIM	Transmit upper limit value to AX register.
	03 0010 06 destination	Store trap register immediate.
	06 0100 06 address COUNT	Increment value of counter.
	06 0010 35 operand 12	Transmit an immediate constant (12) to the general purpose register 35.

CHAPTER 5
GRI-99
INSTALLATION

5.1 INSTALLATION CONSIDERATIONS

Figure 5-1 shows the physical layout of the GRI-99 chassis. The chassis is designed to be easily mounted in a standard EIA 19-in. mounting rack and can be either bolted into the rack or mounted on slides. An expansion chassis is available for mounting above or below the standard unit.

5.1.1 Access Points

The hinged console front panel swings aside to permit quick access to the front of the chassis. Viewed from the front, the power supply is located on the left, the memories are on the right; seven slots are located near the center for large-size PC cards. From the back of the chassis, the nine slots for small-size PC cards are easily accessible.

5.1.2 Temperature

It is recommended that the ambient temperature at the installation site be maintained between 20°C and 30°C (68°F to 86°F). The ambient temperature in the vicinity of the chassis can vary between 0°C and 55°C without adverse effect. Relative humidity should be less than 90% (non-condensing). All exposed surfaces of the GRI-99 have been treated to withstand corrosion; however, exposure to extreme humidity for extended periods of time should be avoided. When mounting the processor in a typical rack mounting configuration with other equipment, the ultimate criterion for proper cooling over the *system's* specified temperature is the maximum temperature of the core stack itself. This temperature must not excel 60°C (140°F) measured on the backside of the core stack board via a thermocouple or temperature tell-tale. In large system configurations with other significant heat loads present, the system designer must consider the use of forced air cooling for the entire system because temperature rises of 40°C are quite possible. Proper attention must be given to the actual flow of air to make sure that fresh air is introduced in the rack, moved freely through the computer and other equipment to be cooled, and expelled from the rack.

5.1.3 Power

The GRI-99 uses 85 – 144 Vac or 160 – 276 Vac, single-phase, 45 Hz – 65 Hz line power. The GRI-99 power cable is equipped with a standard 3-wire plug. The receptacle must be properly grounded and rated at 15A. The line current and power dissipation specification for the computer and associated Teletype are as follows:

	Processor	Teletype
Line Current (115 Vac)	3A	2A nominal 7A turn-on surge
Dissipation	150W – 300W	92W

The +5 Vdc output of the power supply can deliver 20A, all of which is used by the processor with 16K of memory and other optional devices. An expansion module added inside the basic power supply provides power for the I/O expansion chassis.

5.1.4 Specifications

A complete specification for the GRI-99 is included in Chapter 1 of this manual. Power supply specifications are in Chapter 3. The physical dimensions of the GRI-99 and a Teletype Model 33 ASR are as follows:

	Height (in.)	Width (in.)	Depth (in.)	Weight (in.)
Main Chassis	10 1/2	19	22	40
ASR 33	45	22	19	56

5.2 SPECIAL PROCEDURES

No special installation procedures are necessary for the GRI-99. The GRI-99 is built to be rack mounted (see Figure 5-2). For rack mounting, it is important to allow at least 3 in. of space below the computer and 3 in. above the computer for convection cooling. A complete description of installation procedures for the GRI-99 and references for Teletype installation are included in the *GRI-99 Maintenance Manual Volume I.*

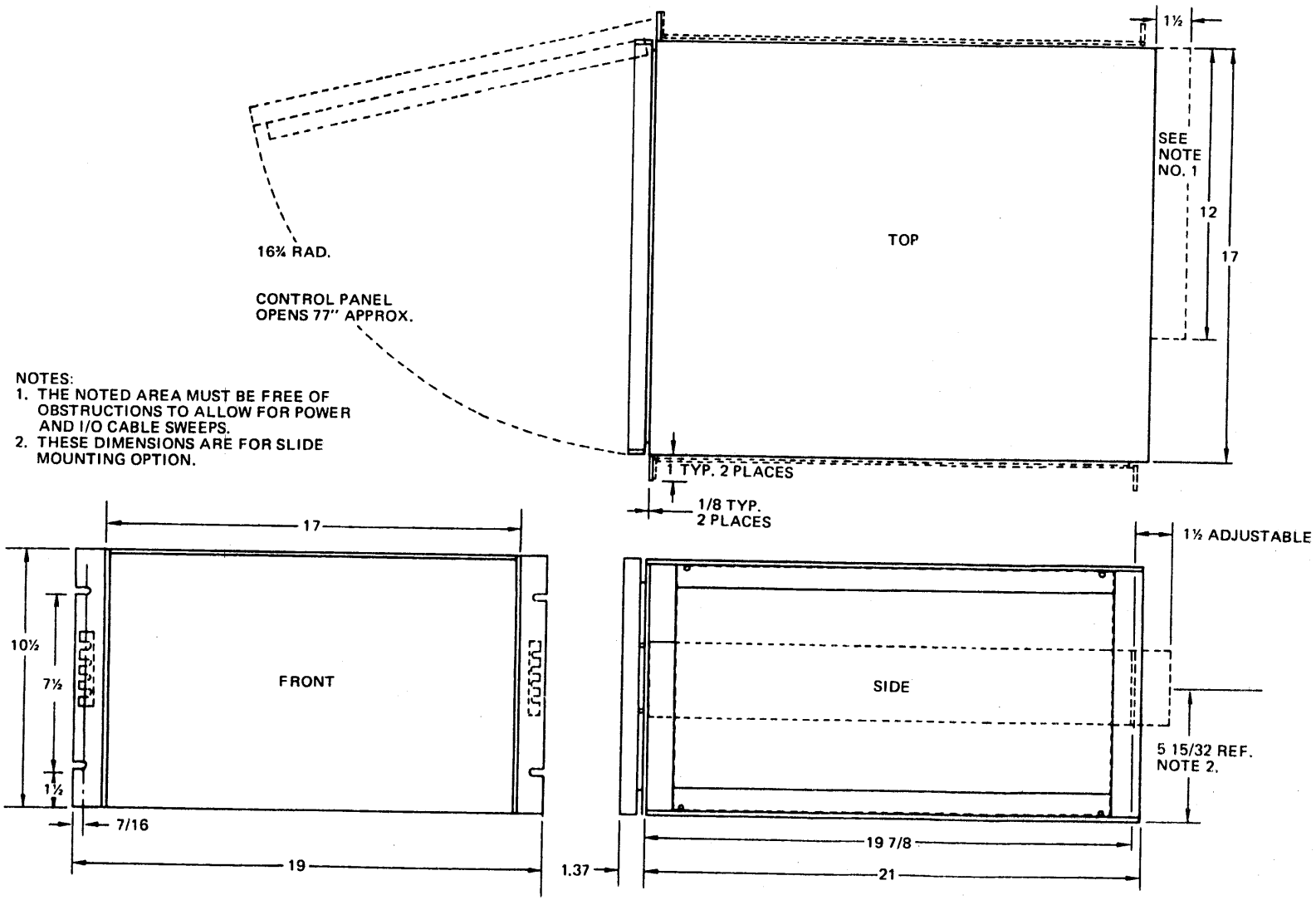


Figure 5-1 GRI-99 Dimensional Drawing

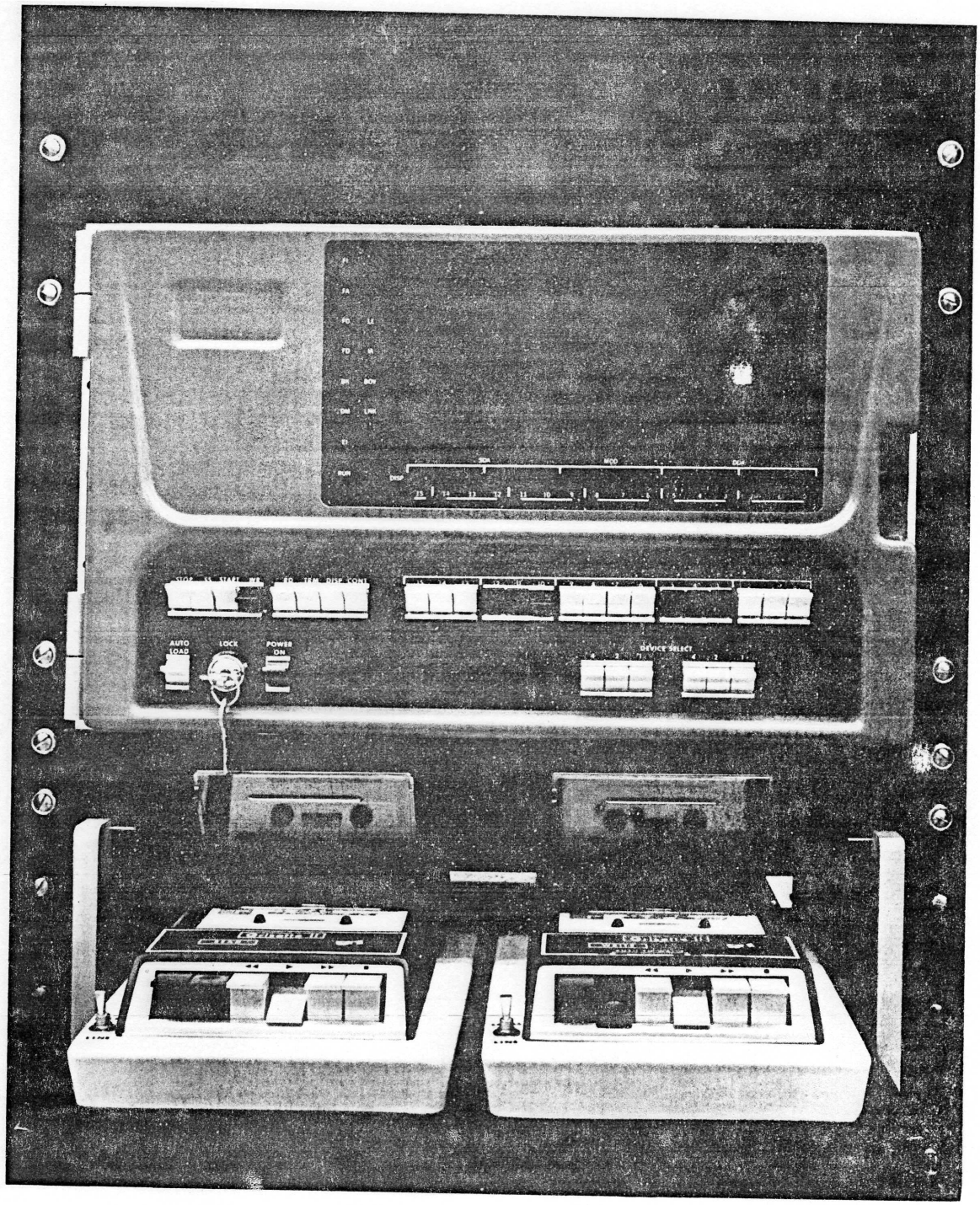


Figure 5-2 Rack Mounted GRI-99 shown with Grisette II full duplex tape I/O system

CHAPTER 6
GRI-99
INTERFACING

A wide variety of GRI devices, as well as user-designed devices, can be easily interfaced to the GRI-99. The addition of new source registers makes data available to all destination devices, and new destination devices can receive data from any source device in the system. Device control is provided by Function Generation instructions to operate the device and Function Test instructions to test the state of the device. Chapter 4 fully describes all GRI-99 instructions to control both internal and external devices. It is important for the designer to understand the architecture of the GRI-99, as well as the busing scheme and PC card setup; Chapter 3 of this manual describes these considerations, and the GRI-99 *Maintenance Manual* contains valuable theory of operation information.

6.1 BUSING SCHEME

The busing scheme and connections internal to the main chassis are shown in Figure 6-1. All PC cards shown in the lower row are of the larger size (9-in. by 13-in.) The processor is contained on four large cards, labeled PC1, PC2, PC3, and A0. Core memory cards are the largest cards used and contain the core planes, and all read, write, inhibit, sense and control circuitry. Two connectors are available in the processor bus for the addition of firmware operators.

Both the processor bus and the I/O bus are actually each two buses: source and destination. In general, the destination bus is associated with output from system devices, whereas the source bus is the source of data and control information for input to system devices.

All connections between the buses and the console are made with ribbon cable running between the processor bus PC and the console PC. An expansion chassis can be placed either above or below the main unit. Ribbon cable connects the mainframe buses to the expansion bus where the bus signals are buffered.

6.2 BUS SCHEMATIC

Figure 6-2 is a bus schematic of the basic GRI-99 configuration. Except for most of the power wiring, all system signals are distributed via the printed bus board. Some of the signals and signal groups shown in the schematic are private communications signals between the three major control assemblies (PC1, PC2, and PC3) that make up the basic processor. The power supply delivers voltages and a power status signal to the bus via a connector that goes to the main processor bus. From PC3 through the last slot on the main processor bus, the signals become the system signals that are distributed in parallel through the entire system, including the I/O bus section in the rear of the machine. Note that the majority of signals that arrive at the back panel bus of the processor are still identical to the processor signals used internally and, therefore, facilitate the use of internal options plugged into these slots, as well as I/O options.

All connections to the main processor bus or the I/O bus assembly are made by 44-pin connectors. Signal Origin drawings are connector lists that describe all the bus signals that appear on the connectors at the various slots in the main processor. These drawings, located in a separate volume titled *GRI-99 Engineering Drawings*, in addition to showing the connector pin number and the signal name, also contain:

a. A brief description of the signal functions.

b. Whether the signal is originated on the board described on the drawing.

c. Whether the signal is simply used on that board, or whether the signal contains bus drivers that are ORed onto one of the common buses that runs through the system.

In general, all signals with the suffix L are bus type signals that are driven from many points throughout the system by open collector gates. Exceptions to this rule can be found by observing whether the signal is marked in the Origin column or the Used column. A bus-type signal has more than one origin. If the signal is a bus-type and the Used column is marked, the notation indicates a point of termination of the signal.

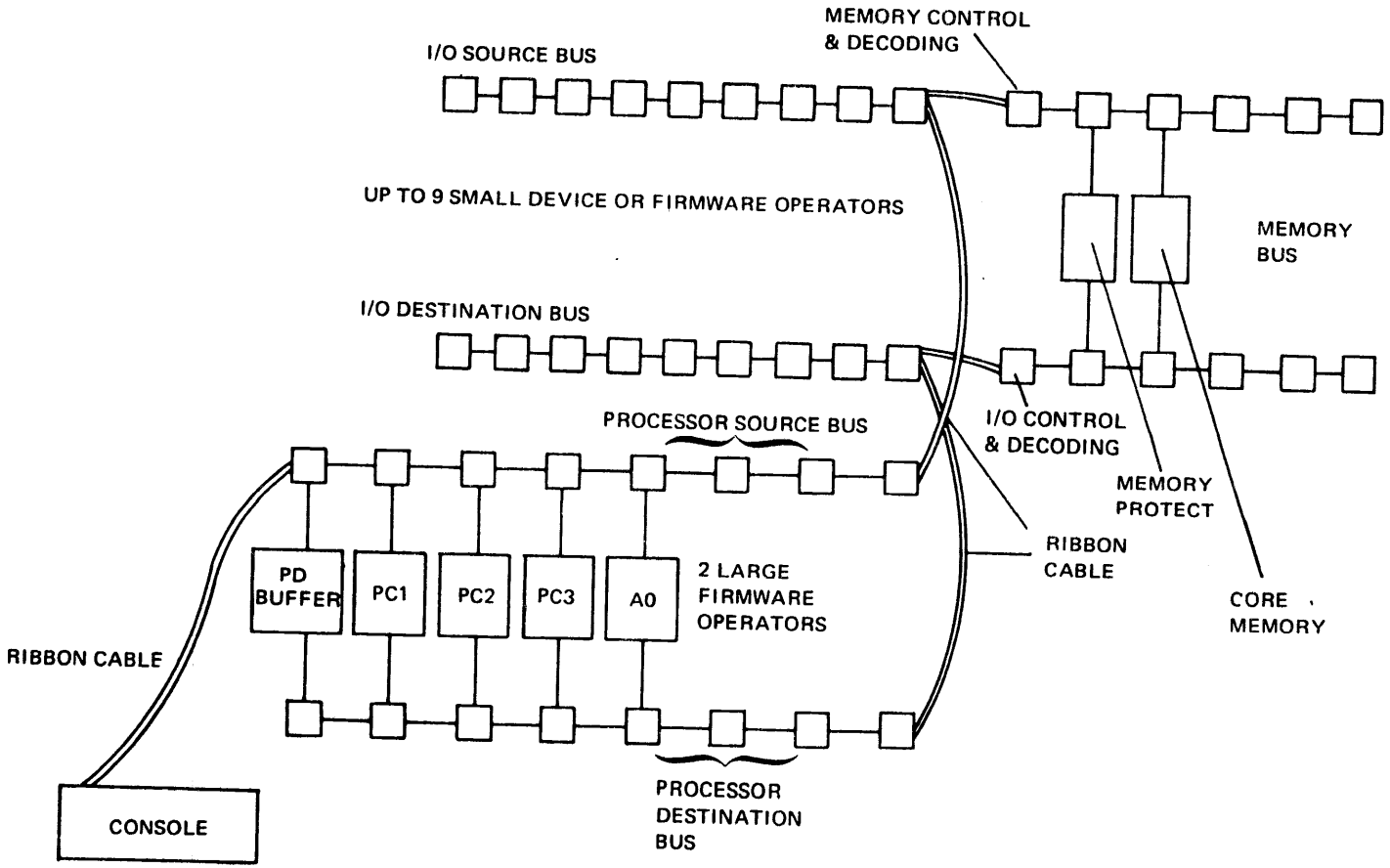
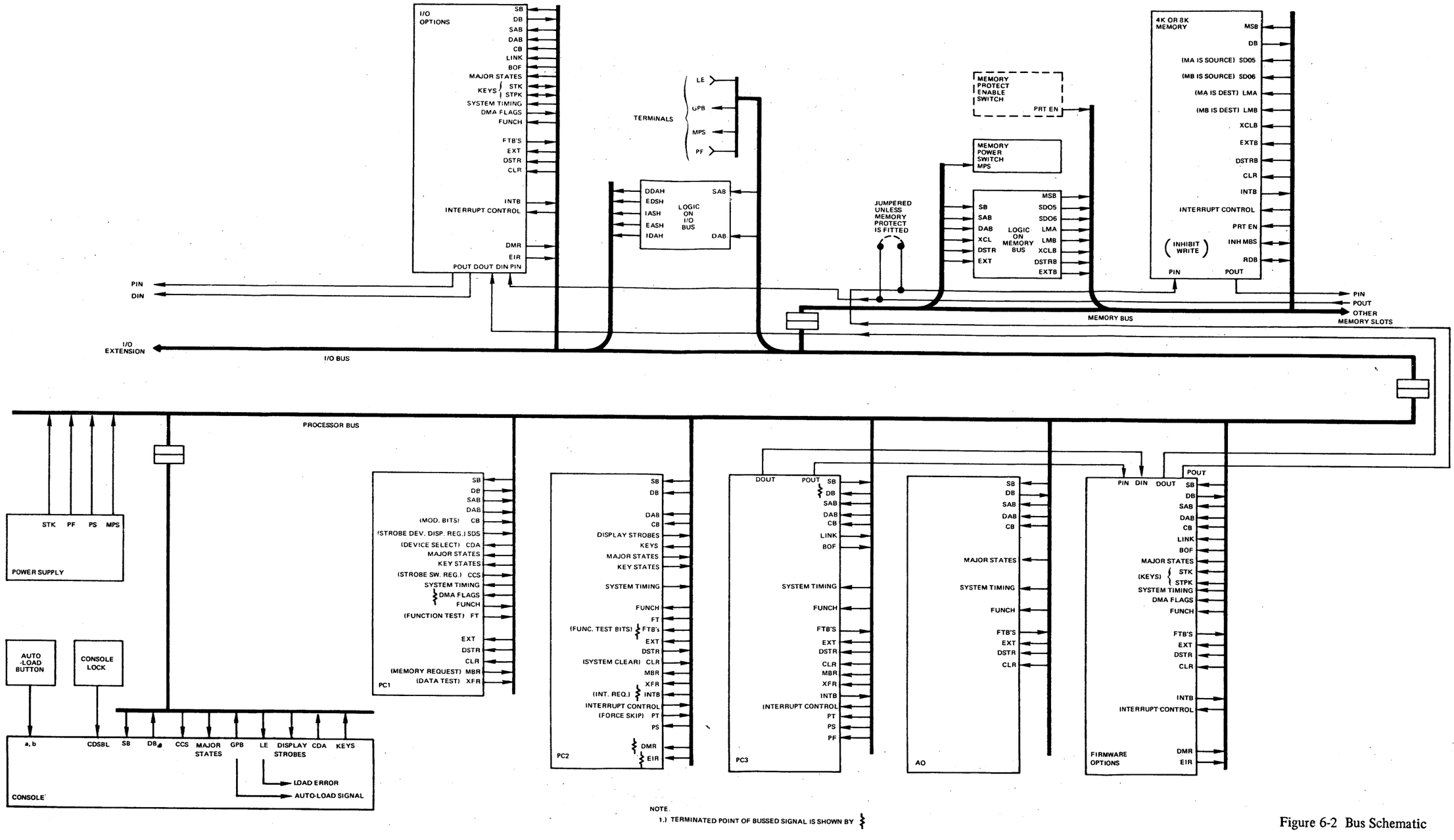


Figure 6-1 System Busing



NOTE.
1.) TERMINATED POINT OF BUSSED SIGNAL IS SHOWN BY ≷

Figure 6-2 Bus Schematic

6.3 SOURCE AND DESTINATION BUS SIGNALS

The paths of the Source and Destination Bus signals are indicated in the block diagram of the GRI-99 architecture. This treatment excludes communication paths between PC1, PC2, PC3 and the power supply. The signals discussed in this section are found on the Source Bus connectors and Destination Bus connectors in the two large card-option slots on the processor bus and the 9 small card-option slots on the I/O bus.

Figures 6-3 and 6-4 are diagrams of the two connectors, showing pin numbers and signals as they would appear looking into the processor at the bus board. Those signals marked by an asterisk (*) appear only on the I/O bus connectors.* A short explanation of the signals is given in Table 6-1.

6.4 PROCESSOR TIMING

The basic clock frequency in the GRI-99 is provided by a 9.09 MHz crystal clock (Y1). This clock produces a signal (XCLL) that is distributed throughout the system on source bus pin L. The origin of the clock is on PC2. The basic clock period is 110 ns. This clock frequency is divided by four to produce the 440 ns microcycle timing (T0, T1, T2, T3). A timing chart demonstrating this relationship is shown in Figure 6-5. The four microcycles within the basic memory cycle are shown with a 110 ns strobe occurring during the last 110 ns of each microcycle, except T0 and T1, which occur 110 ns earlier. All clocking of data transfers is done with these strobe pulses. The balance of the time period between the trailing edge of one strobe and the leading edge of the next is used for propagation and settling time through gates.

---

*These special signals on the I/O bus are decoded combinations of SAB and DAB addresses commonly used in all I/O devices. This feature was provided to minimize the requirements for logic on the small I/O cards.

