6942A Multiprogrammer USER'S GUIDE

HP-IB MULTIPROGRAMMER

HEWLETT PACKARD

HP Part Number 06942-90003

6942A MULTIPROGRAMMER

USER'S GUIDE

HP Part No. 06942-90003

November 1979

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Chapter I
INTRODUCTION

1-1 SCOPE OF THIS USER'S GUIDE

1-2 This User's Guide provides programming, operating, installation, and verification procedures for the entire 6942A Multiprogrammer system (consisting of the 6942A Multiprogrammer, 6943A Extender(s), and complete family of I/O cards). The Multiprogrammer is designed to operate on the Hewlett-Packard Interface Bus (HP-IB) under control of a Desktop Computer or System Computer (HP-IB Controller). Note that the intent of this guide is to provide all of the programming information that is pertinent to the Multiprogrammer System. It does not cover the fundamentals of programming your Controller nor does it describe the structure and operation of the HP-IB. Refer to "Related Publications" for a list of other documents that are presently available for these items.

1-3 The following is an abbreviated listing of all of the major information contained in this guide together with a brief description concerning the scope of this material.

Installation and Checkout. Chapter 2 contains all of the information necessary for interconnecting the controller and the Multiprogrammer units on the HP-IB. Checkout procedures, using the Multiprogrammer's self test feature and the uitlity program cartridges supplied with the User's Guide, verify that the equipment is installed correctly and can be programmed by the controller.

Getting Started. Chapter 3 allows a first time user of the Multiprogrammer to get acquainted with the unit. It contains basic operating and programming information and includes programming examples using a 9825,35, or 45 Desktop Computer.

Programming Concepts. Chapter 4 contains HP-IB message concepts and condensed descriptions of the Multiprogrammer operating concepts that you will need to effectively program the 6942A.

Programming Essentials. Chapter 5 outlines the fundamentals of programming a 6942A Multiprogrammer System. It contains an overall description of the instruction set and detailed descriptions of 21 of the most often used instructions. Programming examples for 9825,9835, and 9845 Controllers are included.

Special Purpose Programming. This chapter describes the remaining 11, special purpose, instructions in the Multiprogrammer's instruction set.

Plug-in Card Descriptions and Programs. Chapter 7 contains a brief functional description and programming examples for each type of I/O card.

Appendices:

A. A brief description of the number systems that apply to the Multiprogrammer.

B. A description of the error codes that are reported by the Multiprogrammer.

C. Descriptions and listings of the utility programs contained on the tape cartridges provided with the unit.

D. A description on how to debug the Multiprogrammer system and avoid potential programming problems.

1-4 RELATED PUBLICATIONS

1-5 The following is a list of other publications that may be helpful to the user.

1-6 Other 6942A Literature. Other 6942A literature includes the pocket-sized "Quick Reference Guide" (06942-90004) and the operating manual that is shipped with each I/O card. Each I/O card manual explains how to change the jumpers or switches that affect the operating characteristics that can be altered for that specific card; e.g. logic sense, logic levels, data formats, etc,. Information on a more general level is contained in the 6942A Multiprogrammer brochure (5952-4034D).

1-7 HP-IB Literature. General information about the HP-IB can be found in the latest Hewlett-Packard "Electronic Instruments and Systems" catalog. More specific information on the HP-IB is contained in the "Condensed Description of the HP-IB" (59401-90030).

1-8 HP-IB Controller Manuals. The following is a list of the manuals for each type of controller that is covered in this User's Guide.

9825 Publications	**HP Part No.**
Operating and Programming	09825-90000
Quick Reference Guide	09825-90010
Advanced Programming ROM	09825-90021
General I/O Programming ROM	09825-90024
Extended I/O Programming ROM	09825-90025
HP-IB Interface Installation	98034-90000

9835 Publications	**HP Part No.**
Operating and Programming	09835-90000
Quick Reference Guide	09835-90010
Beginner's Guide	09835-90001
Owner's Manual	09835-90005
I/O ROM Programming	09835-90060
HP-IB Interface Installation	98034-90000

9845 Publications	**HP Part No.**
Operating and Programming	09845-91000
Quick Reference	09845-91015
Beginner's Guide	09845-91001
I/O ROM Programming	09845-91060
Owner's Manual	09845-91005
HP-IB Interface Installation	98034-90000

1-9 OTHER HP-IB CONTROLLERS

1-10 Any HP-IB Controller, whether it be a full-sized computer or desk-top computer, can be used to program the Multiprogrammer. This guide contains some specific programming examples using a 9825,35, or 45 Controller. However, sufficient information about the Multiprogrammer's instruction set is included so that any Controller can be used provided that the User is thoroughly familiar with the Input/Output operations of that Controller.

Chapter 2
INSTALLATION AND CHECKOUT

2-1 This chapter provides procedures for interconnecting a 9825, 9835, or 9845 controller and a 6942A/6943A Multiprogrammer System using the HP-IB. The procedures include gathering the proper equipment, installing the applicable ROM's in the controller, setting the address switches, and connecting the cables. Checkout procedures, using the utility program cartridge supplied with this user's guide, verify that the equipment is installed correctly and that the Multiprogrammer System can be programmed by the controller.

2-2 EQUIPMENT REQUIRED

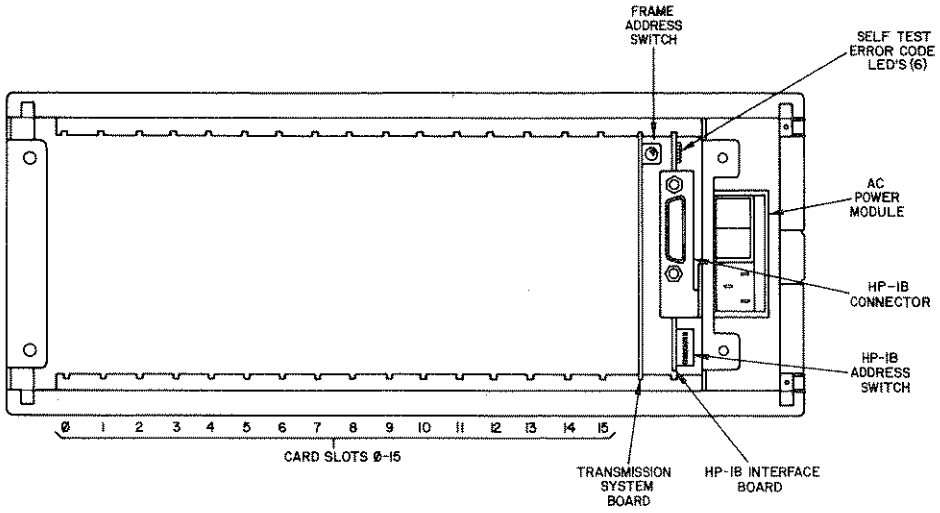
2-3 The following equipment is required to assemble the system:

1. Programmable Controller: 9825A/B, 9835A/B, or 9845A/B/S/T
2. HP-IB Interface Card HP 98034A
3. Controller ROM's
   9825: General I/O, Extended I/O, Advanced Programming, String Variables
   9835/9845: I/O ROM (optional)
4. Utility program cartridge for 9825 (06942-13001) or 9835/9845 (06942-13002)
5. Multiprogrammer HP 6942A (plus 6943A Extender Units if needed).
6. I/O Cards and edge connectors (see paragraph 2-23) for the 6942A/6943A Multiprogrammer System.
7. Multiprogrammer Transmission System Boards (see paragraph 2-18):
   a. Data Strobe (DST) Jumper Board HP Part No. 06942-60020: Installed when only one unit (6942A mainframe) is used in the system.
   b. Transmission System Mainframe Board HP Part No. 14700-60020 (part of Last Extender Kit 14700A): Installed in a 6942A mainframe when one or more 6943A Extender Units are used.
   c. Transmission System Intermediate Extender Board(s) HP Part No. 14701-60020 (Intermediate Extender Kit 14701A): Installed in each 6943A Extender Unit except the last Extender Unit in the line.
   d. Transmission System Last Extender Board HP Part No. 14700-60021 (part of Last Extender Kit 14700A): Installed in the last 6943A Extender Unit in the line.
8. Interconnecting Cables (see paragraph 2-18):
   a. Controller-to-6942A: Cable is part of the 98034A HP-IB Interface Card. Extra HP-IB cables can be ordered separately:
      10631A HP-IB cable, 1 meter (3.3 ft)
      10631B HP-IB cable, 2 meters (6.6 ft)
      10631C HP-IB cable, 4 meters 13.2 ft)
      10631D HP-IB cable, 0.5 meter (1.6 ft)
   b. 6942A-to-6943A etc: Standard 1 meter (3.3 ft) chaining cable HP Model No. 14702A purchased separately. Lengths up to 152 meters are available on special order. The system can operate with a total distance of 152 meters (500 feet) between the 6942A and the last 6943A. This maximum length is the sum of the lengths of all 14702A Chaining Cables used in one chain.

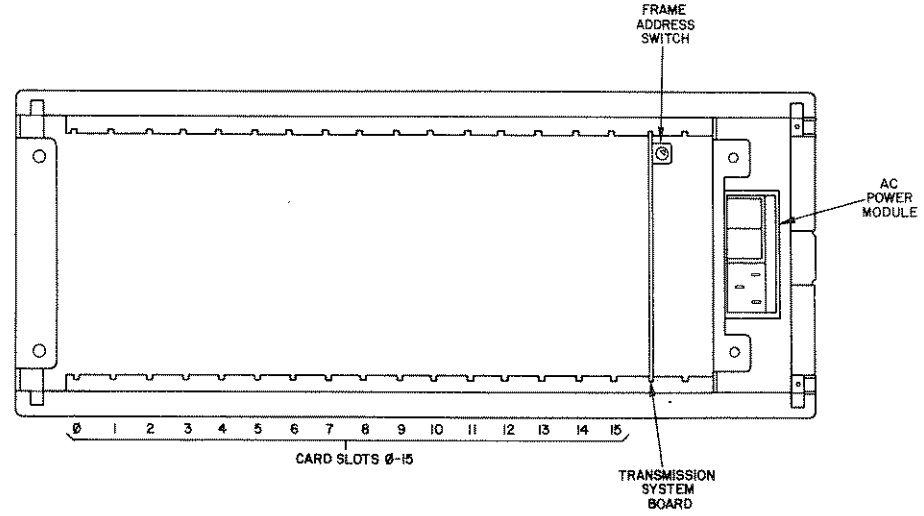
2-4 INSTALLATION

2-5 Rear covers are installed on 6942A and 6943A units to hold the I/O plug-in card connector hoods in place and prevent them from loosening. The cover(s) must be removed in order to gain access to the I/O card slots, the cable connectors, address switches, and the power module. Four quarter-turn fasteners secure the cover to the rear of the unit. Figure 2-1 illustrates the rear views of the 6942A units with their covers removed. The following 8 general steps are required prior to applying power and programming the Multiprogrammer System.

1. Install ROM's and HP-IB Interface Card in controller (paragraph 2-6).
2. Remove rear cover(s) from Multiprogrammer units(s).
3. Ensure proper line voltage is selected and power cord is connected to power module (paragraph 2-9).
4. Check HP-IB address on 6942A (paragraph 2-11).
5. Set the frame address on the 6942A and any 6943A's used in the system (paragraph 2-14).
6. Connect cables between controller and 6942A and between all Multiprogrammer units used in System (paragraph 2-18).
7. Install I/O cards and make connections from I/O cards to user's devices (paragraph 2-23)
8. Verify operation (paragraph 2-32) and then replace rear cover(s).



A. 6942A Multiprogrammer



B. 6943A Extender

Figure 2-1. 6942A and 6943A Units (Rear Views)

2-6 Installing Controller ROM's and HP-IB Interface Card

2-7 Plug the ROM's and 98034A Interface Card into the controller (see applicable service manual). The ROM's and interface card are installed with the controller switched off.

2-8 The 98034A Interface Card provides HP-IB capability for the controller (9825, 9835, or 9845) and can be installed in any of the slots in the rear of the controller. The select code for the HP-IB card is preset to "7". Throughout this guide, all examples and sample programs use a select code of "7". Setting the select code and installing the HP-IB Interface Card are described in Chapter 2 of the 98034A Installation and Programming Manual (98034-90000).

2-9 Input Power Requirements and Line Voltage Conversion for Multiprogrammer System

2-10 The 6942A and 6943A units may be operated continuously from a nominal 100V, 120V, 220V, or 240V (48-63Hz) power source. A printed circuit board located within the ac power module on the rear panel selects the power source. Voltage choices are available on both sides of the PC board. Before connecting the instrument to the power source, check that the PC board selection matches the nominal line voltage of the source. The operating voltage that is selected is the one printed on the lower-left side of the PC board (see Figure 2-2). As shipped from the factory, the PC board in this unit is positioned for the voltage used in the area that the unit is to be

delivered. To select another input voltage proceed as follows:

a. Remove power cable from instrument
b. Move plastic door on power module aside.
c. Rotate-FUSE PULL down and remove line fuse F1.
d. Remove PC board from slot. Select operating voltage by orienting PC board to position the desired voltage on top-left side of PC board. Push board firmly into slot.
e. Rotate FUSE PULL back up into normal position and re-insert fuse F1 in holder using caution to select the correct value for F1 (6A for 100V or 120V and 3A for 220V or 240V).
f. Close plastic door and connect power cable. The unit is shipped with a power cord plug that is appropriate for the user's location.

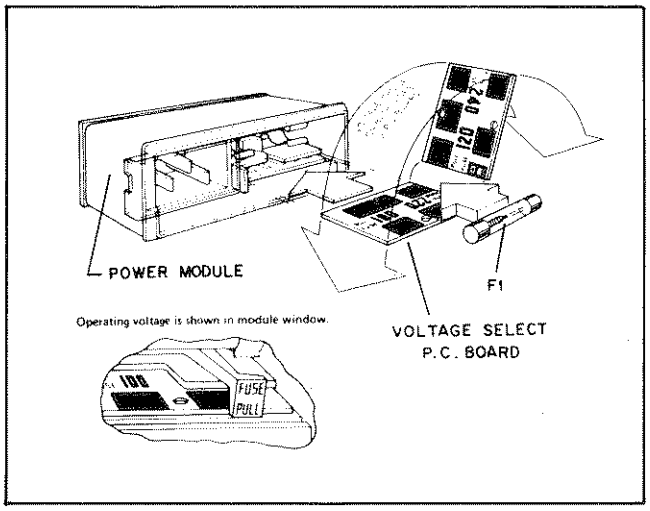


Figure 2-2 Line Voltage Conversion

2-11 Setting Multiprogrammer System HP-IB Address

2-12 The HP-IB address for the Multiprogrammer system is selected by address switches on the HP-IB Interface Board on the rear of the 6942A unit (see Figure 2-1). The switches are set for an address of "23" as the unit is shipped from the factory. Since the controller's HP-IB Interface Code is 7, the Multiprogrammer System's complete address as programmed from the controller is 723. Throughout this guide, all programming examples use the "723" address.

2-13 As shown in Figure 2-3, there are seven address switches on the rear of the 6942A. Switches 1 through 5 are shown set to decimal "23": 1 through 3 and 5 are set to logic "1" while 4 is set to logic "0". The last two switches, 6 and 7, are not used. The dot on the switch assembly identifies the logic "1" side and address switch no. 1 (the LSB).

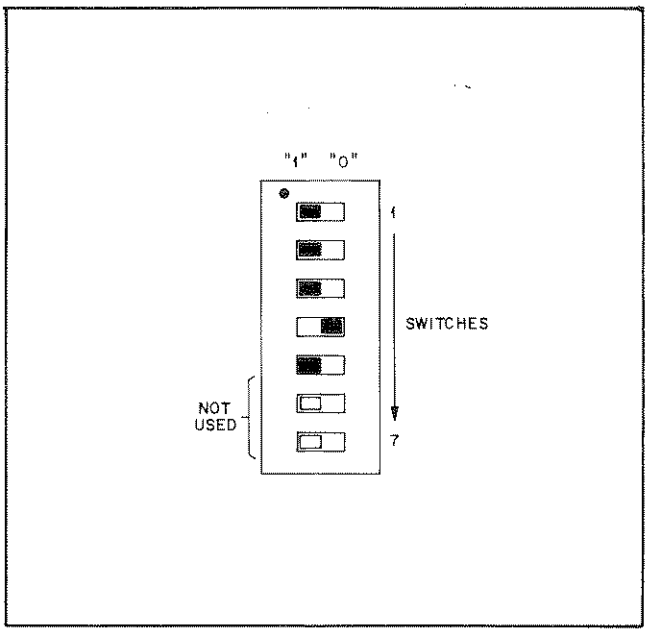


Figure 2-3. HP-IB Address Switches (rear of 6942A)

2-14 Setting Frame Address Switches

NOTE

Make sure that each Multiprogrammer unit in the system is set to a different frame address before power is applied to the system.

2-15 A frame (unit) address switch is located on the rear of each 6942A and 6943A Multiprogrammer unit (see Figure 2-1). The frame address is part of an overall address that identifies a particular I/O card in the system. Since one 6942A Multiprogrammer and up to seven 6943A Extender units can be connected together in a Multiprogrammer system, each unit's frame address switch must be set to a different number in the range from 0 to 7. The following is a list of the 128 possible I/O card slot main addresses in a Multiprogrammer System; i.e. 16 card slots (0 -15) in each of 8 frames (0-7), or $8 \times 16 = 128$. An I/O card assumes the address of the slot and frame in which it is installed. The complete address syntax, including subaddresses, is described in Chapter 5.

NOTE

It is not necessary to send (type in) leading zeros when specifying I/O card slot addresses in your program; consequently the slot addresses for frame 0 can be typed as numbers from 0 through 15.

I/O Card Main Addresses

Frame Slot	
000 - 015 or 0-15:	Specify I/O Cards 0-15 in Frame 0
100 - 115:	Specify I/O Cards 0-15 in Frame 1
200 - 215:	Specify I/O Cards 0-15 in Frame 2
300 - 215:	Specify I/O Cards 0-15 in Frame 3
400 - 415:	Specify I/O Cards 0-15 in Frame 4
500 - 515:	Specify I/O Cards 0-15 in Frame 5
600 - 615:	Specify I/O Cards 0-15 in Frame 6
700 - 715:	Specify I/O Cards 0-15 in Frame 7

2-16 A 6942A unit is shipped from the factory with its frame address switch set to "0" while a 6943A unit is shipped with its switch set to "1". The user can change these settings as required. As shown in Figure 2-4, a frame address switch has 16 positions which are marked in hexadecimal. Note that only the first 8 positions (0-7) should be used. The remaining 8 positions (8-F) are a repetition of the first eight; that is, position 8 is the same as position 0 (frame address 0); position 9 is the same as position 1 (frame address 1), etc.

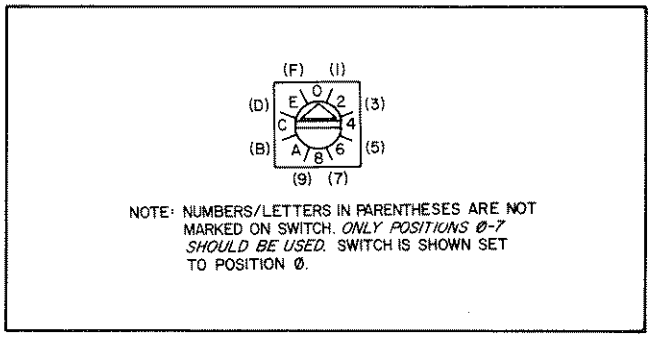


Figure 2-4. Frame Address Switch

To change the frame address of a Multiprogrammer unit, insert a small screw driver into the switch slot and set the pointer to the desired position (0-7). In a multiple unit system (one 6942A and up to seven 6943A's) the 6942A would normally be set to address 0 while the subsequent 6943A units would be set to 1, 2, 3, etc. Note however, that it is not required to set the switches in numerical sequence: e.g. the 6942A can be set to address 2, the first 6943A extender to address 0, etc. In any case, each unit in the system must have a different address.

2-17 Most of the programming examples provided in this guide assume that only one unit (6942A) is used and its frame address is set to 0. When Extender unit(s) are used, the frame address switches not only specify the unit address numbers but also specify the priority of each unit in the system. The lowest numbered frame (unit) has the highest priority. Within a unit, the I/O card slot priority is fixed by slot number order with Slot 0 having the highest priority. The Multiprogrammer system uses this priority when processing data so that data transfers with the high priority I/O cards (low address numbers) are completed first.

2-18 Connecting Cables

2-19 The cable connections and the types of transmission system boards required depend upon the number of Multiprogrammer units used in the system. All possible combinations are illustrated in Figure 2-5. Note that a 9835 or 9845 controller can be substituted for the 9825 controller shown in each figure. The 98034A HP-IB Interface Card must be used with each of these controllers.

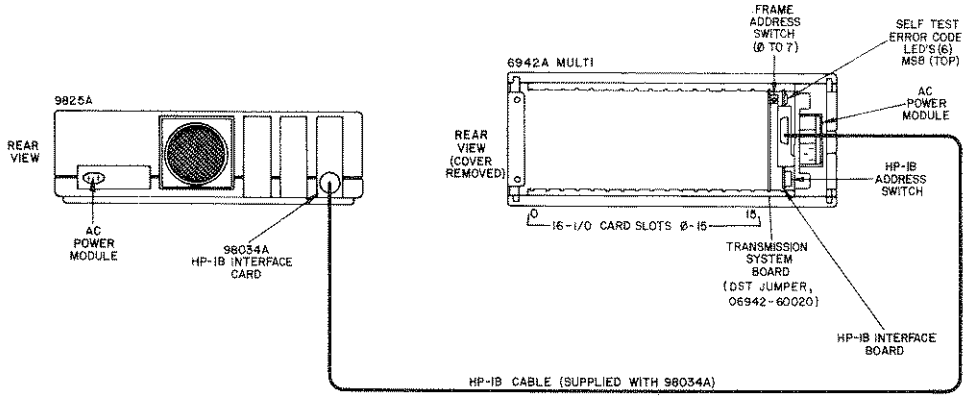
2-20 Figure 2-5A illustrates a one unit (the 6942A) Multiprogrammer system. The 98034A HP-IB Interface Card in the controller is equipped with the proper cable to mate with the HP-IB connector on the HP-IB Interface Board on the rear of the 6942A. For this configuration the DST Jumper Transmission System Board 06942-60020 must be installed in the 6942A.

2-21 Figure 2-5B illustrates a two unit system: one 6942A unit and one 6943A unit. For this configuration, a Mainframe Transmission System Board (14700-60020) must be installed in the 6942A and Last Extender Transmission System Board (14700-60021) in the 6943A. Chaining cable HP Part No. 14702A interconnects the transmission boards in the two units. The maximum distance between the 6942A and the 6943A is 152 meters (500 feet).

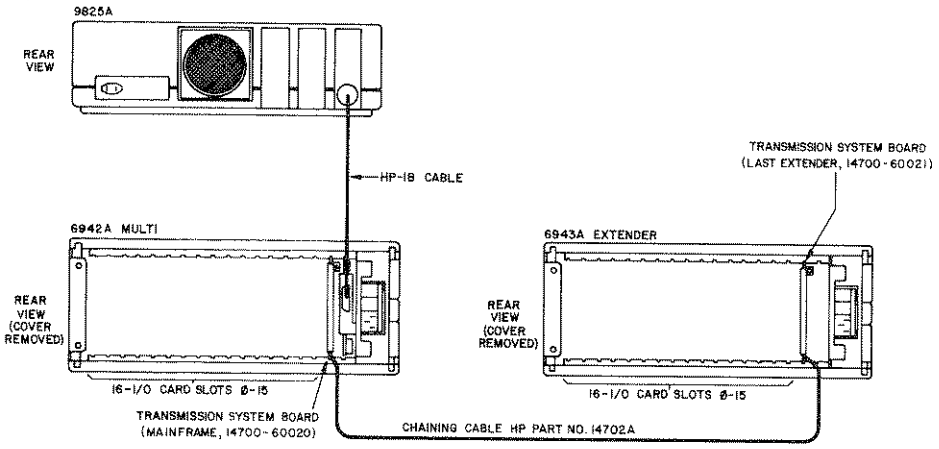
2-22 Figure 2-5C illustrates the cable connections and transmission board requirements for a system comprised of three or more (up to eight) Multiprogrammer Units, one 6942A and up to seven 6943A's. The transmission board requirements for this system are as follows:

Transmission System Boards	HP Part No.
1 Mainframe Board	14700-60020
Up to 6 Intermediate Extender Boards	14701-60020
1 Last Extender Board	14700-60021

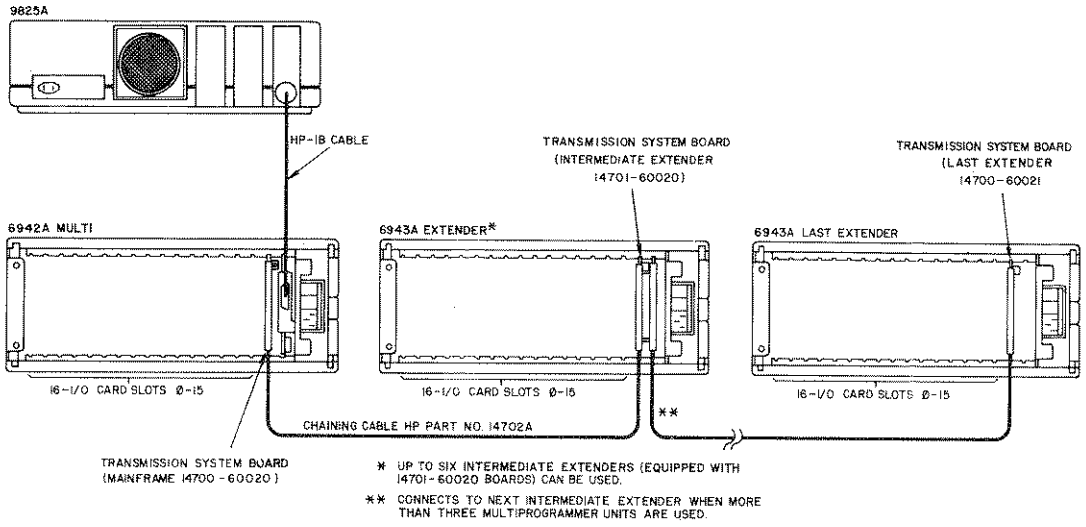
Note that each Intermediate Extender Transmission System Board is equipped with a double connector which provides connections with the units before and after it in a system employing more than one 6943A Extender Unit. The maximum distance between the 6942A and the last 6943A is 152 meters (500 feet).



A. One Multiprogrammer Unit



B. Two Multiprogrammer Units



C. Three or More (up to 8) Multiprogrammer Units

Figure 2-5. Multiprogrammer System Cable Connections

2-23 Installing I/O Cards

CAUTION

Always turn off power at a Multiprogrammer unit before installing or removing an I/O card. If power is not removed, it is possible to short components in the Multiprogrammer when installing or removing a card, thereby causing possible damage.

2-24 Each I/O card is equipped with a handle which is useful when installing or removing the card. The handle is located on the card's outer edge and is labeled to identify the card type (see Table 2-1). The I/O cards are installed in slots 0 through 15 in the rear of 6942A and 6943A units (see Figure 2-1). As stated previously, the I/O card assumes the address of the slot (and unit) in which it is inserted. Within a unit, there is an interrupt priority in slot number order with slot 0 having the highest interrupt priority (see paragraph 2-17). The following paragraphs describe the current requirements for each type of I/O card, the card installation procedures, and the edge connector that is shipped with each card.

2-25 I/O Card Current Requirements. Operating power (+5V, ±12V, Isolated ±18V) as well as data, address, and control signals are supplied to the I/O cards through the connectors in slots 0 through 15. The front panel POWER INTERRUPT and ISOLATED POWER indicators monitor the status of the power supplies (see paragraph 2-33).

+5V Supply: All of the I/O card types require +5V power. The +5V supply has a current rating of 18.5A with 12.8A available for the I/O cards. The current requirements for each type of I/O card are listed in Table 2-2. Note that Memory card assembly 69790A is comprised of two cards which must be installed in adjacent card slots. Each memory card pair requires substantially more current than the other card types. Memory card pairs equipped with Option 002 (2K memory) or Option 004 (4K memory) require even more current. Consequently, the I/O card slot requirements for memory card pairs are as follows:

Option 002 (2K memory): Maximum of six memory card pairs occupying 12 slots in a 6942A or 6943A unit plus 3 slots left vacant.

Option 004 (4K memory): Maximum of five memory card pairs occupying 10 slots in a 6942A or 6943A unit plus 6 slots left vacant.

±12VSupply: The Digital Input and the Counter cards use ±12Vpower. The ±12V supply has 2A (+ 12V) and 1.5A (– 12V) available for the I/O cards. The ±12V supply's current requirements for the Digital Input and the Counter Cards are listed in Table 2-2.

±18V Isolated Supplies: The Voltage D/A, Current D/A and the High Speed A/D cards use the isolated ±18V bias voltages. In addition, the Counter card can use an isolated ±18V supply instead of the ±12V supply. Each Multiprogrammer unit provides 3 regulated ±18V supplies. The ±18V supply current requirements for the applicable cards are listed in Table 2-2. The current capabilities of each ±18V supply are given in Table 2-3.

The status of each ±18V supply is indicated by an LED indicator on the Multiprogrammer unit's (6942A or 6943A) front panel (see paragraph 2-34). The supplies are operational if the indicators are on. Fuses for the three supplies are mounted on a fuse board which is accessible by removing the front panel grill and the fuse board cover (see paragraph 2-49).

Table 2-1. I/O Cards

Label on Card Handle	Card Name	Model No.
RES OUTPUT	Resistance Output	69700A thru 69706A
VOLTAGE D/A	Digital-to-Analog Voltage Converter	69720A
CURRENT D/A	Digital-to-Analog Current Converter	69721A
RELAY OUTPUT	Relay Output	69730A
DIGITAL OUT	Digital Output	69731A
PULSE TRAIN	Pulse Train Output and Stepping Motor Controller	69735A
TIMER/PACER	Timer/Pacer	69736A
HI SPEED A/D	Analog-to-Digital Converter	69751A
ISO DIG INPUT	Isolated Digital Input	69770A
DIGITAL INPUT	Digital Input/Analog Comparator	69771A
COUNTER	Counter Totalizer	69775A
WORD INT	Interrupt	69776A
MEMORY	Memory	69790A
BREAD BOARD	Bread Board	69793A

Table 2-2. I/O Card Current Requirements

Card	+5V	+12V	−12V	+18V(I)	−18V(I)
Res Output	637mA	—	—	—	—
Voltage D/A	387mA	—	—	80mA	38mA
Current D/A	387mA	—	—	110mA	62mA
Relay Output	633mA	—	—	—	—
Digital Out	409mA	—	—	—	—
Pulse Train	717mA	—	—	—	—
Timer/Pacer	733mA	—	—	—	—
Hi Speed A/D	366mA	—	—	150mA	79mA
Iso Dig Input	338mA	—	—	—	—
Digital Input	309mA	17mA	39mA	—	—
Counter	736mA	120mA	12mA	**	**
Word Int	401mA	—	—	—	—
* 1K Memory (Standard)	1.6A	—	—	—	—
* 2K Memory (Option 002)	2.0A	—	—	—	—
* 4K Memory (Option 004)	2.6A	—	—	—	—

* Each Memory Card Assembly consists of two cards which occupy two I/O slots.