SOURCE BUS CONNECTOR

Ground	1	A	Ground
+5V	2	B	+5V
CLRH	3	C	EIH
DMH	4	D	BKH
PINL	5	E	POUTL
DINL	6	F	DOUTL
DAB1H	7	H	DAB0H
DAB3H	8	J	DAB2H
DAB5H	9	K	DAB4H
DSTRH	10	L	XCLL
CLIBH	11	M	INTBL
IMBL	12	N	DIRBL
STPKL	13	P	STKL
EIRL	14	R	DMRL
SB01H	15	S	SB00H
SB03H	16	T	SB02H
SB05H	17	U	SB04H
SB07H	18	V	SB06H
SB09H	19	W	SB08H
SB11H	20	X	SB10H
SB13H	21	Y	SB12H
SB15H	22	Z	SB14H

Figure 6-3 Source Bus Connector

DESTINATION BUS CONNECTOR

DB01L	1	A	DB00L
DB03L	2	B	DB02L
DB05L	3	C	DB04L
DB07L	4	D	DB06L
DB09L	5	E	DB08L
DB11L	6	F	DB10L
DB13L	7	H	DB12L
DB15L	8	J	DB14L
SAB0H	9	K	ISYNH
SAB1H	10	L	FTB1L
SAB2H	11	M	FTB2L
SAB3H	12	N	FTB3L
SAB4H	13	P	LINKH
SAB5H	14	R	B0H
IDAH*	15	S	P2H
EASH*	16	T	ISAH*
EDDH*	17	U	EDSH*
EXTH	18	V	FUNCH
CB3H	19	W	CB2H
CB1H	20	X	CB0H
-A	21	Y	+A
Ground	22	Z	Ground

* APPEAR ONLY ON I/O BUS CONNECTORS

Figure 6-4 Destination Bus Connector

Table 6-1

SOURCE AND DESTINATION BUS CONNECTOR SIGNALS

Signal	Direction	Description
Source Bus		
EIH	Out	External Instruction: the processor external instruction cycle.
CLRH	Out	Clear: system clear level generated by the START key or power on or off.
BKH	Out	Break: the processor interrupt cycle.
DMH	Out	Direct Memory: the processor direct memory access (DMA) cycle.
POUTL	Out	Priority Out: the serial interrupt priority determining level from a higher priority device to the next lower priority device.
PINL	Out	Priority In: the serial interrupt priority determining level into a lower priority device from the next higher priority device.
DOUTL	Out	DMA Out: the serial DMA priority-determining level out of a device.
DINL	Out	DMA In: the serial DMA priority-determining level into a device.
DAB0H-DAB5H	Out	Destination Address Bus: the 6-bit address of the destination device.
XCLL	Out	Crystal Clock: a square wave with a 110 ns period.
DSTRH	Out	Data Strobe: used by a device to clock in data from the Source Bus.
INTBL	In	Interrupt Bus: a common line for all devices to request an interrupt.
CLIBH	Out	Clear Interrupt Bus: clears the Interrupt service flip-flop in the requesting device.
DIRBL	In	Direction Bus: Sent by DMA device to indicate data transfer direction for DMA (0V = in, +4V = out)
IMBL	In	Increment Memory Bus: sent by a DMA device to increment the memory location during a DMA cycle.
STKL	In	Start Key: START request; wire ORed between front panel, power supply and any I/O device.
STPKL	In	Stop Key: STOP request, wire ORed between front panel and any I/O device.
DMRL	In	Direct memory Access Request: a common line for all devices to request DMA.
EIRL	In	External Instruction Request: a common line for all devices to request an external instruction cycle.
SB00H-SB15H	Out	Source Bus Data: the 16 data lines in the Source Bus.
Destination Bus		
DB00L-DB15L	In	Destination Bus: the 16 data lines in the Destination Bus.
ISYNH	Out	Interrupt Sync: generated in each cycle to synchronize interrupt and DMA requests by devices.
SAB0H-SBB511	Out	Source Address Bus: the 6-bit address of the source device.
FTBIL-FTB3L	In	Function Test Bus: devices use these lines for status input during an SF instruction.
LINKH	Out	Link: the Link bit associated with the Bus Modifier.

Table 6-1 (Cont.)

SOURCE AND DESTINATION BUS CONNECTOR SIGNALS

Signal	Direction	Description
BOH	Out	Bus Overflow: the bus overflow flag associated with the Bus Modifier.
P2H	Out	Normally Time 2 Pulse: strobe that gates FO commands into a device; however, during EI cycle, P2H is generated twice (at T1 and T3) due to doubled execution speed in the EI state.
IDAH*	Out	Interrupt Destination Address: the destination address is 04, the interrupt status register.
EASH*	Out	External Address, Source: the source address is 16, the active DMA address register or interrupt address generator.
ISAH*	Out	Interrupt Source Address; the source address is 04, the interrupt status register.
EDSH*	Out	External Data, Source: the source address is 15, the active DMA data register or logic that supplies an external instruction from a ROM.
EDDH*	Out	External Data, Destination: the destination address is 15, the active DMA data register.
FUNCH	Out	Function: the current processor instruction is an SF or an FO.
EXTH	Out	Execute Time: Normally T2, during which the programmed transfer occurs; however, during EI cycle, EXTH is generated twice (at T1 and T3) due to doubled execution speed in the EI state.
CB0H-CB3H	Out	Control Bus: lines that transmit the four MOD bits.
+A	Out	Voltages generated by optional power supply card on the I/O bus. When this card is installed, −A from the power supply is disconnected at a jumper on the bus. These voltages are nominally ±28V.
−A	Out	

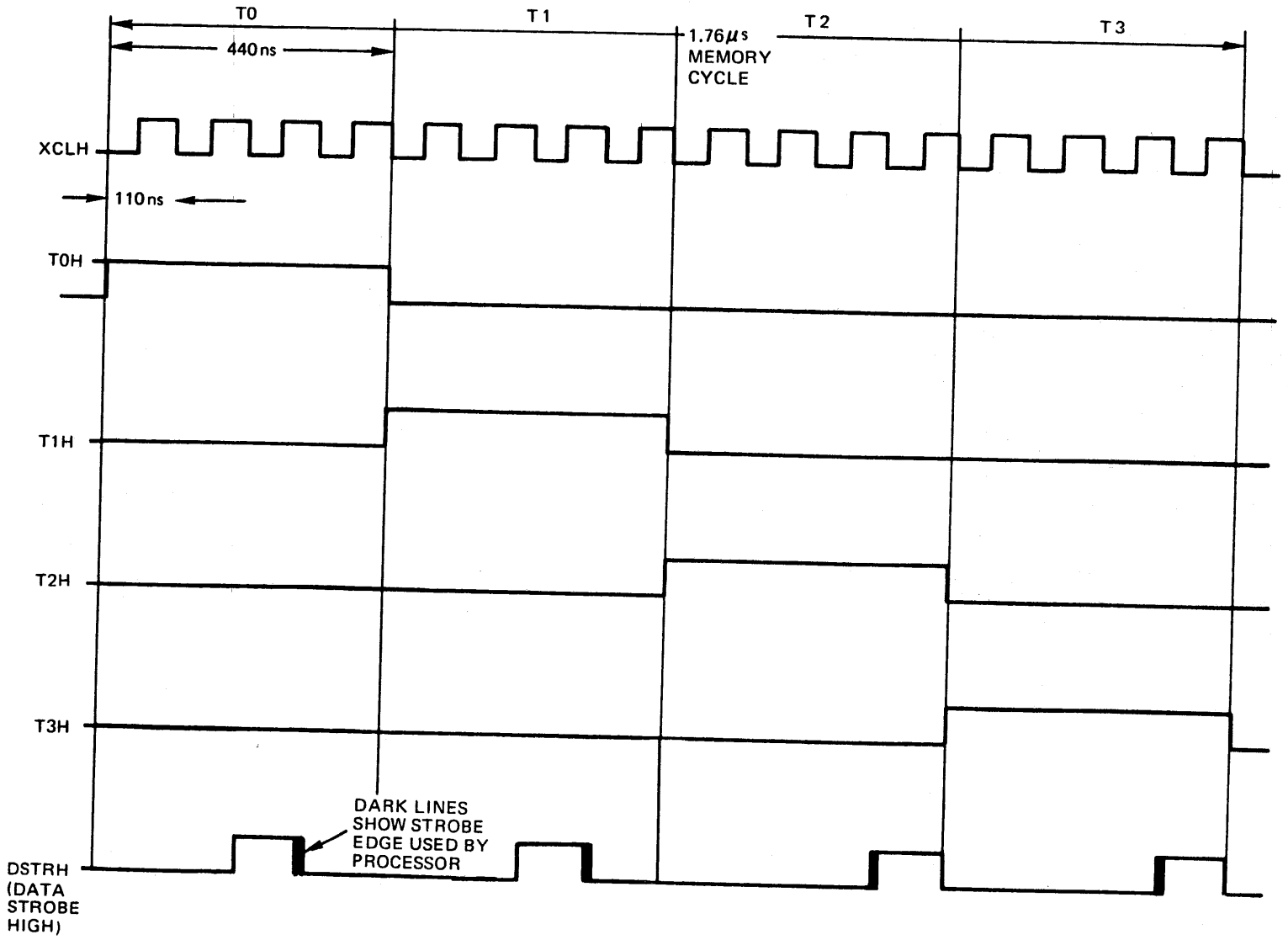


Figure 6-5 Nominal Processor Timing

6.5 INSTRUCTION EXECUTION

Chapter 4 discussed the instructions used by programmers to perform a task within a program. Those instructions are referred to as *macroinstructions.* In the GRI-99, it is necessary to execute a group of *microinstructions* to complete a macroinstruction. The microinstructions are in the same format as macroinstructions; the only difference between the two instructions is the source of the microinstruction and the speed at which it is executed.

The GRI-99 macroinstruction set includes the instructions that require 1, 2, 3, or 4 major states to complete. In addition, the interrupt, direct memory access, and external instruction states are three more major states. When power is applied to the processor, it is in the FI state. During each state, one memory cycle is executed. Each state is given a two-letter designator, as follows:

FI	Instruction cycle 1
FA	Instruction cycle 2
FO	Instruction cycle 3
FD	Instruction cycle 4
BK	Break cycle
DM	Direct memory access cycle
EI	External instruction cycle

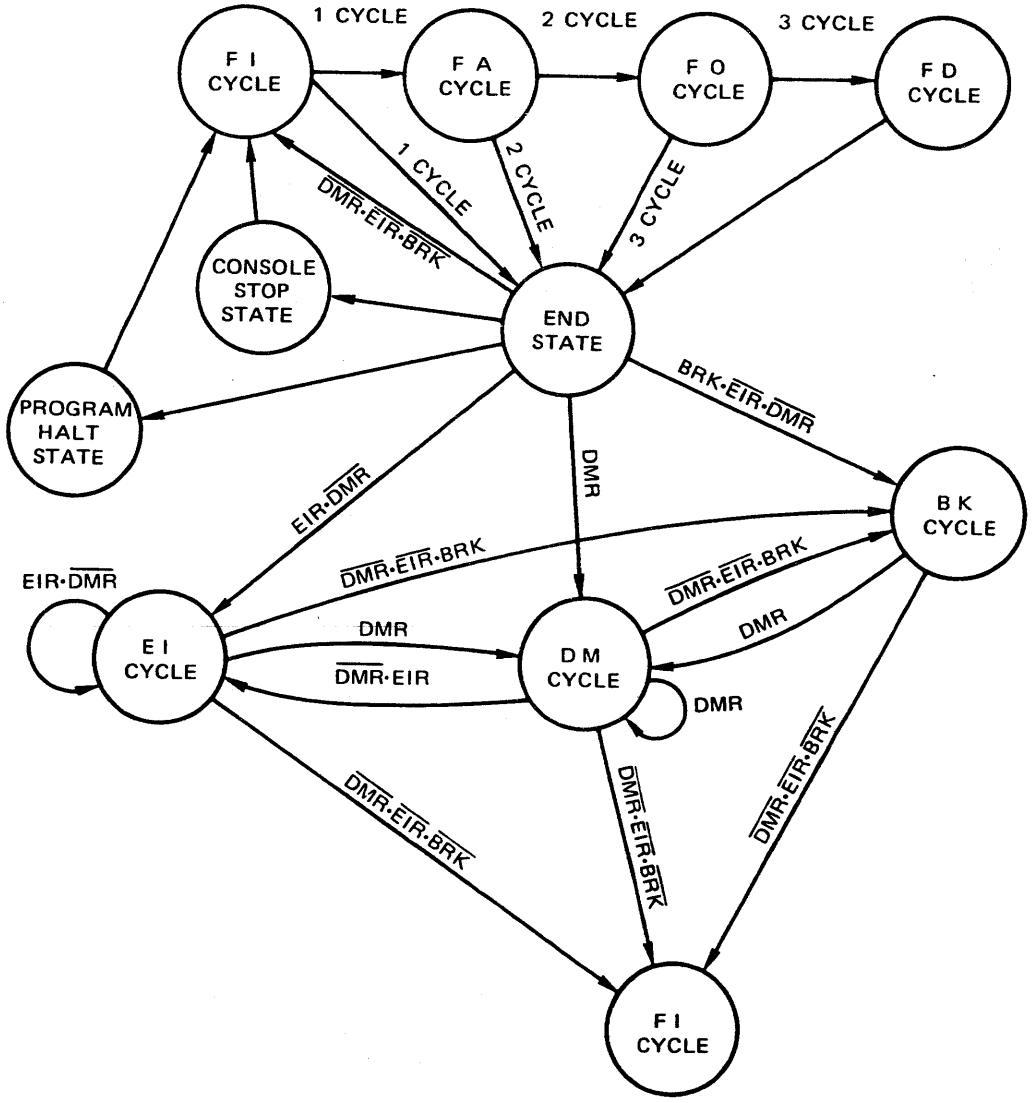
Figure 6-6 is a state flow diagram showing priorities and all possible paths between machine cycles.

The mnemonics INTB, EIR, and DMR are used to represent the request signals for the Break, External Instruction, and Direct Memory Access cycles respectively. The End State, Halt State and Console Stop State are not separate machine states but merely conditions that exist within the processor between machine states. Note that these states represent stop states for the machine, and no new memory cycle is initiated.

During each memory cycle, up to four microinstructions can be performed using either the Instruction Register or the processor's ROM as a microinstruction source. The technique of transferring 16 bits of data from a source to a destination was discussed in detail in Chapter 4.

For a single cycle instruction, at the start of the memory cycle (T0), the instruction word is in memory and the control logic has no indication of what the macroinstruction will be. A data transfer of SC to the memory address (MA) registers is made to read the macroinstruction from memory. During the next period (T1), the instruction word read from memory into the MB is transferred through the bus modifier to the instruction register (IR). At the end of T1, the control logic receives the macroinstruction the programmer wishes to perform (e.g., a data transfer from SDA to DDA). During T2, the transfer is performed. To prepare for the next instruction, the SC must be incremented. During T3, the SC is transferred to itself and incremented as it passes through the bus modifier.

Every macroinstruction is implemented in this manner with 1, 2, 3, or 4 memory cycles and various microinstruction sequences. These sequences are stored in a read only memory (ROM), which is part of the basic machine control logic. The arrangement shown in Figure 6-7 is used to derive microinstructions from either the IR or the ROM. In the sequence previously discussed, during T0, T1, and T3, microinstructions were taken from the ROM. During T2, the IR supplied the instruction to be executed.



PRIORITIES

1. DM DIRECT MEMORY ACCESS
2. EI EXTERNAL INSTRUCTION
3. BK INTERRUPT (BREAK)
4. MACHINE CYCLES (FI, FA, FO, FD)

NOTE:

FI CYCLE IS DRAWN TWICE FOR SIMPLICITY.

Figure 6-6 State Flow Diagram

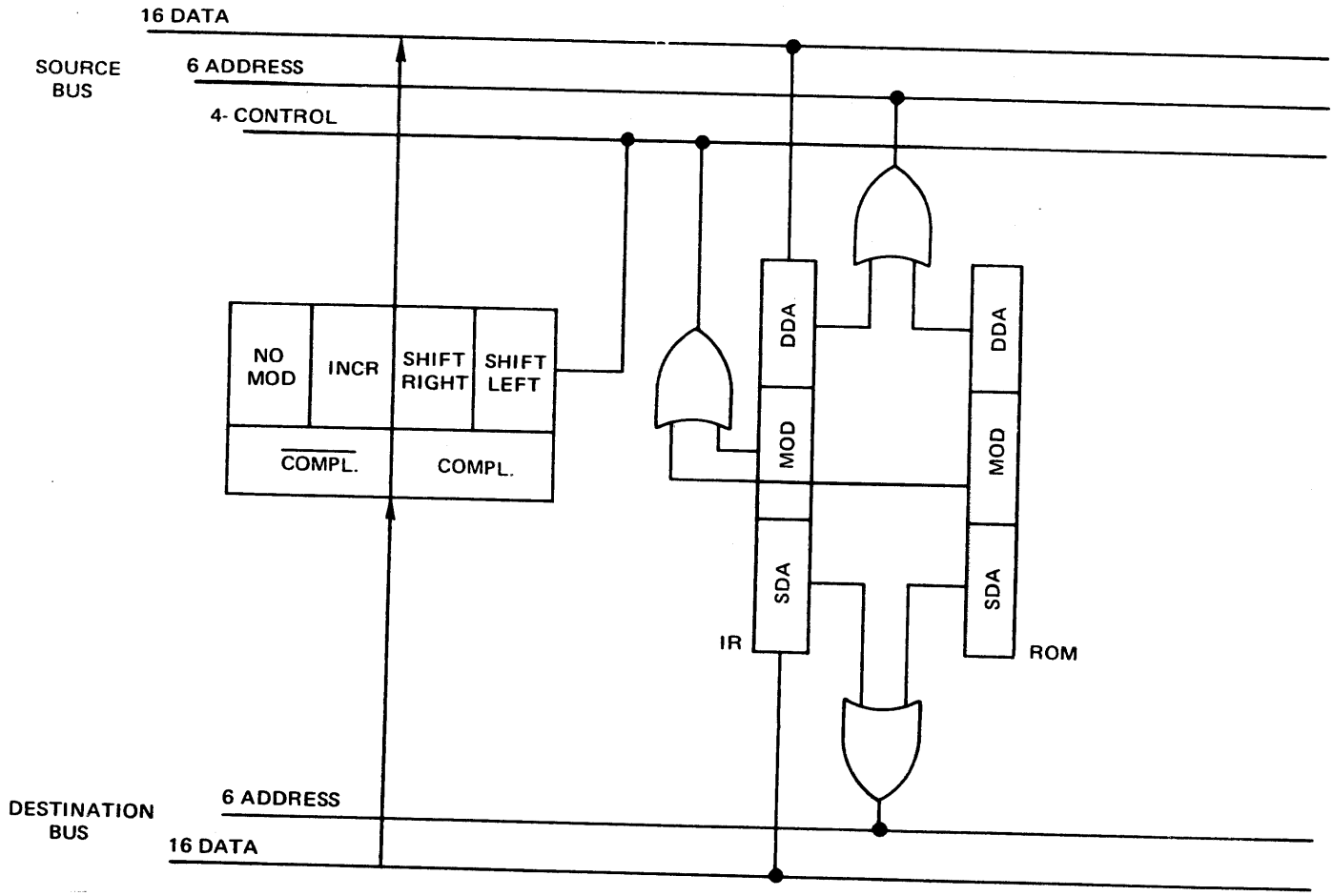


Figure 6-7 Derivation of Addresses and Control Signals from Either the IR or ROM

NOTE

When data are being manipulated in the control registers of the machine (such as the trap or SC), it is quite important to know the event sequence of microinstructions relative to macroinstruction execution.

Figure 6-8 describes all single-cycle instructions. Figure 6-9 describes the Data Test class of instructions, and Figure 6-10 covers memory reference data transmission. In each case, the hexagonal boxes indicate a transfer of information to or from memory. Tables 6-2 through 6-9 are provided to supplement the figures with exact sequence information.

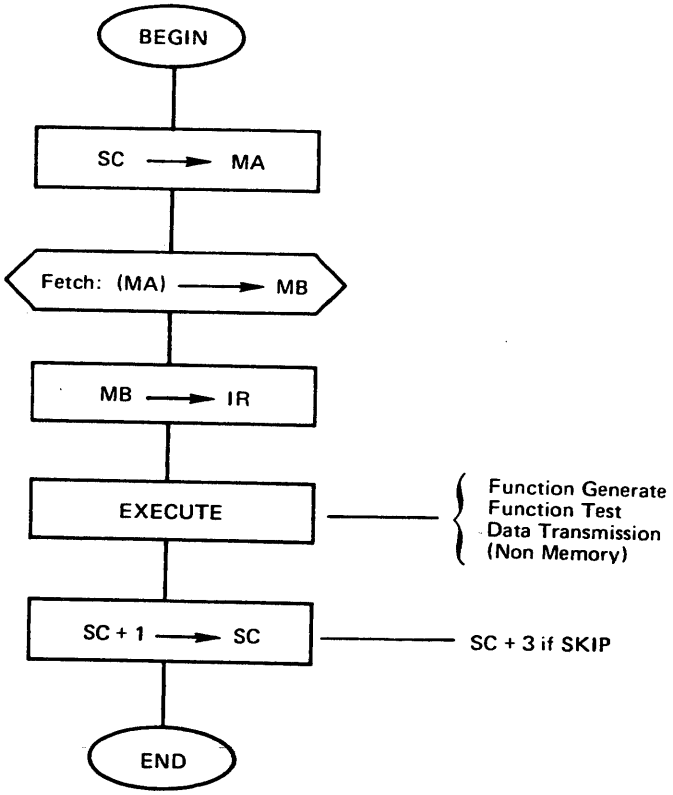


Figure 6-8 Event Sequence for Single-Cycle Instructions

Table 6-2

FUNCTION GENERATE

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begins memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Macroinstruction now executed as a microinstruction.
	T3	SC + 1 to SC	The SC now points to next instruction in memory.

Table 6-3

FUNCTION TEST (SKIP)

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begins memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Macroinstruction executed as micro.
	T3	SC + 1 to SC SC + 3 to SC	Test condition not true. Test condition true.

Table 6-4

NON-MEMORY REFERENCE DATA TRANSMISSION

Major State	Time Slot	Macroinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begins memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Macroinstruction now executed as a microinstruction.
	T3	SC + 1 to SC	The SC now points to next instruction in memory.

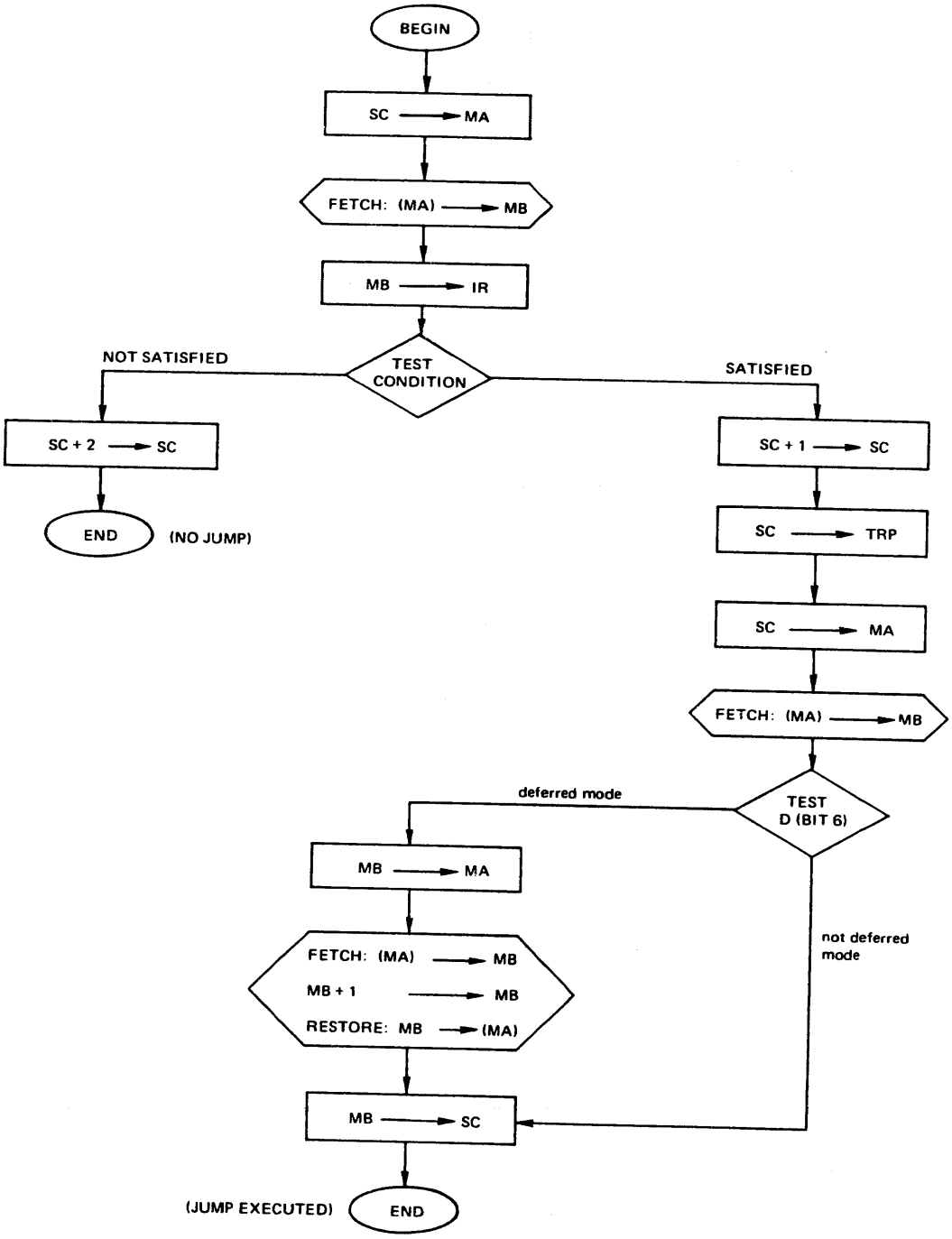


Figure 6-9 Event Sequence for Data Test Instructions

Table 6-5

DATA TEST (JUMP)

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begin memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	Execute (IR)	Perform test.
	T3	SC + 1 to SC	Test condition true; SC now points to jump address; another memory reference required.
		SC + 2 to SC	Test condition not true; SC now points to next instruction.
FA Fetch Address	T0	SC to MA	Begin memory reference.
	T1	SC to TRP	SC now in trap register.
	T2	No op	No Operation.
	T3	MB to SC*	Address is now in SC. Non-deferred jump complete.
FO Fetch Operand (Deferred only)	T0	MB to MA	Begin memory reference.
	T1	No op	No operation.
	T2	MB + 1 to MB	Increment the address.
	T3	MB to SC	Incremented address now in SC.