** Jumpers on Counter card can select the isolated ±18V supply instead of the ±12V supply (non-isolated)

Table 2-3. ±18V Supply Current Capabilities

Power Supply	No.1		No.2		No.3	
	+18V	−18V	+18V	−18V	+18V	−18V
Current Rating	1A	0.6A	0.4A	0.25A	0.2A	0.15A

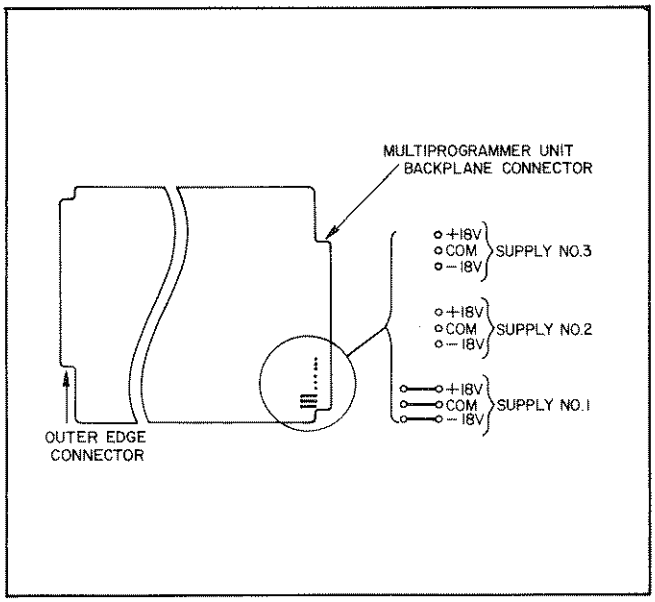


Figure 2-6. I/O card ±18V Isolated Supply Jumpers

2-26 Card Installation Procedure. The procedures outlined below assume that the Multiprogrammer Unit's (6942A or 6943A) rear cover is removed and the proper transmission system board is installed.

a. With the card handle on the bottom and with the card components on the right, slide the card into the desired slot (0-15) until it touches the connector. Note that all cards are slotted between pins 12(n) and 14(R) and all I/O slot connectors of a Multiprogrammer unit. are keyed between the same points. This makes it virtually impossible to plug a card in upside down or into any slot other than an I/O card slot 0-15

NOTE

Memory card assembly 69790A is comprised of two cards and occupies two slots (see Memory card description in Chapter 7).

b. With the I/O card touching the connector, rotate the card handle downward until it engages a groove at the bottom of the I/O slot. Now rotate the handle upward. This will push the card into the Multiprogrammer connector. (To remove a card, rotate its handle downward. This will release the card from its connector).

c. Connect user's equipment to pins on outer edge of card as described in the next paragraph.

d. As installation and wiring are completed for each I/O card, record the following information.

(1) Card type
(2) Application in external system
(3) Card main address:
Slot No. + (Frame No. X100)
(4) Card subaddresses (if applicable)
(5) Data format parameters: data type, LSB value, number of bits.

e. Install rear cover on multiprogrammer unit. The rear cover holds the I/O card connector hoods securely in place, preventing them from backing off or loosening.

2-27 I/O Card Edge Connector (Figure 2-7). Edge Connector Assemblies interface I/O cards with each other or with external devices. One Edge Connector Assembly, is shipped with each I/O card. Order Model 14703A when extra connectors are required. Each connector assembly consists of the following items:

NOTE

The encircled numbers key the items to Figure 2-7.

Hood Assembly Kit, qty 1, consisting of:

① Strain relief
② Cable clamps, 4 sizes
③ Right Hood Assembly, qty 1
④ Left Hood Assembly, qty 1
⑤ Screw, 7/16 inch, qty 1
⑥ Screws, 11/16 inch, qty 4
⑦ Connector Pin Housing, qty 1
⑧ Connector Key
⑨ Solder Pin: 45 plated solder pins are packaged in a plastic bag. Two bags are shipped with card models 69751A, 69770A, 69771AS, 69775, and 69793A. One bag is shipped with the other card models.
⑩ Spring: One plastic bag containing six springs is shipped with each I/O card. The springs are inserted into the connector pin housing to ensure a tight connection to card's edge.

2-28 The Edge Connector Assembly is designed to accommodate two 36-conductor cables with outside diameters of 0.360 inches. An unterminated 36-conductor cable (HP Part No. 8120-1163) can be ordered by the foot from Hewlett-Packard. To assemble and wire an edge connector, gather the tools and materials listed below and perform the assembly procedure. Refer to the applicable I/O card connector diagram in Chapter 7.

Tools and Materials Required:

1. a small pair of long-nose pliers
2. a small pair of diagonal wire cutters,
3. a wire stripper designed for small-gauge wire,
4. a Phillips screwdriver with a No. 0 point,
5. a low-wattage pencil-type soldering iron (a 25-watt iron with a tip 1/16 in. to 1/8 in. wide is about right),
6. some 60/40 alloy, rosin-core solder (small diameter wire, .030 to .045 in),
7. solder pin de-insertion tool, Amphenol P/N 91073-1.

Assembly Procedure (Figures 2-7 and 2-8):

a. Insert the required number of solder pins ⑨ to be wired into connector pin housing ⑦ .Pins are inserted into the applicable pin slots from the rear of the connector housing. Figure 2-8 illustrates a sample pin configuration for a digital output card.

b. Solder each cable wire to the appropriate solder pin.

c. To balance the connector insert extra solder pins ⑨ in the slots next to the wired solder pins (see Figure 2-8).

d. Insert springs ⑩ into connector pin housing. Springs are inserted from front of connector assembly (see Figure 2-7). A pair of springs is inserted into adjacent connector slots in three places on the connector (see Figure 2-8).

e. Install key ⑧ (Figure 2-7) in applicable slot. Connector key is installed from front of housing assembly.

f. Place the hood assemblies ③ and ④ around the connector pin housing ⑦ . Ensure that the bottom (high numbered pin slots) of the housing is at the bottom (cable entrance) of the hood assemblies.

g. Route cable wires through cable entrance and then fasten the left and right hood asssemblies together by inserting and tightening screw ⑤ .

h. Fasten connector pin housing ⑦ to the assembled hoods ③ and ④ using 2 screws ⑥ .

i. Insert strain relief 1 and proper sized cable clamp 2 in the cable entrance and fasten with 2 screws ⑥ .

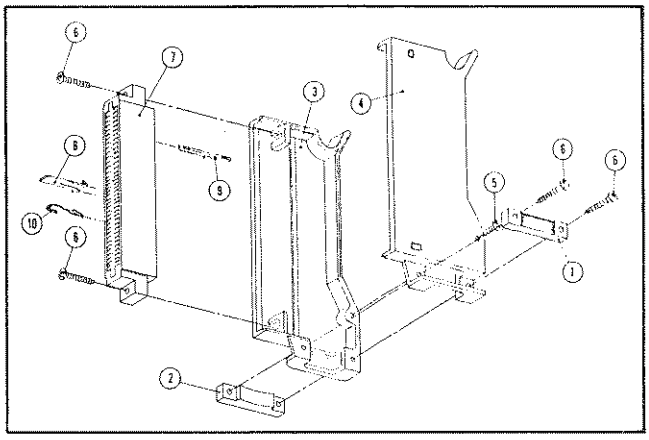


Figure 2-7. I/O Card Edge Connector Assy

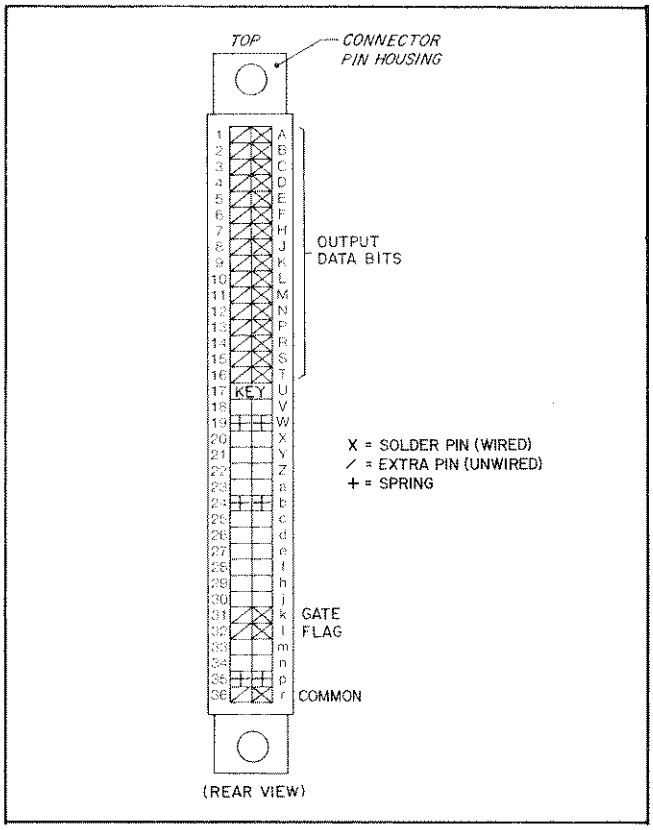


Figure 2-8. Sample Pin Configuration for a Digital Output Card

2-29 Cooling Fan and Air Filter

2-30 The 6942A and 6943A Multiprogrammer Units are equipped with variable speed fans that provide cooling air. A fan speed control circuit changes the fan speed depending upon the temperature within the unit and primary load current changes. Airflow is increased when power dissipation and heat increase, and is reduced when the dissipation and heat decrease. A temperature sensing circuit will shut the unit's internal +5V, ±12V supplies off if the cooling fan fails and causes overheating.

2-31 The grill on the front of a Multiprogrammer unit and the air filter behind it must be kept clean. The grill can be wiped clean with a damp cloth and can be removed by loosening two captive screws. The air filter is held in place on the backside of the grill by 6 Velcro fasteners. The filter can be cleaned by air blasting it in the reverse direction.

CAUTION

After 12 months the air filter loses its "flameproof" properties and must be replaced. Do not wash a dirty filter in a detergent because this will remove its "flameproof" properties. Clean filter with forced air as described above.

2-32 CHECKOUT AND TROUBLESHOOTING

NOTE

If a defective 6942A, 6943A, or I/O card is detected when performing the checkout procedures, contact your HP service engineer. Troubleshooting procedures given in this guide are limited to isolating a malfunction to a defective unit or I/O card. I/O card test programs are provided in Chapter 7.

2-33 Front Panel Controls and Indicators (Figure 2-9)

2-34 6942A (Figure 2-9A). The 6942A Multiprogrammer front panel contains a LINE switch, a SELF TEST indicator and power supply indicators.

LINE switch - The LINE pushbutton switch is pushed-in to turn the unit on and is released (out position) to turn the unit off.

SELF TEST - The SELF TEST indicator (LED) lights green to indicate that the main components of the Multiprogrammer System are operating properly. The self test feature is initiated whenever power is turned-on or the HP-IB "Device Clear" command is executed at the controller (see paragraph 2-41). When the self test runs to completion in approximately 4 seconds, the SELF TEST indicator should light.

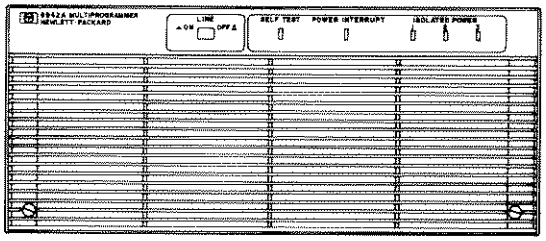
POWER INTERRUPT - The POWER INTERRUPT indicator (LED) lights red to indicate that the 5V and 12V bias supplies in the unit were shut down (via internal crowbar circuit) due to an overvoltage or undervoltage condition in any of the supplies. The 5V and 12V supplies have current limiting circuits that limit the output current if an overcurrent condition exists. For most overcurrent conditions, the crowbar circuit will activate (POWER INTERRUPT will light red) and shut the sup-

POWER INTERRUPT (continued)

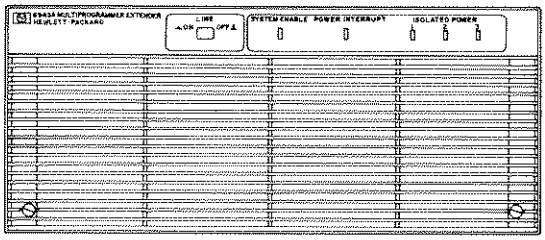
plies down. Note also that a thermal detection circuit will shut down the 5V and 12V supplies (via crowbar circuit) if the cooling fan fails and causes overheating.

ISOLATED POWER 1,2,3 -

The three ISOLATED POWER indicators (LED's) light green to indicate that the associated ± 18V isolated supplies are operational. If any indicator does not light, check the applicable fuse on the fuse board (see paragraph 2-49).



A. 6942A Multiprogrammer



B. 6943A Extender

Figure 2-9. Multiprogrammer Unit Front Panels

2-35 6943A (Figure 2-9B). The LINE switch and power supply indicators on the 6943A front panel are identical to those on the 6942A front panel. However, the 6943A has a SYSTEM ENABLE indicator (LED) instead of a SELF TEST indicator. The SYSTEM ENABLE LED indicates the status of the system enable (SYE) line that runs throughout the Multiprogrammer System. When the SYE line is reset (SYSTEM ENABLE indicator off), all output cards in the system are disabled (e.g. resistance outputs are shorted, voltage outputs are held at 0 volts, digital outputs are held in the open or zero state). This feature protects the external system (user's process) from potentially damaging outputs resulting from the storage registers on the output cards assuming random states at power turn-on. Thus, the SYE line is reset at initial power turn-on and will remain reset until an instruction is executed. As soon as the appropriate instruction is executed, the SYE line is set and the SYSTEM ENABLE indicator(s) on all 6943's will light green. With SYE set, an output card is enabled but will remain in the "safe" state until it is addressed in an instruction. Note that the SYE line is also set and reset by the System Enable (SE) and the System Disable (SD) instruction, respectively (see Chapter 6).

2-36 Pre-Operational Checklist

2-37 Run through the following checklist before applying power and programming the system.

1. The proper ROM's and the 98034A HP-IB Interface card are installed in the controller. Address code "7" is selected on the controller's 98034A HP-IB Interface card (see paragraph 2-6).
2. Correct ac line voltage is selected on 6942A and 6943A's (if used) power modules (see paragraph 2-9).
3. HP-IB address switches on rear of 6942A are set to decimal "23" (see paragraph 2-11).
4. Correct transmission system board(s) are installed, frame switches set, and chaining cables connected (see paragraphs 2-14 through 2-22).
5. The required I/O plug-in cards are installed in the 6942A and 6943A (if used).

2-38 Power-On/Self Test

NOTE

Power must be applied to all 6943A Extenders first and then to the 6942A. If this sequence is not followed, the system will fail "Self Test".

2-39 The Multiprogrammer System has a self test feature that is initiated whenever power is turned-on at the 6942A front panel or when power is already turned on and the HP-IB "Device Clear" command is executed by the controller (see paragraph 2-43). When the self test runs to completion in approximately 4 seconds, the SELF TEST indicator on the 6942A front panel should light (green) indicating that the main components of the system are operating properly. If the SELF TEST indicator does not light it indicates that a major malfunction was detected and the test was aborted (did not complete). The built-in self test feature checks the 6942A's control functions, ROM's, RAM's, HP-IB interface, backplane, and any 6943A extenders that are connected. Self test also makes a partial check of each I/O card installed in the system. I/O card functions such as control, address decoding, and data write and readback through the first rank of storage are checked during self test. Backplane connections to the I/O cards are also verified. The I/O card circuits involved in sending signals to or receiving signals from the external device (user's process) are not checked by the self test feature. Thus, I/O card failures may or may not affect Self Test and can be categorized as follows:

1. Those that affect the backplane portion of self test and cause the test to be aborted (see Self Test Er-

ror Codes Description in paragraph 2-49).

2. Those that are detected during self test but do not abort the self test and can be readback to the controller when self test completes (see Self Test and Card Identifier Program Description in paragraph 2-41).
3. Those that cannot be detected by Self Test and will only be found by a system malfunction.

2-40 Follow the procedures outlined in Figure 2-10 when applying power to the system. If incorrect power supply or self test indications are obtained, procedures are included to isolate the trouble to a defective 6942A unit, 6943A Extender Unit, or I/O card. Additional troubleshooting information is provided in paragraph 2-49.

2-41 Self Test Error Detection and Card Identifier Utility Program

2-42 This utility program is recorded on the 9825 and 9835/45 controller cartridges supplied with this user's guide. Program listings (9825 and 9835/45) are provided in Appendix C. The program is used to verify that the Multiprogrammer system is functioning properly and to provide a list of the I/O card types that are installed in the system. The data type, LSB, and no. of bits are listed with the associated card type and slot number. All of the information is printed out (9825) or displayed (9835/45).

2-43 The subroutine first sends an HP-IB "Device Clear" command to the Multiprogrammer System to reset the system and initiate the system self test. It waits four seconds for the self test to complete and then checks if any errors were detected by the self test. If no errors were detected, it prints out (9825) or displays (9835/45) the slot numbers, card types, and the associated data format parameters (data type, LSB, no. of bits). If any errors were detected, the applicable error printout/display is provided. To run the Self Test Error Detection and Card Identifier Subroutine, follow the procedures outlined in Figure 2-11. Trouble isolation procedures are included if incorrect indicating or error printouts/displays are obtained.

2-44 The following sample printout first shows that the Multiprogrammer System passed self test and then lists the card types installed in the system along with slot numbers and data format parameters. If the slot numbers disagree with the ones in your records, verify that you plugged the card into the correct slot. If the slot numbers agree with the ones you recorded, you are ready to enter your program.

Sample Printout

```
SELF TEST and
card identifier

passed self test

slot              5
D/A card
data type 1
lsb=          0.005
# of bits=      12

slot              7
dig. output card
data type 3
lsb =         1.000
# of bits=      16

slot              9
memory card 1
data type 3
lsb=          1.000
# of bits=      16

slot             10
memory card 2
data type 3
lsb=          1.000
# of bits=      16

slot             15
timer card
data type 4
range code=       M
# of bits=      16

test complete
```

2-45 Self Test Error Codes

2-46 The Multiprogrammer Systems internal self test feature checks approximately 80% of the 6942A/6943A mainframe circuits as well as the read/write and control portions of the 6942A/6943A I/O cards. Major malfunctions that cause the system to fail self test (6942A SELF TEST indicator

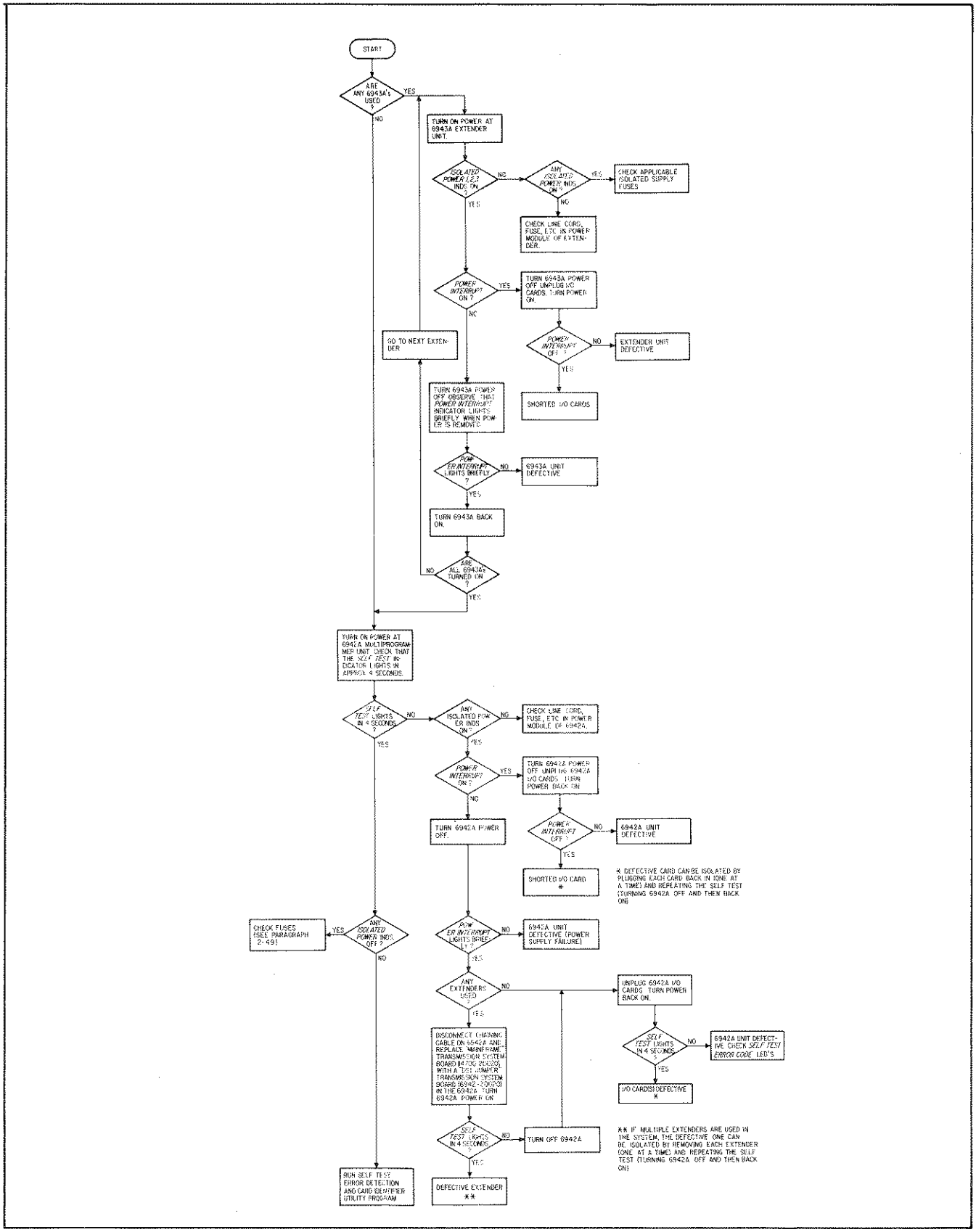


Figure 2-10. Power On/Self Test, Checkout Procedures

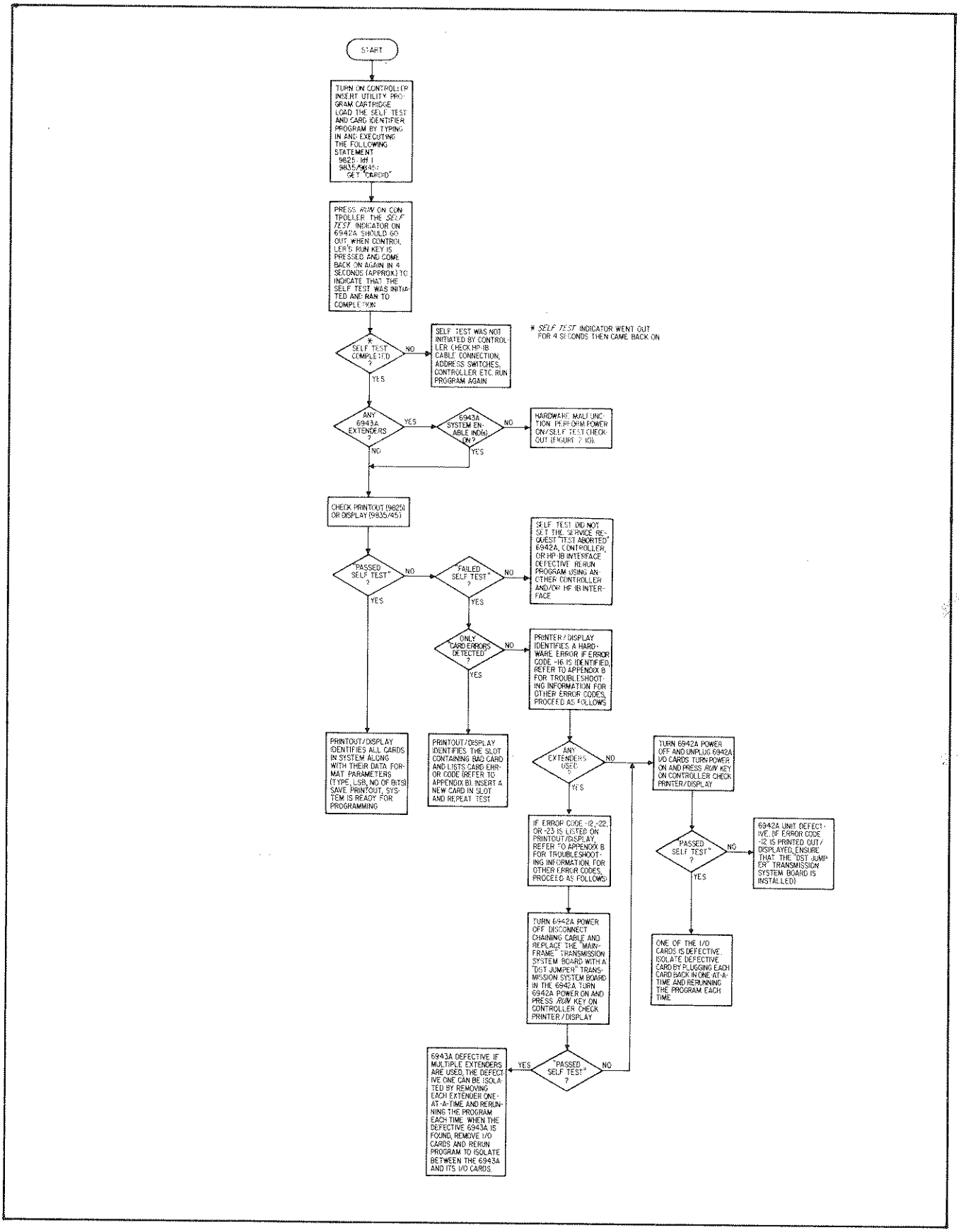


Figure 2-11. Self Test Error Detection and Card Identifier Program, Operating and Troubleshooting Procedures

does not light) are indicated by six LED's located on the HP-IB Interface Board in the rear of the 6942A (see Figure 2-1. The six LED's are arranged vertically with LED (D1) representing the MSB on top and the LSB designated D6 at the bottom. The LED's are visible when the 6942's rear cover is removed and light red to indicate a major failure. The pattern of lighted LED's will be used by the service engineer in isolating the failure to a defective circuit. The general error codes are listed below for your information, however, further troubleshooting is beyond the scope of this User's Guide.

Codes (Led Nos.)						Failure Indicated
(TOP) 1	2	3	4	5	6	
1	1	X	X	X		Microprocessor Board or RAM Data Bus
0	1	X	X	X	X	RAM Board
1	0	X	X	X	X	HP-IB Interface
0	0	X	X	X	X	I/O Card Control Logic

These identify particular circuits

2-47 Power Supply Failures

2-48 Normally when power is applied, the POWER INTERRUPT indicator (red) is off and the three ISOLATED POWER indicators (green) are on. If the POWER INTERRUPT is on, the internal 5V and 12V supplies were shut down (via internal crowbar circuit) due to an overvoltage, undervoltage, or overheating condition. For this condition, first check that the cooling fan is operating properly and then check for shorted I/O cards. Note that the POWER INTERRUPT indicator comes on briefly when the unit is turned-off. If the unit is turned-on again before the POWER INTERRUPT indicator has gone out, the internal crowbar circuit will remain active and the POWER INTERRUPT will stay on.

2-49 If all three ISOLATED POWER indicators do not light when power is applied, check primary power connection and line fuse (see paragraph 2-9). If only one or two ISOLATED POWER indicators do not light, check for a blown fuse on the fuse board. As shown in Figure 2-12, each isolated supply has two fuses. The fuse board is located behind the front grill and to the right of the fan assembly. To gain access to the fuses, proceed as follows:

a. Turn-off power
b. Loosen two captive screws and remove the grill from the front of the unit.
c. Remove two screws securing the fuse board cover. Note that the fuse board cover is also a fuse interlock which disconnects the six fuses when the cover is removed.
d. Pull fuse board straight back and remove from unit. Execise care not to bend the 12-pin interlock connector on the rear of the cover.

2-50 Data Common Ground

2-51 Jumper W1 on the fuse board (Figure 2-12) connects the data common ground [1] to the power (safety) ground ⏚. The jumpers are installed on all 6942A and 6943A units as they are shipped from the factory. They should be left installed for best noise immunity. However, if the 6943A Extender Units are spread over a long distance, the jumpers may be removed from the 6943's to prevent ground loops.

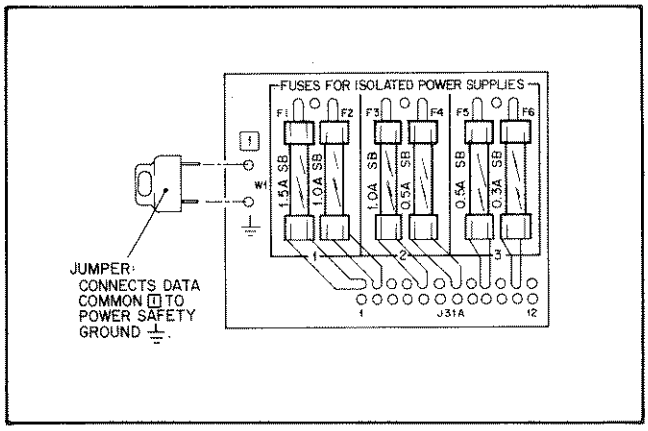


Figure 2-12. Fuse Board for Isolated Power Supplies

Chapter 3
GETTING STARTED

3-1 INTRODUCTION

3-2 The purpose of this Chapter is to acquaint you with some of the basic operating and programming features of the 6942A Multiprogrammer. Using some of the less complex output and input cards of the Multiprogrammer, we will send data to the output cards and read back data from the input cards. In each case, the results will be monitored to verify that the program was sent correctly and was properly executed by the 6942A. Only two Multiprogrammer instructions will be used and working through this Chapter should take approximately one hour.

3-3 The material in this Chapter assumes that you have completed the installation and checkout procedures given in Chapter 2, thus assuring that the 6942A mainframe and the input portion of your I/O cards are operational. After completion of this Chapter, you will need more information about the Multiprogrammer's instruction set, so please read Chapters 4 and 5 (as a minimum) before attempting to write final programs.