*If jump is deferred, this operation is not executed because SC is changed during the next major state.

Table 6-6

MEMORY REFERENCE DATA TRANSMISSION (DIRECT MODE)

Major State	Time Slot	Microinstruction	Comments
FI	T0	SC to MA	Begin memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to address.
FA	T0	SC to MA	Begin memory reference.
	T1	No op	No operation.
	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to next instruction.
FO	T0	MB to MA	MA points to referenced location.
	T1	No op	No operation.
	T2	Execute (IR)	Execute macroinstruction as micro.
	T3	No op	No operation.

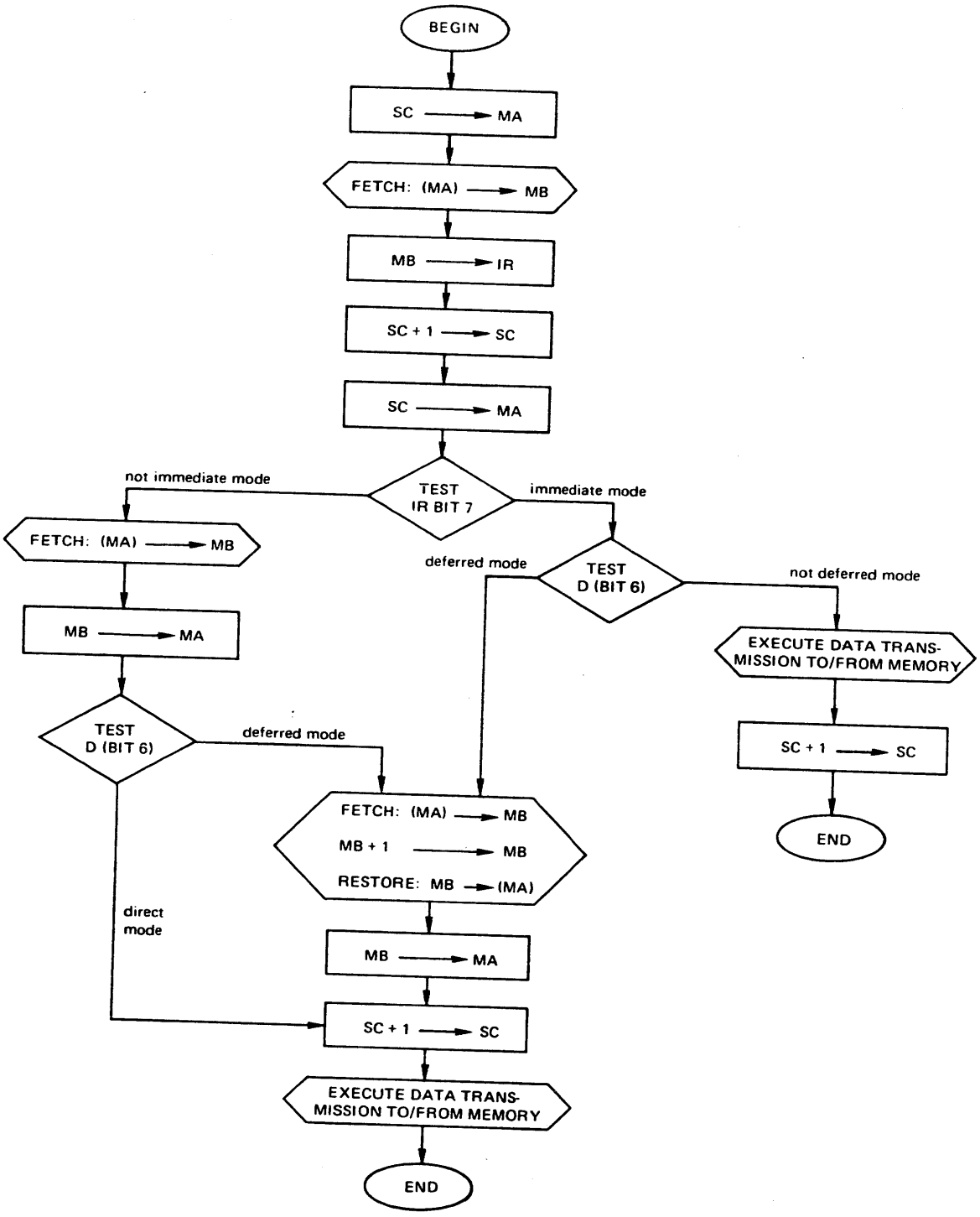


Figure 6-10 Event Sequence for Memory Reference Data Transmission Instructions

Table 6-7

MEMORY REFERENCE DATA TRANSMISSION
IMMEDIATE MODE

Major State	Time Slot	Microinstruction	Comments
Immediate Address			
	T0	SC to MA	Begin memory reference.
FI	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to address.
FA	T0	SC to MA	Begin memory reference.
	T1	No op	No operation.
	T2	Execute (IR)	Macroinstruction executed as microinstruction.
	T3	SC+1 to SC	SC now points to next instruction.

Table 6-8

MEMORY REFERENCE DATA TRANSMISSION
(IMMEDIATE DEFERRED)

Major State	Time Slot	Microinstruction	Comments
	T0	SC to MA	Begin memory reference.
FI	T1	MB to IR	Macroinstruction in IR and decoded.
Fetch Instruction	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to address.
	T0	SC to MA	Begin memory reference.
FA	T1	No op	No operation.
Fetch Address	T2	MB + 1 to MB	Increment address and store.
	T3	SC + 1 to SC	SC now points to next instruction.
	T0	MB to MA	MA now contains incremented address.
FO	T1	No op	No operation.
Fetch Operand	T2	Execute (IR)	Execute macroinstruction.
	T3	No op	No operation.

Table 6-9

MEMORY REFERENCE DATA TRANSMISSION (DEFERRED)

Major State	Time Slot	Microinstruction	Comments
FI Fetch Instruction	T0	SC to MA	Begin memory reference.
	T1	MB to IR	Macroinstruction in IR and decoded.
	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to address.
FA Fetch Address	T0	SC to MA	Begin memory reference.
	T1	No op	No operation.
	T2	No op	No operation.
	T3	SC + 1 to SC	SC now points to next instruction.
FO Fetch Operand	T0	MB to MA	Begin memory reference.
	T1	No op	No operation.
	T2	MB + 1 to MB	Increment address and store.
	T3	No op	No operation.
FD Fetch Deferred	T0	MB to MA	Begin memory reference.
	T1	No op	No operation.
	T2	Execute (IR)	Execute macroinstruction.
	T3	No op	No operation.

6.6 INTERFACES

There are two types of GRI-99 interfaces:

a. *Internal devices* that are simply extensions of the basic processor (e.g., the AO). These devices may be either 9 in. by 13 in. or 9 in. by 4 in. PC cards that plug directly into the processor bus or the I/O bus (see Figure 6-11).

b. *External device interfaces* that connect the bus system to an external device (e.g., an A/D converter). These device interfaces are in the form of 9 in. by 4 in. PC cards that plug into the I/O bus (see Figure 6-12).

Internal devices can be built on the PC card shown in Figure 6-12 without using the external device connectors; however, if the PC card shown in Figure 6-11 is used, one or more such devices can be built on a single card. The large-size card holds up to 108 ICs.

The external device interfaces can be built on the PC card shown in Figure 6-14. This card can contain 33 IC packages. Connections are provided for the Source and Destination I/O Buses and for an external device.

The external PC connector has 48 pins and mates to an Amphenol connector Type 583167-1. Contacts and keys for these connectors are purchased separately, and only those that are needed are used.

The internal device options are summarized in Chapter 2. Chapter 2 also contains brief descriptions of various external devices that can be interfaced to the GRI-99.

6.7 INTERFACE LOGIC AND TIMING

There are two types of I/O data transfer:

a. The transfer of words or characters by the program.

b. The automatic transfer of data by direct memory access (DMA).

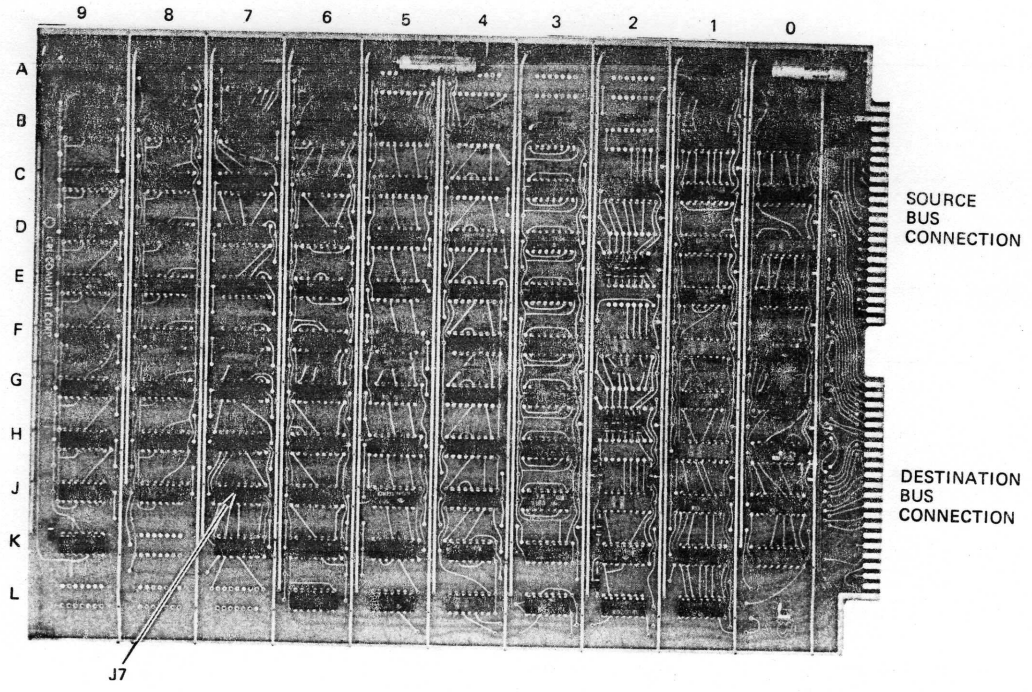


Figure 6-11 Large-Size Internal Device PC Card (Component Side)

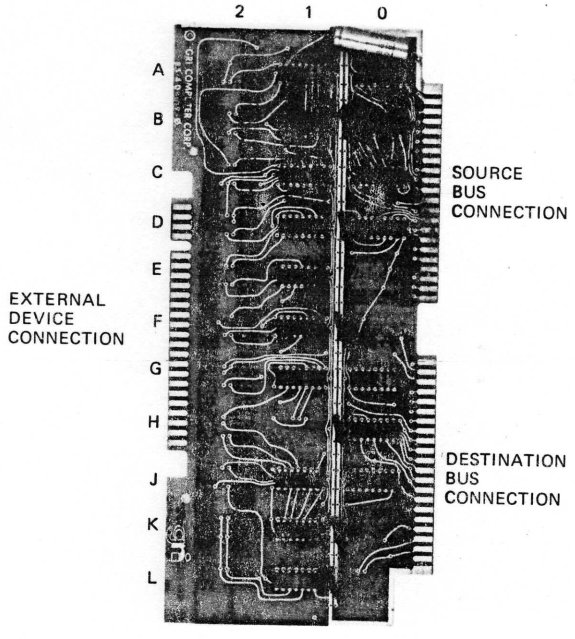


Figure 6-12 Small-Size External Device Interface PC Card (Component Side)

The program handles I/O by Sensing the Ready flag or by allowing the device to interrupt as it requires service. If the device operates automatically, it can use DMA for data transfer. In this case, the program responds only for control purposes (for example: block transfer complete or an error condition to which the program must respond). Also, internal devices (e.g., the AO) may be operated through the use of external instructions.

The two types of I/O transfer can be divided into six categories of functions that are used to operate equipment added to the GRI-99:

a. Programmed data transfers;
b. Function generation;
c. Function testing;
d. Interrupts;
e. DMA;
f. External instructions.

Typical circuit configurations and timing diagrams are provided for the various descriptions of these functions. The timing diagrams show the relationships of the signals involved in each case. In the timing diagrams, the lines for control signals represent the actual voltage levels. For groups of signals that carry binary information (such as data or addresses), a raised section of the line indicates the time during which the information is held on the bus. The individual timing diagrams should be used with the nominal processor and memory timing diagrams in the first part of this chapter.

NOTE

To properly interface any device to the GRI-99, it is very important to understand the instruction execution sequence. Refer to paragraph 6.4 and the accompanying flow charts and tables.

6.7.1 Programmed Data Transfers

Figure 6-13 shows the timing, and Figure 6-14 shows a typical logic setup for data transmission between an interface register and the bus system. If a register is to receive data it must be connected to the source bus data lines SBOOH to SB15H and input gating for the register must include a decoder for the destination address lines DABOH to DAB5H and the data transfer strobe DSTRH so that only this device responds when its device address is specified as the destination.

It is recommended that the data lines be connected to the D inputs of type 7474 flip-flops or registers. That is, a true edge triggered device is required for permitting the register to function with all instructions executed with it as a source and/or destination. These flip-flops also offer the greatest versatility in that they also have dc clear and set inputs for external data entry. The clock input to the data register is derived by combining the data strobe DSTRH with the output of the address decoder. As can be seen in the timing diagram, the strobe occurs at the end of the interval in which the destination address and source data are both valid. The address lines carry high levels; thus, 1s can be recognized by connecting the lines directly to the inputs of a decoder such as the 7430 gate. Lines for 0s must be connected through inverters. For example, to decode address 75, DAB1H is connected through an inverter, the remaining lines are connected directly. If a master clear signal is desired for the data register, the CLRH line is available; this line carries +5V during the power-up and power-down sequences and also every time a start signal is sent from the console or remotely. CLRH is normally low and must, therefore, be inverted to drive the direct clear inputs of the data register.

If a register is to supply data to the system, its outputs must be connected through open collector gates, such as the 7401, to the Destination Bus data lines DBOOL to DB15L. The gating input for the register output into the 7401s is derived by decoding the address that appears on source address lines SABOH to SAB5H. The decoding technique in a 7430 multiple-input gate is identical to that for the destination address. The timing of the transfer is the same as that for output. The SAB address is valid for the same 440 ns period that a valid address appears on the DAB lines. The 110 ns strobe at the input to the receiving register occurs at the end of this interval, allowing a settling time of 330 ns to transmit the data from the DB lines, through the bus modifier, onto the SB lines. Whatever logic is added between the DAB decoder and the open collector gates that connect to the DB lines must not have a delay time greater than 60 ns.

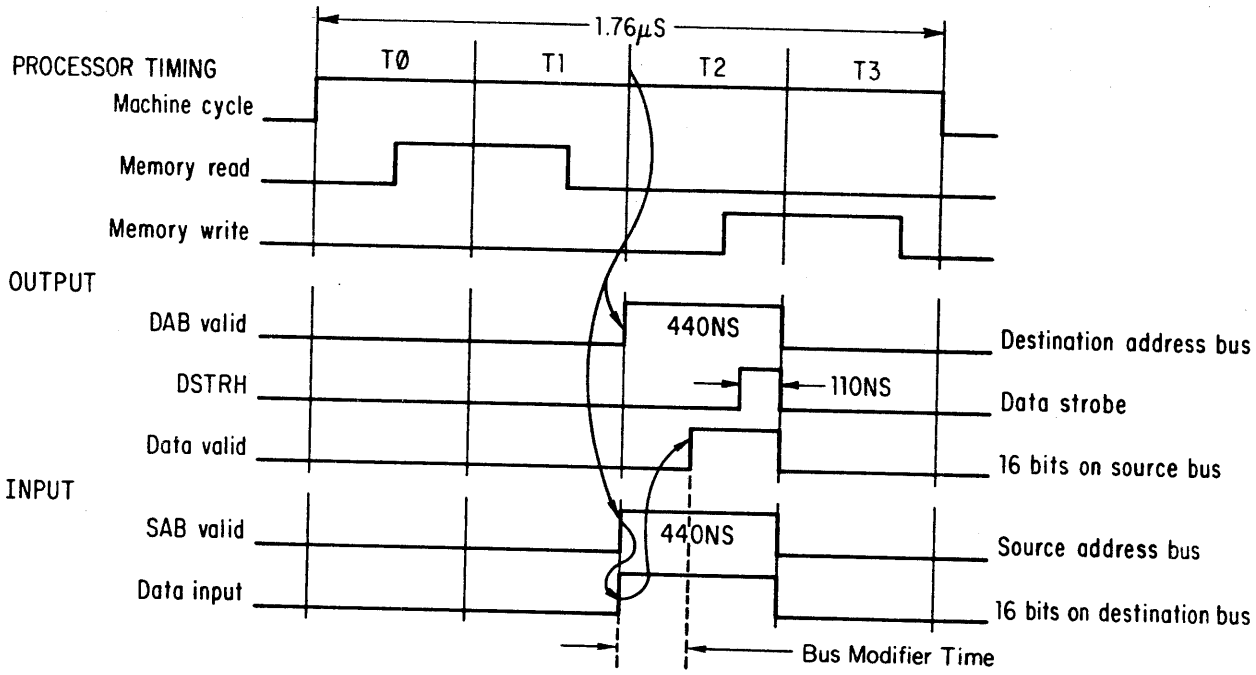


Figure 6-13 Programmed Data Transfer, Timing

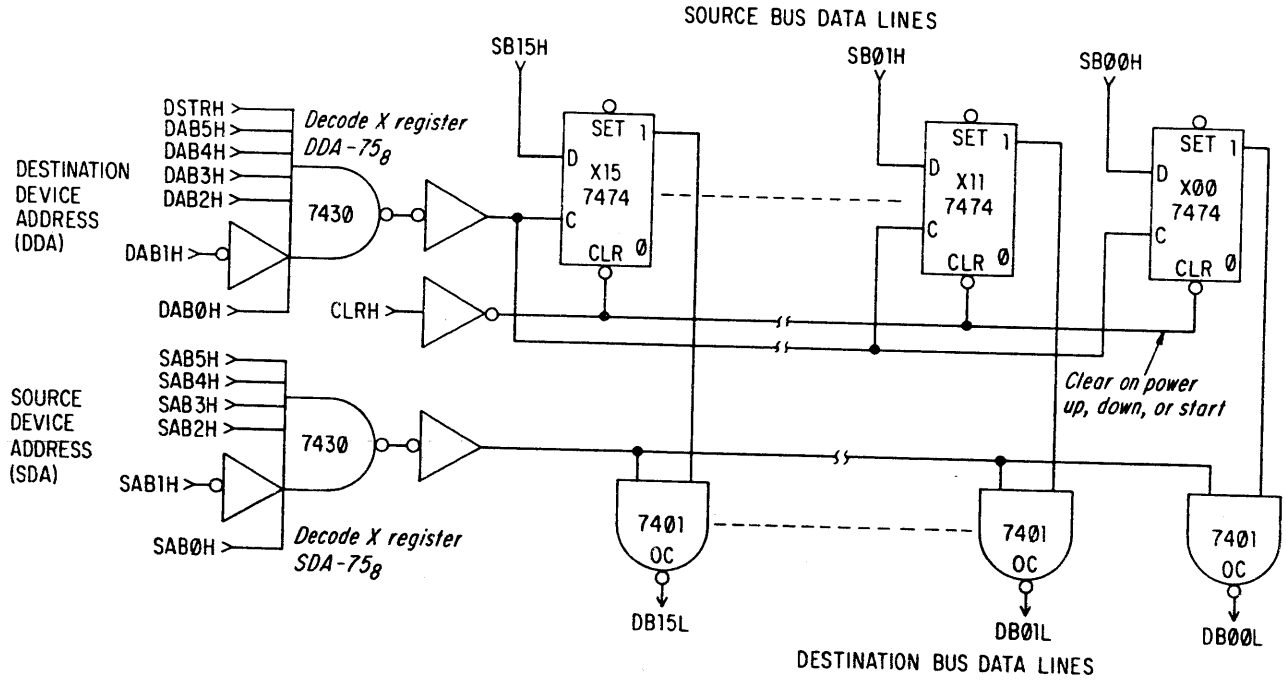
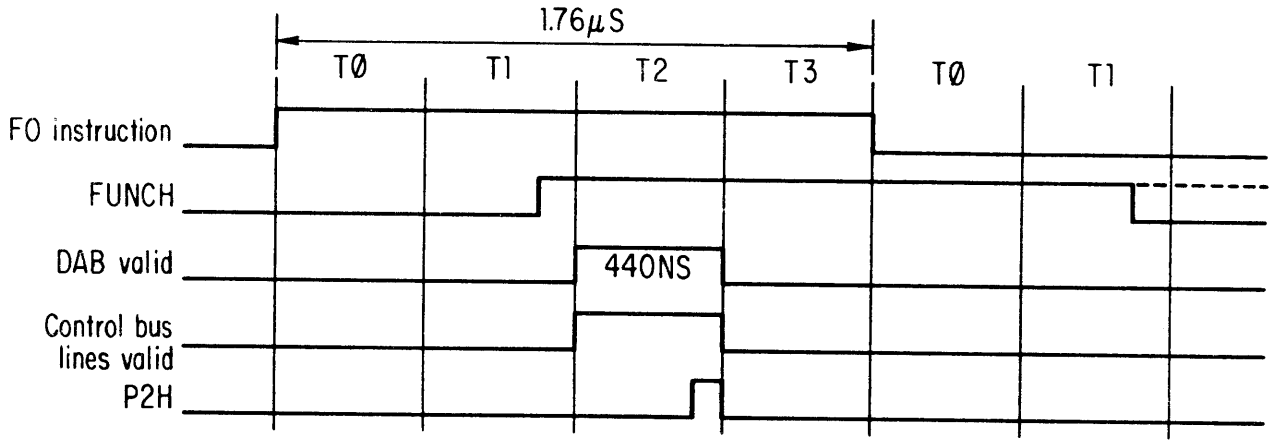


Figure 6-14 Programmed Data Transfers, Logic Diagram

The DDAH and SDAH signals produced by decoding the addresses may also be used by the logic for FO or SF instructions, rather than separately decoding the same addresses for use in testing, setting or clearing flags, or controlling an I/O device.

6.7.2 Function Generation

A function Generation (FO) instruction delivers up to four coded or individually usable pulses to devices for setting, clearing, or complementing flags. These pulses are placed on the control bus lines CBOH to CB3H during T2 of the FO instruction (see Figure 6-15). These lines are strobed by the combination of FUNCH, a signal present during any SF or FO instruction; DDAH, the decoded destination address; and P2H, a strobe pulse occurring at the end of T2.



NOTE: AT THE SECOND T1 TIME, FUNCH MAY REMAIN HIGH IF ANOTHER FO OR SF IS ISSUED.

Figure 6-15 Function Generation, Timing

The examples in Figure 6-16 show two uses for the CB signals. The type of connection at A is for clearing, setting and Complementing a flag and is used with an edge triggered JK type flip-flop (such as SN74110 or SN74111). This permits use of control bits 0 and 1 together to provide a microprogrammed complement of the flip-flop. Example B uses a D-type flip-flop (7474) where the data input is connected to the CBOH line and the clock is provided by the gating of FUNCH and P2H with DDAH. This arrangement permits the transfer of the current state of the CB line (0 or 1) into the flip-flop, and it can be used to transfer up to four coded bits of data into a small function register for multiplexing or selecting up to 16 functions.

By convention, if a device must be placed in operation by an FO instruction, CBOH is used for this purpose because it is already defined in the assembler with the mnemonic STRT. This situation applies to both internal and external devices, but a simple output device may be placed in operation simply by sending data to it. The READY flags in a device should be cleared by either CB3H or CB1H, again because the assembler already carries the definitions of CLIF for CB3H and CLOF for CB2H, Input Ready should be cleared by CB3H and Output Ready by CB1H.

6.7.3 Function Testing

Almost all devices contain flags, relay contacts, or status levels that must be sensed by the program using Function Testing (SF) instructions. The timing for Function Testing is shown in Figure 6-17. The conditions to be sensed are connected to the function test bus lines, FTBL, FTB2L and FTB3L, through open collector gates type 7401 or equivalent (Figure 6-18). The gating function FUNCH may optionally be combined with SDAH and P2H if a pulse is required during the sensing function. Otherwise, SDAH is sufficient since the processor will only examine the signals on the FTB buses during a function test operation.

At the end of the interval defined by SDAH, the processor strobes the function test lines to compare the information on them with the test specification given in the SF instruction. If the test result is positive, the processor increments SC by 3 in the final time time interval of the cycle; otherwise it increments SC by 1.

By convention, the Ready flag in a device is connected to FTB1L or FTB3L with Input Ready connected to FTB3L and Output Ready to FTB1L. The assembler again recognizes the standard mnemonics IRDY for FTB3 tests and ORDY for FTB1 tests. The other line is assigned at the user's discretion to test conditions such as tight tape, low paper, power on, etc.

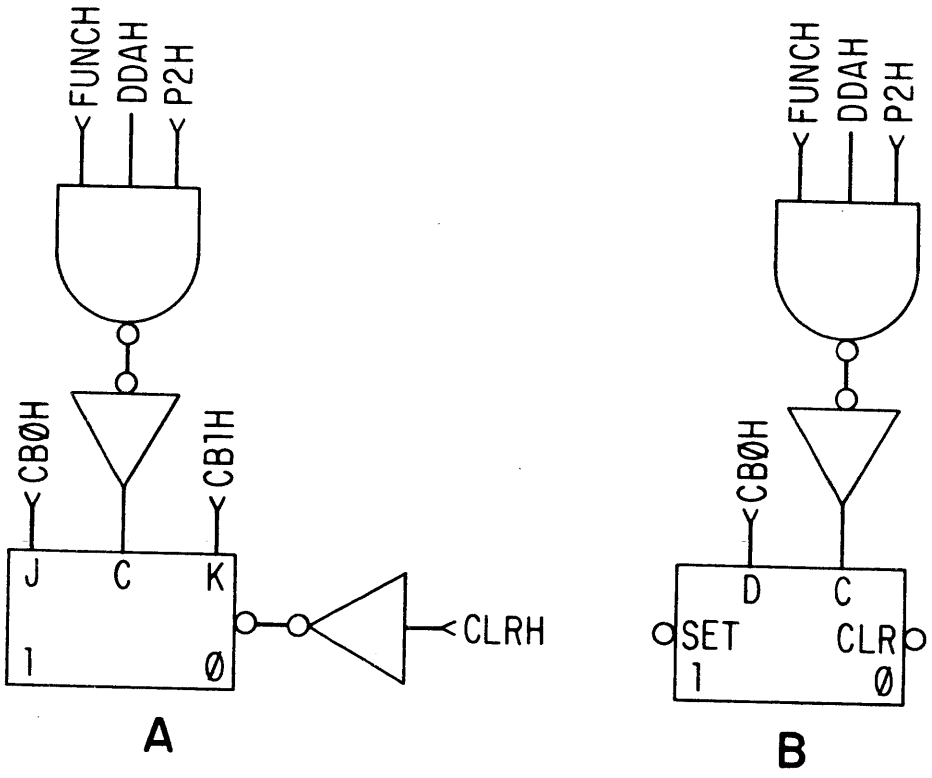


Figure 6-16 Function Generation Examples

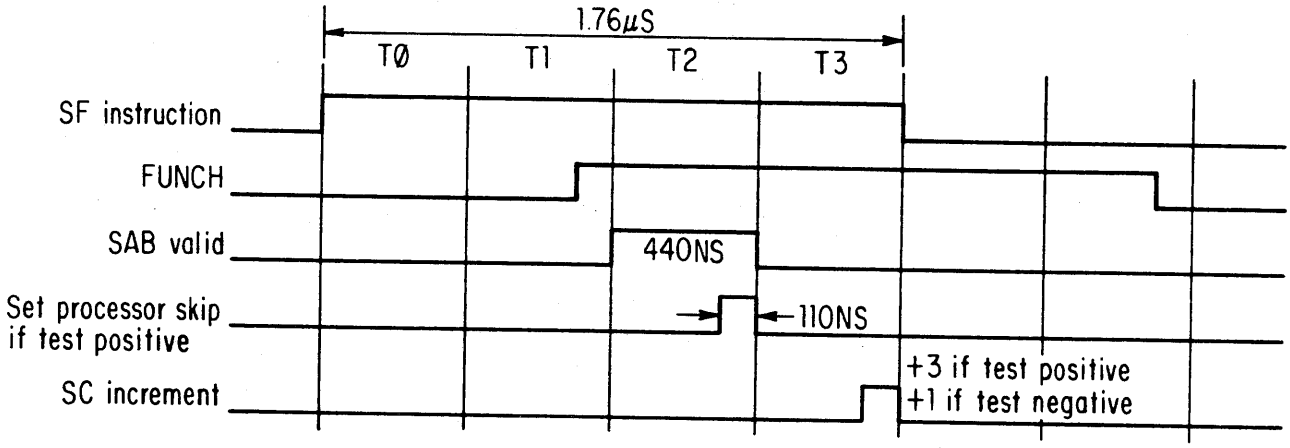


Figure 6-17 Function Testing, Timing

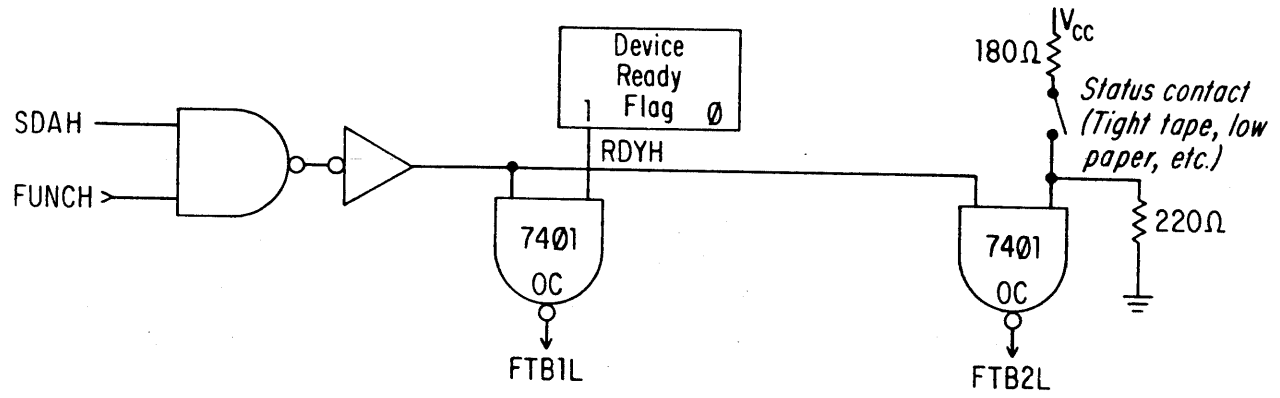


Figure 6-18 Function Testing, Logic Examples

6.7.4 Direct Memory Access

This feature can be set up for any device in the system that must pass data directly to memory, take data directly from memory, or increment the contents of a memory location by one. Each access requires one memory cycle during which the program pauses. The device requesting access must always supply the memory address as well as the control signals that specify mode of DMA access.

A DM request is initiated synchronously within a system device by that device grounding the DM request line through an open collector gate at ISYN time. A DMA is granted if all the following conditions are met:

1. DMRL is true (Request for DMA).
2a. The processor has completed any BK, previous DM, or EI cycle.

or

2b. The processor is at the end of an instruction.

When the DMA is granted the processor goes into the DM state. The DM state is a cycle consisting of four microstate conditions: T0, T1, T2, T3 (refer to Table 6-10).

a. During T0, the external address is sent to the MA, resulting in a memory reference.

b. T1 is dependent on signals DIRB and IMB, which indicates the direction of the DMA transfer. If DIRB is high or if IMB is low, the contents of the MB is transmitted to the external device. If DIRB is low or if IMB is high, a no operation occurs in this microstate time.

c. During T2, if DIRB is low, the word from the external device goes to the MB and is written into memory. If IMB is low, the contents of the MB is incremented and sent back to itself; therefore, the incremented value is stored in memory. If either of the two conditions are not met, this microstate becomes a no operation.

d. T3 is a no operation to allow time for the requesting device to disconnect itself or request the next cycle for a DM.

DMA control is located on PC 2, but decision as to type of DMA is on PC 1.

Figure 6-19 shows timing considerations for DMA transfers. The logic for DMA (see Figure 6-20) is very similar to the logic for an interrupt (Section 6.9.5) except that there is no status flag. An internal condition sets DMA SYNC; DMA REQ sets the next time the processor generates the ISYNH pulse, which occurs in every cycle. Setting DMA REQ gives rise to the DMA request signal DMRL on the bus. The DMA logic in a device also has hardware for a serial priority determining signal that goes from one device to the next. In this case, a device that receives DINL generates DOUTL for the next device if its own DMA REQ flag is clear. The setting of DMA REQ disrupts the serial signal so that it terminates at (and gives priority to) the first device that both receives DINL and in which DMA REQ is set.

At the next available cycle after a request is made, the processor generates the direct memory address signal EASH, which clears DMA SYNC (as a result, the next ISYNH clears DMA REQ) and sets DMA SERV in the device that has priority. DMA SERV combined with EASH gates a memory address onto the Destination Bus data lines; and combined with appropriate control levels in the device it may generate DIRBL to specify input (otherwise output is specified) or IMBL to specify that the word in the addressed memory location is to be incremented.

Table 6-10

DMA EXECUTION

Major State	Time Slot	Microinstruction	Comments
DM	T0	EAS to MA	External address is sent to MA.
	T1	MB to EDD	If DIRB is high or IMB is low, contents of MB go to external device; if DIRB is low, no op.
	T2	EDS to MB MB + 1 to MB	If DIRB is low data from external device go to MB; if IMB is low, MB is incremented and sent to itself.
	T3	No op	No operation.

The next operations depend on the type of cycle. For output, the processor generates EDDH to place the data from memory on the Source Bus data lines and sends a strobe DSTRH to load the data into a register at the device. If the increment function is specified, the processor sends the incremented word out to the device and also writes it back in memory in place of the original data.

For input, the processor generates EDSH to place data from the device on the Destination Bus data lines and generates a strobe internally to load the data into MB. When access is complete CLIBH clears DMA SERV to end the operation.

The logic diagram shows the basic request logic required for any type of cycle. Only input transfer logic and the gates for supplying a memory address are shown. Output transfer logic is not shown; a series of DMA transfers necessitates the use of a memory address counter for addressing consecutive locations and a word counter or end of transfer decoder to determine completion of transfer.

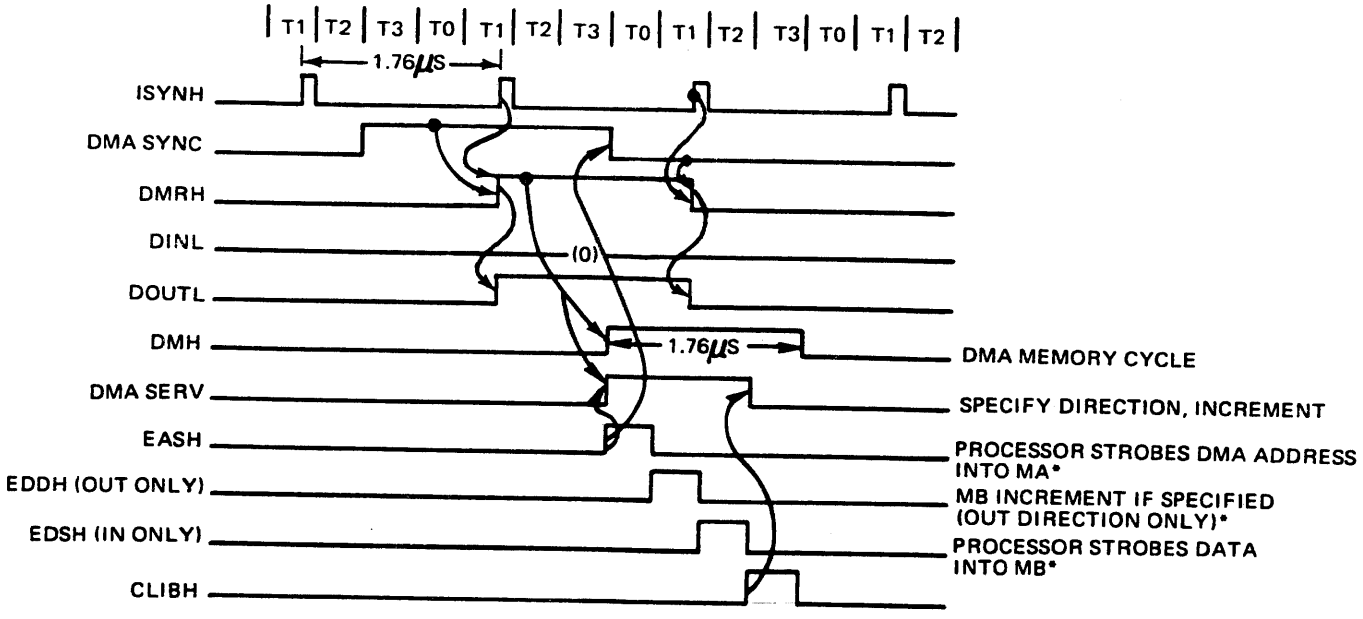


Figure 6-19 DMA Timing

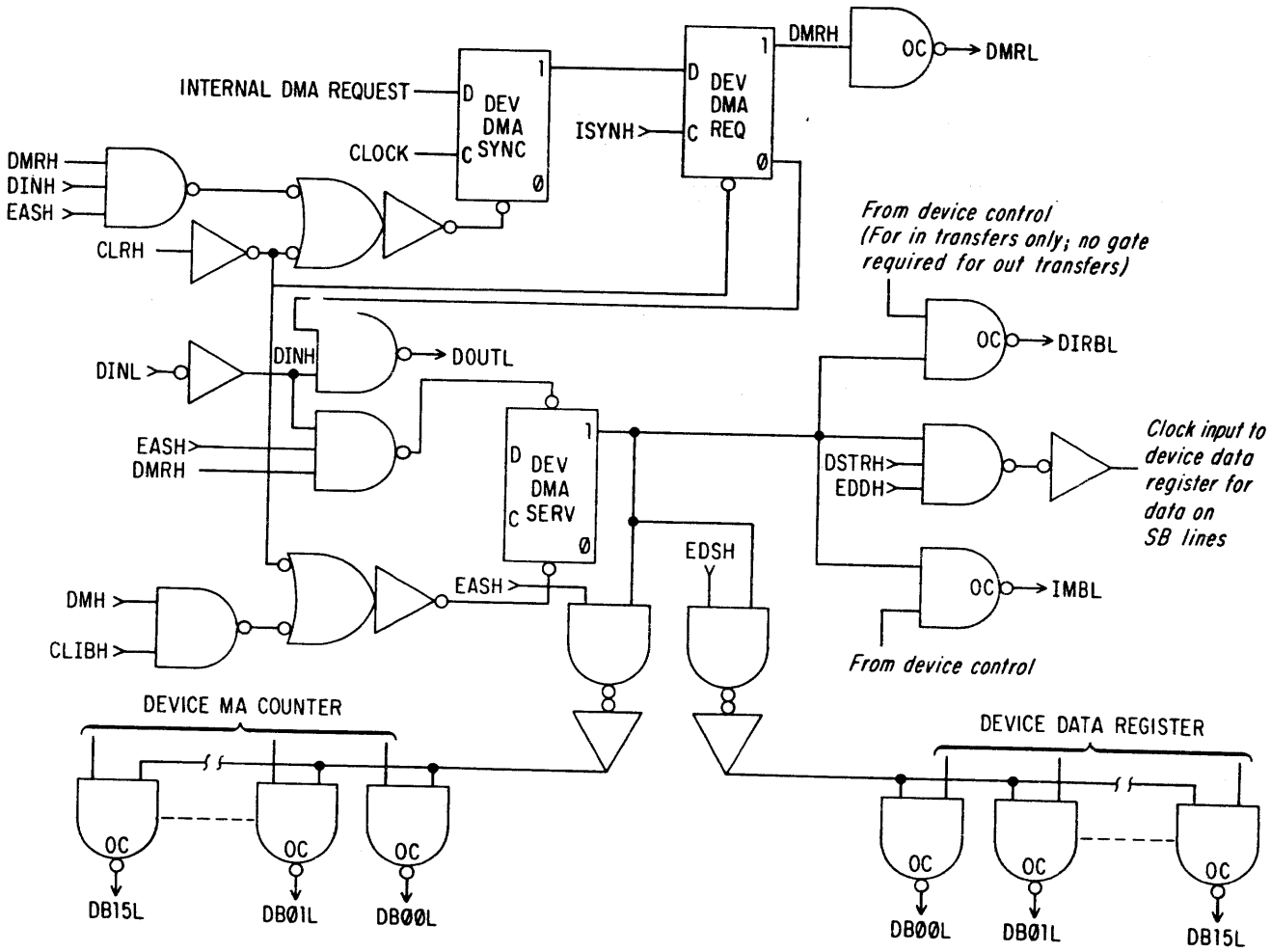


Figure 6-20 DMA, Typical Logic Diagram

6.7.4.1 Real-Time Clock (DMA Example) – An example of a GRI-99 device that uses the DMA feature is the Real-Time Clock (RTC) option. The Real-Time Clock provides programmed variable time intervals for systems software (e.g., event timing, elapsed timing, absolute time of day, etc.)

The Real-Time Clock is a DMA device that increments location 103 by one each time a pulse occurs. When location 103 overflows, an interrupt request is set. By presetting the service routine to a location with a particular negative number, a time base that is a multiple of the basic clock frequency is generated. For example, with a 60 Hz clock, pre-setting location 103 to -74_8 (-60_{10}), generates an interrupt at 1-second intervals. When the interrupt (overflow) occurs, the service routine is entered and the clock location 103 is reset to -74_8 (-60_{10}) again.

Figure 6-21 is a block diagram of the Real-Time clock. Note the variable clock frequencies and the connection of the DMA and interrupt service logic.

The Real Time Clock is contained on one 9 in. by 4 in. I/O PC card that plugs into the back of the GRI-99 in any vacant slot. Device priority relative to other devices for DMA or interrupt operations is determined by its position on the bus. The order of priority is highest on the left side of the GRI-99 when facing the rear of the machine. The Real-Time Clock uses the interrupt system; thus, priority chain jumpers (S40-215) must be inserted in positions SE-5 and SF-6 in all vacant slots between the RTC and the lefthand side of the machine as viewed from the rear.

The external connector, a 48-pin Amphenol (S40-203), is used to:

a. Bring a low level AC signal at line frequency to the RTC.

b. Make the three oscillator frequencies available to other system components.

c. Accept an external time standard.

In terms of DMA, when Clock Enable (CLK EN) is turned on, the clock pulses from the 60 Hz (50 Hz) Schmitt Trigger, or one of the IC oscillator pulse trains (determined by the jumper on the board) causes a DMA by setting the Clock Flag in the DMA service block. This action, in turn, enables the setting of DMA Request at the next ISYNH time. On completing the next full instruction after the DMA Request is set, the processor goes into the DM state.

At T0 of this cycle, the processor requests the memory address of the location that the RTC is to increment. This address is loaded into the Memory Address Register from the RTC when it receives the external source code (EASH).

At T1 time of the DM cycle, the processor executes the microinstruction MB to EDDH. The Source Bus data lines carry the previous value of location 103 during this time period and the value of SB15 is loaded into the OVERFLOW TEST flip-flop.

At T2 time, the processor executes the microinstruction MB P1 to increment the previous value of location 103 in preparation for writing the new value back into location 103. The Source Bus line SB15 now carries the new value of the clock word, and pulse P2H is used to compare the contents of the OVERFLOW TEST flip-flop with the new value of SB15.

This comparison yields one of the following results:

a. At T1, the contents of SB15 was a 0 (OF Test not set), and at T2 C(SB15) is a 1 or a 0; i.e., there was no counter overflow.

b. At T1, the C(SB15) was a 1 (OF Test is set), and at T2 C(SB15) is still a 1, i.e., there was no counter overflow.

c. At T1, the C(SB15) was a 1 (OF Test is set), and at T2 C(SB15) is a 0. There was counter overflow, and the OVFL Flag is set.

The CLK flag flip-flop is cleared each time the processor is in a DMA cycle; as a result, the flip-flop can be set on the next clock pulse and generate another request. This process continues until an overflow does exist. At that time, an Interrupt Request is initiated.

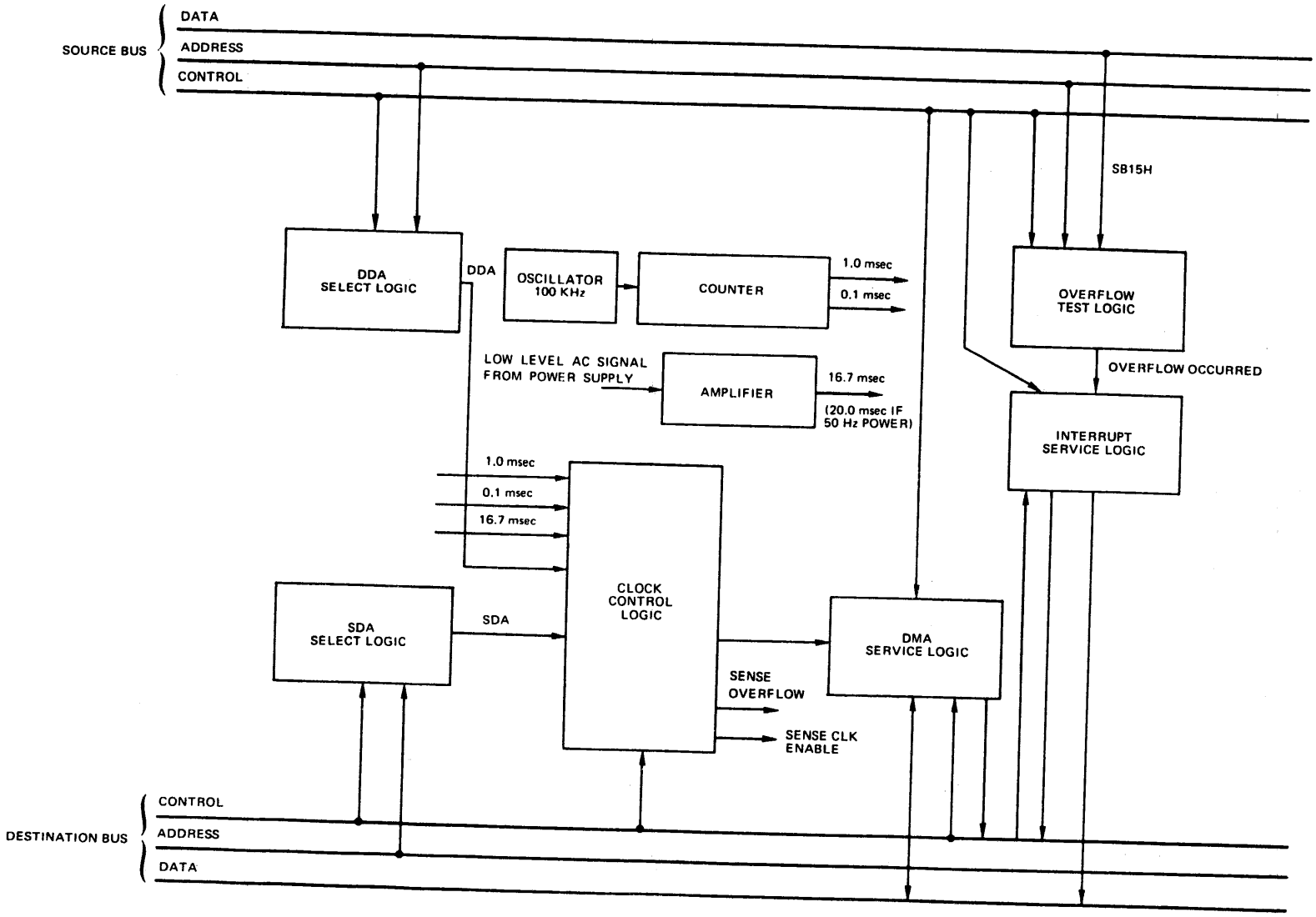


Figure 6-21 GRI Real-Time Clock Block Diagram

To vary the basic time interval between overflow (interrupts), the program can preset the DMA increment location (103_8) to the 2's complement of the number of clock ticks wanted. For example: presetting location 103 to 177400 with the board jumpered for 100 μs clock; it will take 25.6 ms for the clock to overflow. If the board was jumpered for 1 ms operation, the same number requires 256 ms to overflow. If the board was set up for 60 Hz operation, it would take 4.266 seconds to overflow.