3-4 HARDWARE REQUIREMENTS

3-5 The programming procedures in the subsequent paragraphs require a working system, consisting of: a controller, 6942A Multiprogrammer, and several I/O cards.

3-6 Controller/Multiprogrammer Connections and Addresses

3-7 Connect the 6942A to a controller (9825, 9835, or 9845) using the HP-IB Interface as described in Chapter 2.

Table 3-1. Suggested I/O Cards and Associated Equipment

MODEL	DESCRIPTION	OUTPUT MONITORING	INPUT STIMULUS	CARD CONNECTOR PINS (see Chap. 7)
	——— OUTPUT ———			
69731A	Digital Output	Voltmeter (0-10V) or Logic Probe		A thru T (+) and com (rr) (16 Bits)
69730A	Relay Output	Ohmmeter		1A thru 31kk (16 contacts)
69720A	Voltage D/A Converter	Voltmeter (0-10V)		W (+) and Y
69721A	Current D/A Converter	Ammeter (0-20mA)		19 (+) and 21
	——— INPUT ———			
69771A	Digital Input		Clip Lead or Power Supply (0-5V)	A thru T (16 Bits)
69770A	Isolated Digital Input		Power Supply Std: 0-5V 001: 0-12V 002: 0-24V	A thru T and 1 thru 6 (16 Bits)
69751A	A/D Converter		Power Supply (0 to ±10V)	W (+) and Y

Note that throughout this User's Guide, we assume that the Multiprogrammer's HP-IB address is 23, and that the controller's HP-IB Interface select code is 7, resulting in a complete HP-IB address of 723. Also, the programming examples in this chapter assume that the 6942A's frame address switch is at its factory setting of 0.

3-8 I/O Card Selection

3-9 Gather your I/O cards by examining Table 3-1 and selecting two output and two input type cards. If possible, select two different model numbers of each type to provide a wider familiarity. Although the programming examples given in this Chapter apply to most Multiprogrammer I/O cards, the specific cards of Table 3-1 are recommended because they are less complex and programming results can be more easily monitored. If you don't have two of these output or input cards, a minimum number of one output and one input card is required to complete all of the programming examples.

3-10 **Monitoring and Stimulus Devices.** Table 3-1 also lists the devices suggested for monitoring the outputs of the output cards and the stimulus required for simulating an input to the input cards. For example, each output bit of the Digital Output Card can be checked with a voltmeter or logic probe and each bit of the Digital Input Card can be stimulated with an open circuit (HI) or a clip lead to common (LO). During subsequent procedures, these devices must be connected to the edge fingers of the I/O cards. When this becomes necessary, use the connectors and gold pins provided with each I/O card together with the connector diagrams given in Chapter 7.

3-11 Now the I/O cards that you have selected can be inserted into the appropriate slots in the 6942A mainframe. Generally, any I/O card can be placed in any one of the 16 slots in the 6942A mainframe (see Chapter 2). Each I/O slot has a number assigned to it (from 0 to 15) marked on the bottom of the card cage. When an I/O card is plugged into a specific slot, it assumes the address of that slot, so you must be aware of the slot number of each card. The programming examples in this Chapter assume that you have inserted two output cards in slots 0 and 1 and two input cards in slots 2 and 3.

CAUTION

To avoid damaging the I/O cards and the mainframe, always turn the main power switch off before removing or inserting I/O cards.

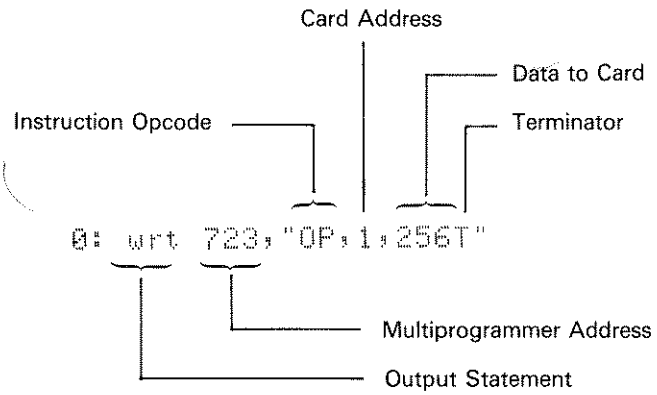
3-12 PROGRAMMING THE OUTPUT CARDS

3-13 The Output Parallel (OP) Instruction

3-14 We will be using the OP instruction to program the output cards. The OP instruction sends data from the controller to your output cards, and then instructs the cards to begin simultaneously processing the data. The instruction completes when all addressed cards have completed processing the data and the results are available at the output edge connectors. For example, an OP instruction sent to a pair of Voltage D/A Converter cards would first send data to storage registers on both cards and then instruct the cards to convert the data into equivalent analog output voltages.

3-15 **OP Format.** Before actually programming your output cards, let's look at the basic elements of a typical OP instruction as programmed from a 9825 Desktop Computer. As shown in Example 3-1, the "wrt" portion tells the controller to output a message to the HP-IB and address 723 establishes the Multiprogrammer System as the device that will obey the characters that follow. The instruction opcode characters (OP in the example) provide control information that tell the Multiprogrammer how to process the message as was explained in the previous paragraph. The card address selects the frame (unit) and card slot (from 0 to 15) that will receive the data (256) that follows. In the example, card address 1 selects slot 1 in frame 0 (the 6942A). Because leading 0's need not be transmitted, the address for any card slot in unit 0 is simply equal to the slot number (0 to 15). The actual data value that is sent to the card is a fixed point real number that depends on the type of card and its data format. This will be discussed in more detail later in this Chapter. The final character is the terminator (T) which indicates to the Multiprogrammer that the instruction is complete and it can begin processing.

Example 3-1. Structure of a Typical OP Instruction



3-16 Send the Program

3-17 To program two output cards (in slots 0 and 1) send the statement of Example 3-2. Refer to the chart below the example for the data values (D) to send to your specific output cards.

3-18 After you have programmed your cards, verify that the results are correct by using the test equipment recommended in Table 3-1. To facilitate connections to the card edge fingers, fabricate the card connectors using the gold pins as outlined in Chapter 2. Use the connector drawings in Chapter 7 and the "Connector Pins" column of Table 3-1 to determine where to connect your measuring device. For example, to check bit 0 on a Digital Output Card, connect your voltmeter or logic probe between pin A(+) and pin rr (common). It should read 5Vdc. Further, to check relay 0 on a Relay Output card, connect your ohmmeter between pins 1 and A. It should read zero ohms.

Example 3.2 Programming Two Output Cards

9825A Controller

send values below

```
0: wrt 723,"OP,0,D,1,DT"
```

9835/45 Controller

```
10   OUTPUT 723;"OP,0,D,1,DT"
```

Output Card	Data (D)	Output Results
69731A, Digital Out.	65535	All 16 Bits = logical 1's (5V)
69730A, Relay Out.	65535	All 16 Relays closed (0 ohms)
69720A, Voltage D/A	5	+5Vdc Output
69721A, Current D/A	10	+10mA Output

3-20 If the measured results do not agree with your program, verify that your program line agrees in every respect with that of example 3-2. If it still does not work, proceed to "Programming Errors" later in this Chapter.

3-21 I/O Card Data Formats

3-22 Before programming your input cards, you may want to program other data values to the output cards. In order to do this, you must be aware of the data format requirements of the Multiprogrammer's I/O Cards.

3-23 As indicated in the last example, the Multiprogrammer accepts data in a variety of formats, converts it to the correct form and then sends it to the card. The idea is to allow you to program in a data format that "makes sense" for the card you are using. For example, the Voltage D/A card is programmed in volts, since the card outputs volts.

3-24 All of the I/O cards "wake-up" to accept data in a certain format. To determine the format for your cards, you can use the Card Identifier Utility program described in Chapter 2. Besides identifying the cards, this program specifies the data type, LSB value, and number of bits.

3-25 Table 3-2 also indicates the wake-up formats for the output cards used in this Chapter. Use the data types, ranges and resolutions listed to program desired data to the cards.

3-26 Notice that the Multiprogrammer allows you to reformat the I/O cards, specifying other data types, LSB values, bit sizes, and ranges. Reformatting of I/O cards is described in Chapter 5. In addition, a more detailed discussion of the concept of I/O card data formats is provided in Chapter 4.

3-27 PROGRAMMING THE INPUT CARDS

3-28 Programming the input cards is a two-step process. First an input instruction is sent to the Multiprogrammer System commanding a card, or group of cards, to capture external data and store it in the 6942A's mainframe memory. Next, an input statement is used to read this data back to the controller.

Table 3-2. Output Card Wake-Up Formats

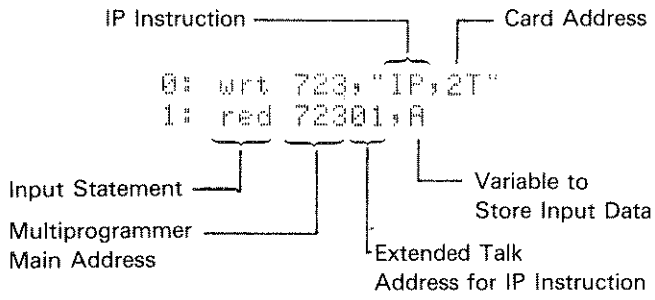
Card Type	Programming Data Type and Range	Card gets Data In	Bit Size	Resolution (LSB)
69731A, Digital Output	0-65535	Unsigned Binary	16	1
69730A, Relay Output	0-65535	Unsigned Binary	16	1
69720A, Voltage D/A	−10.24 to +10.235V	2's Complement	12	0.005V
69721A, Current D/A	−20.48 to +20.475mA	2's Complement	12	0.01mA

3-29 The Input Parallel (IP) Instruction

3-30 The IP instruction will be used to direct your input cards to take the data at their edge connectors, process the data, and then transfer it to memory locations inside the mainframe. How the input data is processed, depends on the card type. The 69751A A/D card converts an analog input voltage into a digital value while a 69771A Digital Input card samples 16 input lines and then returns the resultant values.

3-31 Example 3-3 shows the format of a typical input operation using a 9825 Desktop Computer. In the first line, an IP instruction commands an input card in slot 2 to take a reading. After the card has completed, its input data is stored in 6942A memory for immediate or future readback. The "red" statement takes the data from memory and stores it in variable A.

Example 3-3. Structure of an IP Instruction



3-32 Notice that an HP-IB extended talk address has been added to the Multiprogrammer's main address in the input statement. The extended talk address, which must be specified whenever data or status is read back to the controller, allows the Multiprogrammer to identify the source of the requested data. Right now we will only be concerned with talk address 01 which is associated with the IP instruction. However, a total of 13 different extended talk addresses are used by the Multiprogrammer and these are listed both on your programming card and in Chapter 4.

3-33 Input Simulation and Data Readback

3-34 Before programming the input cards, connect the stimulus devices indicated in Table 3-3. Leave all input pins open on a Digital Input card to simulate 16 logical "1's", and connect a 10V source between pins W (+) and Y on an A/D Converter card. To simulate an all "1's" condition on an Isolated Digital Input card, you must connect the voltage indicated between each lettered pin (A-T) and the associated numbered pin (1-16). For example, make bit 00 a logical "1", by connecting the voltage between pins A (+) and 1.

3-35 Now send the following program to your input cards. The first line sends an IP instruction for the two input cards; assumed to be in slots 2 and 3. The next line reads back the data obtained form the IP instruction, stores it in variables A and B (one variable for each card's data), and then displays the results. The results should agree with the stimulus of Table 3-3. Digital Input cards should have returned a value of 65535 (which is the decimal equivalent of 2^{16}) and the A/D Converter card should have returned 10.00V.

Example 3-4. Programming Two Input Cards

9825 Controller

```
0: wrt 723,"IP,2,3T"
1: red 72301,A,B;dsp A,B
```

9835/45 Controller

```
10  OUTPUT 723;"IP,2,3T"
20  ENTER 723.01;A,B
30  DISP A,B
```

Table 3-3. External Input Simulation

INPUT CARD	STIMULUS	RESULT DISPLAYED
69771A, Digital Input	Input pins (A-T open = logical "1" Input pins connected to common (pins r, 36) = logical "0"	16 logical "1's" = 65535 16 logical "0's" = 0.00
69770A, Isolated Digital Input	Correct Voltage (see below) connected between pins A-T (+) and 1-16 = logical "1" Std Card = 5V Opt. 001 = 12V Opt. 002 = 24V Input pins open = logical "0"	16 logical "1's" = 65535 16 logical "0's" = 0.00
69751A A/D Converter	0 to ± 10V connected between pins W(+) and Y returns identical value to controller	10V input = 10.00 displayed

3-36 If desired, you can now alter the external inputs to your cards and determine if the new data values returned to the controller are correct. For the digital cards, you can simulate other 16-bit binary words by changing the appropriate bits from logical "1's" to logical "0's". The equivalent decimal value of the 16-bit binary number that you select is the value that will be displayed. For the A/D Converter, you can set your external power supply to any voltage between 0 and ±10V (within a resolution of 5mV) and the identical value should be returned to the display.

3-37 Multiple IP Instructions and Delayed Readback. As mentioned previously, the data taken from one IP instruction does not have to read back before performing other IP instructions (or any other Multiprogrammer instructions). During actual procedures, multiple IP instructions can be issued; all of them executed; and then at a later time all of the data can be read back (possibly when the process has completed). This allows your application to run at maximum speed, without taking the time to read data back in the middle of a time-critical process.

3-38 When more than one IP is active in the system, a separate "red" or "ENTER" request must be made for each IP instruction (using extended talk address 01 in each case). The data from each IP will be received in the order that the instructions were programmed; the data from the first IP will be received first, data from the second IP second etc,.

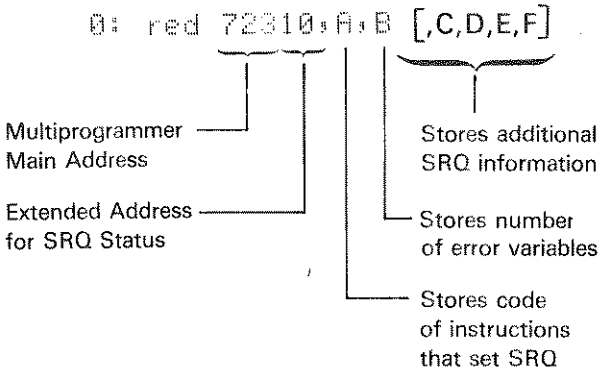
3-39 Checking For Programming Errors

3-40 The Multiprogrammer System performs extensive error checking of every instruction. It checks for syntactical errors, such as leaving out a "T" (terminator) or a card address. General programming errors, such as using illegal opcodes or talk addresses, are also checked. If a programming error is detected by the Multiprogrammer, the appropriate error code is entered in the error buffer and the HP-IB service request (SRQ) line is set. The Multiprogrammer also sets the SRQ line for a number of other reasons; including the detection of hardware errors, completion of certain instructions, completion of self test, and a few other conditions. None of the procedures used in this Chapter will set SRQ unless a programming error is made. Hence, our description here will be limited to the detection of programming errors. Chapter 5, "Multiprogrammer SRQ Status Information," contains a complete description of all the conditions that can set SRQ and describes the possible responses to a service request from the 6942A. In addition, the basic concept of HP-IB service requests is explained in "HP-IB Messages" in Chapter 4.

3-41 Detecting programming errors require two successive read statements. First, Multiprogrammer main SRQ status is read back from extended talk address 10 to determine the number of programming errors that have been made (from 0 to 11 errors can be stored). If any errors were made, a second status read, from extended talk address 11, contains a code that indicates the instruction and/or type of error and a card address, if applicable.

3-42 Reading SRQ Status. As shown in Example 3-5, complete SRQ status information can be read back to a 9825A Controller and stored in up to six variables. Variables C through F contain the status of certain advanced instructions not pertinent to this Chapter. For this Chapter, we will just use variables A and B.

Example 3-5. Reading Complete SRQ Status



3-43 Similarly, the information in variable A is not relevant to this Chapter but must be read back to obtain the number of possible programming errors stored in variable B. Variable A contains a code that indicates whether one or more of certain instructions caused activation of the SRQ line. However, the instructions and operating modes of this Chapter are such that SRQ will not have been set by the previous procedures and thus, the information in variable A can be ignored.

3-44 Variable B indicates how many pieces of error information are in the Multiprogrammers error buffer (from 0 to 11). If B = 0, no errors were detected. If B≠0, errors were detected and the error type code should be read back from extended talk address 11.

3-45 Error Codes. The Multiprogrammer reports two basic types of errors on extended talk address 11: hardware errors and programming errors. For some programming errors, associated card addresses are also reported. Hardware errors are negative decimal numbers (−11 through −24) and are usually detected during self test. Refer to the checkout portion of Chapter 2 for a description of self test. Also, Appendix B contains a description of all hardware and programming error codes.

3-46 As indicated in Tables 3-4 and 3-5, programming errors are subdivided into two groups; those of a general nature and those that can be associated with a specific instruction. For the programming procedures of this Chapter, only two of the general errors are possible; illegal opcodes (−2) or illegal extended talk address (−1).

Table 3-4. General Programming Error Codes

Code	Description
– 1	Illegal extended talk address
– 2	Illegal opcode
– 3	Illegal operation in immediate mode
– 5*	Illegal BCD Code read back

* A second code follows indicating the card address associated with this error. Card addresses are positive numbers (from 0 to 15).

Table 3-5. Instruction Identification and Error Codes

Instruction Identification Codes			
ID Code	**Inst.**	**ID Code**	**Inst.**
– 100	OB	– 1800	RF
– 200	IP	– 1900	DC
– 300	OP	– 2000	AC
– 400	II	– 2100	CY
– 500	OS	– 2200	WC
– 600	IE	– 2300	GN
– 700	MI	– 2400	WF
– 800	MO	– 2500	RS
– 900	OI	– 2600	RV
– 1000	WA	– 2700	GS
– 1100	WU	– 2800	GP
– 1200	Not Used	– 2900	SC
– 1300	GI	– 3000	RC
– 1400	IN	– 3100	SE
– 1500	CC	– 3200	SD
– 1600	SF	– 3300	CW
– 1700	CG		

Instruction Error Codes	
Error Code	**Description**
– 30	No. of cards incorrect
– 31	Illegal character in inst.
– 32	Wrong card address
– 33*	Illegal use of card addr.
– 34*	Data error
– 35*	Data limit exceeded
– 36	Illegal repeat or wait factor
– 37	Illegal use of group number
– 38	Illegal group number
– 39	Illegal OI or II in immediate mode
– 40*	SF parameter Error
– 41*	Illegal card address
– 42*	faulty card at this address
– 43*	no card at this address
– 99	miscellaneous

* A second code follows indicating the card address (positive number from 0-15) associated with this error. For example, – 334 followed by 1 indicates an OP instruction data error for the card in slot one.

3-47 If the Multiprogrammer detects an error while it is processing an instruction, it will report it as an instruction error. An instruction error contains two pieces of information: the instruction that was involved and the specific error. In this Chapter, only two instruction identification errors are possible (– 200 and – 300) because only the OP and IP instructions were used. Notice that the instruction error that is reported by the Multiprogrammer will be the sum of the identification code and the specific error code. For example, if you attempt to send an IP instruction to an empty card slot a code of – 243 will be received when reading back from extended talk address 11. For error codes associated with I/O cards, a positive number (corresponding to the card slot address) will follow the error code.

3-48 Reading Back Error Codes. The routine of Example 3-6 can be run if you suspect that you made programming errors in the previous procedures. The routine first reads SRQ status from extended talk address 10 and stores the number of programming (or hardware) errors in variable B. If no errors were made (B = 0), the program prints "no errors" and stops. If errors do exist, (B = 1 to 11) the error codes are read back (one at-a-time from extended talk address 11), stored in variable r1, and then printed out.

Example 3-6. Programming Error Check

9825 Controller

```
0: red 72310,A,B
1: if B=0;prt "no errors";gto "end"
2: prt "error list";spc
3: for J=1 to B;red 72311,r1
4: prt r1
5: next J
6: "end";spc ;end
```

9835/45 Controller

```
10    OPTION BASE 1
20    DIM R(11)
30    ENTER 723.10;A,B
40    IF B<>0 THEN Errors
50    PRINT "NO ERRORS"
60    GOTO End
70 Errors:  PRINT "ERROR LIST"
80    REDIM R(B)
90    ENTER 723.11;R(*)
100   FOR J=1 TO B
110   PRINT R(J)
120   NEXT J
130 End: PRINT
140   END
```

3-49 If your earlier program procedures all worked correctly, you can intentionally generate some programming errors and than run the check routine of example 3-6 to ensure that the correct error codes are returned. Example 3-7 shows some suggested programming errors and the resultant error codes that should be printed out.

Example 3-7. Programming Errors and Associated Codes

9825 Controller

```
wrt 723,"OD,1,1,T"
```

Error: -2 (Illegal opcode)

```
wrt 723,"OP,1,T"
```

Error: -334,1(No data specified for card 1)

9835/45 Controller

```
OUTPUT 723;"OD,1,1,T"
```

Error: -2 (Illegal opcode)

```
OUTPUT 723;"OP,1,T"
```

Error: -334,1(No data specified for card 1)

Chapter 4
PROGRAMMING CONCEPTS

4-1 This Chapter describes the programming concepts which you must understand before proceeding to the programming information provided in Chapters 5, 6, and 7. The topics covered include both HP-IB message and Multiprogrammer programming concepts. After the applicable HP-IB message types are described a general description of how the Multiprogrammer processes a typical data message is provided. Next, the basic I/O card operations, that result from a data message, are described. Finally, I/O card subaddressing and data format concepts are discussed.

4-2 HP-IB MESSAGES

4-3 Communications on the HP-IB take the form of quantities of information which are transferred from one device to one or more other devices on the bus. These quantitites of information can be thought of as "messages" and most of the programming manuals for Hewlett-Packard desktop and systems computers use the message concept when describing their HP-IB operations. The purpose of this discussion is to summarize all of the bus messages that can be implemented by the 6942A Multiprogrammer. Hewlett-Packard has defined a set of twelve messages that provide complete communication capability on the HP-IB. Of these twelve messages, the 6942A uses five; as listed below.

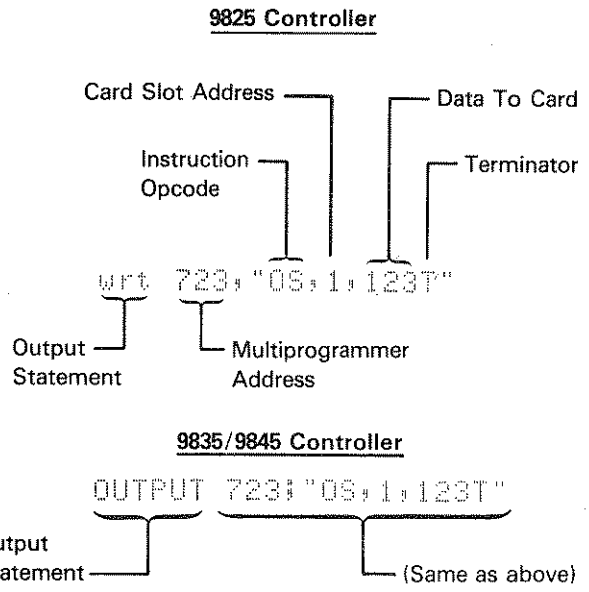
1. Data Message: The 6942A Multiprogrammer accepts address, control, and numerical data from the controller and sends back input data and status information.
2. Clear Message: A clear message from the controller resets the 6942A to its initial state.
3. Require Service Message: The 6942A requests service from the controller by setting the HP-IB service request (SRQ) line. The controller can then conduct a serial or parallel poll of the bus to determine which device requested service.
4. Status Byte: An 8-bit byte sent to the controller in response to a serial poll. The byte indicates whether the 6942A is currently requesting service.
5. Status Bit: A single bit sent to the controller in response to a parallel poll.

4-4 Data Messages

4-5 In terms of HP-IB data messages, the 6942A receives address and control information as well as numerical data from the controller. In addition, input data from the 6942A can be read back to the controller. The following examples show typical data messages that send data to the 6942A and read data back to the controller.

4-6 Sending a Data Message. In Example 4-1, a "wrt"statement (9825) or an "OUTPUT" statement (9835/9845) instructs the controller to send a data message to an HP-IB device. Address 723 establishes the Multiprogrammer as the destination for the message. The message itself begins with control information in the form of an instruction op-code (OS) and is followed by address value "1" which select the I/O card in slot 1 to receive the data (123) that follows. Detailed information about the Multiprogrammer's instruction op-codes and I/O addressing scheme is presented in Chapter 5.

Example 4-1. Sending a Typical Data Message



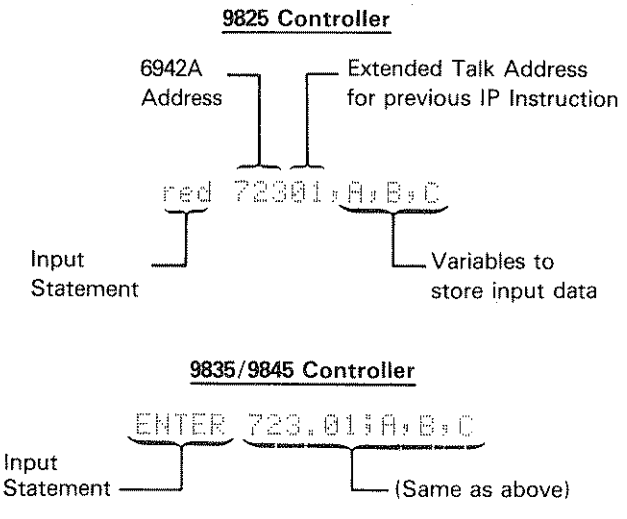
4-7 Receiving a Data Message. Example 4-2 shows a "red" or "ENTER" statement that tells the controller to read back data from the 6942A's memory and store it in variables A,B,C. Each time data is read back to the controller, an extended talk address must be specified in order to identify the source of the data. In the example, it is assumed that the data was collected and stored as a result of an Input Parallel (IP) instruction that was previously sent to the 6942A. As shown in the example, the extended talk address (01) is simply added at the end of the main address (723) when typing the address portion of the statement. Note that extended talk addresses are utilized only when reading data back to the controller. Extended listen addresses are not required for messges sent from the controller to the Multiprogrammer. Overall, there are 13 different extended talk addresses associated with the 6942A: nine are related to instruction opcodes and four are provided

for reading status. As shown in Table 4-1, the extended addresses range from 01 to 14, 07 is not used. Programming examples utilizing these addresses are given in subsequent Chapters.

Table 4-1. Extended Talk Addresses

CODE	INSTRUCTION or STATUS
01	IP
02	II
03	IE
04	RF
05	MI
06	RV
07	(Not Used)
08	RS
09	OI
10	Multiprogrammer SRQ Status
11	Error status List
12	Armed Card Interrupt List
13	Busy Instruction Status
14	RC

Example 4-2. Receiving A Typical Data Message



4-8 Clear Message (Device Clear)

4-9 Example 4-3 shows a "clear" or "RESET" statement that tells the controller to reset the Multiprogrammer to its turn-on, or wake-up, state. The internal effects of the clear message are, as follows:

1. The 6942A's System Enable (SYE) line is turned off. When off, SYE disables the outputs of all output type cards setting them to a "safe" state. For example, all relays on the 69730A Relay Output card are opened.
2. The 6942A is forced to perform a self test. This, in turn, clears Multiprogrammer memory in 6942A of all instructions and and also clears the internal registers on all I/O cards of any data or control information. In addition, the mainframe defaults to the serial mode (see Chapter 5) of instruction processing and the data formats of all I/O cards revert to their wake-up state.

Example 4-3. Sending the Clear Message

```
9825:      clr 723
9835/45:   RESET 723
```

4-10 Service Request and Polling Messages

4-11 Generally, receipt of a require service message indicates to the controller that it should interrupt its normal tasks and take some form of action to maintain proper operation of the bus. The 6942A uses the SRQ line of the HP-IB to send the require service message. Because the SRQ line is a single wire that is connected to every HP-IB device, the controller must conduct a serial or parallel poll to identify the source(s) of the service request. The interrupt system of the controller can be used to monitor the SRQ line and jump to the polling routine when device(s) request service.

4-12 Serial Poll. In a serial poll, the controller polls each device on the bus, one at a time. In response to the poll, each device returns a status byte message to the controller indicating whether or not it requested service (set SRQ). Example 4-4 shows how to serial poll the Multiprogrammer and store its status byte in variable G.

Example 4-4. Multiprogrammer Serial Poll

```
9825:      rds(723)→G
9835/9845: STATUS 723;G
```

4-13 The status byte (an 8-bit character) that is stored in variable G of the example will have a decimal value of 64 (bit 6 on) if the Multiprogrammer requested service, or a value of 128 (bit 7 on) if it had not requested service. None of the remaining bits of the status byte (bits 0 through 5) are used by the Multiprogrammer in a serial poll. Once a service request has been detected, the user should perform a read of Multiprogrammer SRQ status (from extended talk address 10) to determine why service was requested. Notice that in accordance with HP-IB standards, a serial poll will cause the Multiprogrammer to clear the SRQ line. A serial poll will not, however, reset the 6942A's status byte (to 128). Until its status byte is reset, the Multiprogrammer cannot generate new service requests and will continue to respond affirmatively to serial polling. Reading Multiprogrammer SRQ status (red 72310/ENTER 723.10) will reset the status byte message, as well as clear the SRQ line.

4-14 Example 4-5 illustrates using the controller's interrupt system to monitor the SRQ line, serial poll the Multiprogram-

mer when SRQ is set, and then read back Multiprogrammer status. In line 20 (9825) or lines 20 through 40 (9835/9845), an interrupt linkage between HP-IB interface 7 and a subroutine labeled "multi" is established and interface 7 is enabled for interrupts. When an interrupt occurs (SRQ line is set), the program jumps to line 100. Line 100 is a serial poll of the Multiprogrammer followed by a check of the status byte stored in variable G. If the 6942A requested service (G = 64), complete SRQ status information is read back to the controller and stored in variables A through F. If the Multiprogrammer did not request service (G#64), a return to the main program is executed. Chapter 5 describes all of the status information returned and all of the conditions (instruction completions, error detection, etc.) that will cause the Multiprogrammer to set the SRQ line.