For more detailed information concerning installation, testing, and interrupt servicing, refer to the GRI-99 *Real-Time Clock Manual.*

6.7.5 Interrupt

Most I/O devices contain a flag (usually meaning Ready) that is set at the completion of an operation to cause an interrupt. *If the device is for output, the flag is also set by CLRH.* Thus, following power on or use of the START key, the device indicates that it is ready to output data. *In an input device, CLRH clears the flag.* In either case, the flag must also have provision for a programmed clear by an FO instruction to take full advantage of the interrupt system.

An interrupt request is initiated synchronously within a system device by that device grounding the interrupt request line through an open collector gate at ISYN time (P1 time); during a BK cycle, there is no ISYN. At the completion of the current program instruction, the processor grants one cycle, during which the SC contents is saved in memory and the SC is set to a new address as per the interface design.

An interrupt cycle is granted when all the following conditions are met:

1. The main interrupt control is on (an ICO has been issued by the program) i.e., IA on console is lit;
2. INTBL is true (interrupt request);
3a. The processor is not in a DM or EI state;

or

3b. The processor is finished processing a macroinstruction.

When the interrupt cycle is granted, the processor acknowledges this fact by going into the BK state. The signal POUT (originated on PC3) is sent down the bus serially away from the processor. Each device not interrupting propagates the signal. The interrupting device terminates the signal and accepts the interrupt acknowledge. When more than one device requests interrupt at the same time, the device physically closest to the processor on the bus has the highest priority.

When the processor goes into the BK state, four microstate conditions are generated: T0, T1, T2 and T3 (refer to Table 6-11):

a. During T0, the External Address is sent to the MA. This function causes a memory reference.

b. TI is a no operation time slot (the contents of memory is being read).

c. During T2, the contents of the SC is sent to the MB. This action causes the contents of the SC to be stored in memory during the Write phase of the memory reference.

d. During T3, the external address is incremented by one and sent to the SC. The SC is now pointing at the first instruction in the interrupt handling routine.

All the interrupt control logic is located on PC2. Microstate control is on PC1.

Table 6-11

INTERRUPT EXECUTION

Major State	Time Slot	Microinstruction	Comments
BK	T0	EAS to MA	External address sent to MA.
	T1	No op	No operation.
	T2	SC to MB	Contents of SC sent to MB.
	T3	EAS + 1 to SC	External address incremented by 1 and sent to SC.

Figure 6-22 is a timing diagram for the interrupt. Figure 6-23 is a typical logic diagram. The following detailed description of interrupt signals and timing is keyed to Figures 6-22 and 6-23. The discussion, of course, assumes that an FOI ICO instruction has been issued by the program thus activating the interrupt control logic in the processor (IA indicator is lit).

When RDY is set, INT REQ sets at the next ISYNH time, providing that the INT STAT bit for the device is on. INT STAT is a single bit in the Interrupt Status Register (ISR); the ISR can be addressed as a data source or destination. The address of this register is decoded in the processor on the I/O bus and is sent on the I/O bus as IDAH when used as a destination and as ISAH when used as a source. Other registers of this type can be added to the system, but decoding for them must then be done from the DAB and SAB lines.

After INT REQ is set, the interrupt bus line INTBL is pulled to ground, causing the next available cycle to be used for a break. A serial signal PINL and POUTL determines which of the devices requesting an interrupt has the highest priority.

The priority determining signal is passed along the bus from one device to another. If a device receives PINL and its own INT REQ flag is clear, it generates POUTL, which becomes PINL at the input to the next device. INT REQ being set disrupts the serial signal. Consequently, the first device on the bus with INT REQ set is the only one that both receives PINL and has its own INT REQ flag set; this condition allows the acknowledgment signal BKH from the processor to select a device for a break.

BKH is gated at two different times by the external address request signal EASH, which causes a fixed-wired set of open collector gates to produce a hardwired address on the Destination Bus data lines during those periods of the break state when an external address is required by the processor. If no address is generated at the time of the EASH signal, the processor traps to location 0 (i.e., it stores SC in location 0 and resumes operation at location 1). If an address is generated, the processor uses the address generated by the interrupting device for storing SC and uses the next consecutive location to resume the execution of instructions.

Gating the address into the processor twice allows the device to supply a different address the second time. Thus, a device might always store SC in the same pre-assigned location, but then begin program operation at various locations depending on the cause of the interrupt, as for example in the pulse input detector.

6.7.5.1 **Gate Input Card (Interrupt Example)** – An example of a GRI-99 device interface that uses the interrupt system is the Gate Input Card (GIC). The GIC is used to interface most devices that produce parallel data to be read into the GRI-99. Provisions for function testing and control of the device are also made on the GIC. A complete interrupt system interface is provided. The GIC functions as a source of data only. The device address of GIC is normal set for 63, but is selectable by the user through the programming of staples.

The Gate Input Card plugs directly into any vacant slot at the back of the GRI-99. Device priority relative to other devices for DMA or interrupt operations is determined by position on the bus. The order of priority is highest on the left side of the GRI-99 facing the rear of the machine. The GIC uses the interrupt system; thus, priority chain jumpers (S40-215) must be inserted in positions SE-5 and SF-6 in all vacant slots between the left-hand side of the machine, as viewed from the rear, and the GIC board.

All connections to the board are made via the standard 48-pin I/O cable plug and cable clamp assembly, S40-216. Ribbon cable, twisted pair, or miniature coaxial cable can be used. Cable shields are terminated on one of the screws that fastens the cable clamp assembly to the board. Grounding is important, as well as the cable length because the inputs on GIC have no signal conditioning. If sufficient precautions are taken in relation to system grounding, cable length can be up to 20 ft. Every ground connection must be used. Each signal has an adjacent ground to permit running alternate wire grounds in a ribbon cable. EPOL(H)-EP3L(H) can be terminated at the device to both +5V and ground if they are lightly loaded in the device. Table 6-12 is a listing of the external connections and associated pin numbers.

The GIC services many devices for data input to a GRI-99. In most cases, all that is necessary is proper choice of staple arrangements for positive or negative level inputs on flags and status lines. The data line inputs are set up to handle only positive assertion logic. If negative assertion logic is used, the result from the device can be 1's complemented while it is being read into another system register. The user can also increment the value as it is read in from the GIC and, thereby, store the negative of the value (2's complement).

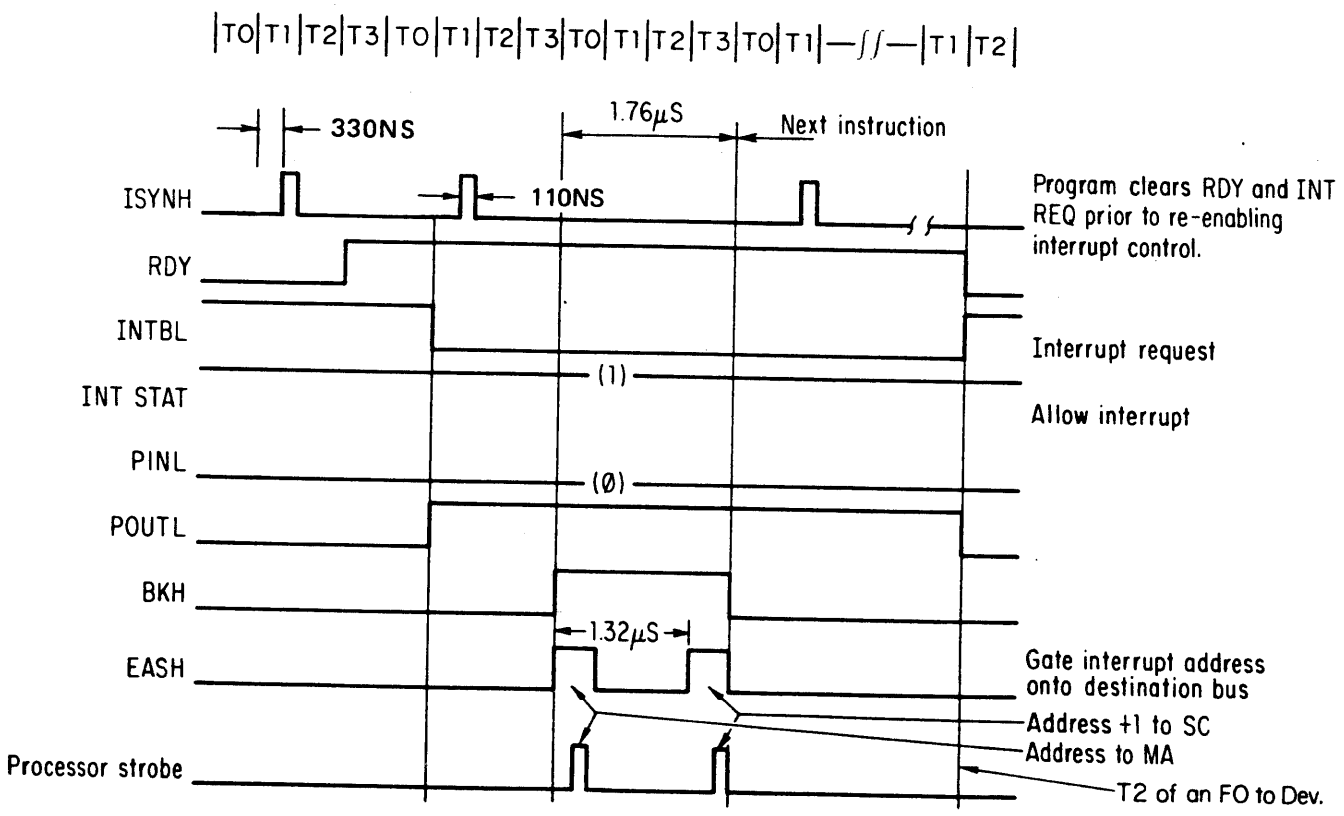


Figure 6-22 Interrupt Timing

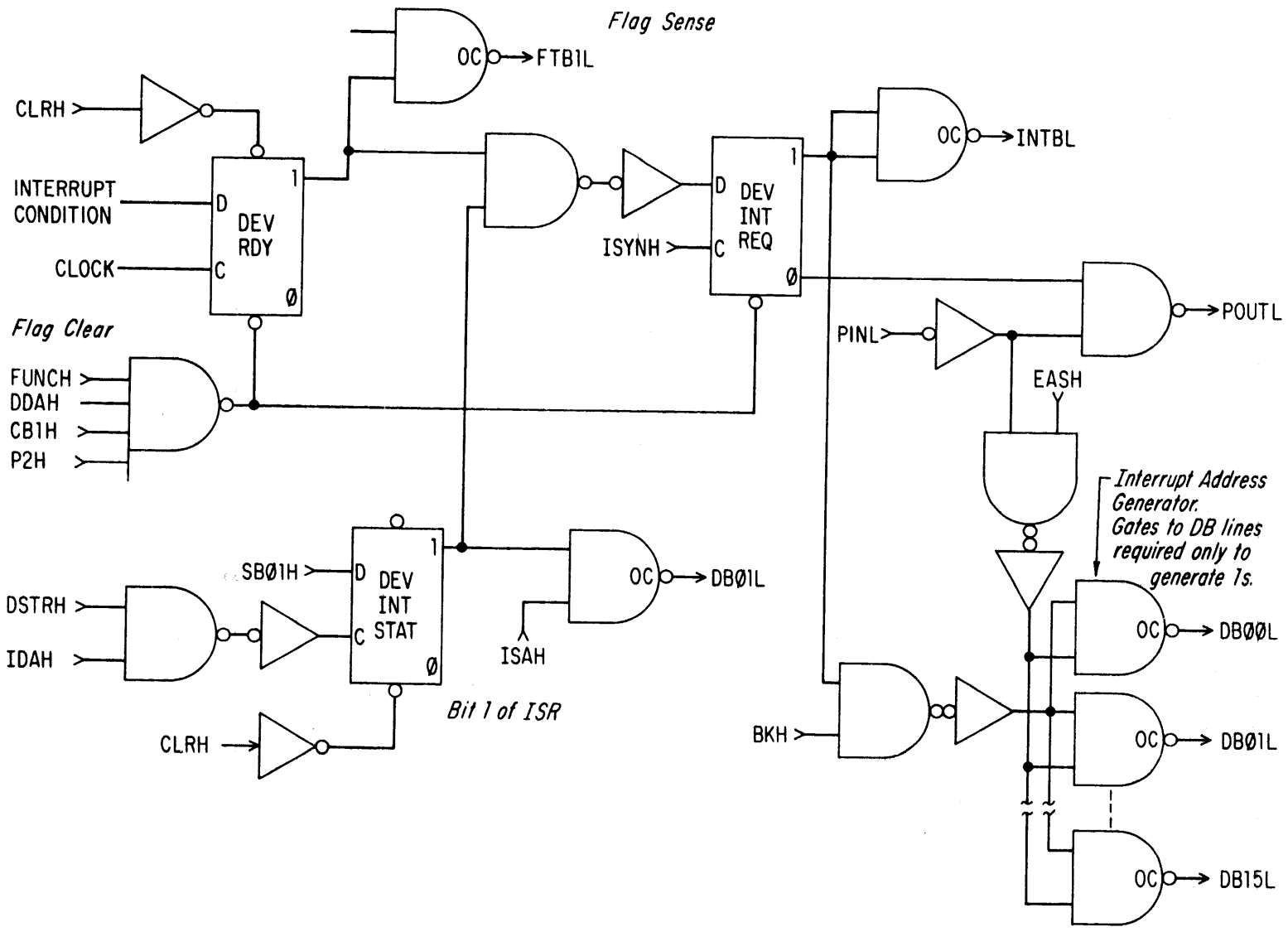


Figure 6-23 Interrupt, Typical Logic

Table 6-12

GIC EXTERNAL CONNECTIONS AND PIN NUMBERS

Signal		Pin	GND
I0H (LSB)	DATA IN	A	1
I1H	DATA IN	B	2
I2H	DATA IN	C	3
I3H	DATA IN	D	4
I4H	DATA IN	F	6
I5H	DATA IN	H	7
I6H	DATA IN	J	8
I7H	DATA IN	K	9
I8H	DATA IN	M	11
I9H	DATA IN	N	12
I10H	DATA IN	P	13
I11H	DATA IN	R	14
I12H	DATA IN	S	15
I13H	DATA IN	T	16
I14H	DATA IN	U	17
I15H (MSB)	DATA IN	V	18
FSPH(L)	FLAG SET IN	W	19
EF1H(L)	STATUS 1 IN	X	20
EF2H(L)	STATUS 2 IN	Y	21
EPOL(H)	FUNC. PULSE 0 OUT	c	25
EP1L(H)	FUNC. PULSE 1 OUT	b	24
EP2L(H)	FUNC. PULSE 2 OUT	a	23
EP3L(H)	FUNC. PULSE 3 OUT	z	22

H = Positive Assertion
L = Low (GND) Assertion
H(L) = Positive or Low Assertion

There are four independent one shots (74121s) on GIC for the user to select a variety of pulse widths for function commands. The nominal factory setting is 1.0 μs and negative polarity. If the user desires a different set of pulse widths, the RC timing networks can be changed, and the polarity is changed by staples on the board.

NOTE

A pulse width in excess of 1.76 μs may cause the control pulse to overlap the next instruction after the FO command that generated the pulse.

The device has a complete interrupt system connection. The ISR bit is normally connected to bit 6 for factory test purposes and can be changed by the user. An address of 44, (45, 46) is normally generated when an interrupt occurs. This address can also be easily changed by the user.

Device Address Selection – The device address selection consists of a dual row of staples marked 1 and 0 surrounding decoders (7430) in positions A1 (DAB decode) and L1 (SAB decode). To set an address in a board, insert staples in Row 1 for the SAB or DAB bits to be decoded as 1s. Insert staples in Row 0 for SAB or DAB to be decoded as 0s. The example shown in Figure 6-24 is for an address of 65_8.

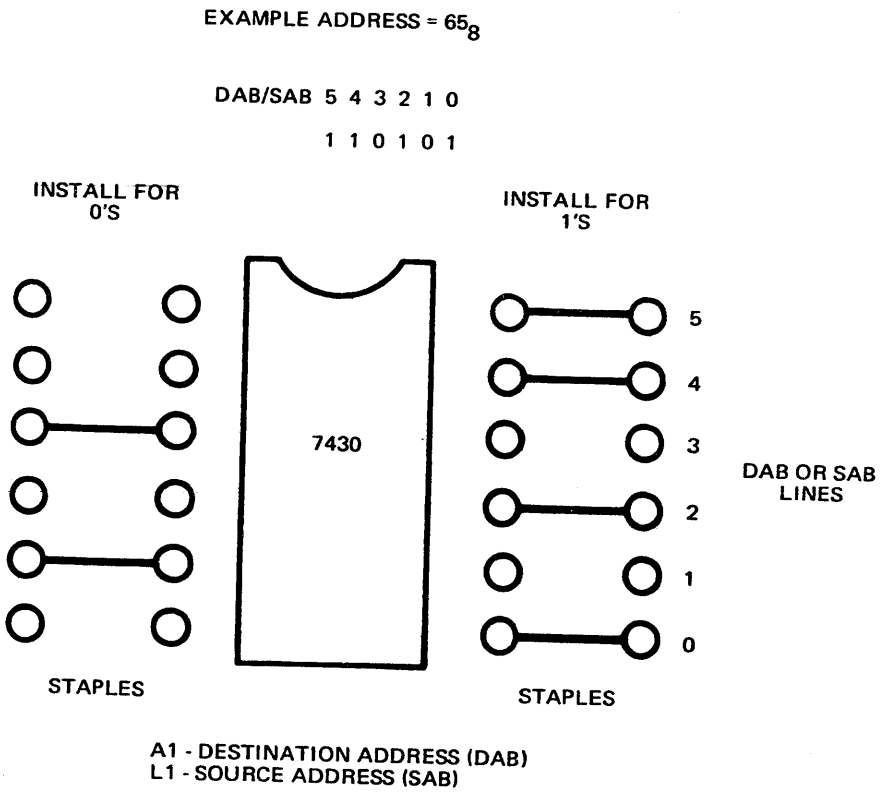


Figure 6-24 Variable Address Selection

Device Interrupt Control – Some devices provide for a choice of interrupt status bit and interrupt address generation. The ISR bit is normally set to a particular bit. For interrupt status bit changes the same SB and DB bits must be chosen. Interrupt address generation provides for up to four Is to be generated on any of the 16 DB lines. For example, assume that the desired interrupt address for a device is 45_8, 46_8, 47_8. Only the first address of the group need be generated. This address is the address in which the SC is stored when the interrupt occurs. The generated address plus 1 (46_8) is the location at which program operation resumes after the interrupt. For 45_8, three Is must be generated:

$$45_8 = 100101_2$$

DB bits 0, 2, 5 must be connected to the address generator gates. Note that one of the four gates is not required and is, therefore, left open.

The wiring of interrupt functions is described with each device manual in tabular form.

NOTE

All devices are set for a specific device address and interrupt controls at the factory to facilitate testing of the boards. The user can alter these addresses if he desires by following the instructions in the device manual. ***In systems where multiples of the same device are used, the user must, of course, change the addresses and interrupt controls.***

The interrupt controls, however, need not all be different. The same status bit, for example, is often assigned to a group of like devices. For example, assume five general-output registers are installed in a system. All five registers may be assigned to the same status bit, but each one will generate a unique interrupt address. When all boards are on the same status bit level, there is a hardware priority imposed by the order in which the boards are plugged into the rear of the GRI-99. This priority is determined by the PINL-POUTL chain and runs from left to right (highest to lowest) looking at the rear of the machine.

The Gate Input Card (GIC) has provisions for user selection of interrupt status bit, interrupt address generation, pulse widths for external function pulses (4), external flag polarity selection, external status flag polarity selection, and clamp type selection on all inputs (diode or resistor pull up).

6.7.6 External Instruction (EIR)

In EIR mode, a system device takes control of the bus and temporarily suspends operation of the stored program until the device is finished. Instructions are presented directly to the IR as 16-bit words to be executed. The instructions do not reference memory; however, each instruction requires only 880 ns (half the normal instruction cycle) to be executed. In a sense, the EIR device borrows registers (such as the AO) from other devices and, thereby, performs certain operations more efficiently than hardware modules, and at least twice as fast as equivalent software.

An EI is granted if the following conditions are met:

1. EIRL is in a true condition (request for EI).
2a. Processor is not in a BK, DM, or EI state.

or

2b. The processor is at the end of an instruction.

The EIR mode is generally initiated by an FO control pulse that sets an EIR flip-flop in a device. The EIR flip-flop is gated onto the EIRL bus, causing EI request to be present when the processor completes execution of the FO instruction. The request is then granted, and the processor enters the EI major state (refer to Table 6-13).

During T0, the processor executes the microinstruction EDSH → IR. Then, the EIR device gates the next instruction from its ROM onto the Destination Bus lines, via the Bus Modifier to the IR. During T1, the instruction is executed by a strobe pulse that occurs in the 330 ns to 440 ns segment of T1. In the EI state, the device must supply an instruction to the Destination Bus on every even time period. In each odd time period, the instruction is executed. During T2,

the processor fetches another instruction and in T3 executes the instruction. The EI mode forces EXT signals to occur during T1 and T3. To allow use of FO and SF class instructions in EI devices, P2H is also generated during T1 and T3, and not during T2. When the EI device completes its last ROM access, the EIR flip-flop is reset. The EI request is terminated, and the processor terminates the EI state. The T1 strobe pulse is reinstated at the 220 ns to 330 ns segment of T1 for normal macro-instruction processing.

Table 6-13

EXTERNAL INSTRUCTION

Major State	Time Slot	Microinstruction	Comments
EI	T0	EDS to IR	External data is sent to the instruction register.
	T1	EXT C (IR)	Execute the external data as an instruction.
	T2	EDS to IR	See T0.
	T3	EXT C (IR)	See T1.

Figure 6-25 shows timing for EIR mode instructions. Figure 6-26 is a typical logic diagram. The sequence begins when an FO instruction sets a flag to generate an external instruction request on the bus from the operator. An external instruction sequence follows immediately, making the FO and the sequence appear as one continuous macro-instruction. The EI state is maintained until a done condition in the device combined with EDSH drops the request.

As previously mentioned, while in the EI state, the device must supply an instruction to the Destination Bus data lines every even time period. Gating is supplied by EDSH and the word is sent to IR through the Bus Modifier. In each odd time period, the instruction is performed. Except for memory references, programs running in the EI state can accomplish the same tasks as a core program, but at twice the speed.