Example 4-5. Serial Poll and SRQ Status Read

9825 Controller

```
20: oni 7,"multi"⟩eir 7
                •
                •
                •
100: "multi":rds(723)→G
101: if G=64⟩red 72310,A,B,C,D,E,F
102: iret
```

9835/9845 Controller

```
 20   ON INT #7 GOSUB Multi
 30   CONTROL MASK 7;128
 40   CARD ENABLE 7
                •
                •
100 Multi: STATUS 723;G
110   IF G=64 THEN ENTER 723.10;A,B,C,D,E,F
120   RETURN
```

4-15 **Parallel Poll.** In time-critical situations, it may be desirable to use the HP-IB parallel poll function. A parallel poll permits the controller to check the SRQ status of up to eight devices at one time. Each device sends a single bit in response to a parallel poll. The parallel poll bit is a "1" if the Multiprogrammer requested service and is "0" if it did not. The Multiprogrammer supports the complete parallel poll function, including parallel poll, parallel poll configure and parallel poll unconfigure commands. Refer to the programming manual of the appropriate controller for a description on the use of this function. Note that the parallel poll message will not affect the SRQ line nor the Multiprogrammer status byte messages.

4-16 DATA MESSAGE PROCESSING

4-17 The following paragraphs provide a general overview of the way in which the 6942A Multiprogrammer processes data messages (instructions and data). Figure 4-1 is a simplified instruction flow diagram which should be referred to when reading this discussion. In addition to the instruction and data flow, the diagram illustrates five microprocessor controlled programs which are required to process the instructions. The five programs are internal (stored in ROM) to the Multiprogrammer and are labeled as follows: HP-IB Communications, Instruction Decoding, Data Conversion, Instruction Sequencing, and Instruction Execution. Although these programs are transparent to the user, they are shown here to aid in describing how the instructions are processed. The Multiprogrammer is designed to allow all five of these programs to run concurrently. Thus, while one or more instructions are executing, additional instructions can be simultaneously read into the Multiprogrammer, decoded, converted to the internal format, and sequenced. This allows the Multiprogrammer to make the most efficient use of its microprocessor, thereby providing enhanced instruction sequencing and control capabilities, and also providing the maximum system throughput.

4-18 To assist in understanding how an instruction is processed, let's follow the execution of first an output instruction (OP) and, then an input instruction (IP). The OP (Output Parallel) and IP (Input Parallel) instructions are part of a powerful set of 32 instructions which provide a high degree of programming flexibility. The instruction set is described in detail in subsequent Chapters. All instructions, either input or output, originate as part of an output statement from the 9825, 9835, or 9845 system controller (see paragraph 4-6).

4-19 Output Instructions

4-20 For this discussion assume that output instruction "OP,1,123T" is sent over the HP-IB to the 6942A Multiprogrammer. A high speed buffer accepts up to eight characters (ASCII) at rates of up to 800,000 bytes per second. The HP-IB Control program takes the data from the first buffer and places it into a 128 character buffer. The purpose of the buffers is to decouple the HP-IB from the 6942A. As instructions are processed, the Multiprogrammer takes one character at a time from the buffer. Without the buffers, the HP-IB would have to wait for the Multiprogrammer to complete this one-at-a-time processing. With the buffers, 128 characters can be stored at a rate of 40,000 characters per second. Thus, instructions and data can be quickly down loaded from the controller into the Multiprogrammer, freeing the controller and HP-IB for other operations while the Multiprogrammer is processing the instructions. Of course if more than 128 characters are sent to the Multiprogrammer at a high rate, HP-IB operations will be slowed down by the Multiprogrammer's processing.

4-21 The Instruction Decoder and Data Conversion programs take characters out of the buffer setting up "instruction modules" in 6942A mainframe memory that contain the card address and card data converted to binary. The instruction modules contain all the information necessary to execute the instruction. In the OP example, after card address 1 is converted to binary, the Data Conversion program determines how the card is formatted and converts its data accordingly. Assuming that card 1 is formatted as a decimal data type with an LSB = 1, the program converts 123 to unsigned binary. The

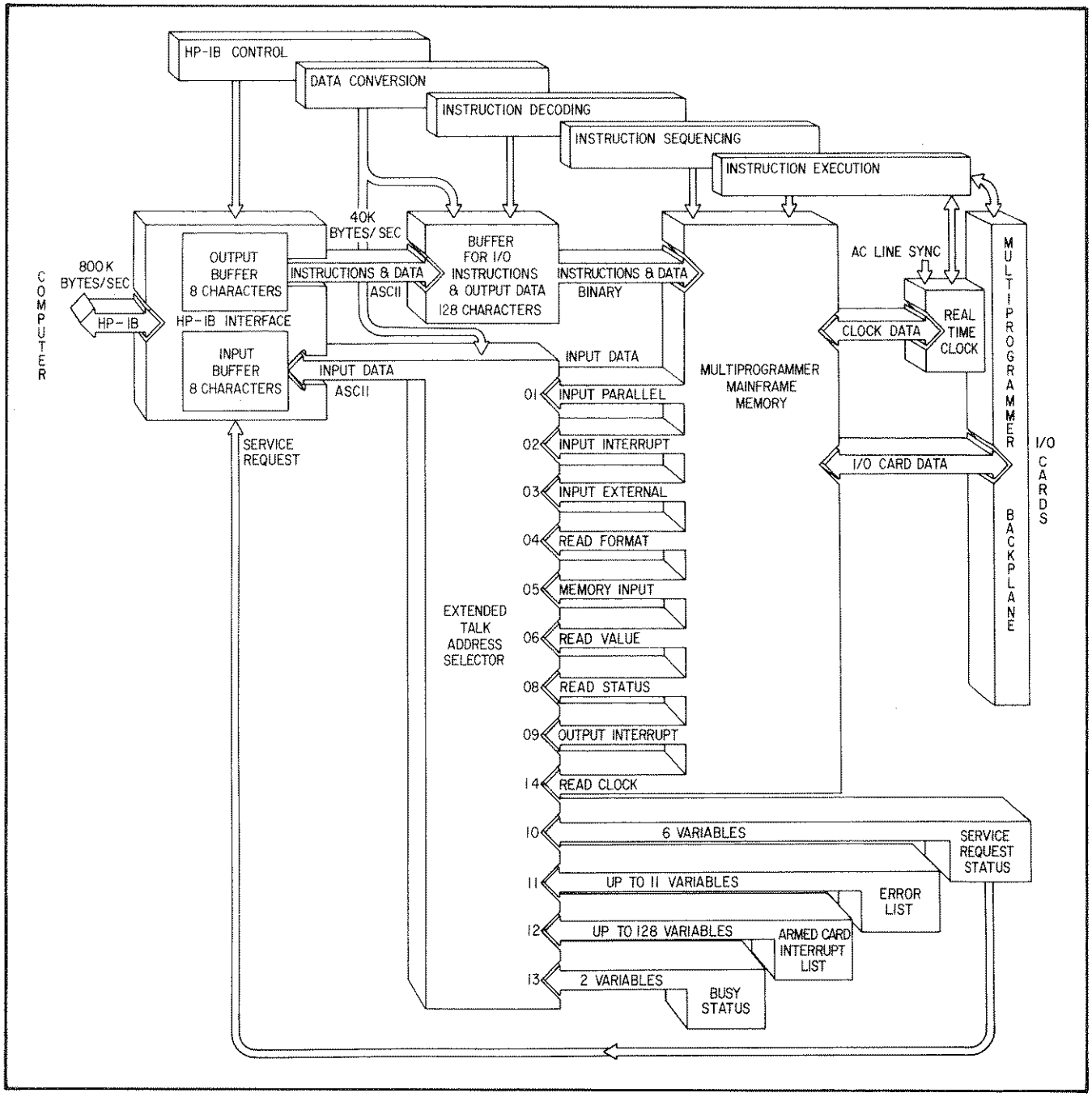


Figure 4-1. Instruction Processing, Flow Diagram

I/O card data formats are described in paragraph 4-64.

4-22 The binary equivalents of the card address and data values are placed in the instruction module in mainframe memory along with all other information required to execute the OP instruction. Because of the sequencing capabilities provided by the Multiprogrammer (e.g. serial or parallel processing modes), the Instruction Sequencing program is required to determine when the instruction can run. The Multiprogrammer "wakes-up" in the serial mode of instruction sequencing when it is first turned-on. In the serial mode, most instructions are processed one-at-a-time and the next instruction cannot start until the previous one has completed. In the parallel mode, the Multiprogrammer allows certain instructions to run concurrently (in parallel). Instruction sequencing is further described in Chapter 5.

4-23 Since, in our example, the OP is the only instruction being processed, it can execute immediately. Thus, the sequencing program informs the Instruction Execution program that the OP can run. The execution program provides the interface to the I/O cards and initiates all card operations. The

execution of our sample OP instruction consists of sending 123 to card 1 and then cycling the card. The term "cycle" means "to cause the card to perform its function". If card 1 is a Digital Output card, cycling causes the binary equivalent of 123 to be transferred from a storage register on the card to the card's output terminals. Card operations including cycling are described in more detail in paragraph 4-32. When the card has completed all operations, the instruction module is terminated and the OP instruction is completed.

4-24 Input Instructions

4-25 The processing of an input instruction, such as "IP,2T", is similar to that described for an output instruction. It is sent over the HP-IB to the 128 character buffer and is decoded and sequenced in a manner similar to the OP. The card address "2" is converted to binary; however, an IP instruction has no data to convert. When it is executed, an IP cycles an input card, reads data from the card, and stores the data in the instruction module which was setup when the instruction was decoded. Although the execution of the instruction is over after the data is stored in the instruction module, the data still has to be sent back to the controller. Therefore, the instruction module is saved until the data can be read back by the controller.

4-26 Reading Back Data from Instructions

4-27 The controller reads back data from an instruction by executing a read statement (see paragraph 4-7) from the applicable HP-IB extended talk address. In our example, the HP-IB extended talk address is 01 (for an IP instruction). The HP-IB Control program detects the read statement, signals the Extended Talk Address Selector to get the appropriate data, converts it to the specified format, and transmits it over the HP-IB to the controller. If the input card in slot 2 is an A/D Converter card, the data will be converted to a value representing volts. When all the data has been sent back to the controller, the instruction module is terminated and the IP instruction is completed.

4-28 Reading Back Status Information

4-29 As shown in Figure 4-1, besides reading back data from instructions (Input Parallel, Input Interrupt, etc), the Multiprogrammer can also read back four different types of status information: Service Request Status, Error List, Armed Card Interupt List, and Instruction Busy Status. The desired status information is processed and read back to the controller by the HP-IB Control program and the Extended Talk Address Selector in a manner similar to that described above for the Input Parallel (IP) instruction. The different types of status information are described in subsequent chapters.

4-30 Real Time Clock

4-31 The Multiprogrammer has a real time clock which provides the time of day in days, hours, minutes, and seconds. It also provides control information to the Instruction Sequencing Program. Real Time Clock processing runs in parallel with all other Multiprogrammer operations. Certain System Timing instructions, described in Chapter 5, use the Real Time Clock in sequencing events and in measuring elapsed time.

4-32 I/O CARD OPERATIONS

4-33 Even the simplest I/O card data transfers, such as with OP and IP instructions, require the cards to perform multiple operations. An explanation of what these operations are and why they are needed will help in understanding how the instructions work. This discussion explains the basic operations performed by both output and input cards, using an OP instruction to a Digital Output card and an IP instruction to a Digital Input card as examples. All I/O cards, both output and input, have a common circuit called the Control chip which is a large 40 pin device near the front of the card. This circuit interfaces the card to the microprocessor via the Multiprogrammer backplane and controls all communications between the backplane and the I/O card. Using identical interfaces for all cards eliminates the need for special instructions for each card type. The result is a general purpose instruction set where all instructions can be used with all cards.

4-34 Output Cards

4-35 Figure 4-2 is a simplified block diagram of a Digital Output card illustrating the data flow, the major control signals, and the signals available at the card's edge connector. Referring to Figure 4-2, let's look at exactly what operations are performed when programming an OP instruction to a Digital Output card. Assume that the card is installed in card slot 1 and instruction "OP,1,123T" is programmed. This instruction sends the binary pattern representing 123 to the card. The OP instruction performs two basic operations on the card. First, the data is sent to the first rank storage register on the card. Although the card now has the data, it has not yet been sent to the output terminals. The second operation consists of cycling the card which transfers the data to the card's edge connector via the second rank storage register, the enable gates, and the logic level drivers. The cycling operation also generates a gate signal which initiates the gate/flag sequence (handshake cycle). Completion of the gate/flag sequence indicates that the card has completed operations. With the gate/flag jumper installed as shown in Figure 4-2, the sequence is completed instantaneously. Removal of this jumper allows the external device to determine the completion of the handshake cycle (see paragraph 4-38). Whenever an I/O card completes its handshake cycle the end-of-process (EOP) signal is generated. Certain instructions (such as OP and IP) require that each card, specified in the instruction, signal (interrupt) the microprocessor when it completes operations. These types of instructions automatically "arm" the card which will cause the EOP signal to generate the "Microprocessor Interrupt" signal (see paragraph 4-44). The arming circuit is inside the control chip on Figure 4-2.

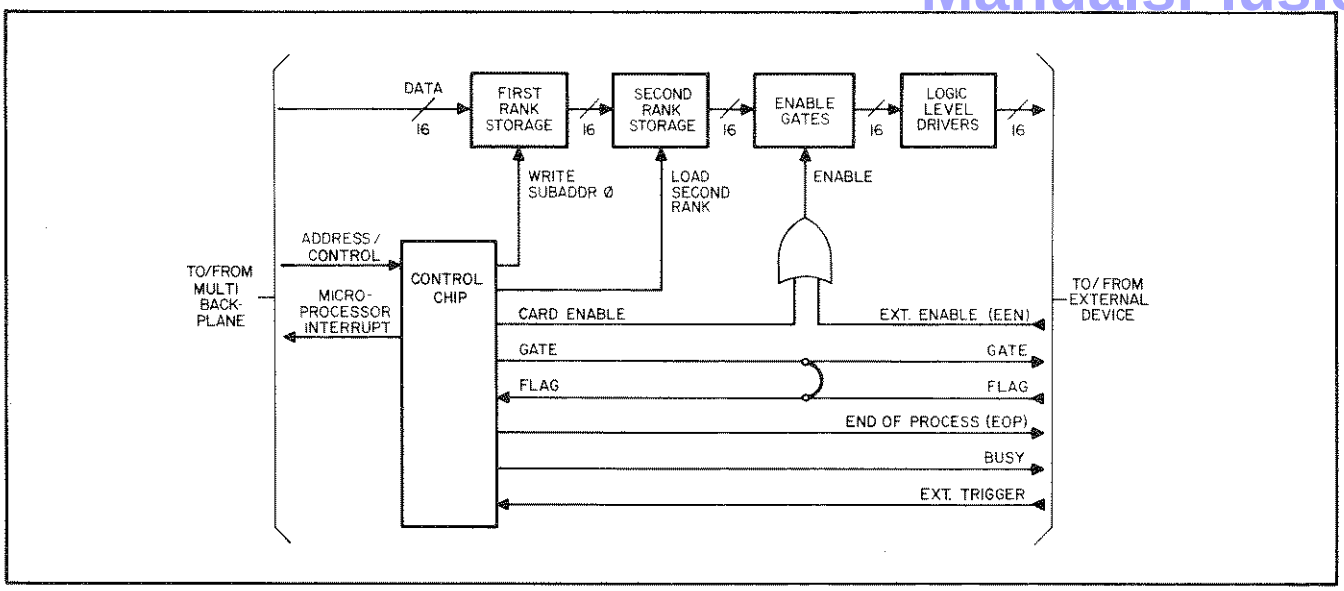


Figure 4-2. Digital Output Card, Simplified Block Diagram

4-36 Internal Gate/Flag. To explain an internal gate/flag operation, assume that data is sent to a Relay card. The relays require 6 msec to close. To ensure that data (in an OP or OS instruction) can not be sent to a Relay card at a rate faster than 6 msec, the relay card uses a 6 msec one-shot timer in the gate/flag circuit (see Figure 4-3). The gate signal starts the timer when the relay card is cycled, and 6 msec later when it completes, the flag is generated indicating that the data transfer process is complete (relays have settled).

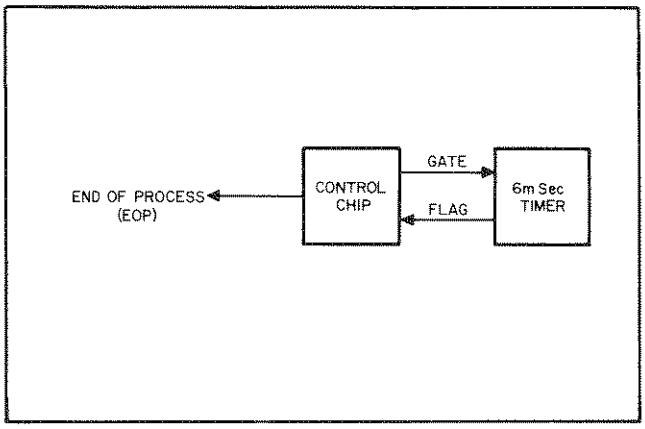


Figure 4-3. Relay Card Internal Gate/Flag Timer

4-37 The gate/flag handshake on the D/A Converter card works in a similar manner, except the timer is only 6 microseconds. Cards like the Digital Output shown in Figure 4-2, require no delay whatsoever. As soon as the data is stored in the second rank of storage, it appears at the output terminals and the gate/flag sequence completes instantly.

4-38 External Gate/Flag. Even though a Digital Output card can process data instantaneously, the external device connected to the card might not be so fast. In this situation, if we send data to the card as fast as the card can take it, we might be sending it to the external device too quickly. As an example, assume we are using a 16-bit Digital Output card to send data to an external 16-bit D/A Converter. Although the Digital Output card will complete instantaneously (remember, flag is tied to gate), the external D/A requires 10 msecs to convert the digital signal to an analog voltage. If we send the data to the output card as fast as possible (faster than 10 msec), we will overwrite the data to the D/A, generating erroneous results.

4-39 To solve this problem, remove the gate/flag jumper (see Figure 4-2) from the Digital Output card to extend the gate/flag handshake to the external device. The external device, such as the D/A in the above example, can then control when the Digital Output card signals completion, thus, guaranteeing that it will always be ready for the next data word. This is called synchronizing the external device to the Multiprogrammer. Each data transfer occurs only when both the Multiprogrammer and the external device are ready for it.

4-40 A card can remain busy for an indefinite period, thus any type of external device, including those that require human interaction, can be synchronzied to the Multiprogrammer with the external gate/flag handshake. The controller can always send a sequence of instructions e.g. (OP's) to the Multiprogrammer and be gauranteed that all of the data will be taken correclty, in the proper order.

4-41 When using the external gate/flag handshake, you must be sure that the external device will always respond with a flag. If for any reason, a flag is not returned, the OP associated with the card will not complete, and because of the serial mode, no other instructions will run until the OP completes, resulting in the system hanging up. Appendix D will discuss methods to recover from this condition. To understand how the gate/flag handshake works, lets examine exactly what the signals do.

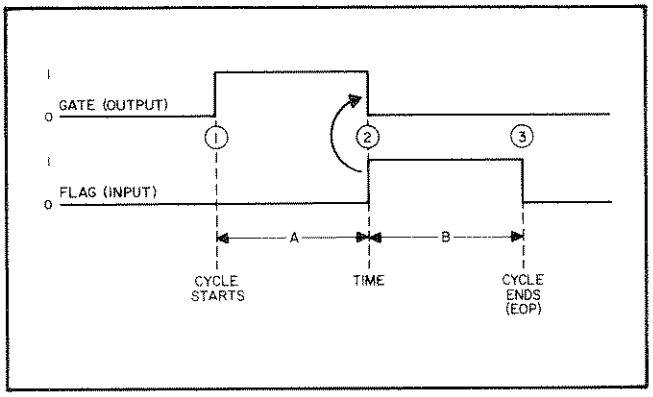


Figure 4-4. Gate/flag Handshake

4-42 As shown in Figure 4-4, the handshake begins at time (1), when the control chip starts the cycle by making GATE = 1. (The data is latched into the second rank storage register simultaneously). At time (2), the external device acknowledges receipt of the gate signal by setting FLAG = 1, and this causes the control chip to clear gate (GATE = 0). The sequence does not complete (and the control chip does not signal the microprocessor) until the trailing edge of flag, at time (3). This gives the external device maximum flexibility. It can, depending upon its particular circuitry, delay the handshake in either time period A or B. For example, an external D/A response to the gate signal by setting a busy status line. At the completion of the conversion (10 msec), the busy line is reset = 0. This busy status line can be tied directly to the flag input, in which case the D/A holds off the output card in time period B.

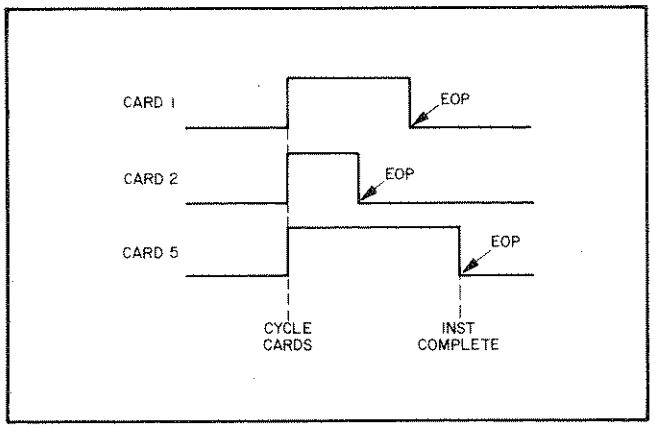


Figure 4-5. Cycling Multiple Cards with OP Instruction

4-43 Multiple Output Cards. Most types of output cards have first rank and second rank storage registers connected in exactly the same sequence. Thus, an OP instruction can be given to a series of various types of output cards with identical results. The cycling step of the OP instruction is performed simultaneously on all cards specifed in the instruction. When a card is cycled, the data from the first rank of storage is loaded into the second rank, where it is proprogated to the output terminals. Until the second rank storage register is updated, the output data will remain unchanged. Assume that an output parallel instruction "OP,1,123,2,456,3,789T" is programmed. As shown in Figure 4-5, the OP loads the data into each card, and then cycles all cards simultaneously. As each card completes, it generates an EOP, however the "OP" instruction does not complete until; all cards have completed.

4-44 Microprocessor Interrupt. As described previously, whenever an I/O card completes a data processing sequence, an EOP signal is generated. If the card is programmed by an instruction that automatically arms the card (OB,IP,OP,II,OS,IE, or OI), the EOP signal will generate an interrupt to the microprocessor which will then determine the instruction that the card is assigned to, disarm the card, and continue the processing according to the type of instruction and processing mode of the Multiprogrammer. For example, if the interrupting card is the only card assigned to an "OP" instruction and the Multiprogrammer is in the serial mode, the next instruction is allowed to start.

4-45 Not all instructions arm the card. The write and cycle (WC) instruction used to program output card, simply sends data to the card(s) then cycles the card(s). The cycle (CY) instruction simply cycles the card. Cards may also be cycled by appling an external trigger to the external trigger input on the card (see Figure 4-2). The Arm Card (AC) instruction can be used to enable cards that are being externally triggered or programmed with a "WC" or "CY" instruction to interrupt the microprocessor upon completion of their data processing. These interrupts are called "armed card interrupts" and are discussed in detail in Chapter 6. Note that if a card is cycled without first being armed, the EOP signal will go high indicating that the card completed its operation but will not be reset by the microprocessor until another instruction addressing the card is programmed.

4-46 Card Enable. All output card types have a card enable circuit that disables the card outputs when power is initially applied. (The enable circuit for the Digital Output card is shown in Figure 4-2.) A card's outputs remain disabled until the Multiprogrammer's system enable (SYE) line is turned on and the particular card is cycled for the first time. The SYE line is one of the control signals applied to the control chip on each card and is turned on when the first instruction is executed. An individual card is not enabled however until it is cycled. When disabled, an output card is set to a "safe" state (e.g. relays on a Relay Output card are opened).

4-47 Input Cards

4-48 Figure 4-6 is a simplified diagram of a Digital Input card. The input operation is the reverse of the output. The card is cycled, loading the data on the input terminals into the input latch. At this point the control chip signals the microprocessor that the card has completed and the microprocessor responds by reading the data from the input latch. The identical sequence, sampling the data on the edge connector, loading it into the input latch, signalling the microprocessor, and then having the microprocessor read the input latch, is performed on every type of input card. Note that the card enable circuit shown in Figure 4-2 for the Digital Output card does not apply to the input cards.

4-49 Assume that the input card is installed in slot 4 and instruction "IP,4,T" is programmed. The sequence of events for an input from the card in slot 4 is as follows:

a. Cycle card
   1) Initiate gate/flag handshake
   2) Load data into input latch at completion of handshake
b. Signal microprocessor that card has completed (interrupt)
c. Send data from card to mainframe memory.

At a later time the controller can read the data from the mainframe memory over the HP-IB using the IP instructions extended talk address of 01 (red 723.01). Refer to paragraph 4-7.

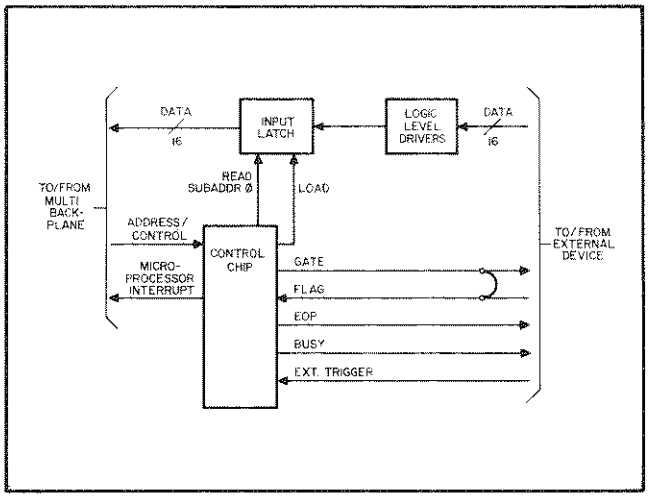


Figure 4-6 Digital Input Card, Block Diagram

4-50 **Internal Gate/Flag.** Just as with output cards, some input cards complete their cycles instantly while others take varying lengths of time to complete. The A/D input card, for example, takes 33 microseconds to take a reading. Input cards also make use of the gate/flag handshake. When an A/D card is cycled, the gate tells the A/D converter to process a reading. At the completion of the conversion, the A/D signals the control chip via the flag line. This causes the converted data to be stored in the input latch, and a card completion (interrupt) signal to be sent to the microprocessor from the control chip. Some cards, such as the Digital Input, require no processing time, so just as with the Digital Output card, the flag signal is tied to the gate. This forces the card to complete as soon as the Gate is generated.

4-51 **External Gate/Flag.** Again, the situation arises when the card can complete very quickly, but the external device might be slow. In the case of inputs, if the data is loaded into the input latch before the external device can transmit the correct data to the input card, erroneous data will be stored in the input latch. To avoid this problem, a way to synchronize the external device with the Multiprogrammer is required so the data is not stored until the external device has transmitted it. Just as with output cards, the gate/flag handshake can be extended to the external device. Again refer to the gate/flag handshake in Figure 4-4. Although the card never completes until the end of the cycle, the user can specify whether the data should be stored in the input latch on the leading or trailing edge of the flag signal (see Digital Input card Manual).

4-52 Additional information on the operation of the I/O cards will be given throughout Chapters 5 and 6. A complete summary of all the I/O card information, both general card information, and specifiic information about each card type, including example programs, is given in Chapter 7.

4-53 I/O CARD SUBADDRESSES

4-54 The Multiprogrammer utilizes an I/O card subaddressing scheme that allows programming the more complex (multi-functioned) I/O cards as well as the simple 16-bit Digital Input and 16-bit Digital Output cards described in the previous paragraphs. There are eight subaddresses (4 write and 4 read) that can be used to program specific functions on an I/O card. The appropriate subaddress number (0, 1, 2, or 3) is appended to the card slot address in the instruction syntax. Note that the subaddress number defaults to 0 if it is not specified in the address; therefore, it may be omitted when addressing subaddress 0. For example address "1.0" specifying subaddress 0 on card 1 can also be programmed as "1". To program subaddress 3 on card 1, "1.3" must be programmed. Refer to Chapter 5 for a detailed description of instruction syntax.

4-55 Subaddress 0 is the main address and is used when sending data (write 0) to an output card or when reading data (read 0) from an input card. The OP and IP instructions, described in paragraphs 4-35 and 4-49, used subaddress 0 to program the Digital Output and Digital Input cards, respectively. Write subaddresses are specifed when output instructions (e.g. OP, OS) are executed, and read subaddress are specified when input instructions (e.g. IP, IE) are executed. The data values than can be sent (write 0) to an output card or read (read 0) from an input card depend upon the particular I/O card's data format parameters (see paragraph 4-64). The following paragraphs describe the subaddressing scheme used with each type of I/O card that is currently available with the 6942A/6943A Multiprogrammer System. Table 4-2 summarizes the subaddressing scheme by listing each I/O card by type and specifying the applicable subaddresses.

4-56 Output Card Subaddresses

4-57 The desired output value is sent to the applicable output card using write subaddress 0. Read subaddress 3 can be used to ensure that the output value sent was actually received by the output card. The Pulse Train and Timer/Pacer output cards use additional subaddress to program special functions on each card. The Pulse/Train card uses the write 1 and write 2 subaddresses to program the frequency (period) of its output pulses. The Timer/Pacer card uses write subaddress 2 to program its mode of operation (one-shot or recirculate). Also, as noted in Table 4-2, for certain special conditions the write 1 subaddress can be used to program the period of the pulses produced by the Timer/Pacer card. Complete details on programming output cards are provided in Chapter 7, pages 7-1 through 7-17.

Table 4-2. I/O Card Subaddresses

I/O CARD TYPE	SUBADDRESSES							
	WRITE				READ			
	0	1	2	3	0	1	2	3
Output Cards								
Resistance Output 69700A-69706A	Output Value	—	—	—	—	—	—	Output Value
Voltage D/A 69720A	Output Value	—	—	—	—	—	—	Output Value
Current D/A 69721A	Output Value	—	—	—	—	—	—	Output Value
Relay Output 69730A	Output Value	—	—	—	—	—	—	Output Value
Digital Output 69731A	Output Value	—	—	—	—	—	—	Output Value
Pulse Train 69735A	No. Of Pulses	Period Multr	Period Magtd	—	—	—	—	No. of Pulses
Timer/Pacer 69736A	Pulse Width	* Period Multr	Mode	—	—	—	—	Pulse Width
Input Cards								
High Speed A/D 69751A	Self Test Value	—	—	—	Input Value	—	—	Self Test Value
Isolated Digital Input 69770A	Self Test Value	—	—	—	Input Value	—	—	Self Test Value
Digital Input 69771A	Self Test Value	—	—	—	Input Value	—	—	Self Test Value
Counter Card 69775A	Count Preset	—	—	—	Present Count	—	—	Count Preset
Interrupt Card 69776A	Ref Word	Mode	Mask	—	Intrpt Word	Ext Word	Mask	Ref Word
Memory Cards								
Memory Card 1 P/O 69790A	Write Data	Mode	—	—	Read Data	—	—	—
Memory Card 2 P/O 69790A	Ref Word	Diff Counter	Write Pointer	Read Pointer	Read Pointer	Diff Counter	Write Pointer	Ref Word

*Subaddress 1(period multiplier) on a Timer/Pacer card is not programmable with the card set to the special auto-ranging data type (see paragraph 4-73). Subaddress 1 on the Timer Card can only be programmed if another data type is selected (see Timer Card manual).