6.7.6.1 Devices Using EIR – Typical devices that use the EIR mode are:

a. Medium Speed Multiply Operator (MPO)

b. Medium Speed Divide Operator (DVO)

c. Arithmetic Right Shift Operator (ARS)

d. Normalize Operator (NORM)

These devices, in conjunction with the AO, the Six General-Purpose Registers, and the Extended Arithmetic Operator, are inexpensive *hardware* devices that perform hardware multiplication, division, double-precision shift and normalize, and firmware double-precision and floating point arithmetic. In the GRI-99 Model 40, these devices are standard features.

The GRI-99 EIR channel effectively extends the ROM that is used for controlling the processor. For simplicity, the EIR device is restricted to register-reference microinstructions only. These microinstructions are actually executed from the IR, rather than directly from the ROM added to the EIR device. The EIR device is normally initialized by the execution of an FO instruction.

For example, the instruction

```
FO   STRT, MPO
```

initializes the Medium Speed Multiply Operator.

The FO control pulse causes the EIR flip-flop to set in the EIR device. The output of the EIR flip-flop is gated onto the EIRL bus, which causes an EI request to be present when the processor completes execution of the FO instruction. The EI request is granted at the completion of the FO instruction, and the processor goes into the EI major state. Processor timing now cause a fetch-execute microcycle every 880 ns, which is half the normal instruction cycle. During the EI state, the processor sets up and executes the microinstruction sequence previously described, and the EIR flip-flop is reset. A more detailed example of typical EIR device logic is shown in Figure 6-27. For detailed operation and programming information about EIR devices, refer to the GRI-99 *EIR Devices Manual.*

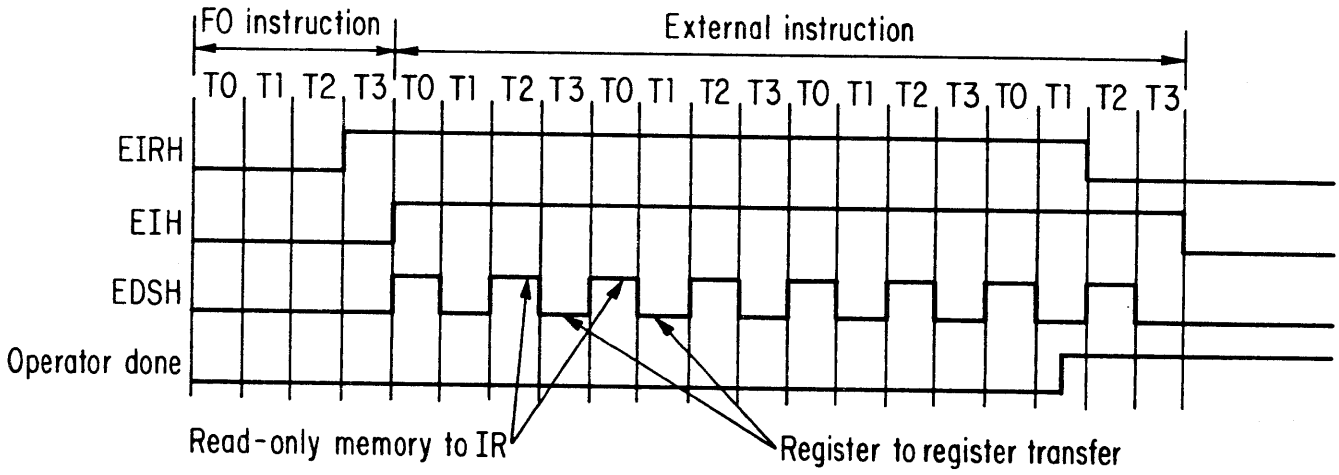


Figure 6-25 EIR Timing

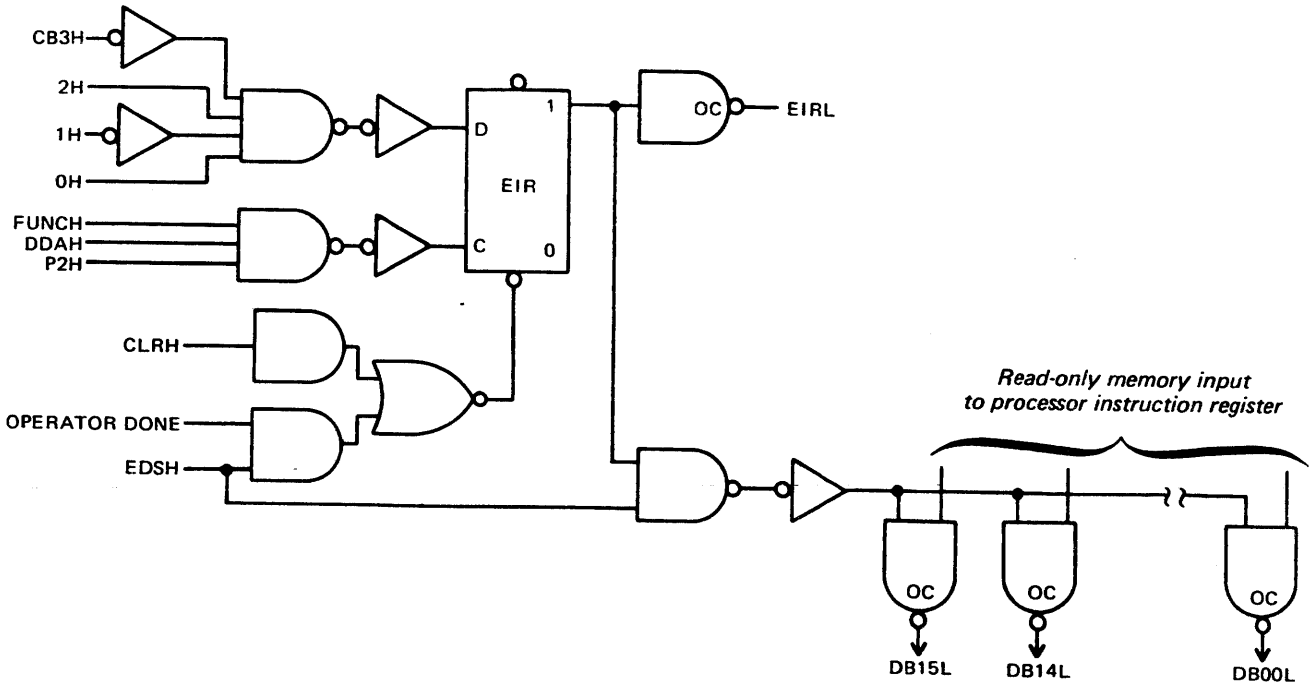


Figure 6-26 EIR, Simplified Logic Diagram

6.8 DESIGN EXAMPLES

A common operation in many process control applications is mechanical positioning of a device on command from the computer. Figure 6-28 shows a valve positioner interfaced to the GRI-99 Source Bus. Valve position in digital form is loaded into the D/A converter when that device address is specified as a destination. The D/A converter output goes directly to the external device connector, through the Amphenol plug to a positioning servo motor external to the computer.

Figure 6-29 shows a device that monitors the status of 32 relay contacts. An FO instruction selects one of the two sets of 16 contacts, which can then be addressed as a data source, where the 16 data bits equal the relay contact status (e.g. 0 indicates an open relay, 1 indicates a closed relay).

6.9 MEMORY EXPANSION

Power is available within the computer to drive most 32K systems. However, in the case of large configurations, especially those utilizing an I/O chassis, an auxiliary supply may be required.

6.9.1 8K Expansion

A pair of staples or switch closings are located at position E11 on the memory module. The position of one staple or switch closing (0-3) defines the 8K segment of a possible 32K configuration. The second staple or switch closing is placed in the L4K position to specify the first address of the 8K block selected by the staple in pins 0-3. Figure 6-30 shows examples of staple or switch arrangement for an 8K memory module.

6.9.2 4K Expansion

A pair of staples or switch closings are located at position E11 on the memory module. One staple or switch closing (0-3) defines the 8K segment of a possible 32K configuration. The second staple or switch closing defines the upper 4K (U4K) *or* lower 4K (L4K) segment of the 8K block. Figure 6-31 shows examples of staple or switch arrangement for a 4K memory module.

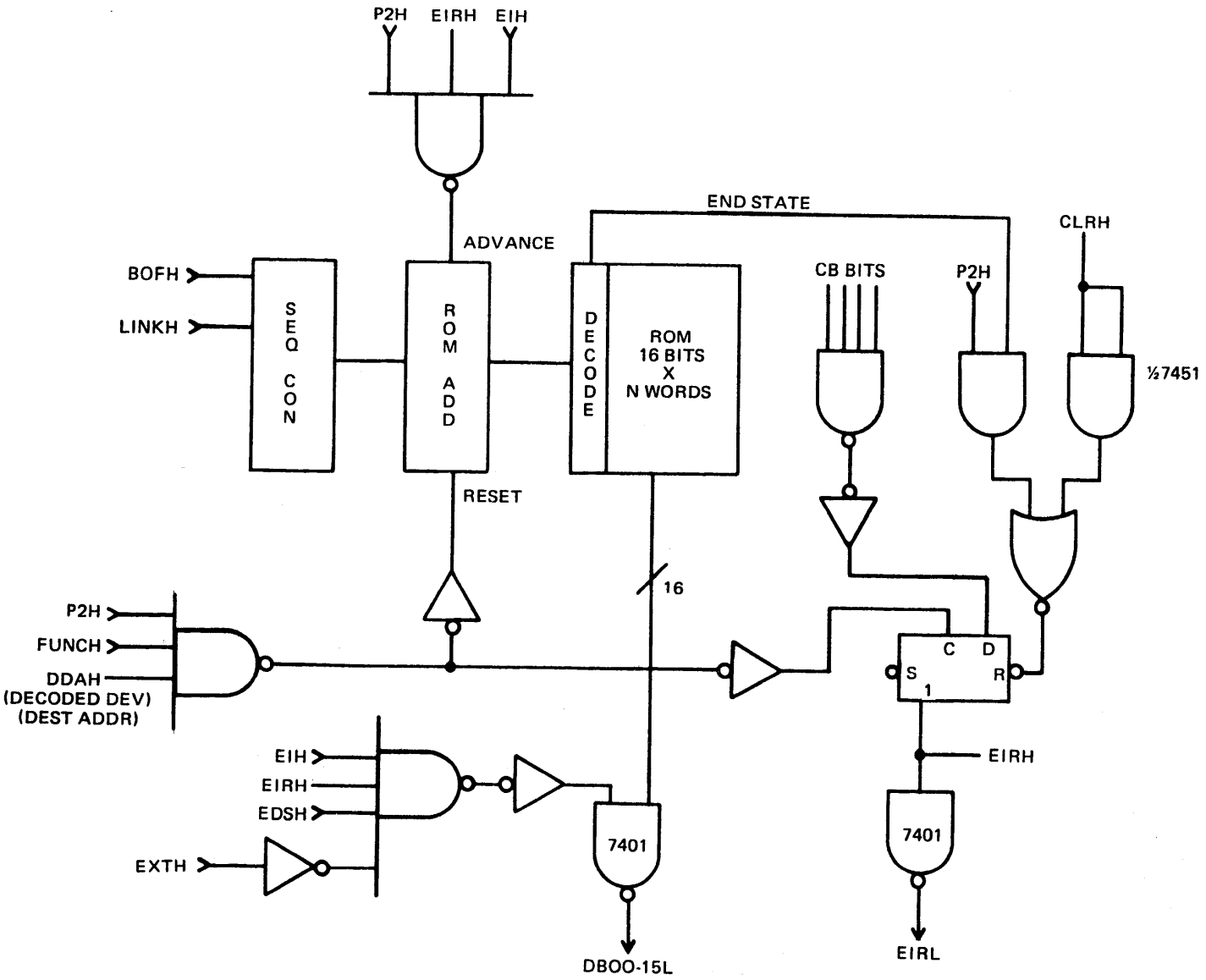


Figure 6-27 Typical EIR Device Logic

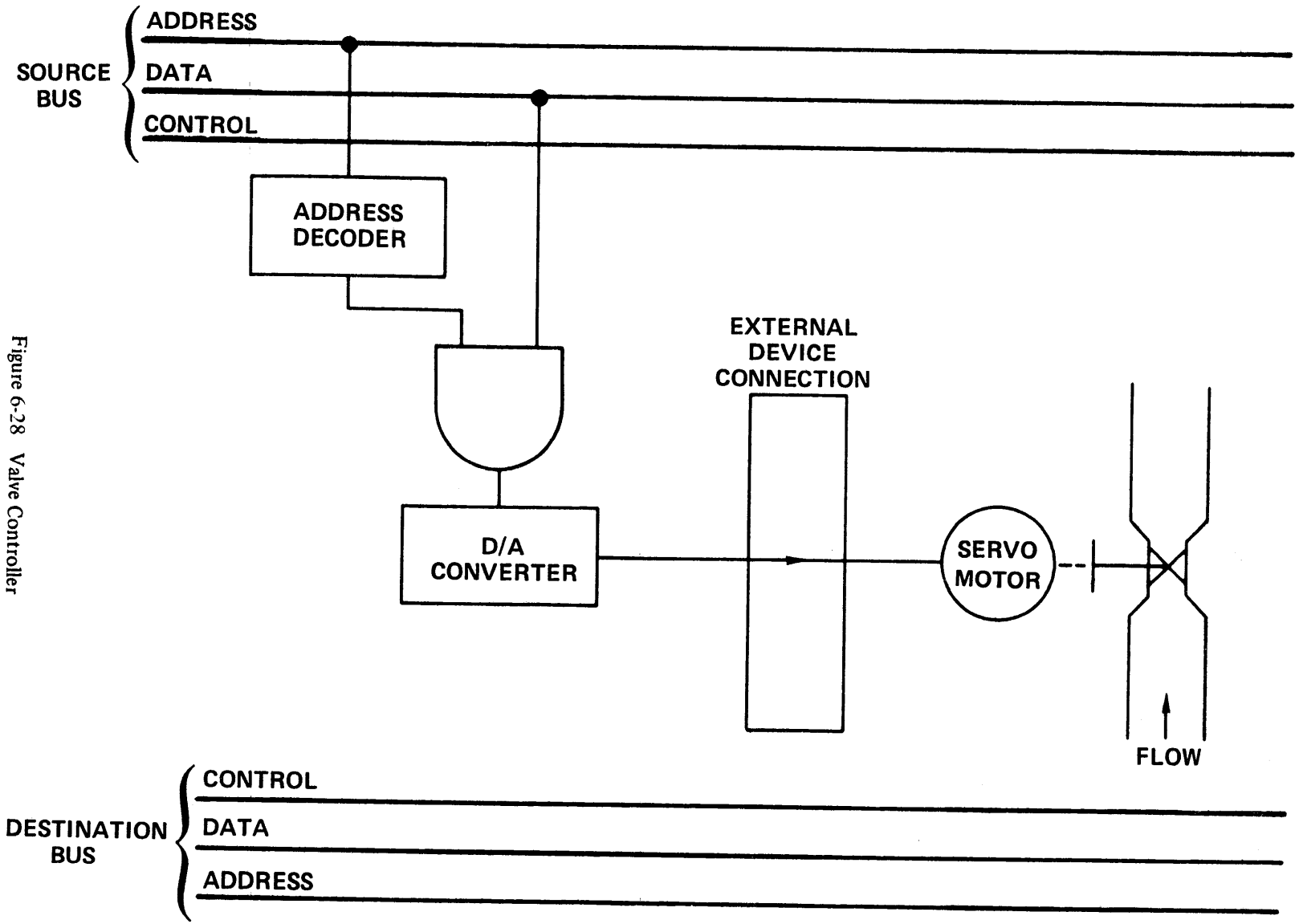


Figure 6-28 Valve Controller

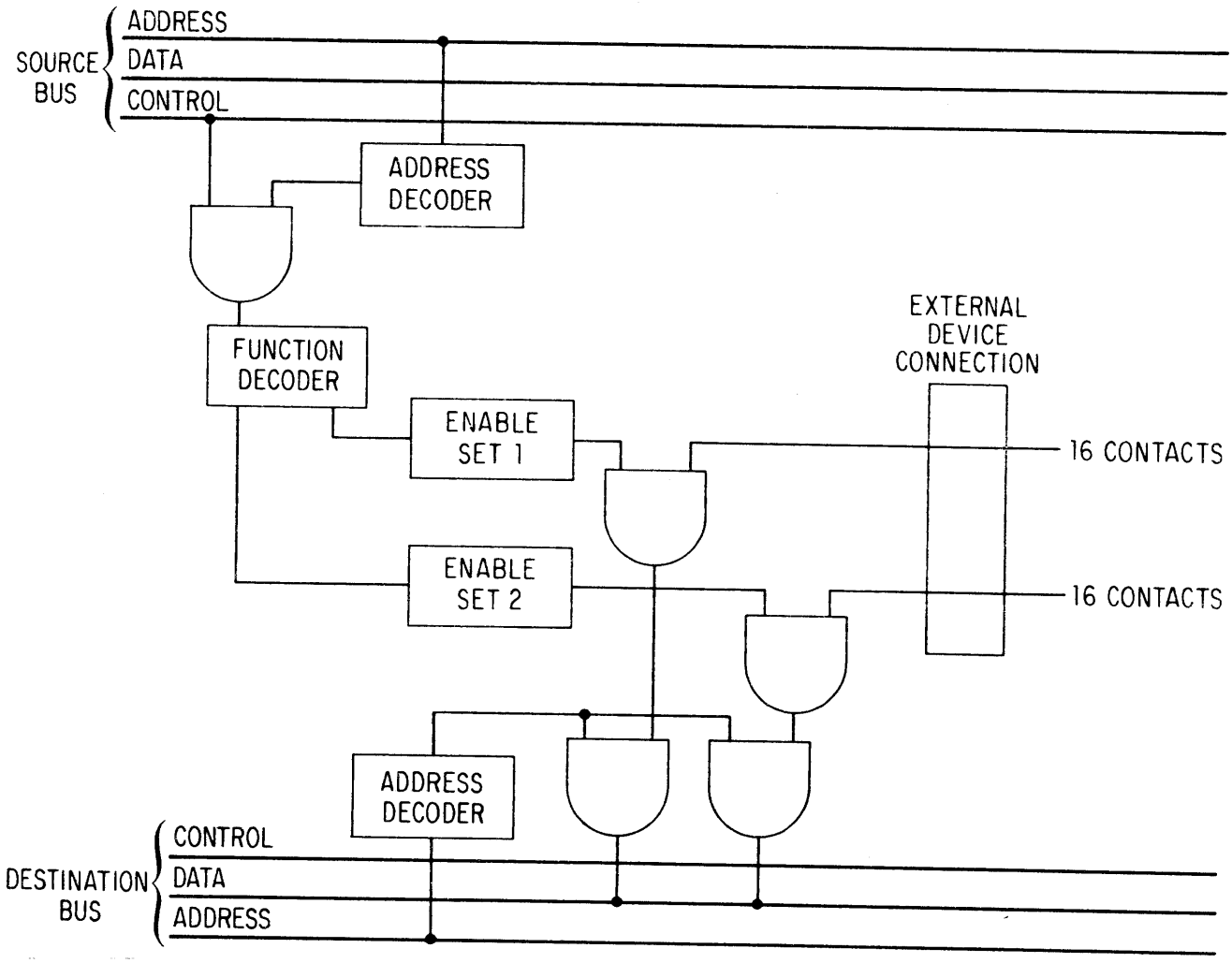


Figure 6-29 Relay Contact Monitor

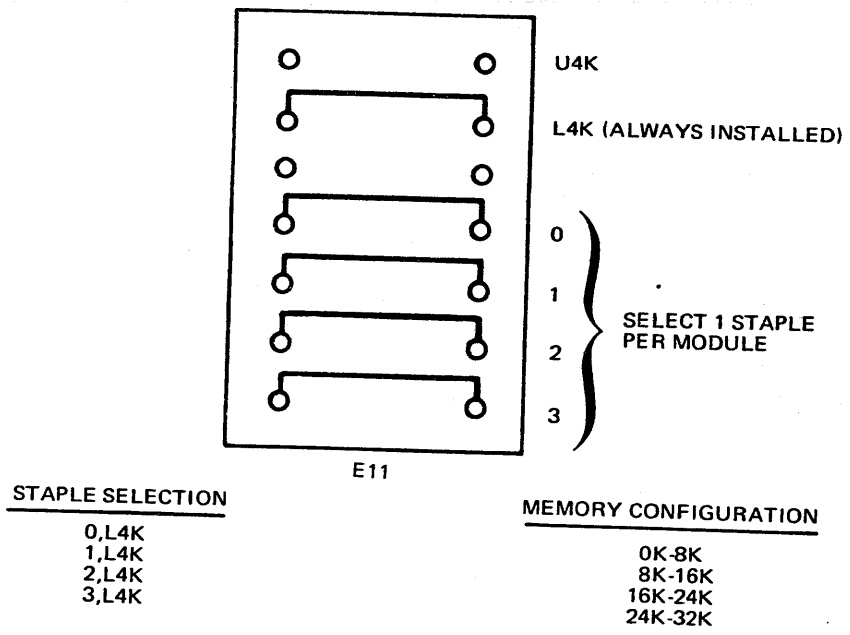


Figure 6-30 Staple Pattern for 8K Memory Module

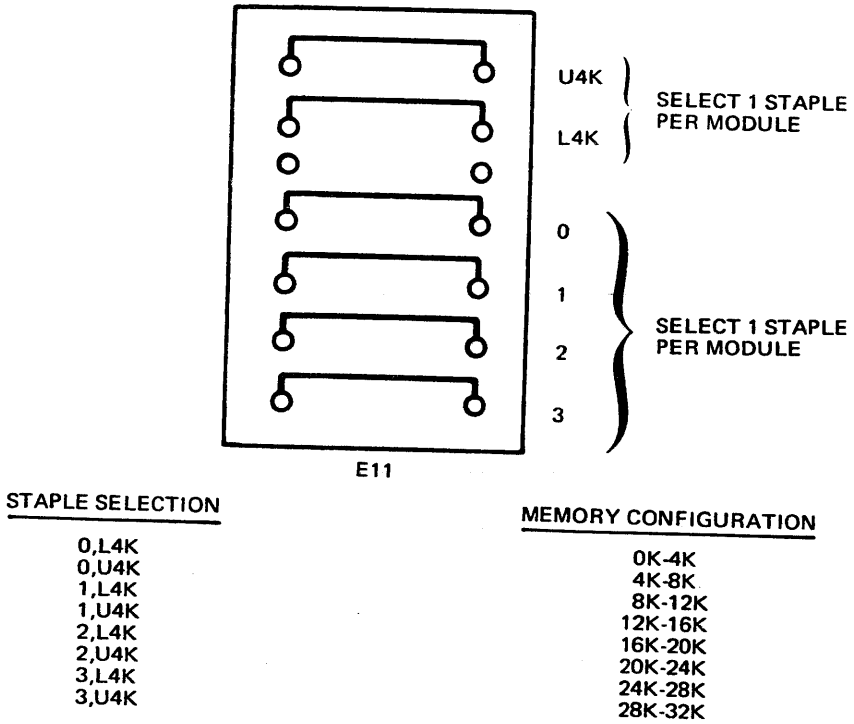


Figure 6-31 Staple Pattern for 4K Memory Module

APPENDIX A
HARDCOPY EQUIPMENT

This appendix discusses the simpler peripheral devices: Teletype, tape reader, tape punch, card reader, card punch, plotter and line printer. These devices are used principally for communication between the computer and the user using a paper medium: tape, cards, form paper or graph paper. All transfers for them are made by the program.

The program can type out characters on the Teletype printer and can read characters that have been typed in at the keyboard. This device has the slowest transfer rate of any device, but it provides a convenient means of man-machine interaction. The KSR Teletypes comprise only a keyboard and printer; the ASR models also have a slow-speed 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 on tape can be brought into the system by the high-speed tape reader or the one mounted in the Teletype.

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 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 on-line 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.

A.1 TELETYPE

Two Teletype models are regularly available for use with the GRI-99; the ASR-33 and KSR-33, both of which are capable of speeds up to ten 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 on tape and read characters from a tape.