4-58 Input Card Subaddresses

4-59 The input value is read from the applicable input card using read subaddress 0. Input cards use the write 0 and read 3 subaddresses for test purposes. Write 0 is used to send test data to a dummy register on an input card. The test data can then be read back using the read 3 subaddress to ensure that the Multiprogrammer can communicate with the input card. Complete details on programming input cards are provided in Chapter 7, pages 7-17 through 7-23.

4-60 Counter Card Subaddresses

4-61 As shown in Table 4-2, the count preset value is sent to the card using the write 0 subaddress. Read 3 can be used to ensure that the desired count preset value was received by the card. The actual count in the card can be read at any time using the read 0 subaddress. Details in programming a Counter Card are provided in Chapter 7 on pages 7-23 through 7-25.

4-62 Interrupt and Memory Card Subaddresses

4-63 As shown in Table 4-2, these cards use all of the subaddresses. The subaddresses allow various input and output operations to be programmed. Note that the Memory Cards are two interconnected cards that function as a single memory. Memory card 1 contains the memory and the mode control circuits. Memory card 2 contains a reference register, differential counter, write pointer, and a read pointer. All subaddresses are used on Memory Card 2. Refer to Chapter 7 for complete programming details. The Interrupt Card is covered in pages 7-26 through 7-28 while the Memory Cards are described in pages 7-29 through 7-34.

4-64 I/O CARD DATA FORMATS

4-65 One of the major features of the Multiprogrammer is its ability to communicate with the controller using "Engineering Units" data values. Instead of requiring the programmer to converse with the I/O cards in a specific data format, such as octal or decimal, the Multiprogrammer allows the user to program in units that "make sense" for the particular application. A Voltage D/A card is programmed in volts, a Current D/A card in milliamps, and a Timer card in seconds, milliseconds, or microseconds. Cards are designed to "wake-up" in the most appropriate units when power is first applied or when the system is reset. The units can be respecified however, for any card at any time.

4-66 Programming in "Engineering Units" means that a specific linear relationship (a scale factor) is assigned between the units the programmer is conversing in (engineering units), and the binary data at the I/O card. For example, a binary value of 1 sent to a Voltage D/A card will cause the card to output .005 volts. Once this relationship is defined in the Multiprogrammer firmware, the card can be programmed directly in volts. If we program .05V (.005V × 10), the Multiprogrammer will send the binary value of 10 to the Voltage D/A card, which will then output .05V. The use of engineering units is described in paragraph 4-86.

4-67 Each I/O card has five programmable data format parameters which specify how the Multiprogrammer will process the data it sends to or receives from a particular card. The five parameters consist of the data type, least significant bit (LSB) value, size (no. of bits), an optional programmable limit, and a card identifier. The data format parameters for all I/O cards in the system are reported and stored in Multiprogrammer memory so that each card's data can be processed properly. The Multiprogrammer accepts data, converts it to the form specified by the applicable data format parameters, and then sends the converted data to the card for output instructions, or reads it back to the controller for input instructions. Two instructions interact with the data format parameters; the Set Format (SF) and the Read Format (RF) instructions. The RF instruction reads the current value of all five parameters while the SF instruction is used to modify one or more of the parameters. When the SF instruction is used to modify a card's "wake-up" parameters, the Multiprogrammer will recognize the new parameters until another SF changes them or until the Multiprogrammer is reset. Once the Multiprogrammer is reset, the I/O card's data format parameters revert to their "wake-up" values. Executing an HP-IB Device Clear command or turning the power on will initiate the system self test and also reset the Multiprogrammer (see paragraph 2-38). The RF and SF instructions are described in detail in Chapter 5. Descriptions of the five data format parameters are provided in the following paragraphs.

4-68 Data Type

4-69 The data type parameter specifies the conversion routine that the Multiprogrammer will apply to the data before sending it to the card (data output) or reading it back to the controller (data input). The Multiprogrammer supports six different data types. Each data type is identified by a code between 1 and 7. When using either the RF or SF instruction, the data types are specified by the data type codes. Note that the data types listed below include the form that the data is in when it is processed at the I/O card as well as the form that the user programs in.

Data Type Codes

Code	User Programs in	Card Data
1	*Decimal	2's Complement Binary
2	*Decimal	Signed/Magnitude Binary
3	*Decimal (pos. only)	Unsigned Binary
4	*Decimal + Range	Timer-Auto Range
5	Not Used	—
6	*Decimal (pos. only)	Unsigned BCD
7	Octal integer	Unsigned Binary

* Fixed point decimal number with a maximum of 3 places to the right and 7 places to the left of the decimal point.

4-70 All of the data types except octal provide engineering units (e.g. volts, milliamps, milliseconds, etc). programming

capability where the user programs in fixed point decimal numbers that represent the value of the applicable engineering units. Additional information on the various data types (2's complement, sign/magnitude binary, etc.) is given in Appendix A. The octal data type (code 7) is provided for applications that require accessing individual bits on a card independent of the other bits. Octal is particularly useful when programming the 16-bit relay output card. The octal pattern is easily converted to the desired relay contact closures.

4-71 Least Significant Bit Value (LSB Value)

4-72 The LSB value is the scale factor that specifies the relationship between the engineering units programmed and the binary number at the I/O card. It indicates what engineering unit value corresponds to the smallest binary value. This scale factor is used by the firmware to perform linear translations from engineering units to binary for output instructions, and from binary to engineering units for input instructions. A complete description of the methods involved, and the procedure for computing the LSB value, are described in paragraph 4-86. Since the Octal data type (code 7) does not use engineering units, the LSB value does does not apply. The Timer data type (code 4), because of its auto-ranging capability, uses a range code. I/O cards that use any of the other data types "wake up" with an appropriate value assigned to the LSB parameter. The LSB "wake-up" values can be changed using the SF instruction. When using an SF instruction, the allowable range of values for the LSB parameter is from .001 to 65.535. Negative LSB values are not allowed.

4-73 Range Code

4-74 Because of the auto-ranging capabilites of the Timer data type (code 4), a Range Code is specified instead of an LSB value. Auto-ranging is used because the programmable range of the Timer/Pacer card far exceeds the standard 16-bit range. The range code specifies one of three engineering units the card can be programmed with. The actual LSB value assigned to the card is automatically computed by the firmware for each output to the card. The range codes are: S for seconds, M for milliseconds, and U for microseconds. As described in paragraph 4-81, the timer wakes up being programmed in milliseconds. The full programming ranges for the three codes are:

Code	Range
U (Microseconds)	1.0 to 4,294,836 Microseconds
M (Milliseconds)	.001 to 4,294,836.225 Milliseconds
S (Seconds)	.001 to 65,535 Seconds

Refer to the Read Format and Set Format instructions in Chapter 5, and the Timer card description in Chapter 7 for complete programming details.

4-75 Size

4-76 The size parameter on each card specifies whether the card utilizes 12 or 16-bits of data. For certain applications, it may be desirable to change the size parameter on a particular card. For example assume that the external digital outputs from a 12-bit High Speed A/D card are connected to the digital inputs on a 16-bit Memory Card. In this example, an SF instruction can be used to change the size parameter of the Memory card to 12-bits.

4-77 Limit

4-78 The limit parameter is a positive engineering units number that must be within the range allowed for the specific card's data type, LSB, and size parameter values. The absolute value of the data (number) programmed to the card must be less than or equal to the limit specified, or an error is detected. All cards wake-up with their programming range specified by the data type, LSB, and size parameters. For example, a Voltage D/A card wakes-up with a programmable range from −10.240 to +10.235 (see paragraph 4-81). If desired, the SF instruction can be used to set the limit parameter on the D/A card to 10.000. Now if the absolute value of the number programmed (e.g. −10.15 or 10.15) exceeds the limit (10), the card will not be programmed and an error will be detected. Note that limits cannot be set for I/O cards using the octal or the special timer data types.

4-79 Card Identifier

4-80 Each I/O card type is assigned a card identifier code in the range from 0-63. The card ID codes are used to enable card dependent features built into the firmware and also to identify what card types are installed. The card ID code is normally used when data is read back from an RF instruction. The card ID codes are listed below along with the corresponding card type.

ID Code	Card Type
1	Timer/Pacer
4	Memory Card 2
6	Memory Card 1
12	Word Interrupt
42	Digital Output
44	Digital Input
45	Isolated Digtal Input
46	Relay Output
48	Voltage or Current D/A
52	A/D Converter
56	Resistance Output
58	Pulse Train
63	Counter

4-81 I/O Card Data Format Wake-up Values

4-82 Circuits on each I/O card specify the "wake-up" values for the data type, LSB, size, and card ID parameters. Note that these circuits do not affect how the card performs its function. Their only purpose is to specify the data type and LSB parameters which determine how the Multiprogrammer

firmware will process the data it sends to or receives from the particular I/O card. Jumpers on each I/O card allow the "wake-up" values of the data type and LSB parameters to be changed from their factory selected values. If an application calls for a specific I/O card to be used with a data type and/or LSB value that will always be different from the factory set "wake-up" values, it is recommended that the jumpers be changed. The card ID and size parameter circuits are hard-wired on each I/O card and can only be changed from their "wake-up" values with the SF instruction. As mentioned previously, the SF instruction can be used to change some or all of the data format parameters.

4-83 Whenever the system is run through self test (see paragraph 2-38), every card slot is accessed, and a "wake-up" word for each card is read into the Multiprogrammer's firmware. The wake-up word specifies the values of all five of the data format parameters. Each card wakes-up with its limit set, a card can be programmed to its maximum value. The wake-up values for the card ID, data type, LSB and size parameters are given in Table 4-3 for each type of I/O code. The associated card name and model number are given along with the applicable parameters. Note that most cards wake-up in data type 3 with an LSB of 1 and a size of 16-bits. The range of values that can be programmed is calculated for each card (except the Timer) from the data type, LSB, and size parameters. The Timer/Pacer card uses a range code instead of an LSB value (see paragraphs 4-73).

Table 4-3. I/O Card Wake-Up Values

CARD	MODEL	ID CODE	DATA TYPE		LSB	SIZE	RANGE
			USER PROGRAMS IN	CODE			
Timer/Pacer	69736A	1	Decimal + Range	4	M*	16	.001 to 4,294,836.225 msec
Memory Card 2	P/O 69790A	4	Decimal (pos only)	3	1	16	0 to 65,535
Memory Card 1	P/O 69790A	6	Decimal (pos only)	3	1	16	0 to 65,535
Interrupt	69776A	12	Decimal (pos only)	3	1	16	0 to 65,535
Digital Output	69731A	42	Decimal (pos only)	3	1	16	0 to 65,535
Digital Input	69771A	44	Decimal (pos only)	3	1	16	0 to 65,535
Isolated Digital In	69770A	45	Decimal (pos only)	3	1	16	0 to 65,535
Relay Output	69730A	46	Decimal (pos only)	3	1	16	0 to 65,535
Voltage D/A	60720A	48	Decimal (pos or neg)	1	.005V	12	−10.240 to +10.235V
Current D/A	69721A	48	Decimal (pos or neg)	1	.01mA	12	−20.48 to +20.47mA
High Speed A/D	69751A	52	Decimal (pos or neg)	1	.005V	12	−10.240 to +10.235V
Resistance Output	69700A	56	Decimal (pos only)	3	1	12	0 to 4095
Resistance Output	69701A, 04A,05A	56	Decimal (pos only)	3	.01V**	12	0 to 40.95V**
Resistance Output	69702A,06A	56	Decimal (pos only)	3	.025V**	12	0 to 102.350V**
Pulse Train	69735	58	Decimal (pos or neg)	2	1	716	-32767 to +32767
Counter	69775A	63	Decimal (pos only)	3	1	16	0 to 65,535

*Special auto-ranging capability ensures best possible resolution.
**LSB and Range values in volts are specified for Resistance Output Card/Option 40 Power Supply combination.

4-84 Error Processing

4-85 After all 5 parameters have been set up (either with their wake-up values or with an SF instruction), the Multiprogrammer will start processing the card's data according to these specifications. Extensive error checking is performed on the data to guarantee that it is a legal value for the specific card. If any error is detected, the Multiprogrammer will not process the instruction and will store an error message explaining what the problem was. The following are some of the checks made by the Multiprogrammer for output instructions.

Range Check - After converting the data to binary, the Multiprogrammer checks to see if the data has overflowed out of the 12 or 16-bit card (determined by the size parameter). This check is performed before the limit check, thus an overflow will be reported (as a "data error") regardless of the limit programmed, (or if no limit was programmed).

Sign Check - Attempts to program a negative number when using data types 3 (unsigned binary), 4 (timer), 6 (unsigned BCD), or 7 (octal) will cause the system to store a "data error".

Roundoff- The Multiprogrammer always rounds off the engineering units value to the closest even multiple of the LSB value. For example, if the LSB = .005V, and the value programmed is 1.234V, the Multiprogrammer will round this off to 1.235V (247 × .005).

4-86 Programming in Engineering Units

4-87 As was explained before, engineering units allow the user to customize the way he communicates with his I/O cards. He can program the card in units that "make sense" for the application the card is used in. To do this, the LSB value is used as a scale factor by the firmware to make the appropriate conversions. The relationship between the engineering units, the binary value at the card, and the LSB value are as follows:

for outputs: (Engin Unit) / (LSB) = Binary Value

for inputs: (Binary Value) × (LSB) = Engin Units

4-88 Using the above formulas, it is easy to compute the maximum range of engineering unit values allowed. A 16-bit card, with data type 3 (unsigned binary), has a maximum binary value of 65,535. The maximum LSB = 65.535, thus:

max engin units = (binary) × (LSB)
= 65535 × 65.535
= 4,294,836.225

4-89 To compute the maximum negative number, use data type 1 (2's complement). The maximum negative binary value = -32,768, thus:

max engin units = (binary) × (LSB)
= −32768 × 65.535
= −2,147,450.880

4-90 To customize the engineering units for a particular application, re-arrange the expression.

(Engin Units) / (Binary Value) = LSB Value

As can be seen, all that must be known is a single engineering unit value, and the corresponding binary value to complete the LSB value.

Example: Some applications call for programming the D/A card in a percentage of the full scale value. The D/A card is a 12-bit, 2's complement card, thus it has 2^{12} or 4096 values. Since it is 2's complement, it is a bipolar card, with 2048 positive values (2047 positive values and 0), and 2048 negative values. To program as a percentage of full scale, the following relationship is used:

100% = 2047

thus LSB = (engin units) / (binary)
= 100/2047
= .049

If we assign an LSB = .049, we can program the D/A card directly in percentage of full scale.

4-91 Another example is using a pulse train output card to control a stepping motor. The pulse train card generates a programmable number of square-wave output pulses. The square-wave pulses can be connected to a stepping motor translator to control the magnitude (no. of revolutions), direction, and the speed of motor rotation. The 16-bit pulse train card "wakes-up" set for sign/magnitude programming (data type 2) with an LSB value of 1. For these data format values, the card is programmed directly in the number of output pulses desired (−32767 to +32767), where the polarity specifies direction of rotation.

4-92 In this application, it makes sense to customize the engineering units to be the "number of revolutions desired" instead of the "number of pulses" desired. Assuming that 200 pulses from the card translates into one stepping motor revolution, the LSB value is scaled as follows:

LSB = (engin units) / binary)
= 1/200
= .005

Now if we set the LSB value to .005 and program an engineering units value of 1 revolution to the card, the binary value of 200 will be sent to the card. This causes the card to generate 200 square-wave pulses which are translated into one stepping motor revolution. With the LSB set to .005 the maximum number of revolutions that can be programmed is:

.005 (LSB) × −32767 (max neg 16-bit value) = −163.835
and
.005 (LSB) × +32767 (max pos 16-bit value) = +163.835

or
−163.835 to +163.835 revolutions

Chapter 5
PROGRAMMING ESSENTIALS

5-1 This chapter covers the fundamentals of programming the Multiprogrammer system using a 9825, 9835, or 9845 Desktop Computer. Included is an overall description of the instruction set, instruction syntax conventions, basic processing modes of the 6942A, and detailed descriptions of twenty one of the basic instructions that are most often used in programming the 6942A. The remaining eleven instructions of the 32-instruction set are described in Chapter 6. These eleven instructions are for more advanced or special purpose use.

5-2 INSTRUCTION SET

5-3 Table 5-1 lists all 32 instructions. The opcode and name of each instruction is given, together with a brief description of the instruction and the paragraph(s) where a more detailed description can be found.

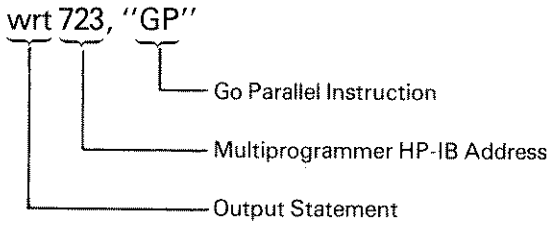
5-4 As indicated in Table 5-1, the instruction set can be subdivided into five functional groups; system control, output, input, card control, and system timing. A brief description of these five groups is presented in the following paragraphs.

5-5 System Control Instructions

5-6 These instructions establish the basic operating modes of the Multiprogrammmer system. Many of the instructions in this group control the way in which the Multiprogrammer sequences the other instructions that it receives from the HP-IB computing controller.

5-7 When the Multiprogrammer is first switched on, it "wakes-up" in the serial mode of instructions sequencing. In serial mode, most instructions are processed one-at-a-time and the next instruction cannot start until the previous one has completed. Example 5-1 shows how to place the Multiprogrammer in the parallel mode of instruction sequencing using a 9825 Desktop Computer.

Example 5-1. Sample System Control Instruction

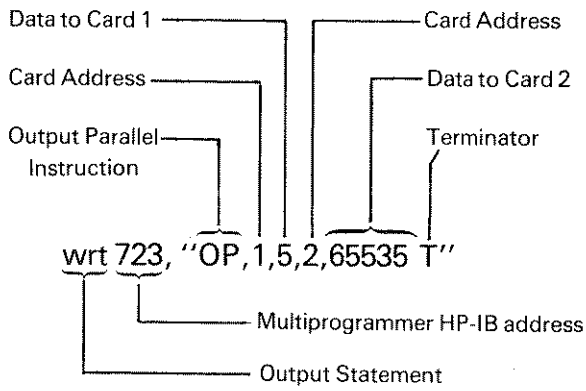


5-8 In parallel mode, the Multiprogrammer allows different instructions to run concurrently (in parallel). This mode is useful when controlling processes that are relatively independent of each other and it is necessary to reduce execution times.

5-9 Output Instructions

5-10 These instructions are used to program the Multiprogrammer's various output type cards. Example 5-2 illustrates the Output Parallel (OP) instruction. The OP instruction sends data to a group of output cards and then instructs them to begin simultaneously processing the data. The instruction completes when all addressed cards are producing an output that is equivalent to the data sent by the instruction. In the example, a 69720A Voltage D/A card in slot number 1 is instructed to produce a 5 Volt output and a 69731A Digital Output card in slot 2 is programmed so that it's 16 output bits are all logical 1's.

Example 5-2. Typical Output Instruction



5-11 Input Instructions

5-12 These instructions apply to the input cards. Obtaining data from an input card is a two-step process. First, one of the input instructions is sent to the Multiprogrammer directing a card, or group of cards, to capture data from the external world and store it in the 6942A's memory. Next, an input statement is used to read back the data to the controller.

5-13 In Example 5-3, an Input Parallel (IP) Instruction commands two input cards, in slots 5 and 6, to take simultaneous readings. After the cards have completed, the input data is stored in 6942A memory locations for immediate

Table 5-1. Instruction Set

Opcode	Name	Brief Description	Described in Paragraph
SYSTEM CONTROL			
GP	Go Parallel	Sets system to parallel mode of operation. In the parallel mode instructions having different opcodes are executed simultaneously.	5-57
GS	Go Serial	Restores system to the serial mode of operation in which all instructions (except OI and II) are executed one-at-a-time.	5-57
GI	Go Immediate	Sets system to immediate mode of operation which suspends all currently running instructions and allows a specified subset of instructions to run immediately. The GI instruction is used whenever an immediate response to an emergency situation is required.	6-52
GN	Go Normal	Returns system to the mode of operation that existed prior to execution of the GI instruction.	6-58
CG	Clear Group	Clears user defined group instructions from system memory. Group instructions provide a programming convenience.	6-9
SE	System Enable	Enables the outputs of all output cards in the system. The SE is used to enable the system after an SD was executed.	6-67
SD	System Disable	Disables the outputs of all output type cards in the system setting them to a "safe" state. The SD could be used in conjunction with the GI instruction to respond to an emergency condition.	6-63
OUTPUT			
OP	Output Parallel	Simultaneously programs a group of output cards. Instruction completes when all cards have completed.	5-63
OS	Output Sequential	Sequentially programs a card or a group of cards. As each card completes, the next card is programmed. Instruction completes when last card completes.	5-70
OB	Output Bit	Simultaneously sets or clears specified bits on an output card without affecting the other bits. Instruction completes when card completes.	5-77
OI	Output Interrupt	Same as OP, except that it always runs in parallel with other instructions. As each OI card completes, its adresss is stored in 6942A memory and SRQ is set.	5-193
WC	Write and Cycle	Same as OP, except WC completes immediately (does not wait for cards to complete).	5-87
WF	Write First Rank	Sends data to first rank storage on a card or group of cards. The WF provides "low level" control of a card and is used in conjunction with other instructions.	5-92
MO	Memory Output	Loads data word into a memory card. It then cycles the card to allow the next data word to be loaded into the next memory location.	6-71
INPUT			
IP	Input Parallel	Simultaneously programs a group of input cards to take readings; waits for card to complete; and then reads each card and stores the data in 6942A memory. If desired a repeat factor can be used to take multiple readings from the input cards. Also, a wait time can be specified between readings.	5-106
IE	Input External	Same as IP except the IE waits for an external trigger from user's process to initiate the take reading cycle. The repeat and wait factors can also be used with the IE instruction.	5-148

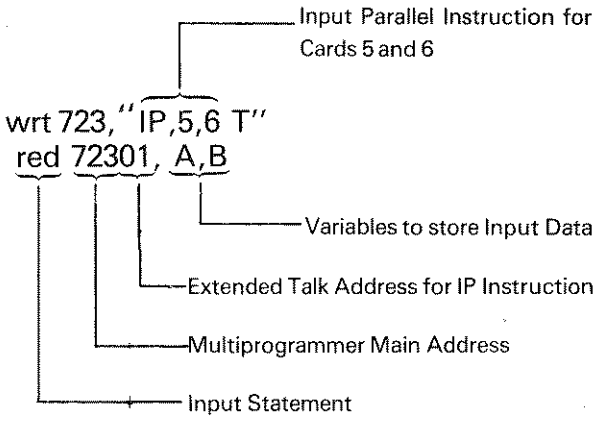
Table 5-1. Instruction Set (Continued)

Opcode	Name	Brief Description	Described in Paragraph
INPUT (continued)			
II	Input Interrupt	Same as IP, except it always runs in parallel with other instructions and does not allow repeat and waits. As each card completes, its data and addresses are stored in 6942A memory and SRQ is set.	5-201
MI	Memory Input	Specifies the address of the particular memory card to be read.	6-75
RV	Read Value	Reads the current data value on a card (or group of cards) and stores the data in 6942A memory. The RV does not initiate a take reading cycle. Consequently, the values read are the current value (values obtained from last take reading cycle).	5-163
CARD CONTROL			
AC	Arm Card	Selectively "arms" a card or group of cards. Armed cards can be used in conjunction with external triggering techniques.	6-21
DC	Disarm Card	Selectively "disarms" a card or group of cards. This instruction is used if the programmer decides to disarm a card he had previously armed with an AC instruction.	6-33
CY	Cycle Card	"Cycles" the card(s) without sending data or arming the cards(s). The term cycle means to "cause the card to perform its function".	5-98
CC	Clear Card	Clears the timing circuits on a card or a group of cards. For example, a CC can be used to clear the gate/flag circuit on a card that is waiting for a flag signal from an external device.	6-45
RS	Read Status	Obtains a data word that gives the data transfer status of the card and the identity of the instruction (if any) currently being used to program the card.	6-39
RF	Read Format	Reads card data format. Five variables (card ID, data type, LSB, size, limit) are returned.	5-223
SF	Set Format	Allows the card(s) data format to be changed. All or some of the parameters (data type, LSB, etc) can be changed.	5-236
SYSTEM TIMING			
SC	Set Clock	Sets the Multiprogrammer real time clock to predetermine values of days, hours, minutes, and seconds.	5-254
RC	Read Clock	Reads the internal real time clock values.	5-260
WA	Wait	Cause a pause between two instruction sequences. In parallel mode, it set SRQ when the specified time has elapsed and can be used as a timer.	5-267
WU	Wait Until	Causes a sequence of instructions to start at a programmable time. In the parallel mode, it sets SRQ when the programmed time is reached.	5-224
CW IN	Clear Wait Interrupt Now	Clears all WA or WU instructions from the system. Sets SRQ to indicate that all previously programmed instructions have completed.	5-282 5-215

or future readback. The read back statement (red) causes the data from the cards in slots 5 and 6 to be stored in variables A and B, respectively.

5-14 Notice that an extended talk address is added to the Multiprogrammer's main address in the input statement. HP-IB extended addresses are provided to allow identification of a particular source inside a bus device. The Multiprogrammer System utilizes this extended addressing scheme whenever reading back data or status to the controller.

Example 5-3. Typical Input Instruction

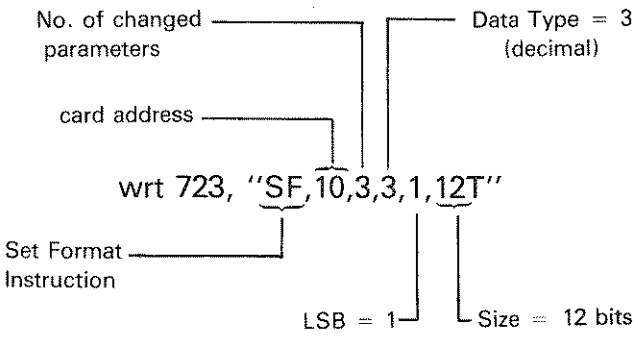


5-15 Card Control Instructions

5-16 The card control instructions allow access to various circuits on the I/O cards to enable close control and monitoring of their various operations. The following example shows how to change the data format of an I/O card using a Set Format (SF) instruction.

5-17 The Multiprogrammer allows the I/O cards to send and receive data in several different forms e.g; binary, 2's complement, and BCD. This enables you to program in a manner that is convenient for your particular application. In the example, a card in slot 10 is reformatted so that it can be programmed with a decimal number having a resolution of 1 (LSB = 1) and with a maximum bit size of 12.

Example 5-4. Sample Card Control Instruction

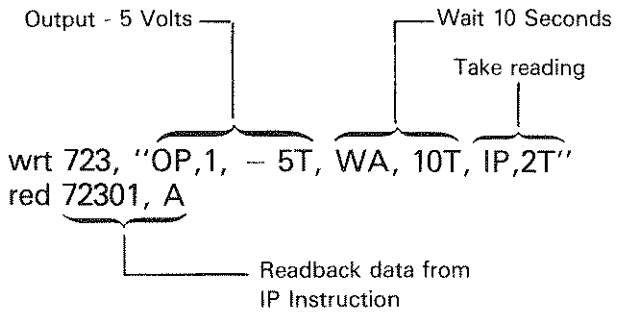


5-18 System Timing Instructions

5-19 These instructions permit sequencing events and measuring elapsed time. All but one of these instructions (the IN instruction) utilize the Multiprogrammer's real time clock, which has a resolution of 0.1 second and a maximum range of about 1.5×10^6 hours.

5-20 In the following example, a Wait (WA) instruction is used to establish a time delay between an analog stimulus and an associated response.

Example 5-5. Sample Timing Instruction



5-21 First, an OP instruction commands a D/A card in slot 1 to provide an analog stimulus (– 5 Volts). This is followed by a WA instruction that directs the Multiprogrammer to wait 10 seconds before taking the associated reading from an input card in slot 2. Finally, the input data is read back to the controller

5-22 High Level And Low Level Instructions

5-23 The instruction set contains both "high" and "low" level instructions to provide flexibility in programming the 6942A. The high level instructions (such as the "OP" and "IP" instructions described previously) simplify programming by delegating the tasks of instruction processing and sequencing to the Multiprogrammer. The "OP" instruction, for example, automatically initiates many operations within the Multiprogrammer. First, it "arms" the addressed output card so that it can generate an internal microprocessor interrupt after the card completes its operation. At the same time, it loads the data included in the instruction into the card's "first rank" of storage. Next, the card is "cycled" so that it produces an output that is proportional to the programmed data. After a gate/flag data transfer, the card's EOP (End of Process) signal will generate a microprocessor interrupt to indicate completion of the instruction. The microprocessor will then disarm the card and, if the Multiprogrammer is in the serial mode, allow the next instruction to be processed. If the Multiprogrammer is in the parallel mode of instruction processing, the HP-IB Service Request (SRQ) line will be set when the "OP" instruction completes. Note that all of these operations are accomplished automatically, and the user does not have to be concerned with the internal workings of the

Multiprogrammer when using a high level instruction. The OP,OS,OB,OI,IP,IE, and II instructions are all high level instructions.

5-24 In many instances, the high level instructions provide the most convenient and effective means of solving your application problems. However, for certain situations, the low level instructions may prove useful. These instructions allow you to access the Multiprogrammer I/O functions directly, without all of the control that the high level instructions provide. Moreover, low level instructions complete immediately and, therefore, provide faster access to the I/O functions.

5-25 All of the card control instructions of Table 5-1 are low level instructions as well as the "WF" and "WC" output instructions and the "RV" input instruction. Some of the capabilites provided by these instructions are, as follows:

a. Data can be transferred to and from I/O cards without cycling them.
b. The cards can be cycled without transferring data.
c. The microprocessors interrupt system can be used, or ignored, at the discretion of the User.

5-26 Proper use of the low level instructions requires a more detailed knowledge of the internal operations of the 6942A and its I/O cards. Therefore, be sure to read Chapter 4 (especially "I/O Card Operations") before attempting to use the low level instructions.

5-27 INSTRUCTION SYNTAX CONVENTIONS

5-28 Before describing specific instructions or processing modes, the syntax conventions that apply to the 32 Multiprogrammer instructions will be outlined. With a few minor variations, all of the instructions take one of these four basic forms:

(1) "XX"
(2) "XX,A1,D1,A2,D2,...T"
(3) "XX,A1,A2,...T"
(4) "XX,P1,P2...T"

where:
XX = Instruction opcode
A1, A2 = card addresses
D1, D2 = card data
P1, P2, = control parameters
T = Instruction terminator

5-29 Form (1) instructions consist of just an opcode. Examples of this type are the system control instructions ("GP", "GI", etc,). The output instructions ("OP", "OS", etc.), are examples of form (2) instructions. The input and card control instructions are formatted as form (3) types and most of the system timing instructions follow the form (4) format.

5-30 Opcode

5-31 All instructions begin with the opcode, a two-letter mnemonic that specifies which one of the 32 instructions is to be executed. If the opcode is entered incorrectly, the Multiprogrammer will set the SRQ line and report a programming error.

5-32 Card Address

5-33 Instructions that communicate with an I/O card (i.e., output, input, or card control instructions) must specify the address of the card. As shown in Figure 5-1, the card address specifies the frame (unit) number, the card slot number within that frame and a subaddress number.

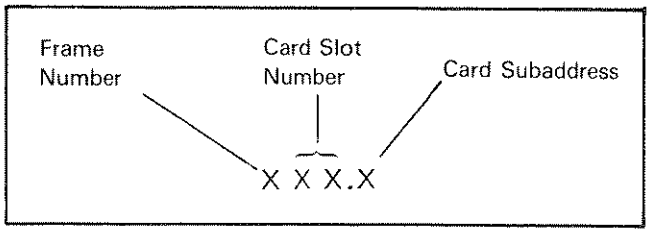


Figure 5-1. Card Address Structure

5-34 The frame number selects the unit number (from 0 to 7) in which the card is located. If no frame number is specifed, frame zero is automatically selected. As indicated in Chapter 2, a frame address switch selects the frame number assigned to each unit. Each 6942A is shipped with its frame address switch set to zero, and each 6943A with its switch set to one. Most of the instruction examples in this User's Guide are sent to the 6942A at an assumed frame address of 0. However, when addressing frames 1 through 7, the frame number must be multiplied by 100 to enable the Multiprogrammer to distinquish the frame number from the slot number. This is described in Paragraph 5-37.

5-35 The card slot number designates a specific slot number (from 00 to 15) within the addressed frame. A slot number must always be included in the instruction; even for slot 00.

5-36 As shown previously, in Table 4-2, the card subaddress selects a specific register (or circuit area) on the addressed I/O card. Write subaddresses (from 0 to 3) apply to output instructions while read subaddresses (0 to 3) are for input instructions. If no subaddress is specified, it defaults to the main write or read subaddress of 0. Subaddresses 1, 2, or 3 must be preceded by the decimal point shown in Figure 5-1.

5-37 In summary, the following examples shows how to send card addresses to frame 0 and frame 1:

Frame = 0
Slot = 7
Subaddress = 0
} Send 7

Frame = 1
Slot = 3
Subaddress = 1
} Send 103.1

5-38 The following list shows all the possible card addresses for a complete Multiprogrammer system that includes frames 0 through 7.

000 - 015 (or 0-15):	Specify I/O Cards 0-15 in Frame 0
100 - 115:	Specify I/O Cards 0-15 in Frame 1
200 - 215:	Specify I/O Cards 0-15 in Frame 2
300 - 315:	Specify I/O Cards 0-15 in Frame 3
400 - 415:	Specify I/O Cards 0-15 in Frame 4
500 - 515:	Specify I/O Cards 0-15 in Frame 5
600 - 615:	Specify I/O Cards 0-15 in Frame 6
700 - 715:	Specify I/O Cards 0-15 in Frame 7

5-39 Card Data

5-40 Card data is numerical information sent to a card by an output type instruction. The kind of data sent to the card depends upon its type and data format. Table 4-3 shows the wake-up data format of each I/O card. Note that all of the cards wake-up to accept a fixed point decimal number with the maximum ranges indicated on Table 4-3. The Multiprogrammer will accept but ignore more than three digits to the right of the decimal point. Just as with the card address, leading and trailing zeros are allowed and ignored.

5-41 The cards will also recognize whole octal numbers, provided that they are reformatted to the octal data format first, with a Set Format (SF) instruction.

5-42 Control Parameters

5-43 Control parameters specify control information for the Multiprogrammer system or a specific I/O card. Control parameters are neither card data nor card address information. The system timing instructions that are associated with the real time clock are one example of instructions that use control parameters.

5-44 Parameters are either whole or fixed point decimal numbers, depending upon the instruction type. As always, leading and trailing zeros are allowed but not required.

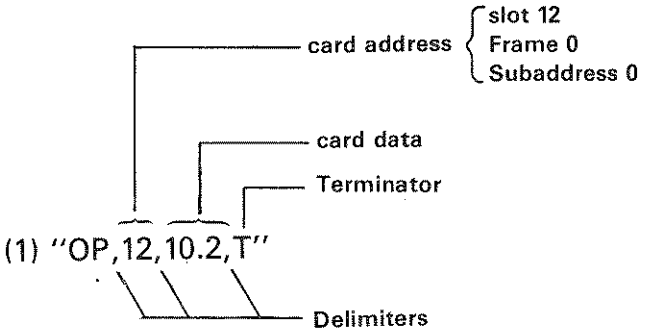
5-45 Instruction Terminator

5-46 The terminator (T) is used exclusively to indicate the end of an instruction. A "T" is always required at the end of any instruction that contains numerical data (card addresses, card data, and control parameters). If the "T" is omitted or incorrectly placed in an instruction, an error is reported. A "T" is allowed but not required after instructions that do not contain numerics; i.e., instructions that consist of an opcode only.

5-47 Delimiters

5-48 Delimiters are required to separate one numeric quantity from another within an instruction. The Multiprogrammer accepts either a comma or a space as a delimiter. Although they are only required between numeric quantities, delimiters can also be used between any parts of the instruction. Example 5-6 shows three valid ways to send the same instruction. Number one shows maximum use of delimiters with a comma between each segment of the instruction. Number two is exactly the same except that spaces are used rather than commas. The third example shows the minimum required usage of a delimiter; placed between the address and data values of the instruction. Note that if a delimiter was not included in this instruction, the Multiprogrammer would be unable to distinguish between the address and data values.

Example 5-6. Three Valid Uses of Delimiters



(2) "OP12 10.2T"

(3) "OP12,10.2T"

5-49 When spaces are used as delimiters, the Multiprogrammer will allow more than one space between the various parts of an instruction. With commas, however, an erstruction. Commas and spaces can be used together as delimiters.

5-50 INSTRUCTION PROCESSING MODES

5-51 The Multiprogrammer provides three different instruction processing modes which are designed to give the user maximum flexibility when writing programs for his particular application. The three modes are the Serial, Parallel, and Immediate modes.

Serial Mode - The serial mode is provided for user applications that require a series of dependent sequential operations. In the serial mode, most instructions are executed sequentially (one-at-a-time) and the next instruction cannot start until the previous one has completed. The majority of user applications will

probably require the serial mode of instruction processing. For this reason, the Multiprogrammer "wakes-up" in the serial mode when it is first switched on or when it is reset (clr 723/RESET 723).

Parallel Mode - The parallel mode is provided to simultaneously program two or more independent functions. In the parallel mode, most instructions run concurrently (at-the-same-time). Another significant feature of the parallel mode is that the "high level" instructions (IE, IP, OB, OS, OP) will each set service request upon completion. Thus, in the parallel mode, the user is notified when an operation is completed allowing him to do his own instruction sequencing. In the parallel mode, instructions run about 5% faster than instructions in the serial mode.

Immediate Mode - The immediate mode allows an instant response to an emergency condition. When the immediate mode is invoked, it suspends all currently active instructions and allows a selected subset of instructions to run instantly and resolve the emergency condition. After the condition has been resolved, the user can return to the normal mode (serial or parallel) and continue where the program was suspended.

5-52 The Multiprogrammer decodes instructions in the same sequence that they are sent by the controller. The instructions are processed by the firmware according to the instruction type and the processing mode that is in effect at that time. Table 5-2 summarizes how each of the Multiprogrammer's 32 instructions is affected by the serial, parallel, and immediate modes. As indicated in the table, certain instruction types are not affected by the processing modes and are executed instantly as soon as they are decoded. The following paragraphs provide detailed descriptions on how instructions are processed in each mode. The mode control instructions are also described.

Table 5-2. Instruction/Processing Mode Summary

INSTRUCTIONS	PROCESSING MODES		
	SERIAL (Paragraph 5-53)	PARALLEL (Paragraph 5-55)	IMMEDIATE (Paragraph 5-59)
High Level Instructions (Instructions that complete when cards complete) Input External (IE) Input Parallel (IP) Output Bit (OB) Output Sequential (OS) Output Parallel (OP)	Run sequentially	Run concurrently. Each instruction generates service request when it completes	Not permitted
Low Level Instructions (Instructions complete immediately do not wait for cards to complete) Arm Card (AC) Cycle Card (CY) Disarm Card (DC) Read Value (RV) Write and Cycle (WC) Write First Rank (WF)	Run sequentially	Run instantly	Run instantly
Card Control Instructions Clear Card (CC) Read Format (RF) Read Status (RS) Set Format (SF)	Run instantly	Run instantly	Run instantly

Table 5-2. Instruction/Processing Mode Summary (continued)

INSTRUCTIONS	PROCESSING MODES SERIAL	PARALLEL	IMMEDIATE
System Control			
Clear Group (CG)	Runs instantly	Runs instantly	Not permitted
Go Immediate (GI)	Runs instantly	Runs instantly	Not applicable
Go Normal (GN)	Not applicable	Not Applicable	Runs instantly
Go Parallel (GP)	Run sequentially	Run sequentially	Not permitted
Go Serial (GS)			
System Disable (SD)	Run sequentially	Run instantly	Run instantly
System Enable (SE)			
Interrupt Instructions			
Input Interrupt (II)	Start in sequence, but once started run concurrently with a sequential string of instructions Service request is generated when any card in an interrupt instruction completes.	Run concurrently. Service request is generated when any card in the interrupt instruction completes.	Run instantly
Output Interrupt (OI)			
Interrupt Now (IN)	Runs sequentially and generates service request when executed.	Separates two groups of parallel instructions by not allowing second group to start until first group completes. Generates service request when encountered.	Not permitted
Memory Instructions			
Memory Input (MI)	Run instantly	Run instantly	Not permitted
Memory Output (MO)			
System Timing			
Clear Wait (CW)	Runs instantly	Runs instantly	Runs instantly
Read Clock (RC)	Runs sequentially	Runs concurrently	Runs instantly
Set Clock (SC)	Runs sequentially	Runs concurrently	Not permitted
Wait (WA)	Run sequentially	Run concurrently. Generate service request when instruction completes	Not permitted
Wait Until (WU)			

5-53 Serial Mode

5-54 Figure 5-2 illustrates the execution sequence of five instructions in the serial mode. The instructions are executed in the sequence that they were sent out by the controller. In the example, I_1 was the first instruction sent; I_2 was the second; etc. Also, I_2 will not be executed until I_1 completes; I_3 will not be executed until I_2 completes; etc.

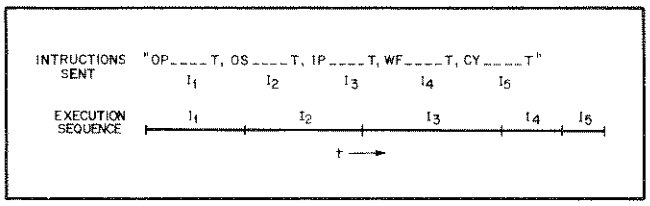


Figure 5-2. Instruction Execution in Serial Mode

Instructions that run Differently in Serial Mode:

1. The II and OI interrupt instructions are started in sequence but once they start, they run in parallel with the remaining instructions in the sequence. Figure 5-3 illustrates the execution of six instructions in the serial mode where the third instruction (I_3) in the sequence is an OI. Notice that I_3 is started sequentially; however as soon as it starts, it allows instruction I_4 to start even through I_3 has not completed. Note also that when any card in the OI instruction (I_3) completes, service request is generated. The OI and II instructions are described in detail in paragraphs 5-187 through 5-220.

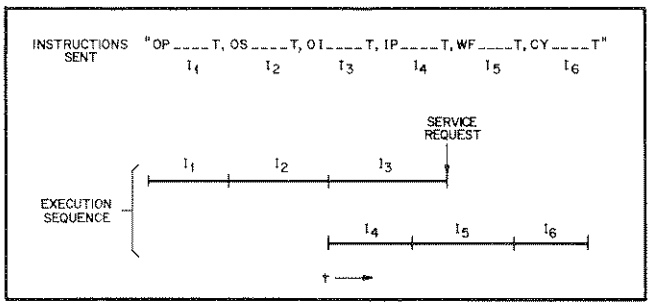


Figure 5-3. Interrupt Instruction in Serial Mode

2. The Interrupt Now (IN) instruction runs sequentially in the serial mode and sets service request when it completes. Since most instructions do not set service request in the serial mode, IN can be used as a completion indicator. Figure 5-4 illustrates an IN programmed as the fourth instruction in a sequence. When the IN instruction is encountered, it will set service request indicating that the previous instructions have completed. Use of the IN instruction is described in greater detail in paragraphs 5-187 through 5-200.

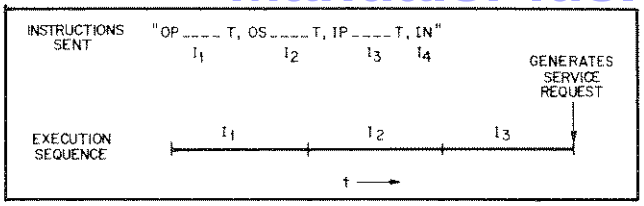


Figure 5-4. Interrupt Now Instruction in Serial Mode

3. Certain instructions run instantly and are not affected by the mode of operation (see Table 5-2). Figure 5-5 illustrates an SF (which runs instantly) programmed as the third instruction in a sequence sent from the controller. The diagram illustrates that each instruction is decoded in sequence. I_1, I_2, and I_4 then are executed sequentially with I_2 starting when I_1 completes with I_4 starting when I_2 completes. Note however, that I_3 is unaffected by serial mode and is executed while it is decoded independently of the other instructions.

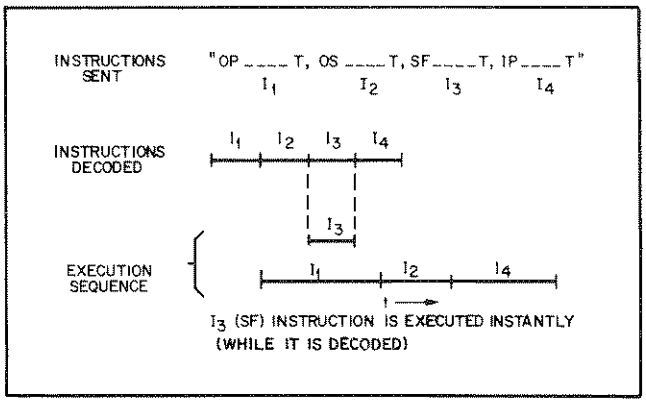


Figure 5-5. Instructions that are Executed Instantly

5-55 Parallel Mode

5-56 Figure 5-6 illustrates the execution of four instructions in the parallel mode. Note that when the "high level" instructions (I_1, I_3, I_4) complete, they generate service requests.

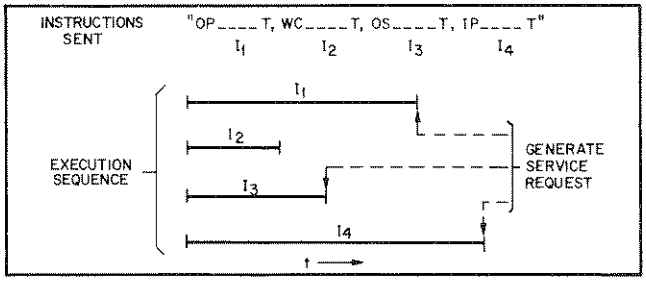


Figure 5-6. Instruction Execution in Parallel Mode

Exceptions in Parallel Mode:

1. Only one instruction of each type (except OI and II) can be running at any one time. Thus, instructions of the same type run in sequence. Figure 5-7 illustrates how two OP's and two OI's would run in the parallel mode.

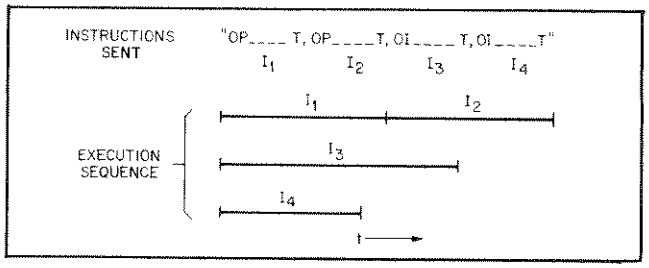


Figure 5-7. Like Instructions in Parallel Mode

2. Only one instruction can be active for any one card at the same time. Multiple instructions (like or unlike) that address the same card will run in sequence. The following instructions are affected by this restriction; IE, II, IP, OB, OI, OP, OS, and WC. Note that, even though two or more OI (or II) instructions can be executed in parallel, the same card restriction applies since one card cannot be busy in two or more OI (or II) instructions at the same time. Figure 5-8 illustrates how four instructions affected by this restriction (I_1 I_2, I_4, and I_6) and two that are not (I_3 and I_5) are executed in the parallel mode. In the example six instructions are addressing card 7.

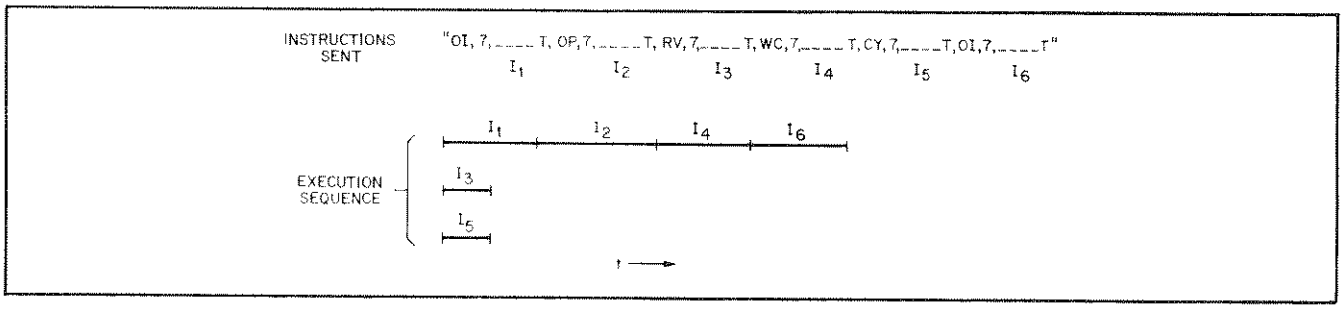


Figure 5-8. Same Card Restrictions in Parallel Mode

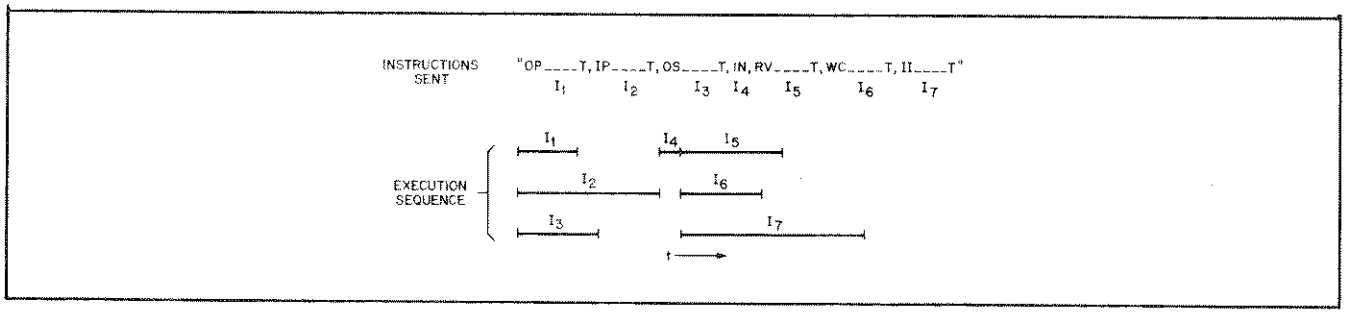


Figure 5-9. Interrupt Now Instruction in Parallel Mode

Instructions that run differently in Parallel Model:

1. The interrupt Now (IN) instruction separates two groups of parallel instructions by not allowing the second group to start until all instructions (except OI and II) in the first group have completed. Figure 5-9 illustrates an IN instruction (I_4) separating two parallel groups (I_1, I_2, I_3 and I_5, I_6, I_7). When the last instruction in the first group completes (I_2 in this example), I_4 executes and allows the next group I_5, I_6, I_7) to run concurrently.
2. In the serial mode, only the interrupt instructions, (II, IN, and OI) generate service requests. In the parallel mode, the interrupt instructions, the "high level" instructions (IE, IP, OB, OS, and OP), and the wait instructions (WA, WU) all generate service requests when they complete.

a. The high level instructions may be used to program I/O cards that require a flag from an external device to complete the operations. In the parallel mode a high level instruction will generate a service request when all cards addressed in the instruction have completed. The controller can monitor service request and determine which instruction(s) have completed. This feature allows the user to sequence high level instructions in the parallel mode.

b. The wait instructions (WA, WU) do not hold up execution of later instructions in the parallel mode like they do in serial mode. In parallel mode, WA is used as a timer generating a service request when the specified time has passed; and WU is used as an alarm clock generating a service request at the specified time.

5-57 Serial-Parallel Mode Control Instructions

5-58 The Go Serial (GS) and Go Parallel (GP) mode instructions place the system in the specified mode for all subsequent instructions. Since the Multiprogrammer "wakes-up" in the serial mode, the GS instruction is only used to restore the Multiprogrammer to the serial mode after a GP instruction has been executed and the user wants to return to the serial mode. In any case, GS and GP instructions will not run until all previous instruction sent by the controller have completed. The only exception to this rule is that a GP given in the parallel mode (with no previous GS) is ignored. Figure 5-10 illustrates how these instructions can be used to change from one processing mode to another.

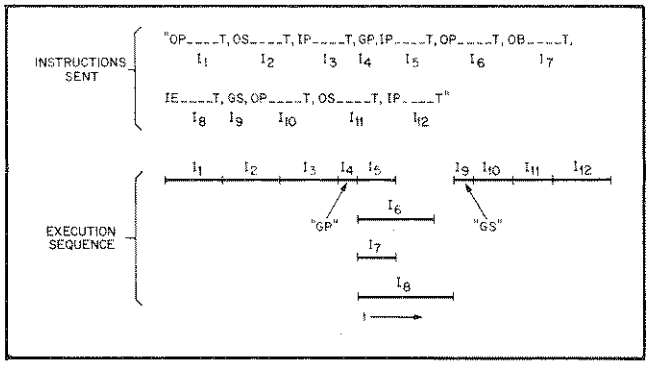


Figure 5-10. GP and GS Mode Control Instructions

Syntax: Since only the opcode is sent the terminator "T" is not required but will be accepted if it is sent.

"GS" or "GST"
"GP" or "GPT"

Examples:

9825: wrt 723, "GS"

9835/9845: OUTPUT 723; "GP"

5-59 Immediate Mode

5-60 Figure 5-11 illustrates using the immediate mode to respond to an emergency condition. The instruction sequence starts off in the serial mode with the main line program instructions (I_2 through I_8) controlling a series of dependent sequential operations. Instruction I_1 is an interrupt instruction running in parallel with the main program and is used to monitor an alarm condition. If the alarm is triggered it will be detected by the interrupt instruction which will generate a service request causing the program to jump to I_{12} (the GI instruction). The GI instruction will suspend the currently active instruction (I_5 in this example) and allow I_{12} to I_{16} to run immediately. These instruction (I_{12}-I_{16}) will resolve the alarm condition and will be executed as soon as they are decoded and in the same sequence that they were sent by the controller. The GN instruction (I_{16}) will return the Multiprogrammer back to the mode that existed prior to executing the GI. In our example the system returns to the serial mode. The program will now continue where it left off completing I_5 and sequentially executing I_6 through I_{11}. Note that when I_{11} (IN instruction is executed, a service request is generated to indicate that all instructions have completed. As noted in Table 5-2 all instructions are not permitted in the immediate mode. Also, since the CC, CW, RF, RS, and SF instructions always execute immediately regardless of mode, it is not necessary to go to the immediate mode when using these instructions. Refer to Chapter 6 for a more detailed description of operations and restrictions in the immediate mode.

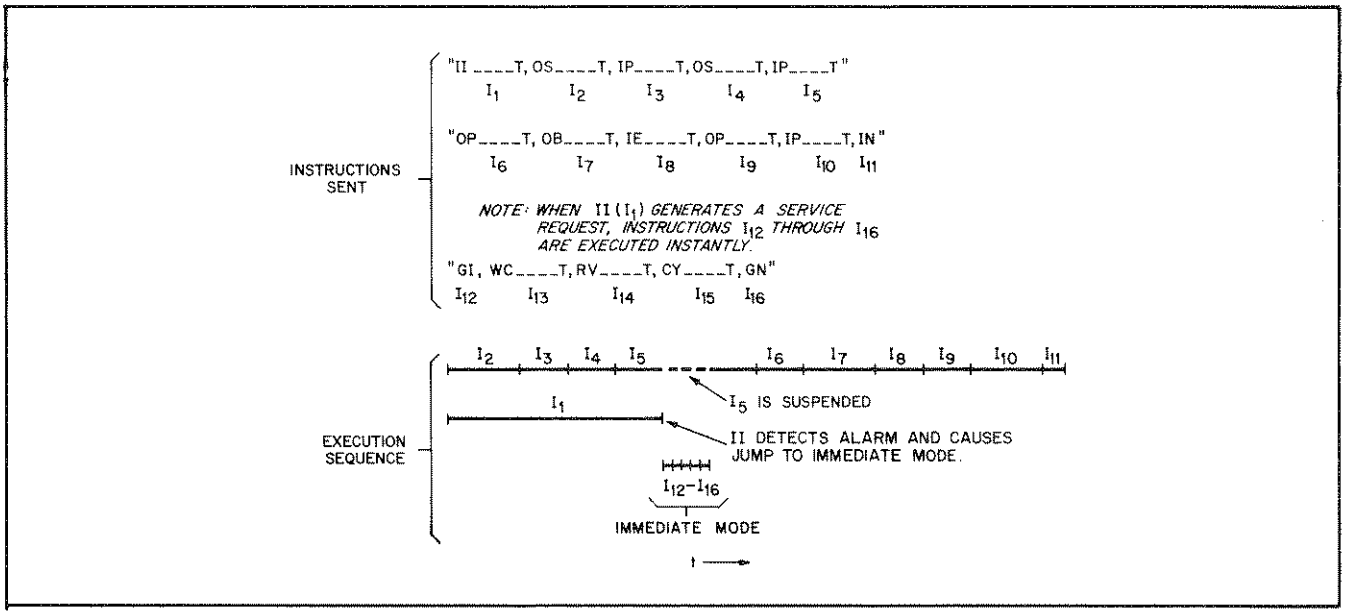


Figure 5-11. Use of Immediate of Mode

5-61 BASIC OUTPUT INSTRUCTIONS

5-62 The following paragraphs describe three of the most commonly used high level output instructions.

5-63 Output Parallel (OP) Instruction

5-64 The OP instruction, as stated previously, is used to simultaneously program a group of output cards. The syntax of the OP instruction is as follows:

"OP,A1,D1,A2,D2,...T"

The card addresses and associated data are specified by each address/data pair (A1/D1, A2/D2,etc,). Of course, an OP instruction can also be used to program one output card.

5-65 As shown in Figure 5-12, the OP instruction first sends the data simultaneously to the specified subaddress (usually subaddress 0, first rank storage) on all cards. It then cycles the addressed cards and the instruction does not complete until the last card has signaled completion with its end-of-process (EOP) signal.

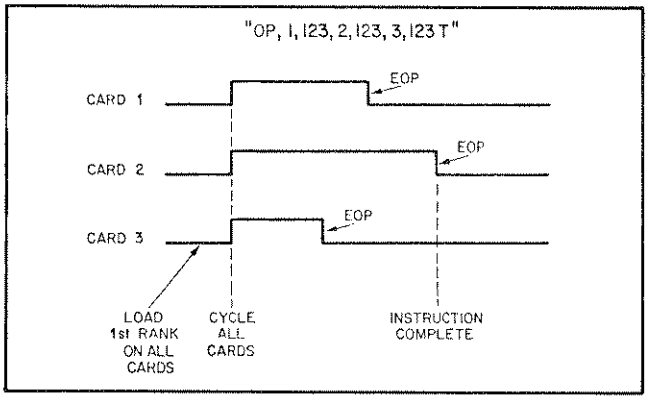


Figure 5-12. Typical OP Instruction

5-66 Processing Modes. After an OP instruction completes, the next instruction can begin if the Multiprogrammer is in the serial mode of instruction processing. In the parallel mode, service request is set when an OP instruction completes. The OP is not permitted in the immediate mode. (Paragraphs 5-50 through 5-60 outline the effects of the Multiprogrammer modes on each type of instruction.)

5-67 Restrictions and Error Checking. Because all cards are cycled in parallel in an OP instruction, a single card can only be addressed once in a specific instruction. This restriction also applies to subaddresses. Two different subaddresses on the same card cannot be programmed in a single OP instruction. If either of these restrictions is violated, the Multiprogrammer will set SRQ and report a programming error.

5-68 Besides checking for the two restrictions described above, the Multiprogrammer performs other error checks on each OP instruction before it is executed. The data is checked to make sure it is in the correct format and within the range allowed by the card. If a programmable limit was specified, the data is also checked to ensure that it doesn't exceed the limit. Additional checks ensure that there are no illegal characters, that a legal card address was specified, and that a terminator (T) was given. These "standard error checks" are performed on all of the instructions that are received by the 6942A.

5-69 Sample OP Instruction. Example 5-7 uses two OP instructions to program two different data values to a D/A Converter card in slot 1 and one data value to a Digital Output card in slot 2. The first line programs the D/A card to a 1.51 Volt output by sending a "constant" data value. The second OP instruction sends "variable" data to both cards.