A.1.1 Customer Supplied Teletypes

The Teletype Corporation Model ASR-33 Teletype is available in a variety of model numbers. These model numbers are designated by the ending two- or three-character codes on the name plate of the Teletype. The three most common Teletypes being used, all of which may be easily used with the GRI-99 with minor modifications, are the ASR-33 *TC*, the ASR-33 *TU*, and the ASR-33 *TZ*. The models *TC* and *TU* are identical except that the numeral zero on the *TU* has a slash through it (ϕ), whereas the numeral zero on the *TC* printwheel looks like the letter O. The *TC* and *TU* are both what is known as no-parity units; that is, they always produce a 1 in channel 8. The model *TZ* is an even-parity model that produces a 1 in channel 8 in order to make the total number of ones in the 8-bit character an even number. All three of these models are equipped with "answer back," which means that transmission of the "who are you" code (WRU) to the printer mechanism causes the "here is" drum to be tripped and a string of characters to be transmitted. This feature must be disabled before using the Teletype with the GRI-99 software packages. Disabling of the answer back feature is a relatively simple modification which can be performed by any Teletype serviceman. All three of these Teletypes are the most common Teletypes available, and they all have friction-feed typing units. The Model ASR-33 *TY* is identical to the Model *TZ*, except for the sprocket-feed typing unit. The common 50-cycle versions of the ASR-33 are the ASR-33 *TAC* or *TAJ*, which are identical to the *TZ* unit (the *TAC* comes without stand and chad box). The Model ASR-33 *TBM* is identical to the *TAC* unit with the exception of a sprocket-feed typing unit for pin-feed paper.

None of the Model 33 Teletypes are capable of running in a remote reader/run mode, where the reader may be selected by *external command pulses.* This feature is essential to the operation of the assemblers. This modification is installed via the GRI-99 Teletype mod kit (Model number S40-212). Teletypes that are equipped with an automatic reader control function are not recommended for use with the GRI-99. If a Teletype utilizing this option is to be used with the GRI-99, the X-ON and X-OFF remote reader control functions must be disabled: otherwise, generation of a binary tape by the assembler may cause the reader to turn on or off overriding the remote reader/run control, which is installed with the Teletype mod.

There is a Model ASR-33 *TBE* that has an even-parity feature, which has been disabled. Disabling of the even-parity, although it apparently produces a no-parity tape, does not produce proper code when reading binary tape with the reader. If this Teletype is to be used with the computer, then the disabling of even-parity must be removed and the unit converted back to an even parity unit. The model *TBE* has a momentary reader/run manual select switch on it and is often equipped with the remote reader control functions X-ON and X-OFF.

Before any of the Teletypes can be used with the GRI TTI and TTO option cards, the Teletypes must be converted according to the standard Teletype instructions to a 20-mA current loop operation and to full-duplex operation. New Teletypes generally come wired for 60-mA loop current and simplex operation. Instructions for making this conversion are included in the instructions that come with the Teletype mod kit. If the GRI Teletype mod kit is not used with the Teletype and the user desires to make up his own cables and add his own reader/run relay control, it is recommended that at least the instructions for making the GRI mod be followed explicitly, because the proper grounding of the Teletype and the proper usage of thyractor suppressors on 110V switches is required to ensure safe system operation.

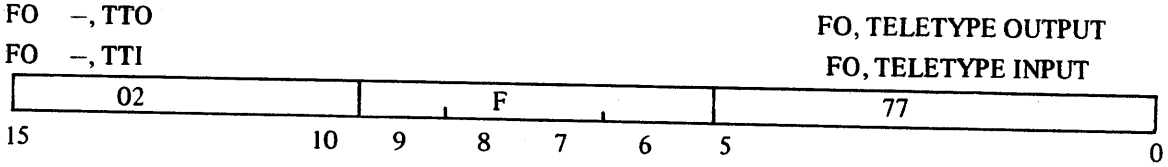
A.1.2 Input/Output Commands

The Teletype separates its input and output functions and is really two distinct devices that share the same device address. Each device has its own Ready and Interrupt Status flags, as well as its own interrupt channel and status bit assignments. Placing a code for a character in the output buffer causes the Teletype to print the character or perform the designated control function. Striking a key places the code for the associated character 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; the most significant bit is always 1, but the printer ignores this bit in characters transmitted to it (e.g., codes 123 and 323 print the same character). Lower case characters (codes 340-376) are not available on the keyboard, 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 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 printer. Horizontal tabs are usually simulated by spaces, with tab settings at preselected columns.

The Teletype input and output both use device address 77, mnemonic TTI or TTO. As the source in a data transmission (or Data Testing) instruction, this code retrieves a character from the Teletype input buffer; as the destination in data transmission, it sends a character to the output buffer. In Function Generating or Testing instructions, it represents both devices.



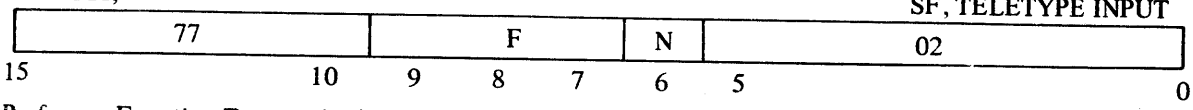
Perform the functions specified by 1s in *F* as follows:

Bit	Mnemonic	Function
6	STRT	Read one character from tape into the input buffer.
7	CLOF	Clear Output Ready.
9	CLIF	Clear Input Ready

Programming 1s in bits 6 and 9 by combining the mnemonics CLIF STRT clears Input Ready and starts the reader.

SF TTO, SF, TELETYPE OUTPUT

SF TTI, SF, TELETYPE INPUT



Perform a Function Test on the flags selected by 1s in *F* as follows:

Bit	Mnemonic	Flag
7	ORDY	Output Ready
9	IRDY	Input Ready

A.1.3 Teletype Output

Output Interrupt Status is bit 0 of the status register, and the Teletype output interrupts to location 11.

Sending a character from bits 0-7 of any source register to the output buffer clears Output Ready (removing the interrupt request) and 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 on tape, with bit 0 corresponding to channel 1 (a 1 produces a hole in the tape). Completion of transmission sets Output Ready, requesting an interrupt if Output Interrupt Status is set.

A.1.3.1 Timing – The Teletype can type or punch up to ten characters per second. After Output Ready is set, the program has 18.18 ms to send another character to keep typing or punching at the maximum rate. The carriage return and line feed, when given in that order, allows sufficient time for the type block to get to the beginning of a new line.

A.1.4 Teletype Input

Input Interrupt Status is bit 1 of the status register, and the Teletype input interrupts to location 14.

Reception from the keyboard requires no initiating action by the program: striking a key clears Input Ready and transmits the code for the character serially to the input buffer. Completion of reception sets Input Ready, requesting an interrupt if Input Interrupt status is set. On retrieving the character in bits 0-7, the program gives an FO CLIF, TTI to clear Input Ready and remove the interrupt request if more input is expected.

If the reader is under program control, giving an FO CLIF STRT, TTI clears Input Ready (removing the interrupt request) and causes the reader to read all eight channels from the next frame on tape. The reader transmits the frame serially to the buffer, with channel 1 corresponding to bit 0 (the presence of a hole produces a 1 in the buffer). Completion of reception sets Input Ready, requesting an interrupt if Input Interrupt Status is set.

A.1.4.1 Timing – After Input Ready is set the character is available for retrieval for 20.45 ms before striking another key can destroy it. If the reader is in use, the program has 20.45 ms to give an FO CLIF STRT, TTI and keep the tape in continuous motion.

A.1.5 Programming Examples

There are basically two procedures for using the Function Testing instructions in a loop to process a series of characters. Consider this loop for typing out characters from a table beginning at location TAB (assume the printer is not in use).

```
OUT:    MRID    TAB-1,TTO     ;Type out
        SF      TTO,ORDY      ;Wait till trans
        JU      .-1           ;mission done
        .                     ;Compute
        .
        .
        JU      OUT           ;Go back
```

This procedure is inefficient because most of the time is spent waiting during the transmission, and there is very little time to do anything afterwards if the next character is to be typed out at full speed. But with this arrangement:

```
OUT:    SF      TTO,ORDY      ;Wait till printer
        JU      .-1           ;free
        MRID    TAB-1,TTO     ;Type out
        .                     ;Computer, etc.
        .
        .
        JU      OUT           ;Go back
```

almost all of the time is useful and full speed is attained, provided only that there is a jump back to OUT before the entire Teletype cycle time is over. Also the first time into the loop, there is a delay until any previous (perhaps unknown) Teletype output operation is finished.

Of course, using the interrupt eliminates all waiting time. Suppose 20 characters are to be typed out (one per location) beginning at TAB, using one of the general-purpose registers to count the characters. The main program might resemble *Example 1* (assume that the program left Output Ready on the last time the Teletype output was used). Hence, an interrupt occurs immediately for the first character. The interrupt routine might resemble *Example 2.* If we do not care whether TRP is affected, the following could be substituted:

```
        JC      XR,ETZ,DONE
```

for testing overflow and loading SC. This method saves no time, but it takes only two locations instead of three.

Example 1 (Setup)

```
        MRI     -24,XR        ;Set up XR: 20(10) = 24(8)
        MRI     014207,TRP    ;Set up channel locations 12, 13
        RM      TRP,12        ;014207 = 06 0010 07 = MRI -, SC.
        MRI     OUT-1,TRP
        RM      TRP,13
        FOI     ICO           ;Turn interrupt on
        ZR      P1,ISR        ;Set TTO ISR bit
        .                     ;Continue program
        .                     ;Locations 12, 13 say MRI OUT-1,SC
        .
```

Example 2 (Interrupt Service)

```
OUT:    MRID    TAB-1,TTO     ;Type out character
        RMI     MSR,O         ;Save machine state
        RS      XR,P1         ;Count character
        SFM     BOV           ;Done yet?
        MRI     DONE-1,SC     ;Yes, go to DONE
        MR      OUT+3,MSR     ;Restore machine state
        FOI     ICO           ;Turn interrupt back on
        MR      11,SC         ;Return to main program
DONE:   MR      OUT+3,MSR     ;Restore machine state
        ZR      ISR           ;Disable interrupt
        MR      11,SC         ;Return
```

Without using the interrupt, the same situation as described above exists for input operations. The following is an example:

```
IN:     FO      CLIF STRT, TTO      ; Read character
        SF      TTI, IRDY           ; Wait till recep-
        JU      .-1                 ; tion done
        RMID    TTI, TAB-1          ; Store character
        .                           ; Decide whether to
        .                           ; read another, etc.
        .
        JU      IN                  ; Go back
```

This sequence is more efficient:

```
IN:     FO      CLIF STRT, TTI      ; Read character
        .                           ; Lots of time
        .
        .
        SF      TTI, IRDY           ; Wait till recep-
        JU      .-1                 ; tion done
        RMID    TTI, TAB-1          ; Store character
        SF      TTO, ORDY           ; Lets make a copy
        JU      .-1                 ; of the tape while
        RR      TTI, TTO            ; we are at it
        FO      CLIF, TTI
        :                           ; Decide whether to
        :                           ; read another
        JU      IN                  ; Do this if want
                                    ; another
        :                           ; Skip to here if
        :                           ; not
```

A.1.6 Operation

A KSR Teletype is actually two independent devices, keyboard and printer, which can be operated simultaneously. An ASR Teletype is really four devices, keyboard, printer, reader and punch, which can be operated in various combinations. Power must be turned on by the operator: the switch is beside the keyboard and is labeled LINE/OFF/LOCAL or ON/OFF and has an unmarked third position opposite ON. When this switch is set to LOCAL or the unmarked position, power is on, but the machine is off line and can be used as 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, and the tape can be moved by hand through the punch mechanism. Pressing BSP 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 are transmitted by using the shift key; most control codes require use of the control key.

The line feed spaces the paper vertically at six lines to the inch, and must be combined with a return to start a new line. The local line feed and return keys affect the printer directly and do not transmit codes. (Appendix B lists the complete Teletype code, ASCII characters, and key combinations.) Pressing the REPT button and striking any character key causes transmission of the corresponding code if 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 and changing the ribbon. An abbreviated version of this procedure is described here for convenience.

A.1.6.1 Tape – The tape moves in the reader from back to front with the feed holes, closer to the left edge. To load tape in the reader:

Step	Procedure
1.	Set the switch to FREE.
2.	Release the cover guard and place the tape so that the sprocket wheel teeth engage the feed holes.
3.	Close the cover guard.
4.	Set the switch to STOP.

To load tape in the punch:

Step	Procedure
1.	Raise the cover.
2.	Feed the tape manually from the top of the roll into the guide at the back.
3.	Move the tape through the punch by turning the friction wheel.
4.	Close the cover.
5.	Turn on the punch with the unit in LOCAL and punch about 2 feet of leader by pressing HERE IS or the BREAK key to generate null codes.

A.1.6.2 Paper – The printer has an 8½-in. 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.

A.1.6.3 Ribbon – Replace the ribbon whenever it becomes worn or frayed or the printing becomes too light. To replace the ribbon:

Step	Procedure
1.	Disengage the old ribbon from the ribbon guides on either side of the type block.
2.	Remove the reels by lifting the spring clips on the reel spindles and pulling the reels off.
3.	Remove the old ribbon from one of the reels and replace the empty reel on one side of the machine.
4.	Install a new reel on the other side.
5.	Push down both reel spindle spring clips to secure the reels.
6.	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.
7.	Engage the hook on the end of the ribbon over the point of the arrow in the hub.

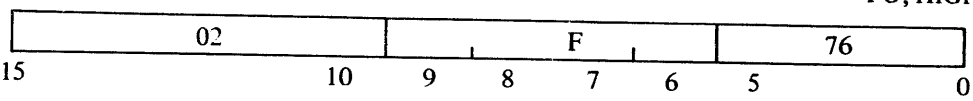
Step	Procedure
8.	Wind a few turns of the ribbon to make sure that the reversing eyelet has been wound onto the spool.
9.	Make sure the ribbon is seated properly and feeds correctly.

A.2 PAPER-TAPE READER AND PUNCH

The high-speed reader and punch are totally separate devices, but they share a single device address code in the same manner that the Teletype input and output do. The common device address is 76, mnemonic HSR or HSP. Each interface contains an 8-bit buffer that corresponds to bits 0-7 of a computer word; the reader buffer is addressable as a source of data, the punch buffer as a destination.

FO –, HSR — FO, HIGH SPEED READER

FO –, HSP — FO, HIGH SPEED PUNCH



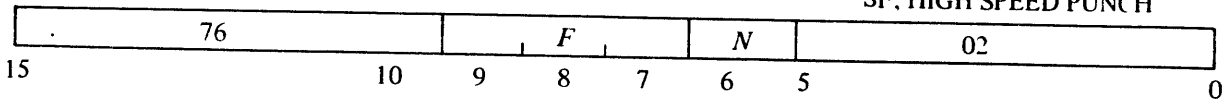
Perform the functions specified by 1s in *F* as follows:

Bit	Mnemonic	Function
6	STRT	Read one character from tape into the reader buffer.
7	CLOF	Clear Punch (Output) Ready.
9	CLIF	Clear Reader (Input) Ready.

Programming 1s in bits 6 and 9, combining the mnemonics CLIF STRT, clears Reader Ready and starts the reader.

SF HSR, — SF, HIGH SPEED READER

SF, HSP, — SF, HIGH SPEED PUNCH



Perform a function test on the flags selected by 1s in *F* as follows:

Bit	Mnemonic	Flag
7	ORDY	Punch Ready
9	IRDY	Reader Ready

A.2.1 Paper-Tape Reader

The reader reads 8-channel perforated paper or mylar tape photoelectrically at a speed of 300 frames per second. Reader Interrupt Status is bit 3 of the status register, and the reader interrupts to Location 22.

Giving an FO CLIF STRT, HSR clears Ready (removing the interrupt request) and causes the reader to read all eight channels from the next frame on tape into the buffer. Channel 1 corresponds to bit 0 (the presence of a hole produces a 1 in the buffer). When the operation is complete the reader sets Ready, requesting an interrupt if Interrupt Status is set.

A.2.1.1 Timing – At 300 frames per second the reader takes 3.3 ms per character, but the program must read several frames before the reader reaches maximum speed. After Ready is set, the program has 1.5 ms to retrieve the character and give an FO CLIF STRT, HSR to keep the tape in continuous motion. Waiting longer forces the reader to operate at a speed no greater than 150 frames per second.

A.2.1.2 **Operation** – Unoiled paper tape can be used, but it must be opaque. To load the reader:

Step	Procedure
1.	Place the fanfold tape stack vertically in the bin at the right, oriented so that the front end of the tape is nearer the read head and the feed holes are away from you.
2.	Lift the gate and take three or four folds of tape from the bin.
3.	Slip the tape into the reader from the front.
4.	Carefully line up the feed holes with the sprocket teeth to avoid damaging the tape, and close the gate.
5.	Make sure that the part of the tape in the left bin is placed to correspond to the folds, otherwise it will not stack properly.
6.	Turn on the power switch so the reader can respond to the program.

A.2.2 Paper-Tape Punch

The punch perforates 8-channel paper tape at speeds up to 60 frames per second. Interrupt Status is bit 2 of the status register, and the punch interrupts to location 17.

Sending a character from bits 0-7 of any source register to the punch buffer clears Ready (removing the interrupt request) and causes the punch to punch the contents of the buffer in the tape, with bit 0 corresponding to channel 1 (a 1 produces a hole in the tape). After punching is complete, the device sets Ready, requesting an interrupt if Interrupt Status is set.

A.2.2.1 **Timing** – Punching is synchronized to a punch cycle of 16.7 ms. After Ready sets, the program has 10 ms to send another character to keep punching at the maximum rate; after 10 ms punching is delayed until the next cycle.

A.2.2.2 **Example** – With direct function processing a program for duplicating a tape is quite simple:

```
DUP:    FO      CLIF STRT, HSR      ; Read
        SF      HSR, IRDY           ; Wait for character
        JU      .-1

        SF      HSP, ORDY           ; Got it, wait for
        JU      .-1                 ; punch

        RR      HSR, HSP            ; Move character to
                                    ; punch
        JU      DUP                 ; Read another
```

A.2.2.3 **Operation** – Punch power must be on all the time that the punch might be used; otherwise it will not respond to the program. 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 can 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 depressed (provided power is on).

To load tape:

Step	Procedure
1.	Empty the chad box.
2.	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).
3.	Set the box in the frame and thread the tape through the punch mechanism.

Step	Procedure
4.	The arrows on the tape should be on the bottom and should point in the direction of tape motion. If they are on top, 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.
5.	To facilitate loading, tear or cut the tape carefully at one of the folds. Thread the tape under the out-of-tape plate.
6.	Open the guide plate (over the sprocket wheel), and push the tape beyond the sprocket wheel.
7.	Close the guide plate.
8.	Press the feed button long enough to punch about a foot and a half of leader. Make sure the tape feed holes are in a straight line. If not, tear the tape and rethread through the punch (steps 5-7).

To remove a length of perforated tape:

Step	Procedure
1.	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).
2.	Tear the tape at a fold within the area in which only feed holes are punched.
3.	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.

APPENDIX B
CODES

B.1 DEVICE SELECTION CODES

The GRI-99 architecture allows 6 bits for addressing source and destination operators. This provides a range of 00_8 – 77_8 or 64_{10} addresses of each type. Several of these addresses are used in conjunction with the basic machine and others are assigned to some of the most popular options. Those addresses currently assigned are included in the Table B-1.

Table B-1

DEVICE ADDRESSES

Device Address	Abbreviation	Description	Source or Destination of Data
00	–	Null	S
00	–	Null	D
01	IR	Instruction Register	S
01	IR	Instruction Register	D**
02	FO	Function Output	S
02	SF	Sense Function	D
03*	–	Data Tests	D
04	ISR	Interrupt Status Register	S
04	ISR	Interrupt Status Register	D
05	MA	Memory Address	S
05	MA	Memory Address	D**
06	MB	Memory Buffer	S
06	MB	Memory Buffer	D
07	SC	Sequence Counter	S
07	SC	Sequence Counter	D
10	SWR	Data Switch Register	S
11	AX	AX Register	S
11	AX	AX Register	D
12	AY	AY Register	S
12	AY	AY Register	D
13	AO	Arithmetic Operator	S
13	AO	Arithmetic Operator	D**
14	–	Unused	–
14	EAO	Extended Arithmetic Operator	D**
15	ED	External Data	S
15	ED	External Data	D**
16	EAS	External Address	S
16	EAS	External Address	D**
17	MSR	Machine Status	S
17	MSR	Machine Status	D
20	–	Unused	S
20	–	Unused	D
21	–	Unused	–
21	–	Unused	–
22	XR	Index Register	–
22	XR	Index Register	D
23	TRP	Trap Register	S
23	TRP	Trap Register	D
24	BSW	Byte Swap	S
24	BSW	Byte Swap	D

*Alternate address for Trap Register as a *source*, or as destination for a memory reference.

**It appears as DDA in a data transfer instruction, it is equivalent to sending data to the NULL (0) register. Except for MA (05), ED (15), and EAS (16), these generally appear as DDA for an FO instruction.

Table B-1 (Cont.)

DEVICE ADDRESSES

Device Address	Abbreviation	Description	Source or Destination of Data
25	BPK	Byte Pack	S
25	BPK	Byte Pack	D
26	BCA	Byte Comparator A	S
26	BCA	Byte Comparator A	D
27	BCB	Byte Comparator B	S
27	BCB	Byte Comparator B	D
30	GP1	General Purpose Register 1	S
30	GP1	General Purpose Register 1	D
31	GP2	General Purpose Register 2	S
31	GP2	General Purpose Register 2	D
32	GP3	General Purpose Register 3	S
32	GP3	General Purpose Register 3	D
33	GP4	General Purpose Register 4	S
33	GP4	General Purpose Register 4	D
34	GP5	General Purpose Register 5	S
34	GP5	General Purpose Register 5	D
35	GP6	General Purpose Register 6	S
35	GP6	General Purpose Register 6	D
36	–	Unused	S
36	–	Unused	D
37	–	Unused	S
37	–	Unused	D
40	–	Unused	S
40	–	Unused	D
41	–	Unused	S
41	–	Unused	D
42	–	Unused	S
42	–	Unused	D
43	–	Unused	S
43	–	Unused	D
44	–	Unused	S
44	–	Unused	D
45	–	Unused	S
45	–	Unused	D
46	–	Unused	S
46	–	Unused	D
47	–	Unused	S
47	–	Unused	D
50	BIM	Binary Input Mux	S
51	BOM	Binary Output Mux	D
52	–	Unused	S
52	–	Unused	D
53	–	Unused	S
53	–	Unused	D
54	–	Unused	S
54	–	Unused	D
55	CRDR	80 Col Card Reader	S
55	–	Unused	
56	–	Unused	S
56	–	Unused	D
57	CARD	80 Col Card Reader (CAR)	S
57	CARD	80 Col Card Reader (CAR)	D
60	WIT	Watchdog Interval Timer	S
60	WIT	Watchdog Interval Timer	D

Table B-1 (Cont.)

DEVICE ADDRESSES

Device Address	Abbreviation	Description	Source or Destination of Data
61	DAC	D/A Converter	D
62	GOR	General Output Register	S
62	GOR	General Output Register	D
63	GI	Gate Input Register	S
64	MUX	Multiplexer	S
64	MUX	Multiplexer	D
65	ADC	A/D Converter	S
65	–	Unused	D
66	WCT	Disk Word Count	S
66	WCT	Disk Word Count	D
67	CAD	Disk Core Address	S
67	CAD	Disk Core Address	D
70	DISK	Disk Controller	S
70	DISK	Disk Controller	D
71	LPR	Line Printer	D
72	–	Unused	S
72	–	Unused	D
73	CRD	Card Reader	S
73	CRD	Card Reader	D
74	GRI	Grisette II	S
74	GRI	Grisette II	D
75	RTC	Real-Time Clock	(DMA Only)
75	RTC	Real-Time Clock	
76	HSP	High-speed Reader	S
76	HSP	High-speed Punch	D
77	TTI	Teletype Input	S
77	TTO	Teletype Output	D
*NA	PID	Pulse Input Detector	

B.2 INTERRUPT STATUS AND TRAPS

Interrupt devices can interrupt (TRAP) to locate 0 or to a memory location of choice. These devices utilize one bit of the interrupt status register (ISR) for ease of program interrupt control. Those trap locations and status bits currently assigned and considered standard are included in Table B-2.