Example 5-7. Using Two OP Instructions

9825 Controller

```
0: wrt 723,"OP,1,1.51T"
1: -3.9→A;163→B
2: wrt 723,"OP",1,A,2,B,"T"
```

9835/45 Controller

```
10   OUTPUT 723;"OP,1,1.51T"
11   A=-3.9
12   B=163
20   OUTPUT 723;"OP",1,A,2,B,"T"
```

5-70 Output Sequential (OS) Instruction

5-71 The OS instruction is useful when it is necessary to program a group of cards, or one card, in a sequential manner. Although the same operations could be executed with a series of OP instructions, using a single OS instruction is more efficient. This is because a single instruction is processed faster and uses much less buffer memory. When programming only one card, the OS and OP instructions are equally efficient.

5-72 The syntax of the OS instruction is the same as that of the OP with address/data pairs following the opcode.

"OS,A1,D1,A2,D2,....T"

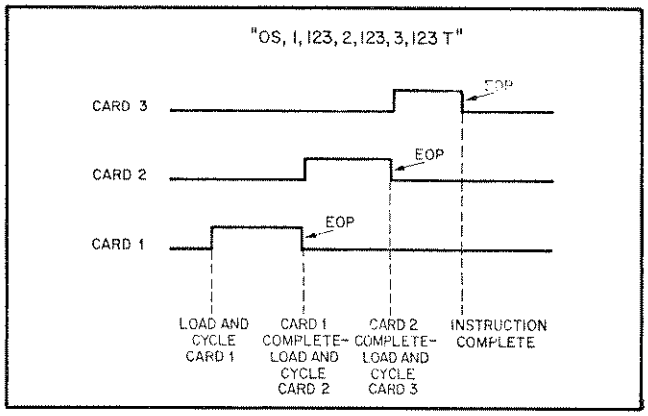


Figure 5-13. Typical OS Instruction

5-73 As indicated in Figure 5-13, each card is programmed in sequence in accordance with its position in the instruction. Card 1 receives its data first, which is loaded into first rank storage, and then the card is cycled. When card 1 completes its data transfer and generates an EOP signal, the second card addressed in the instruction is programmed. The OS instruction completes when the last card listed has completed.

5-74 **Error Checking and Processing Modes.** The OS instruction undergoes the "standard error checks" described for the OP instruction except that a single card can be referenced as many times as desired in one OS instruction.

5-75 The OS instruction is affected in the same manner as the OP instruction by the 6942A processing modes. It runs in a serial fashion with the other instructions in the serial mode and sets SRQ when it completes in the parallel mode (refer to Paragraphs 5-50 through 5-60).

5-76 **Sample OS Instructions.** Example 5-8 shows how to use an OS instruction to ramp the output voltage of a D/A Converter card in five sequential steps. Example 5-9 uses an OS instruction to ensure that an analog output voltage (−3.64 Volts from a D/A Card) is stable before programming a Digital Output Card. The D/A card is assumed to be located in slot 1, frame 0, and the Digital Output Card in slot 2, frame 0.

Example 5-8. Using an OS instruction to obtain a Ramp

9825 Controller

```
0: wrt 723,"OS,1,1.0,1,1.5,1,2.0,1,2.5,1,3.0,T"
```

9835/45 Controllers

```
10 OUTPUT 723;"OS,1,1.0,1,1.5,1,2.0,1,2.5,1,3.0,T"
```

Example 5-9. Using an OS Instruction to Sequence Two Events

9825 Controller

```
0: wrt 723,"OS,1,-3.64,2,194T"
```

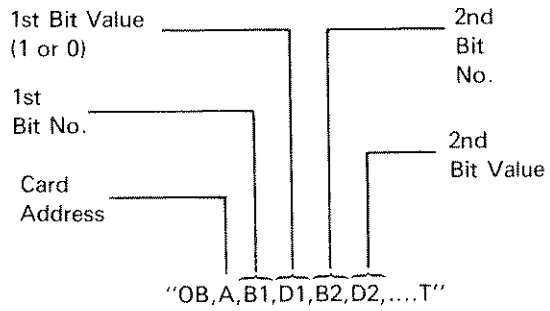
9835/45 Controllers

```
10  OUTPUT 723;"OS,1,-3.64,2,194T"
```

5-77 Output Bit (OB) Instruction

5-78 In some applications, a single card (such as a Relay Output Card) is used to control several independent output channels. In these instances, it is often convenient to change the state of some of these channels (bits) without affecting any of the others. This is the function of the OB instruction; it allows the user to simultaneously set or clear individual bits to "1" or "0", without affecting the other output bits on the card.

5-79 The syntax of the OB instruction is shown below.



5-80 **Special Conditions.** Several special conditions and restrictions apply when using OB instructions. One is that only one card address per instruction can be specified. Another is that only subaddress 0 (the main address) can be specified. Subaddresses 1, 2, and 3 are illegal and will result in an error. All bit values must be either a "1" or a "0". Because all bits are updated simultaneously, no bit can be specified more than once in the same instruction. The OB instruction should not be used to program the Memory or Input type cards (Models 69790A, 69751A, and 69771A).

5-81 The user need not be concerned with the data format of the card. Using up to 16 bits, it is programmed exactly as indicated previously, regardless of the data type.

5-82 Figure 5-14 shows a typical OB instruction that changes bits 3, 5, and 6 on a card in slot 1, to "0", "1", and "1", respectively. When the Multiprogrammer receives the instruction, it reads the last data stored on the card (from read subaddress 3), merges in the three new bit values, and then loads a new data word into the card's first rank of storage. It then cycles the card and waits for it to complete.

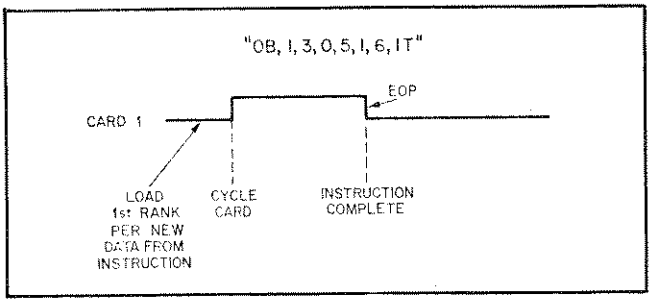


Figure 5-14. Typical OB Instruction

5-83 In addition to the restrictions described in Paragraph 5-80, the Multiprogrammer provides standard error checking (described with the OP instruction) of each OB instruction.

5-84 **Sample OB Instruction.** In example 5-10, an OB instruction is sent to a Digital Output card in slot 1 that was originally programmed to all 0's. The new output bit values, after the OB is executed, are shown below the instruction.

Example 5-10. Changing Output Bits

9825 Controller

```
0: wrt 723,"OB,1,0,1,5,1,7,1T"
```

Card 1 = 0 000 000 010 100 001 = 161_{10}

9834/45 Controller

```
10   OUTPUT 723;"OB,1,0,1,5,1,7,1T"
```

Card 1 = 0 000 000 010 100 001 = 161_{10}

5-85 Low Level Output Instructions

5-86 The following paragraphs describe three of the most often used low level output instructions. Unlike the three high level output instructions just described, these instructions are executed faster because they do not wait for the cards to complete their programmed operation.

5-87 **Write and Cycle (WC) Instruction.** The WC instruction is the same as the OP instruction (Paragraph 5-63) except that it does not wait for the card(s) to complete. The syntax of the WC instruction is identical to that of the OP.

5-88 Figure 5-15 illustrates how the WC is executed. The data is written to the specified subaddress (usually the first rank of storage) on all cards addressed by the instruction. All cards are then cycled in parallel and the instruction completes immediately. Assuming that the Multiprogrammer is in the serial mode, the next instruction after the WC will begin processing even though all the card may not have completed. The WC is very useful when programming an Output card that requires a long completion time and the application can not wait for it. One example of this is using a Pulse Train card (69735A) to program a stepper motor. Under certain conditions, the card could take an hour to complete turning the stepper motor. A WC could be used to start the operation, and then subsequent instructions could begin to implement other operations.

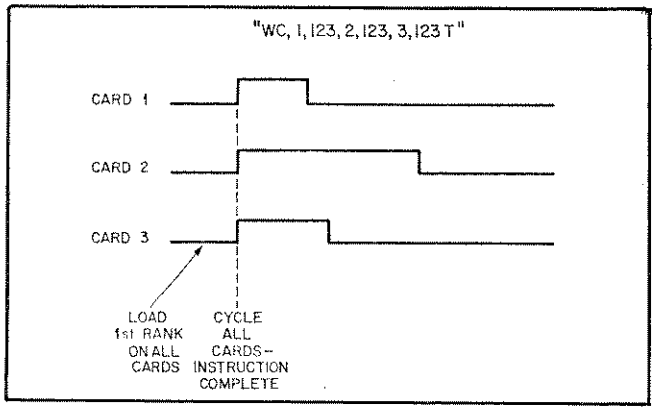


Figure 5-15. Typical WC Instruction

5-89 Because of the parallel cycling of the addressed cards, a single card can be specified only once in each WC instruction. If this restriction is violated a programming error will result. The WC instruction is subjected to the standard error checks described previously.

5-90 The effects of the serial, parallel, and immediate modes on the WC instruction are described in paragraphs 5-50 through 5-60. This instruction, when run by itself, will not set SRQ in either the serial or parallel modes.

5-91 Example 5-11 uses two WC instructions to send two different data values to a D/A card in slot 1 and one data value to a Digital Output card in slot 2. A "constant" (1.57) is sent first, followed by two "variable" values.

Example 5-11. Using Two WC Instructions

9825 Controller

```
0: wrt 723,"WC,1,1.57T"
1: -7.12→A:123→B
2: wrt 723,"WC",1,A,2,B,"T"
```

9835/45 Controllers

```
10   OUTPUT 723;"WC,1,1.57T"
11   A=-7.12
12   B=123
20   OUTPUT 723;"WC",1,A,2,B,"T"
```

5-92 **Write First Rank (WF) Instruction.** This instruction sends data to a card, or group of cards, without cycling the card(s). The subaddress specified in the WF instruction will often be main subaddress 0(first rank storage). However, on the more complex output cards, other write subaddress (1, 2, or 3) can be specified to establish control conditions.

5-93 Like the other low level instructions, the WF completes immediately; as soon as the data is loaded into the cards subaddress. After this, no other action is performed by the card, unless it receives another instruction (such as a Cycle instruction).

5-94 The syntax of the WF is as follows:

"WF,A1,D1,A2,D2...T"

Like the WC and OP instructions, the card addresses and associated data for a WF instruction are listed in pairs (A1/D1, A2/D2, etc,).

5-95 The WF instruction is checked for the standard errors described previously. Note that the same card can be specified more than once in a single WF instruction, because the card(s) are not cycled by the instruction.

5-96 Sample WF Instruction. The following example (5-12 shows how to send a WF instruction to two cards in slots 1 and 2, unit 0. Note that main subaddress 0 (first rank storage) is specified for both cards.

Example 5-12. Sending a CY Instruction

"WF,1,123,2,564T"

5-97 As stated previously, the WF can be sent to the control subaddresses of the more complex cards to set up control conditions. Example 5-13 programs a 69735A Pulse Train output card (assumed to be in slot 6) to generate 100 pulses with periods of 10 milliseconds each. The subaddress list for the Pulse Train card is shown below. In the first line of Example 5-13, a WF instruction is used to set up a pulse width of 10ms using subaddresses 1 and 2. In the second line, an OP instruction is sent to subaddress 0 to obtain 100 output pulses from the card.

Pulse Train Write Subaddresses

Subaddress	Function
0	No. of Output Pulses
1	Period Multiplier (1,10, or 100)
2	Period Magnitude (in μsec)
3	Not Used

Example 5-13. Sending a WC to Control Subaddresses.

"WF,6.1,1,6.2,10000T" (Set up pulse width)
"OP,6,100T" (Generate 100 pulses)

5-98 Cycle (CY) Instruction. The only function of the CY instruction is to cycle a card or group of cards. The output of a card, after it has been cycled, is proportional to the data that was stored on the card prior to the execution of the CY instruction. If this instruction is sent to an input card, it causes the card to take a reading.

5-99 The syntax consists of the opcode followed by a card address list.

"CY,A1,A2,A3,...T"

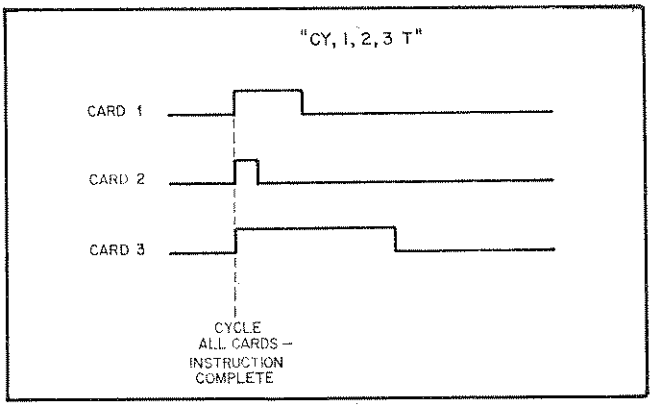


Figure 5-16. Typical CY Instruction

5-100 As shown in Figure 5-16, the CY instruction cycles all addressed cards in parallel, and then completes immediately, without waiting for the cards to complete.

5-101 The CY is subjected to standard error checking. It allows cards to be specified more than once in the address list, however since all the cards are cycled in parallel, additional references to the same card are ignored.

5-102 The effects of the operating modes on the CY instruction are described in Paragraphs 5-50 through 5-60. When run by itself, this instruction will not set SRQ in either the serial or parallel modes.

5-103 Sample CY Instruction. The CY instruction allows the user to cycle a card using whatever data is currently in its first rank of storage. The application just described for the WF instruction sets up a Pulse Train card (in slot 6) to generate 100, 10ms-wide, pulses. After Example 5-13 has been executed, a CY instruction could be sent to the Pulse Train card to regenerate these 100 output pulses. Example 5-14 shows how this instruction would be sent.

Example 5-14. Sending a CY Instruction

"CY,6T"

5-104 BASIC INPUT INSTRUCTIONS

5-105 This section describes three of the most often used input instructions: consisting of the high level IP and IE instructions and the low level RV instruction.

5-106 Input Parallel (IP) Instruction

5-107 The IP is, perhaps, the most commonly used input instruction. As indicated previously, the IP instruction simultaneously initiates a "take reading" cycle on a group of input cards. The syntax of the IP consists of the opcode followed by a card address list. Of course, the IP instruction can also be used to program one input card.

"IP,A1,A2,A3,...T"

5-108 Figure 5-17 illustrates a typical IP instruction sent to three input cards. The instruction "cycles" the cards in parallel and when all cards have taken readings (completed) it stores the data from each card in 6942A system memory locations.

5-109 Normally, the IP instruction specifies read subaddress 0 to obtain the data stored in the input latch on the card. However, on the more complex cards (e.g., the 69776A Interrupt Card) control data can be read from the other read subaddresses (1,2, or 3).

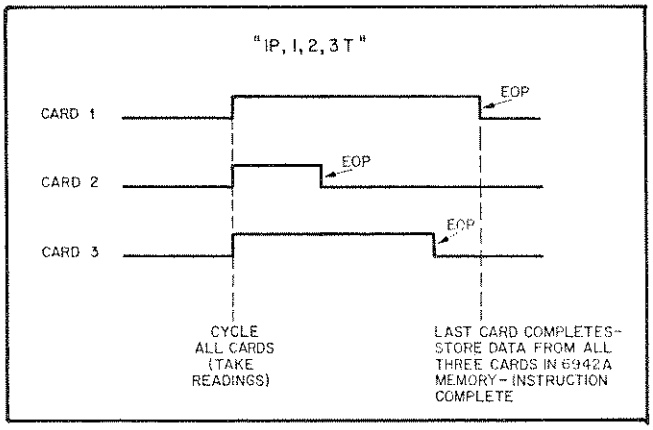


Figure 5-17. Typical IP Instruction

5-110 Processing Modes. After the IP instruction completes, the next instruction can begin if the Multiprogrammer is in the serial mode of instruction processing. In the parallel mode, service request is set when an IP instruction completes. The IP is not permitted in the immediate mode. (Paragraphs 5-50 through 5-60 outline the effects of the Multiprogrammer modes on each type of instruction.)

5-111 Restrictions and Error Checking. Because all cards are cycled in parallel in an IP instruction, a single card can be addressed only once in a specific instruction. This restriction also supplies to subaddresses. Two different subaddresses on the same card cannot be specified in a single IP instruction. If either of these restrictions is violated, the Multiprogrammer will set SRQ and report a programming error. Besides these two restrictions, each IP instruction is examined for the standard errors described for the output instructions.

5-112 Reading Back Data From an IP Instruction. As described previously, once the data from an IP instruction has been stored in Multiprogrammer memory, it can be read back to the controller immediately or at a later time.

5-113 Immediate Readback. Data can be read back to the controller as soon as the IP instruction has completed by immediately following the IP with an input statement ("red" or ENTER, see Example 5-15). As described previously, extended talk address 01 specifies the IP instructions as the source of the requested data. Note that the card data is always sent back in the order that the cards were specified in the instruction and not in the order that the cards completed. In Example 5-15, variable A contains the data from the first card specified in the address list (card 1). Variable B contains the data from card 2, and Variable C contains the data from Card 3.

Example 5-15 Reading Back Input Data

9825 Controller

```
0: wrt 723,"IP,1,2,3T"
1: red 72301,A,B,C
```

9835/45 Controllers

```
10   OUTPUT 723;"IP,1,2,3T"
20   ENTER 723.01;A,B,C
```

5-114 Delayed Readback. In time-critical applications, it is often desirable to store the data from one or more IP instructions in Multiprogrammer memory and then read the data back after the process has completed. Before readback, data can be stored in 6942A memory for as long as necessary and does not ordinarily interfere with the execution of successive instructions. However, if you will be using very large IP instructions that contain many readings, you must keep in mind that each data reading will require a certain amount of memory space and that the size of the 6942A memory is not infinite. Paragraph 5-126 discusses memory and its proper utilization in detail.

5-115 Multiple IP Readbacks. The Multiprogrammer requires that a separate input statement be used for each IP instruction that was issued. Only the data from one instruction is returned with each red/ENTER statement. Data from input instructions is always read back in the order that the instructions were programmed. Data from the first instruction issued is received first, data from the second instruction is received next, etc,.

5-116 Example 5-16 indicates the correct way to read back data from two IP instructions. The first red/ENTER command will store the data from cards 1, 2, and 3 in variables A, B, and C, respectively. The second command stores the data from cards 4, 5, 6 in D, E, and F, respectively.

Example 5-16 Using Separate Input Statements

9825 Controller

```
0: wrt 723,"IP,1,2,3T,IP,4,5,6T"
1: red 72301,A,B,C
2: red 72301,D,E,F
```

9835/45 Controllers

```
10   OUTPUT 723;"IP,1,2,3T,IP,4,5,6,7T"
20   ENTER 723.01;A,B,C
30   ENTER 723.01;D,E,F
```

5-117 Observe caution if your program is halted before all of the data from multiple IP's has been read back. If the program is re-started and any IP instructions are re-executed, your readbacks will contain data obtained from the first IP(s) that were sent (before the program halted) and not the latest IP's. Thus, your readbacks will be out of syncronization with the instructions. To avoid this problem,you can clear memory by executing a clear message (clr 723/RESET 723) prior to restarting your program. An alternative way is to read the busy instruction status on extended talk address 13 to determine if there is active IP data still available in memory. If there is, you can read back this data on extended talk address 01. Keep checking the status and reading back the data until the IP instruction is no longer active.

5-118 Specifying Too Many Variables with 9835/9845 Controllers. In the preceding examples, the number of variables specified in each input statement correctly matched the number of readings taken by the associated IP instruction. Now consider the following example:

"IP,1,2,T"

9825	9835/45
red72301,A,B,C,D	ENTER 723.01;A,B,C,D

5-119 As indicated above, the number of variables requested exceeds the data readings available from the IP instruction. For 9825 Controllers, this does not present a problem. Variables A and B will contain the data from cards 1 and 2 and variables C and D will remain unchanged. 9835 and 9845 Controllers, however, will report an I/O error 159 and halt execution of the program whenever the data received is less than that requested. Paragraph 5-135 "Additional Readback Considerations", shows a sample way to avoid this problem. This Paragraph also describes the data throw-away that occurs, if you specify fewer variables than the data available from the Multiprogrammer.

5-120 Sixteen-Channel Scanning Application. For this application example, assume that we have an environmental control system that must monitor the temperatures in each 16 zones in a factory. The temperature valves are available to the 6942A as 16 analog voltages in the range of ±10V. For this applications, we can use a Relay Output card as an analog scanner, feeding the selected channel to an A/D input card. This application assumes the input signals are all single-ended. One side of all the realys are tied to the analog input pin of the A/D input card and the other side of each relay is tied to one of the analog channels. Remember, with a single ended scanner, all channels must have their commons tied together, and this tie point must be connected to the A/D card's isolated common (see Figure 7-3 in chapter 7). Assume that the relay card is installed in slot 0 and the A/D card in slot 1. Example 5-17 will take readings from 16 channels and store them in array, R. It is assumed array R is dimensioned R(16) in the main routine. Note that all the relays are opened ("OS",0,0) before selecting the next channel to avoid shorting adjacent channels.

5-121 The first method in Example 5-17 takes the data, one instruction at-a-time, as soon as the IP completes. The second method waits until all the IP's have completed and then reads the data all at once. With the Multiprogrammer in the serial mode, it is guaranteed that the data will be stable on each channel before an input is performed.

Example 5-17. Sixteen-Channel Scanning Application

9825 Controller

```
0: for I=0 to 15
1: wrt 723,"OS",0,0,0,2↑I,"T,IP,1T"
2: red 72301,R[I+1]
3: next I
```

OR

```
0: for I=0 to 15
1: wrt 723,"OS",0,0,0,2↑I,"T,IP,1T"
2: next I
3: for I=1 to 16
4: red 72301,R[I]
5: next I
```

9835/45 Controllers

```
10   FOR I=0 TO 15
20   OUTPUT 723;"OS",0,0,0,2↑I,"T,IP,1T"
30   ENTER 723.01;R(I+1)
40   NEXT I
```

OR

```
10   FOR I=0 TO 15
20   OUTPUT 723;"OS",0,0,0,2↑I,"T,IP,1T"
30   NEXT I
31   !
32   !
33   !
40   FOR I=1 TO 16
50   ENTER 723.01;R(I)
60   NEXT I
```

5-122 Using Repeat Factors with the IP Instruction. The repeat factor is very useful for applications that involve taking multiple readings from one or more input cards. As shown next, the repeat factor (Rn) is simply placed between

the opcode and the card addresses. Note that n is a positve integer that specifies the number of readings to be taken from each card addressed by the instruction.

"IP,Rn,A1,A2,...T"

5-123 Figure 5-18 shows an IP instruction with a repeat factor of 3. The first reading is initiated by cycling all three cards in parallel, just like the previously described IP instruction. When all cards have completed, the data from each card is stored in Multiprogrammer memory and the first reading is complete. The second and third readings are performed in an identical manner and the instruction completes after the data from the third reading has been stored in memory.

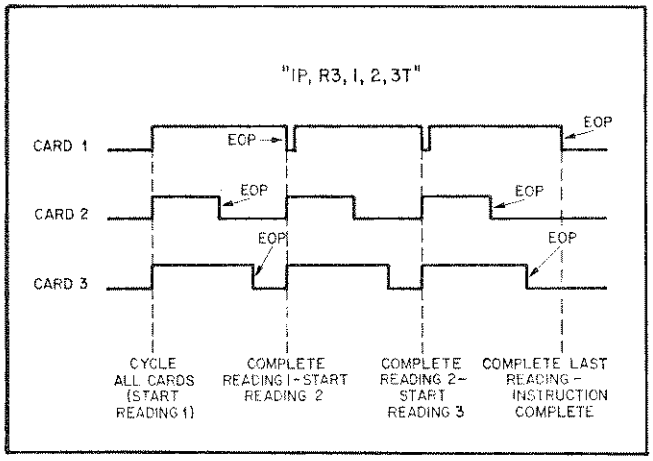


Figure 5-18. IP Instruction with Repeat Factor

5-124 Reading Data Back. IP's with repeat factors are read back in a manner identical to that of a normal IP instruction. The extended talk address is still 01 and the data can be read back immediately or at a later time using the previously shown input statements.

5-125 During readback, all of the data for the first addressed card (n readings) is received first, followed by all of the data for the second addressed card, and so on. Referring back to Figure 5-18, reading number 1 for card one is received first, reading number 2 for card one is received second, etc,. In most cases, a storage variable must be set aside within the controller for each data reading. If the number of readings is to be large, "continuation lines" can be used (9825 controllers only) or an array can be set up (refer to "Additional Readback Considerations" later in this section). With large readings you must also be careful to avoid overflowing memory (refer to next paragraph).

5-126 Memory Utilization, General Information. The number of instructions, of any type, that can be stored in the 6942A depends upon the amount of available memory. Every instruction in the system requires a certain amount of memory, which it keeps while it is active. An active instruction is defined as an instruction that is waiting to execute, executing, or one that has completed executing but still has data to be read back to the controller. An output instruction (such as an OP) remains active until the instruction completes, at which time it relinquishes its memory. An input instruction (such as an IP) remains active after it has completed until the controller has read back all of the data (or it has been thrown away), at which time it relinquishes its memory.

5-127 Note that if your application program attempts to store more active instructions then will fit into memory, your system will hang up. If this occurs, the only recourse is to send a device clear (clr 723/RESET 723) which clears all of memory and all the I/O cards. Because the Multiprogrammer does not monitor the amount of available memory, you should have an idea of how much memory space your instructions will require in order to avoid potential hang-up problems. Chapter 6 describes how to compute the exact amount of memory that each instruction uses and the following paragraph provides some general rules-of-thumb.

5-128 The 6942A memory can store up to about 80 minimum size instructions that specify a single card. As a general rule, if you keep the number of active instructions below 35, you will avoid coming close to filling system memory and any resultant hang-ups. However, special precautions must be observed when using very large input instructions, especially those with a large number of repeats. The 6942A memory can hold approximately 1300 readings; so if you specify a repeat factor of 1300 to one input card you will be using all of memory, with no room for any other active instructions. Of course, memory limitation can also be exceeded by using several smaller instructions. If you execute three successive input instructions that each require 600 readings and do not read back the data from the first instruction, you will again overflow memory and cause a hang-up. To avoid this, the data readings from each instruction should be read back immediately.

5-129 Without calculating exactly how much memory you are using, it is suggested to never fill more than one-third of memory (approximately 450 readings) with input instructions unless they are the only active instructions.

5-130 IP's with Repeat Factors and Waits. In some instances it is desirable to take a series of readings that are paced by a clock; e.g., take a reading every minute, or every 0.5 second. The IP with a repeat factor and a wait time has this self pacing capability. As shown below, the wait time (Wm) is placed directly following the repeat factor, where m is the wait time in the range of 0.1 seconds to 6,535.5 seconds, in increments of 0.1 second.

"IP,Rn,Wm,A1,A2,...T"

5-131 As shown in Figure 5-19, this instruction is the same as the IP with a repeat factor except that there is a specified wait time between readings. All cards are cycled at the beginning of the first reading. As each card completes, the data is read from the card to the 6942's memory. At the completion of

the first reading the instruction pauses for the specified wait time (5-seconds). When the wait time has elapsed, the IP performs the second reading just like the first. This process continues until the last card has completed and the instruction completes.

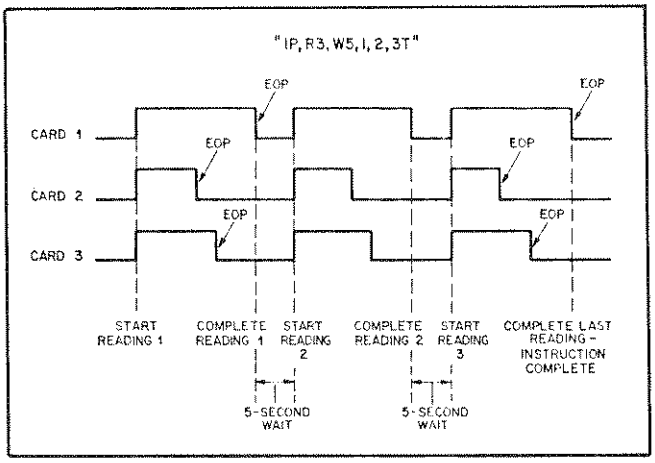


Figure 5-19. IP with Repeats and Waits

5-132 Notice that the IP instruction is executing during the entire instruction sequence, including the wait time between readings. Thus, in the serial mode, no other instruction will run until this IP completes.

5-133 Sample Applications. The most common application using repeats and waits is for a self-paced analog input (see Example 5-18). This application assumes that seven channels must be read (using seven A/D input cards) once every 10 minutes for 24 hours.

Example 5-18. Self-Paced Analog Input Readings

9825 Controller

```
0: dim R[1008]
1: wrt 723,"IP,R144,W600,1,2,3,4,5,6,7T"
2: for I=1 to 1008
3: red 72301,R[I]
4: next I
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(1008)
30   OUTPUT 723;"IP,R144,W600,1,2,3,4,5,6,7T"
40   FOR I=1 TO 1008
50   ENTER 723,01;R(I)
60   NEXT I
```

NOTE

In this Chapter, all 9835/45 examples that include dimension (DIM) or redimension (REDIM) statements, will use OPTION BASE 1. OPTION BASE 1 specifies a lower limit of 1 (instead of the default base of 0) for the array variables in DIM or REDIM statements. If OPTION BASE 0 is used, you must subtract 1 from any DIM or REDIM statements in this Chapter.

5-134 Note that this instruction will take 24 hours to complete and no other instructions can be executed. If you are in serial mode. The Interrupt Now (IN) instruction can be used at the end of this routine to set the SRQ line and thus notify the controller that the instruction has completed (refer to "interrupt Instructions" later in this Chapter). If you cannot tie up the Multiprogrammer for 24 hours, you may be able to utilize the parallel mode (refer to Paragraphs 5-50 through 5-60). In parallel mode, the SRQ line is set when the IP instruction completes.

5-135 Additional Readback Considerations

5-136 The following paragraphs provide an in-depth discussion of some of the particulars of data readback. Although we have only discussed the IP instruction up to this point, the following material applies to most input type instructions that send data back to the controller.

5-137 Specifying Too Many Variables. As described previously, in Paragraph 5-118, if a 9835/45 controller specifies more variables than the 6942A has to offer, it will report an I/O error 159 and halt execution of the program. This represents a potential problem in cases where you are unsure of the number of data readings that will be read back or when you want to use a standard readback routine that must specify enough variables in its ENTER list to cover all situations.

5-138 The way to avoid this problem is to use the 9835/45 "ON ERROR GOSUB" statement, and then write an error recovery routine (see Example 5-19). The "ON ERROR GOSUB" statement in the first line, will cause the 9835 or 9845 to perform a subroutine branch to an error trap routine instead of stopping the controller, whenever any controller error is detected. The subroutine then checks the error code. If the error was not 159, the subroutine prints out the error message and halts, as the system normally does. If the error was 159, the routine executes a subroutine return, which returns control to the statement after the ENTER statement. All of the variables that had received data from the Multiprogrammer will still contain valid data and all the variables that had not yet received data will retain their original values. The result is now exactly the same as that which occurs on a 9825 when too many variables are specified in a "red" statement.