Table B-2

TRAP LOCATIONS

Interrupt				
Status Bit	Trap Location	Device	Abbreviation	Device Address
	0	Power Failure		00
	0	Breakpoint		01
0	11	Teletype Output	TTO	77
1	14	Teletype Input	TTI	77
2	17	High-speed Punch	HSP	76
3	22	High-speed Reader	HSR	76
4	25	Card Reader	CRD	73
5	30	Line Printer	LPR	71

*NA – not applicable

Table B-2 (Cont.)

TRAP LOCATIONS

Interrupt		Device	Abbreviation	Operator Code
Status Bit	Trap Location			
6	36*	General Output Register	GOR	62
	44*	Gate Input Card	GI	63
7	33	80 Col Card Reader	CRDR	55/57
8	104-124	Pulse Input Detector	PID	NA
9	44	Grisette-write	–	–
10	47	Grisette-read	–	–
11	100	Real-time Clock	RTC	75
12	52	A/D Converter	ADC	65
13	–	Unused	–	–
14	53-55	Disk	DISK	66-70
15	25	Watchdog Interval Timer	WIT	61

*For test purposes only.

B.3 TELETYPE CODES

Table B-3 lists the complete Teletype code set. Codes generated by the keyboard may have a 1 or 0 in the most significant bit depending on the Teletype model. For no parity Teletypes, the eighth channel is always punched 1's. In the case of even parity, the eighth channel is either punched or not punched, depending on the number of bits in the particular frame.

The lower-case character set (codes 340-376) is not available on the Model 33. Specifying one of the lower-case codes causes the Teletype to print the corresponding upper case character. Definitions of control codes are those given by the ASCII code set. Most control codes, however, have no effect on the Teletype and their definitions bear no necessary relation to the use of the codes in conjunction with the GRI-99 software.

Table B-3

TELETYPE CODES (No Parity TTY)

8-Bit Octal Code	Character	Remarks
200	NUL	Null, tape feed. Control shift P.
201	SOH	Start of heading; also start of message (SOM). Control A.
202	STX	Start of text; also end of address (EOA). Control B.
203	ETX	End of text; also end of message (EOM). Control C.
204	EOT	End of transmission; shuts off TWX machines. Control D.
205	ENQ	Inquiry; also WRU, "Who are you?" Triggers identification ("Here is .....".)
206	ACK	Acknowledge; also RU, "Are you ...?" Control F.
207	BEL	Rings the bell. Control G.
210	BS	Backspace; also format effector (FEO). Backspaces some machines. Control H.
211	HT	Horizontal tab. Control I.
212	LF	Line feed or line space; advances paper to next line. Duplicated by Control J.
213	VT	Vertical tab. Control K.
214	FF	Form feed to top of next page. Control L.
215	CR	Carriage return to beginning of line. Control M.
216	SO	Shift out; changes ribbon color to red. Control N.
217	SI	Shift in; changes ribbon color to black. Control O.

Table B-3 (Cont.)

TELETYPE CODES (No Parity TTY)

8-Bit Octal Code	Character	Remarks
220	DLE	Data link escape. Control P (DCO).
221	DC1	Device control 1, turns transmitter (reader) on. Control Q. (X ON).
222	DC2	Device control 2, turns punch or auxiliary on. Control R (TAPE, AUX ON).
223	DC3	Device control 3, turns transmitter (reader) off. Control S (X OFF).
224	DC4	Device control 4, turns punch or auxiliary off. Control T (AUX OFF).
225	NAK	Negative acknowledge; also error (ERR). Control U.
226	SYN	Synchronous idle. Control V.
227	ETB	End of transmission block; also logical end of medium (LEM). Control W.
230	CAN	Cancel. Control X.
231	EM	End of medium. Control Y.
232	SUB	Substitute. Control Z.
233	ESC	Escape, prefix. This code is also generated by control shift K.
234	FS	File separator. Control shift L.
235	GS	Group separator. Control shift M.
236	RS	Record separator. Control shift N.
237	US	Unit separator. Control shift O.
240	SP	Space
241	!	
242	"	
243	#	
244	$	
245	%	
246	&	
247	'	Accent acute or apostrophe.
250	(	
251	)	
252	*	
253	+	
254	,	Comma.
255	–	
256	.	
257	/	
260	0	
261	1	
262	2	
263	3	
264	4	
265	5	
266	6	
267	7	
270	8	
271	9	
272	:	
273	;	
274	<	
275	=	
276	>	
277	?	

Table B-3 (Cont.)

TELETYPE CODES (No Parity TTY)

8-Bit Octal Code	Character	Remarks
300	@	
301	A	
302	B	
303	C	
304	D	
305	E	
306	F	
307	G	
310	H	
311	I	
312	J	
313	K	
314	L	
315	M	
316	N	
317	O	
320	P	
321	Q	
322	R	
323	S	
324	T	
325	U	
326	V	
327	W	
330	X	
331	Y	
332	Z	
333	[	Shift K.
334	/	Shift L.
335	]	Shift M.
336	↑	
337	←	
340	'	Accent grave.
341	a	
342	b	
343	c	
344	d	
345	e	
346	f	
347	g	
350	h	
351	i	
352	j	
353	k	
354	l	
355	m	
356	n	
357	o	
360	p	
361	q	
362	r	
363	s	
364	t	

Table B-3 (Cont.)

TELETYPE CODES (No Parity TTY)

8-Bit Octal Code	Character	Remarks
365	u	
366	v	
367	w	
370	x	
371	y	
372	z	
373		
374		
375		On early versions, either of these codes may be generated by either the ALT MODE or ESC key.
376		
377	DEL	Delete, rub out.

Teletype Keys That Generate No Codes

REPT	Causes any other key that is struck to repeat continuously until REPT is released.
LOC LF	Local line feed.
LOC CR	Local carriage return.
BREAK	Opens the line (machine sends a continuous string of null characters).
BRK RLS	Break release (not applicable).
HERE IS	Transmits predetermined 20-character message.

APPENDIX C
LOADERS

Before a program can be executed, it must be brought into memory. To bring a program into memory, a loading program must already reside in core. If the memory is empty, the console switches are used to load in a bootstrap loader, which is ordinarily used only to bring in a more extensive absolute loader. This absolute loader program is then used to read the object GRI-99 software. If an undebugged user routine accidentally destroys the absolute loader, it can be restored by first reloading the bootstrap manually. There are several bootstrap loaders, depending on which functional devices are included in the system.

For a complete description of all loaders, refer to the *GRI-99 Loaders Manual.*

NOTE

In the loader descriptions, the letter X designates the portion of the address that varies according to the core size of the individual machine. The digits that replace the X's are determined by the highest locations in core. For example: if the machine has 4K of core memory and a load address of X7555 is specified, the actual load address is 07555. (The highest location is 07777 for a 4K machine.)

Specific addresses are assigned for using the bootstrap and absolute loader tapes. The tape number specifies the tape format. The tape number is in the following form:

7 n – nn – nnnY-Z

where:

7 = Software engineering
n = Numeric designations as to category and release order
Y = Letter designating tape format
- A = Absolute
- B = Bootstrap
- D = Directory
- R = Relocatable
- S = Absolute source
- X = Relocatable Source

Z = Letter or number designating revision level

C.1 BOOTSTRAP LOADER (%BLD)

The primary purpose of the bootstrap loader is to load the absolute loader (%ALH); however, for the purposes of this manual the basic processor bootstrap loader, which cannot be used to load %ALH, is described first. Section C.1.2 describes the version of %BLD that is used to load %ALH.

C.1.1 Basic Processor Bootstrap Loader

The basic bootstrap loader reads a bootstrap format object from paper tape and loads it into memory. No tests are performed on the data from the reader to verify that the data have been read correctly.

The basic processor bootstrap loader does not use the A0 registers or functions. As a result, this loader is useful for troubleshooting if the A0 is faulty (or is suspected) and the A0 diagnostic must be loaded (D99P1 or D99P2).

To load the basic processor bootstrap loader, proceed as follows:

NOTE

If the loader is to be used with a reader other than the Teletype reader, all reference to device address 77 must be changed to the new device address (e.g., for the high-speed reader, change 77 to 76).

Step	Procedure
1.	Key in the following sequence of instructions:

Location	Instruction
X7605	02 0100 00
X7606	02 1001 77
X7607	77 1000 02
X7610	00 0100 03
X7611	0X7607
X7612	77 0100 03
X7613	0X7605
X7614	77 0000 06
X7615	0X7637
X7616	77 0000 06
X7617	0X7650
X7620	02 1001 77
X7621	77 1000 02
X7622	00 0100 03
X7623	0X7621
X7624	77 0001 06
X7625	0X7646
X7626	02 1001 77
X7627	77 1000 02
X7630	00 0100 03
X7631	0X7627
X7632	77 0000 06
X7633	0X7635
X7634	06 1010 06
X7635	0
X7636	06 1110 06
X7637	0
X7640	00 0100 02
X7641	00 0100 03
X7642	0X7634
X7643	06 1000 06
X7644	0X7635
X7645	06 1000 06
X7646	
X7647	06 1110 06
X7650	0
X7651	00 0100 02
X7652	00 0100 03
X7653	0X7643
X7654	00 0100 03
X7655	0X7606

Key load address – 1 into X7646.
Start at X7606.

Step	Procedure
2.	Set X7646 in the SWR.
3.	Set the DEVICE SELECT switches to 07 and depress TRM.
4.	Set the first address – 1 of the program to be loaded in the SWR.
5.	Push the WR key up.
6.	Set X7606 in the SWR.
7.	Depress TRM.
8.	Mount the bootstrap format tape in the reader and turn the reader on.
9.	Depress START. The loader halts each time it reads a zero word until it reads a non-zero word.
10.	Depress CONT as many times as necessary.

NOTE

This version of the bootstrap loader is for basic diagnostics purposes only and cannot be used to load the absolute loader. Use the version detailed in Section C.1.2.

C.1.2 %BLD to Load %ALH

It is assumed that after the absolute loader has been loaded, the bootstrap loader is no longer necessary; therefore, location zero has been chosen as the starting address. In this location, the bootstrap will most likely be destroyed by other programs loaded later. To locate the loader in a different core area, simply increase all addresses by the value of the address that is chosen to start keying in the program (e.g., to key in at 7000, add 7000 to all internal addresses).

To load the version of %BLD that is used to load %ALH proceed as follows:

NOTE

If the loader is to be used with a reader other than the Teletype reader, all references to device address 77 must be changed to the new device address (e.g., for the high-speed reader, change 77 to 76).

Step	Procedure
1.	Key in the following:

```
00000 00 0100 03    JU    GETO
00001 000023
00002 11 1000 12    RR    AX, L1, AY
00003 12 1000 12    RS    AY, L1
00004 12 1000 12    RS    AY, L1
00005 12 1000 12    RS    AY, L1
00006 12 1000 12    RS    AY, L1
00007 12 1000 12    RS    AY, L1
00010 12 1000 12    RS    AY, L1
00011 12 1000 12    RS    AY, L1
00012 00 0100 03    JU    GETO
00013 000023
                    ; FOR OTHER THAN %ALH SET
                    ; SECOND WORD OF NEXT INSTRUCTION
                    : TO ADDRESS –1 OF LOAD ADDRESS
```

Step	Procedure		
	00014 13 0011 06	RMID	AO
	00015 000014		
	00016 00 0100 03	START: JU	GETO ; START PROGRAM HERE
	00017 000023		
	00020 11 0110 03	JC	AX, NEZ, .−20; PRESS CONTINUE
	00021 000000		
	00022 02 0100 00	FOM	HLT ; UNTIL NON 0 READ
	00023 02 1001 77	GETO: FO	CLIF STRT, TTI; OR HSR
	00024 77 1000 02	SF	TTI, IRDY
	00025 06 0010 07	MRI	−2, SC
	00026 000023		
	00027 77 0000 11	RR	TTI, AX
	00030 03 0000 07	RR	TRP, SC
2.	Set 16 on the SWR.		
3.	Depress TRM.		
4.	Depress START once. The loader halts each time it reads a zero word until it reads a non-zero word.		

C.1.3 Bootstrap Tape Format

The bootstrap tape format is:

Blank Tape

.
.
.

Space
Control Code (200_8)
Bits 15-8 of Data Word 1
Bits 7-0 of Data Word 1
Control Code (200_8)
Bits 15-8 of Data Word 2
Bits 7-0 of Data Word 2

.
.
.

Space

C.2 ABSOLUTE LOADER (%ALH)

The absolute loader loads the user's object program into memory. It differs from the bootstrap loader in that it loads a tape of a different format, checks to ensure correct loading, and is capable of loading data into non-sequential areas of memory.

An object tape in absolute format consists of a series of data blocks as follows:

Space	
Control Code	(001)
Checksum	(2 frames)
Block Start Address	(2 frames)
Data Word Count	(2 frames)

Data Word	(2 frames per word)
.	
.	
.	
.	
.	
Data Word	
Block Trailer	(20_8 null frames)

The *Control Code* indicates the beginning of a block. The *Checksum* is the 16-bit sum, ignoring overflows, of the *Block Start Address*, the *Data Word Count*, and the *Data Words*. The *Block Start Address* is the first location into which this block of data is to be loaded sequentially. The *Data Word Count* is the number of data words contained in the block.

C.2.1 Using %ALH

The following procedure is used to read a tape in absolute format:

Step	Procedure
1.	Using the key-in bootstrap loader, load %ALH into memory.
2.	Set 07 on the DEVICE SELECT switches.
3.	Set X7661 on the SWR.
4.	Depress TRM.
5.	Position bit 15 of the SWR to select a reader, 0 = High-speed reader 1 = Teletype reader
6.	Mount the absolute format object tape in the reader. Ensure a null frame preceding the first block is under the head.
7.	Depress START.
8.	If the SC = X7661, the program has loaded correctly, and %ALH is ready to load another. Begin at Step 3.
9.	If execution is halted at another location, there is an error: *a.* SC = X7676 indicates a control code error. Reposition to beginning of block and go to Step 8. *b.* SC = X7733 indicates a checksum error. Reposition to beginning of block and depress START.

NOTE

The last location loaded is X7720.

The absolute loader (block loader) is available in many versions, depending on the available firmware and amount of memory. Refer to the *GRI-99 Loaders Manual* for a complete description.

APPENDIX D
RAS EXAMPLES

To show the wide variety of individual instructions available for the GRI-99, the following paragraphs list all the variations of the basic instruction types for a single pair of devices (in this case, the AX register and general-purpose register 1). The assemblers recognize mnemonic or structural forms for 11 instruction types, as follows:

1. Register to Register
2. Zero to Register
3. Register to Self
4. Register to Memory
5. Zero to Memory
6. Memory to Register
7. Memory to Self
8. Conditional Jump
9. Unconditional Jump
10. Function Generate
11. Sense Function

Indexing examples are shown in the simplest form. Increment, complement, and shift are variations on the basic format.

D.1 REGISTER TO REGISTER

Data transmission from one register to another is the fastest and most commonly used class of machine instruction. The modifications performed on the data by the Bus Modifier while the data are in transit expand this form into the set listed below. For the purpose of these examples, the arithmetic register AX is being sent to general-purpose register 1 (GR1)

Data Modification	RAS	
None	RR	AX, GR1
Increment by 1	RR	AX, P1, GR1
Shift left 1	RR	AX, L1, GR1
Shift right 1	RR	AX, R1, GR1
Complement	RRC	AX, GR1
Complement and increment by 1	RRC	AX, P1, GR1
Complement and shift left 1	RRC	AX, L1, GR1
Complement and shift right 1	RRC	AX, R1, GR1

D.2 ZERO TO REGISTER

This instruction is a special case of the register to register instruction where the source is the null device. With no source register delivering data to the Destination Bus, the data supplied to the bus modifier is zero, and the following set of instructions results.

Data Modification	RAS	
None	ZR	AX
Increment by 1	ZR	P1, AX
Shift left 1	ZR	L1, AX
Shift right 1	ZR	R1, AX
Complement	ZRC	AX
Complement and increment by 1	ZRC	P1, AX
Complement and shift left 1	ZRC	L1, AX
Complement and shift right 1	ZRC	R1, AX

D.3 REGISTER TO SELF

The Bus Modifier can be made to act directly on the contents of a single register by specifying that register as both source and destination. This special case of register to register gives this instruction set.

Data Modification		RAS
Increment by 1	RS	AX, P1
Shift left 1	RS	AX, L1
Shift right 1	RS	AX, R1
Complement	RSC	AX
Complement and increment by 1	RSC	AX, P1
Complement and shift left 1	RSC	AX, L1
Complement and shift right 1	RSC	AX, R1

D.4 REGISTER TO MEMORY

To transfer data from a register to memory, the memory buffer is designated as the destination. The next consecutive memory location in the program is then used as a memory address or as an immediate data sortage location. This instruction type expands into the set given here when memory addressing modes and Bus Modifier options are considered. AX is the source, *MDATA* is the address of a location containing data, and *ADATA* is the address of a location containing a deferred address. In immediate mode, the second instruction location will be used to receive data, so we specify its contents initially as 0.

Memory Addressing	Data Modification		RAS
Direct	None	RM	AX, *MDATA*
Direct	Increment by 1	RM	AX, P1, *MDATA*
Direct	Left 1	RM	AX, L1, *MDATA*
Direct	Right 1	RM	AX, *MDATA*
Direct Indexed	None	RM	AX, *#MDATA*
Deferred	None	RMD	AX, *ADATA*
Deferred	Increment by 1	RMD	AX, P1, *ADATA*
Deferred	Left 1	RMD	AX, L1, *ADATA*
Deferred	Right 1	RMD	AX, R1, *ADATA*
Deferred Indexed	None	RMD	AX, *#ADATA*
Immediate	None	RMI	AX, 0
Immediate	Increment by 1	RMI	AX, P1, 0
Immediate	Left 1	RMI	AX, L1, 0
Immediate	Right 1	RMI	AX, R1, 0
Immediate deferred	None	RMID	AX, *ADATA*
Immediate deferred	Increment by 1	RMID	AX, P1, *ADATA*
Immediate deferred	Left 1	RMID	AX, L1, *ADATA*
Immediate deferred	Right 1	RMID	AX, R1, *ADATA*
Immediate deferred Indexed	None	RMID	AX, *#ADATA*

D.5 ZERO TO MEMORY

This is a special case of register to memory where the source is the null device. The source word is therefore zero and the data sent to memory depends only on the operations performed in the Bus Modifier. The set of instructions for this special case are given using the same terminology as for register to memory.

Memory Addressing	Data Modification		RAS
Direct	None	ZM	*MDATA*
Direct	Increment by 1	ZM	P1, *MDATA*
Direct	Left 1	ZM	L1, *MDATA*
Direct	Right 1	ZM	R1, *MDATA*
Deferred	None	ZMD	*ADATA*
Deferred	Increment by 1	ZMD	P1, *ADATA*
Deferred	Left 1	ZMD	L1, *ADATA*
Deferred	Right 1	ZMD	R1, *ADATA*
Immediate	None	ZMI	0
Immediate	Increment by 1	ZMI	P1, 0

Memory Addressing	Data Modification	RAS	
Immediate	Left 1	ZMI	L1,0
Immediate	Right 1	ZMI	R1,0
Immediate deferred	None	ZMID	*ADATA*
Immediate deferred	Increment by 1	ZMID	P1, *ADATA*
Immediate deferred	Left 1	ZMID	L1, *ADATA*
Immediate deferred	Right 1	ZMID	R1, *ADATA*

D.6 MEMORY TO REGISTER

This instruction type is the exact inverse of register to memory discussed above. *MDATA* and *ADATA* have the same meaning as before, but in immediate mode we must specify the data to be supplied from the second instruction location. For an example, 5 is used as the data word.

Memory Addressing	Data Modification	RAS	
Direct	None	MR	*MDATA*, AX
Direct	Increment by 1	MR	*MDATA*, P1, AX
Direct	Left 1	MR	*MDATA*, L1, AX
Direct	Right 1	MR	*MDATA*, R1, AX
Direct Indexed	None	MR	*#MDATA*, AX
Deferred	None	MRD	*ADATA*, AX
Deferred	Increment by 1	MRD	*ADATA*, P1, AX
Deferred	Left 1	MRD	*ADATA*, L1, AX
Deferred	Right 1	MRD	*ADATA*, R1, AX

Memory Addressing	Data Modification	RAS	
Deferred Indexed	None	MRD	*#ADATA*, AX
Immediate	None	MRI	5, AX
Immediate	Increment by 1	MRI	5, P1, AX
Immediate	Left 1	MRI	5, L1, AX
Immediate	Right 1	MRI	5, R1, AX
Immediate deferred	None	MRID	*ADATA*, AX
Immediate deferred	Increment by 1	MRID	*ADATA*, P1, AX
Immediate deferred	Left 1	MRID	*ADATA*, L1, AX
Immediate deferred	Right 1	MRID	*ADATA*, R1, AX
Immediate deferred Indexed	None	MRID	*#ADATA*, AX

D.7 MEMORY TO SELF

A memory reference instruction can access only one memory data location, but when the memory buffer is designated as both source and destination, the contents of that location can be modified directly, giving this instruction set.

Memory Addressing	Data Modification	RAS	
Direct	Increment by 1	MS	*MDATA*, P1
Direct	Left 1	MS	*MDATA*, L1
Direct	Right 1	MS	*MDATA*, R1
Deferred	Increment by 1	MSD	*ADATA*, P1
Deferred	Left 1	MSD	*ADATA*, L1
Deferred	Right 1	MSD	*ADATA*, R1
Immediate	Increment by 1	MSI	0, P1
Immediate	Left 1	MSI	0, L1
Immediate	Right 1	MSI	0, R1
Immediate deferred	None	MSID	*ADATA*
Immediate deferred	Increment by 1	MSID	*ADATA*, P1
Immediate deferred	Left 1	MSID	*ADATA*, L1
Immediate deferred	Right 1	MSID	*ADATA*, R1

D.8 CONDITIONAL JUMP

This is a Data Test instruction. The source data are sent directly to the data tester and is not subject to action by the Bus Modifier. Jump addressing and conditions for testing the data produce this instruction set.

Memory Addressing	Data Test	RAS	
Direct	Data = 0	JC	AX, ETZ, BEGIN
Direct	Data < 0	JC	AX, LTZ, BEGIN
Direct	Data ≤ 0	JC	AX, LEZ, BEGIN
Direct	Data ≠ 0	JC	AX, NEZ, BEGIN
Direct	Data ≥ 0	JC	AX, GEZ, BEGIN
Direct	Data > 0	JC	AX, GTZ, BEGIN
Deferred	Data = 0	JCD	AX, ETZ, BEGIN
Deferred	Data < 0	JCD	AX, LTZ, ABGIN
Deferred	Data ≤ 0	JCD	AX, LEZ, ABGIN
Deferred	Data ≠ 0	JCD	AX, NEZ, ABGIN
Deferred	Data ≥ 0	JCD	AX, GEZ, ABGIN
Deferred	Data > 0	JCD	AX, GTZ, ABGIN

D.9 UNCONDITIONAL JUMP

In BASE, the unconditional jump is written as follows:

Memory Addressing	RAS	
Direct	JU	BEGIN
Deferred	JUD	ABGIN

D.10 FUNCTION GENERATE

The four function bits in an FO instruction word permit up to 16 unique FO instructions per destination device address. These bit combinations can be given symbolic names or can simply be written in octal in RASX. The instruction set shown here generates functions for a device with a DDA mnemonic of DEV.

RAS		RAS	
FO	0, DEV	FO	10, DEV
FO	1, DEV	FO	11, DEV
FO	2, DEV	FO	12, DEV
FO	3, DEV	FO	13, DEV
FO	4, DEV	FO	14, DEV
FO	5, DEV	FO	15, DEV
FO	6, DEV	FO	16, DEV
FO	7, DEV	FO	17, DEV

D.11 SENSE FUNCTION

The four control bits in an SF instruction word allow up to 14 different skip instructions per source device address. The rightmost bit determines whether the skip shall occur if any condition specified by the other three is satisfied, or if no condition specified by them is satisfied. The conditions may be given in octal as is done here in RAS.

RAS		RAS	
SF	DEV, 2	SF	DEV, NOT 2
SF	DEV, 4	SF	DEV, NOT 4
SF	DEV, 6	SF	DEV, NOT 6
SF	DEV, 10	SF	DEV, NOT 10
SF	DEV, 12	SF	DEV, NOT 12
SF	DEV, 14	SF	DEV, NOT 14
SF	DEV, 16	SF	DEV, NOT 16