Example 5-19. 9835/45 Error Recovery Routine

```
10 ON ERROR GOSUB Ertrap
                :
                :
          Main body
          of program
                :
                :
1999 STOP
9900 Ertrap: IF ERRN = 159 THEN RETURN
9910 PRINT ERRM$
9920 STOP
```

5-139 Using Continuation Lines on 9825 Controllers. When reading back many data values from an instruction, it

Example 5-20. Using Continuation Lines on the 9825

```
0: wrt 723,"IP,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15T
1: red 72301,A,B,C,D,E
2: red 72301,F,G,H,I,J
3: red 72301,K,L,M,N,O
```

Example 5-21. Dimensioning an Array for Large Readbacks on the 9835/45

```
10    OPTION BASE 1
20    DIM A(15)
30    OUTPUT 723;"IP,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15T"
40    ENTER 723,01;A(*)
```

may not be possible for a 9825 Desktop Computer to read back all of the data with one "red" statement. In these cases, additional "red" statements may be given to input the rest of the data, provided that no other instructions or "red" commands with a different extended talk addresses are given until all of the data has been taken. These additional commands are called continuation lines (see Example 5-20). As many continuation lines as desired may be given.

5-140 **9835/45 Controllers.** Continuation lines are not used with the 9835 or 9845. These controllers read all of the data that the Multiprogrammer sends regardless of how many variables are specified in the ENTER list these controllers automatically throw away any additional data that was not specified in the ENTER list. Thus, all of the data from one instruction must be readback with a single ENTER statement. When reading more than 10 variables, it is suggested that an array be used (example 5-21).

5-141 **Specifying Too Few Variables.** In some instances, the user may find that data that was stored in the Multiprogrammer with an input instruction is worthless. If this is the case, the 6942 requires that the controller only specify one variable, take the first data word, and the rest of the data can be thrown away. If you are using a 9835 or 9845, it will always read back all of the data associated with an instruction and throw away any additional data received after the ENTER variable list has been satisfied.

5-142 For 9825 Controllers, throw away must be done by the Multiprogrammer and not the controller. This is accomplished as follows. After data has been read back that is sufficient to satisfy the "red" variable list, the Multiprogrammer must receive another instruction (any type) or another "red" statement with a **different** extended talk address. Once this is done, the Multiprogrammer assumes that the user does not want the data associated with that specific input instruction, and throws it away. Remember, if your next operation is a "red" statement **with the same extended talk address,** the Multiprogrammer will assume that you have requested a continuation line and send back the data that you want it to throw away.

5-143 External Triggers

5-144 When the Multiprogrammer cycles an I/O card, the card is cycled when the controller specifies it, not necessarily at the optimum time for the external device. Many applications require taking inputs, or performing an operation at a precise time determined by an external signal (for example, taking a reading from an A/D when a external strobe pulse is received). Because of the nature of the signal (a single pulse, not a two-wire handshake), the gate/flag circuitry included on some cards is not an acceptable interface. This problem could be solved using a series of IP's to monitor the strobe line, or a combination of multiple instructions, but these solutions would have a response time of approximately 10 milliseconds or longer.

5-145 To solve this problem, every I/O card was designed with the capability of being cycled externally. When a pulse is applied to the External Trigger Input of any card, it causes the card to perform a cycle, just as if a CY instruction had been executed. The response time of the external trigger (from leading edge of the pulse until the cycle is started) is guaranteed to be under 20 microseconds (under 10 microseconds if the card is not being cycled simultaneously by the Multiprogrammer).

5-146 With an external trigger, the A/D will respond to a read strobe within the 20 microseconds, however, the Multiprogrammer is unaware that the card was cycled. Recall in Chapter 4 the discussion of card cycling and the EOP signal. It was stated there than when a card completed a cycle, it generated the EOP signal back to the microprocessor in the Multiprogrammer, which then found the instruction associated with the card, and based on the instruction, did additional processing.When a card is cycled with an external trigger, although the same EOP signal is generated, it is inhibited from signalling the microprocessor. This is done so as not to clutter the microprocessor with many interrupts that require no processing (no instructions associated with them). The control that determines if the EOP signal will be sent to the microprocessor or not is called the "arm" flip-flop. If the card is armed (the flip-flop set), EOP will be sent to the microprocessor. If the card is disarmed (the flip-flop clear),

EOP is inhibited. All of the high level instructions that wait for the card to complete, arm the card before cycling them so the EOP signal will be seen. The WC, and other low level instructions that do not wait for the card to complete, do not set the arm flip-flop.

5-147 In summary, with the use of external triggers we can synchronize the I/O cards to the external devices, but in so doing we lose the synchronization with the mainframe. The Input External, (IE) instruction is provided to allow complete synchronization, between the external device, the I/O card, and the Multiprogrammer mainframe.

5-148 Input External (IE) Instruction

5-149 The IE instruction is identical to the IP, but instead of arming and cycling the card, it only arms the card and waits for an external trigger to cycle it. The syntax of the IE is the same as the IP except for the opcode.

"IE,A1,A2,A3,...T"

5-150 Figure 5-20 shows a typical IE instruction sent to three input cards. It starts by simultaneously arming all addressed cards and then waiting for the application of each external trigger. As each trigger is received, the card is cycled and when all cards have taken their readings, the instruction stores all data in 6942A memory. Generally, the IE instruction is very similar to the IP. In serial mode, no other instruction can run until the IE completes (including the time that it may be waiting for external triggers). Just like the IP, the IE does not allow specifying the same card (even with different subaddresses) twice in the same instruction.

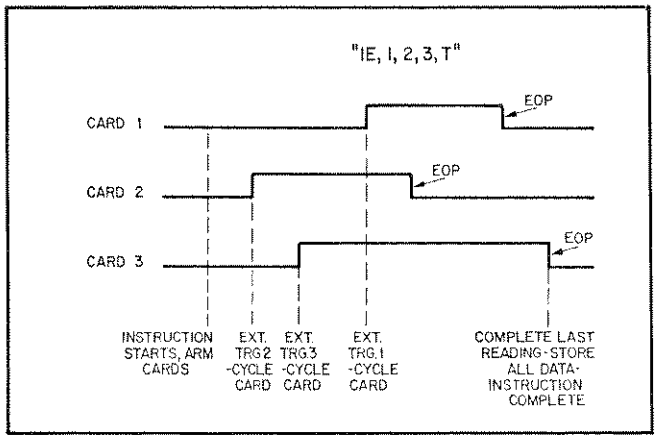


Figure 5-20. Typical IE Instruction

5-151 Reading Back Data. Reading back data from an IE instruction is performed in the same manner as reading back data from the IP, except the IE's extended talk address is 03. Just as with the IP, data can be read back immediately, or it can be left in the Multiprogrammer memory for readback at a later time. Delayed readback follows the same procedures as IP delayed read back; i.e., instructions are readback in the order they were programmed.

Example 5-22. IE Instructions and Input Statements

9825 Controller

```
0: wrt 723,"IE,1,2T,IE,3,4T,IE,5,6T"
1: red 72303,A,B
2: red 72303,C,D
3: red 72303,E,F
```

9835/45 Controllers

```
10   OUTPUT 723;"IE,1,2T,IE,3,4T,IE,5,6T"
20   ENTER 723.03;A,B
30   ENTER 723.03;C,D
40   ENTER 723.03;E,F
```

5-152 IE Application Example. This example assumes that an operators panel in an industrial heating control system allows the operator to specify a desired temperature range for each of one of 16 zones. The operator specifies the zone number, and the desired temperature range on a series of Thumbwheel switches and then presses the execute button, which automatically updates the control system with the new information.

5-153 To implement this, we will use one 16-bit Digital Input card installed in slot 1, that is connected to the Thumbwheel switches. The execute button is connected to the external trigger input of the card. The subroutine shown in Example 5-23 waits until the button is pushed, and then reads the information into variable A.

Example 5-23 External Trigger Application

9825 Controller

```
0: "settemp":wrt 723,"IE,1T"
1: red 72303,A
2: ret
```

9835/45 Controllers

```
10 Settemp:  OUTPUT 723;"IE,1T"
20   ENTER 723.03;A
30   RETURN
```

5-154 Special Considerations. The special considerations for the IE instruction are identical to those associated with the IP. Included are: specifying too many or too few variables, continuation lines, memory utilization, etc,.

5-155 IE Instruction with Repeats. Multiple readings may be required with the IE instruction just as they were with the IP instruction. As shown in Figure 5-21, the IE begins the first of two readings by arming all cards and waiting until the

cards have been cycled by external triggers. When all three cards have completed, their data is read back into memory and the second reading is started by re-arming the cards. The second reading is performed in a similar manner until the last card completes, whereupon the instruction completes.

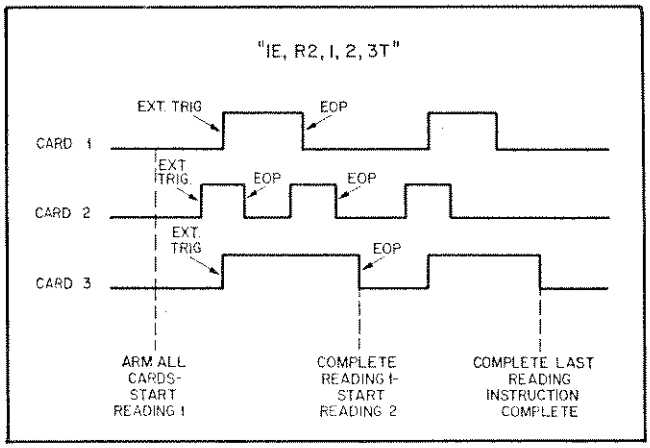


Figure 5-21. IE Instruction with Repeats

5-156 As shown on Figure 5-21, it is possible for two external triggers to occur during a single reading interval. This will cause the card to cycle twice and take two readings, as shown for card 2 during the first read interval. If this occurs, the data stored in the Multiprogrammer at the end of the reading will be the data obtained from the last cycle (on Figure 5-21, the second cycle of card 2 that occurs during the first reading).

5-157 Reading the data back from an IE with repeat factors follows the same rules and procedures as a normal IE instruction. The extended address is still 03. The order in which the data is read back is the same as that for an IP with repeats; i.e., all data from the first addressed card is sent back first, all data from the second card next, etc,.

5-158 The same precaution that applied to IP's with large repeat factors also applies to IE's with large repeats. Remember, there is only room for a maximum of 1300 readings in system memory. The general recommendation given for the IP, which suggests not using more than one-third system memory for all active IP's (unless they are the only active instructions), also applies to IE's, and combinations of IP's and IE's.

5-159 IE with Repeats and Waits. There are occasions when an application calls for a mixture of self pacing and external pacing. For example, a series of externally triggered readings spaced at least five minutes apart. The IE instruction supports a self pacing mode just like the IP. The syntax for this instruction is the same as that for an IP with repeats and waits, except for the opcode.

"IE,Rn,Wm,A1,A2,A3...T"

Again m is within the range of 0.1 second to 6553.5 seconds, in increments of 0.1 second.

5-160 As indicated in Figure 5-22, the IE with repeats and waits operates very similarly to the IE with repeats. They both arm all cards in the address list to start the first reading, and then wait for all the cards to be externally triggered and complete. They then both readback the data to the 6942 at the end of the reading. The only difference in operation is that the IE with repeats starts the next cycle immediately, while the IE with repeats and waits pauses for the specified time before starting the next cycle. This process continues until all the cycles have completed.

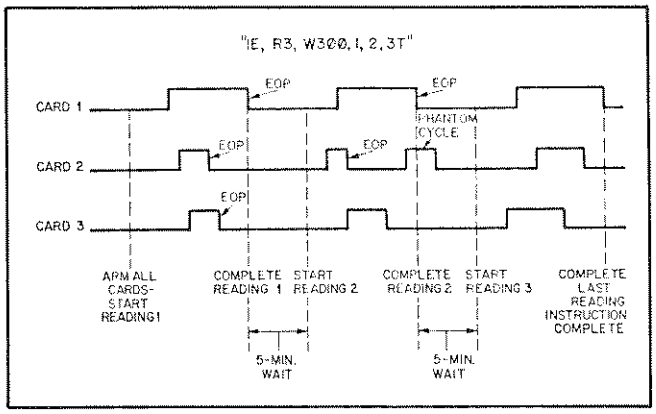


Figure 5-22. IE with Repeats and Waits

5-161 Just as with the IE with repeats (Figure 5-21) it is possible for two external triggers to occur during one reading interval. In the example of Figure 5-22, two external triggers are received during the second read interval by card 2. Because card 2 happens to complete during the 5-minute wait period, its data for that cycle is lost (referred to as a phantom cycle on Figure 5-22).

5-162 The following table compares the IE and IP instruction capabilities.

INST.	Normal	Repeats	Reps. and Wts.
IP	Single Read No Pacing	Multiple Read No Pacing	Multiple Read Clock Pacing
IE	Single Read Ext Pacing	Multiple Read Ext Pacing	Multiple Read Clock + Ext Pacing

5-163 Read Value (RV) Instruction

5-164 This low level instruction allows you to read the current data on a card, or group of cards, and stores the data in Multiprogrammer memory. The RV does not cycle the card(s) and, thus, the data that is read is the value that was stored on

the card prior to the execution of the RV instruction. This instruction completes immediately and there is no restriction on reading the same card more than once in a single instruction.

5-165 This instruction, along with the WF instruction, can be useful in setting-up and then verifying the control subaddresses of some of the more advanced cards.

5-166 Readback. The RV's data is read back from HP-IB extended talk address 06. As usual, the data comes back in the order that it was programmed. Example 5-24 shows a typical RV instruction, sent to cards in slots 1, 2, and 3, followed by the associated input statement. Variable A will contain the data from card 1, B the data from card 2, etc.

Example 5-24. RV Instruction and Input Statement

9825 Controller

```
wrt 723,"RV,1,2,3T"
red 72306,A,B,C
```

9835/45 Controllers

```
OUTPUT 732;"RV,1,2,3T"
ENTER 723.06;A,B,C
```

5-167 The RV instruction, just like the IP and IE, provides two options for reading the data back: take the data immediately, or wait until a later time and read back the data from multiple RV's at once. The same rules apply for the RV: instructions are read back in the order that they were programmed, and only one instruction's data can be read back with a single input statement from the controller.

5-168 The RV operates in the same manner as described for the IP with respect to too many variables on readback, continuation lines on the 9825, throw away mode, and memory utilization. Because the RV instruction completes immediately, it can't hold up the controller from reading back its data. However, in the serial mode, if an instruction sent out before the RV is holding up the system, and an attempt is made to read the RV's data, the 6942 will wait until the other instruction (or instructions) complete, execute the RV, and then send the data back to the controller. Although the RV is similar to the IP and IE in many respects, it does not allow repeats or waits.

5-169 SYSTEM STATUS

5-170 Many tasks can be going on inside the Multiprogrammer at any given instant during the execution of a program. Because of this, detailed status information must be available to the controller at all times. The Multiprogrammer provides different types of status information on HP-IB extended tlak addresses 10 through 13. To obtain the status information the applicable extended talk address is appended to the Multiprogrammer HP-IB address (723) in a controller read statement (e.g. red 723/ENTER 723.10) as described in Chapter 4.

5-171 Certain conditions, such as completion of self test, completion of some instructions, or detection of errors, will cause the Multiprogrammer to set the SRQ (service request) line to the controller (see paragraph 4-10). Extended talk addresses 10, 11, and 12 are used to report all of these conditions. Extended talk address 13 is used for reporting which instructions are busy. The specific status information associated with each extended talk address is summarized in Table 5-3.

5-172 Multiprogrammer SRQ Status Information

5-173 As described in Table 5-3, extended talk address 10 is used to read back status words into variables to indicate the conditions that generated a Multiprogrammer service request. These conditions are: instruction completion, self test completion, error detection, and detection of armed card interrupts. Also as indicated in the table, extended talk address 11 is used to read back error codes when errors are detected and extended talk address 12 is used to read back the addresses of armed cards that interrupted.

5-174 When any conditions described above are detected, the Multiprogrammer performs the following operations: generates the appropriate SRQ status words (and if applicable, compiles a error list and/or armed card interrupt list), sets the HP-IB SRQ line to the controller, and sets the status byte equal to 64 to indicate that the Multiprogrammer is requesting service. This sequence of operations will be referred to as "generating a service request" in the following discussion.

5-175 Once the controller detects that the SRQ line is set, the controller must perform a serial or parallel poll to determine which device(s) on the HP-IB require service. Serial and parallel polling are described in paragraphs 4-10 through 4-15. When the Multiprogrammer requires service, the specific condition(s) that generated the service request can be determined by reading back all six variables (A-F) from extended talk address 10 (red 72310,A,B,C,D,E,F/ENTER 723.10;A,B,C,-D,E,F). As noted in Table 5-3, the last four variables (C,D,E,F) provide status information on armed card interrupts and interrupt instructions. If these features are not being used, just read the Multiprogrammer SRQ and error status words into the first two variables (red 72310,A,B/ENTER 723.10;A,B). Reading the status from extended talk address 10 will reset the Multiprogrammer status byte as well as clear the SRQ line. Reading the six status words (variables A through F) is described in the following paragraphs.

5-176 Instruction/Self Test Completion. The first status word (variable A) read back from extended talk address 10, indicates if any instruction(s) or self test completed and generated a service request. Depending upon the mode (serial or parallel), some instructions will generate a service request when they complete. The controller can use this information to help it sequence through its control program. For instance the controller can wait for a service request to indicate that a sequence of instructions completed before continuing the program.

Table 5-3. System Status Summary

Extended Talk Address	Status Information
10	Multiprogrammer SRQ Status - 9825: red 72310,A,B,C,D,E,F 9835/9845: ENTER 723.10;A,B,C,D,E,F Six words are read into 6 variables (A thru F in examples above) to indicate the following: A - Instruction/self test completion. B - Number of errors. (see extended talk address 11) C - Number of armed card interrupts. (see extended talk address 12) D - Indicates the first card completed in an OI/II instruction. E - Address of card reported in variable D. F - Data from card reported in variable E (applicable only if instruction is II).
11	Error List - 9825: red 72311, [Up to 11 Variables] 9835/9845: ENTER 723.11; [Up to 11 Variables] Gives error codes and may also include card addresses and instruction identification codes (see Appendix B). A maximum of eleven words of error information can be read. The exact number of variables required to store the error codes is specified in variable "B" from extended talk address 10.
12	Armed Card Interrupt List - 9825: red 72312, [Up to 128 Variables] 9825/9835: ENTER 723.12; [Up to 128 Variables] Gives adresses of cards that generated armed card interrupts. The list has space for storing all 128 card addresses in the system.
13	Busy Instructions List - 9825: red 72313,A,B 9835/9845: ENTER 723.13;A,B Two 16-bit words, read into variables A and B in the examples above indicate which instructions are busy.

5-177 Whenever self test completes, a service request is generated to let the user know that the system has been initialized. If, in the middle of an application, the system indicates that it has completed self test, it means that there was either a momentary power failure, or that the controller sent the HP-IB device clear message (clr 723/RESET 723).

5-178 One bit in the status word (variable A) is assigned to each of 11 conditions. Any bit on indicates a service request has been generated. As shown in Table 5-4, 10 different instructions and self test can generate a service request when they complete. Note that for II and OI instructions, a service request is generated when any card in the instruction completes. In parallel mode, all ten instructions can generate a service request. However, in serial mode, only three of the ten generate a service request upon completion. Self test completion always sets service request regardless of mode; however, the system is always placed in serial mode after a self test. How the bits in the status word are set and how service requests are implemented depends upon the specific instruction involved.

Table 5-4. Instruction/Self Test Completion Status Word (Variable A)

Bit No.	Bit Value	Instruction	Service Request Generated
0	1	OB	Inst complete (parallel mode)
1	2	IP	Inst complete (parallel mode)
2	4	OP	Inst complete (parallel mode)
3	8	II	Card complete (serial or parallel)
4	16	OS	Inst complete (parallel mode)
5	32	IE	Inst complete (parallel mode)
6	64	–	—
7	128	–	—
8	256	OI	Card complete(serial or parallel)
9	512	WA	Inst complete (parallel mode)
10	1024	WU	Inst complete (parallel mode)
11	2048	—	—
12	4096	—	—
13	8192	IN	Inst complete (serial or parallel)
14	16,384	Self Test	Self Test complete (always leaves system in serial mode)
15	32,728	—	—

OB, OP, OS, WA, WU, IN, Instructions or Self Test Completion:
In the parallel mode, these instructions generate a service request and set the associated status word bit each time an instruction of this type completes. The bit is cleared after the next read from extended talk address 10 is performed.

IP and IE Instruction:
In the parallel mode, these instructions generate service requests and set the assoicated status word bit, each time an instruction completes and has data available to read back to the controller. The bit remains set until all available data has been read back, however, the service request is reset as soon as the status word is read from extended talk address 10. Therefore, it is possible for the status word bit to be set and the service request to be reset, when a second instruction of the same type completes. In this case, the bit remains set, and the service request is reactivated. There are two conditions that will cause the service request to be reactiviated when the status word bit is already set. They are:

1. A second instruction of the same type completed (as described above).
2. The data from multiple IP or IE instruction is being read back and all the data from one instruction was taken, but there is additional data available from another instruction(s) for readback. Service request is regenerated to inform the controller that there are other instructions ready for readback.

NOTE

If you are sure that the cause of the service request is an IP or IE completion, you may read back the data from the appropriate extended talk address (01 for an IP, 03 for an IE). When all of the data is taken, the service request is reset and the bit in the status word is cleared, even though no read from extended talk address 10 or HP-IB serial poll was performed.

OI and II Interrupt Instructions:
These instructions generate a service request and set the associated status word bit whenever a card programmed in the instruction completes. The bit stays set until all available data has been read back. However, the service request is reset as soon as the status word is read from extended talk address 10. It is possible for the status word bit to be turned off and on multiple times for a single instruction. This can occur if the instruction programs multiple cards, some cards complete, and the data is read back before the other cards complete. Remember the bit in the status word is cleared as soon as all available data is taken (even if all cards in the instruction have not completed). There are two conditons that can cause the service request to be generated when the status word bit is already set:

1. An additional card associated with the instruction type completes.
2. All of the data available for readback is not taken, in which case the service request is generated again to inform the controller that there is more data available.

5-179 Error Detection. The second status word (variable B) read back from extended talk address 10, gives the error status. It indicates the number of error words (0 to 11) present in the error buffer (error list). If the status word = 0, no errors were detected. A non-zero value indicates that errors were detected and a service request generated. The specific value (1 to 11) indicates the number of error variables to read back to the controller. Not that the error list can include card addresses as well as error codes (see description below). The error status word always gives the current number of error words in the error buffer and is updated whenever more errors are detected or whenever errors are read back from extended talk address 11 (see description below). The error buffer has the capability of storing up to 10 error codes. If more than 10 errors are detected, the later errors are lost. Because of this, whenever the controller detects Multiprogrammer errors, all errors are detected, the later errors are lost. Because of this, whenever the controller detects Multiprogrammer errors, all errors should be read back. If a partial read back of the error list is performed, the service request will be reset and will not be set again until another error (or instruction completion, etc.) is detected. Consequently, all errors should be read back as soon as they are detected or they could be lost.

Error List:
The error codes can be read from the error list (error buffer) by addressing extended talk address 11 (red 72311/ENTER 723.11). As stated above, the error status word can specify up to 11 error codes to read back. Three types of error codes can be read back from extended talk address 11: hardware errors, general programming errors, and programming errors related to a specific instruction. The error codes are reported as negative decimal numbers. A detailed desription of all error codes is given in Appendix B. The following examples are given to aid you in understanding the error codes.

Examples:
Hardware errors are usually detected during self test. They are reported as codes −11 through −24. Self test operating and troubleshooting procedures are provided in Chapter 2. General programming errors are reported when the Multiprogrammer is unable to identify the specific instruction where the mistake occurred. For instance, error −1 indicates that a non-existant extended talk address was specified. Another common error is typing the wrong opcode. Typing "OR" instead of "OP" will result in a −2 error, illegal opcode. (See Appendix B).

If the Multiprogrammer detects an error while it is decoding an instruction, it will report it as an instruction error. When an instruction error is reported, there are two pieces of information: what instruction was involved, and what the specific error was. The error code reported will be the sum of the specific error code (−30 or −99) and the instruction identifier (−100 to −3300, in increments of −100).

Example: −332 is −300 + (−32) or unrecognizable card address for an OP instruction.
−230 is a −200 + (−30) or number of cards incorrect for an IP instruction.

For some instruction related errors, the card address programmed is also indicated. When error codes −33, −34, −35, −40, −41, −42, and −43 are reported, the card address associated with the error is also reported. If a card address is reported, it is given as a second error variable, directly following the error code. To distinguish between error codes and card addresses, the card address is always reported as a positive number, while the error code is reported as a negative number.

Example: −334
7
Meaning: Data error in an OP instruction to Card 7.

The above error would occur if you tried sending the value 89 to the card in slot 7 which is formatted to accept octal data. Another example would be programming a larger number than can be represented by the number of bits in the card.

Additional error examples are as follows:

Example	Error Code	Meaning
"OD,1,1,T"	−2	Illegal opcode
"OP,1,T"	−334,1	Data error in OP instruction for card in slot 1. In the example, no data is specified.
"OP,1,0,IP,1,T"	−331	The OP instruction decoding is not completed because the "T" was not sent. Consequently, the I in the IP instruction is an illegal character. Once an error is found, all additional characters are ignored until a "T" is decoded. Thus, in this example, "P,1,T" is ignored.

Error Code Printout Routines:
The following routines will read the error buffer and printout the results.

9825A Controller

```
0: red 72310;A,B
1: if B=0;prt "no errors";sto "end"
2: prt "error  list:";spc
3: for J=1 to B;red 72311,r1
4: prt r1
5: next J
6: "end";spc 1end
```

9835/9845 Controllers

```
10    OPTION BASE 1
20    DIM R(11)
30    ENTER 723.10;A,B
40    IF B<>0 THEN Errors
50    PRINT "NO ERRORS"
60    GOTO End
70 Errors:    PRINT "ERROR LIST:"
80    REDIM R(B)
90    ENTER 723.11;R(*)
100   FOR J=1 TO B
110   PRINT R(J)
120   NEXT J
130 End:  PRINT
140   END
```

5-180 Armed Card Interrrupt Detection. The Arm Card (AC) instruction allows I/O cards to generate service requests when they are cycled by external trigger signals, the Write and Cycle (WC) instruction, or the Cycle (CY) instruction. When an "armed" card is cycled, it will interrupt the microprocessor when it completes causing the Multiprogrammer to generate a service request to the controller. This type of service request is called an "armed card interrupt". More details on armed card interrupts are provided in Chapter 6.

5-181 The third status word (variable C) read back from extended talk address 10 indicates the armed card interrupt status. If C = 0, no cards generated armed card interrupts. A non-zero value indicates that armed card interrupt(s) were detected and a service request generated. A list containing the addresses of the interrupting cards can be read back from extended talk address 12 (red 72312/ENTER 723.12). When an interrupt is detected, the associated card address is placed on the list, the status word (variable C) is incremented, and a service request is generated. The service request is generated every time a new armed card interrupt is detected, and after a readback of the list, if all the data isn't taken. Reading back the entire list clears the status word (variable C), and the service request. The armed card interrupt list, read back from extended talk address 12, indicates which cards generated armed card interrupts. The interrupts are listed in numerical order by card address. The list does not indicate how many times a particular card interrupted or in what order the interrupts occurred. Thus, if a card generates armed card interrupts multiple times it will only be in the list once, and will only generate service request once. Reading the card addresses back to the controller clears the addresses out of the list and allows them to generate another armed card interrupt.

5-182 The armed card interrupt list has space for storing all 128 card addresses possible in a maximum system. The card addresses are read back in numerical order, starting with card 0, regardless of the actual order that the interrupts were received. Sample programs that read the armed card interrupt list and print out the card addresses are provided in Chapter 6.

5-183 Interrupt Instruction Status. The last three status words (variables D,E, and F), read back from extended talk address 10, can be used to obtain data and/or the address of the first card that completes in an OI or II instruction. This feature allows the controller to respond quickly to a single card completion in an OI or II instruction. If the card was programmed by an OI instruction, the card address is read back. If the card was programmed by an II, both the card address and data from the card are read back. Only one card address can be read back. If both an II and an OI instruction are active, the address of the first card that completed is read back.

Example:

```
9825: red 72310,A,B,C,D,E,F
9835/9845: ENTER 723.10;A,B,C,D,E,F
                          |
                  These variables indicate
                  special OI/II status
```

Variable D contains 400 (II) or 900 (OI) indicating the instruction type associated with the first card that completed in an interrupting instruction. Contains 0 if no card completed in an II or OI instruction.

Variable E contains the address of the interrupting card. Only the slot and frame number are indicated, not the subaddress.

Variable F contains the card data if the instruction is an II. If the instruction is an OI, variable F contains 0.

5-184 The information returned in these variables does not in any way affect the normal readback from II and OI extended talk addresses 02 and 09, respectively. The data and/or card address of the first interrupting card will be stored in the Multiprogrammer memory for these extended addresses also. For this reason, care must be exercised when using variables D,E, and F of extended talk address 10. Performing a read from extended talk address 10 clears the SRQ status data stored in Multiprogrammer memory for variables A and D through F. However, the II and OI readback data stored in memory for extended talk addresses 02 and 09 is not read. Thus, if readbacks from extended talk addresses 02 and 09 are not performed, it is possible for the Multiprogrammer memory to be eventually filled with II and OI read back data. Use of OI and II instructions are described in paragraphs 5-187 through 5-200.

5-185 Busy Instruction Status

5-186 In the process of executing a control program, there may be instances when the controller might want to know the status of all instructions in the system. For example, when the Multiprogrammer is in the serial mode, the controller may need to know which instructions are still active (busy) before it can begin the next stage of the test. The busy instruction status is obtained by reading two variables from HP-IB extended talk address 13 (red 72313 A,B/ENTER 723.13;A,B). Each instruction is assigned to a single bit in one of the two variables. The values read back are the summation of all the bit values set. If the bit = 1, there are instructions of this type active in the system. If the bit = 0, there are no instructions of this type active in the system. The busy instruction status does not indicate the number of instructions of each type active in the system. Table 5-5 lists the variables and the bit assignments.

Table 5-5. Busy Instruction Variables:

Bit No.	Value	VAR. A	VAR. B
0	1	OB	RF
1	2	IB	DC
2	4	OP	AC
3	8	II	CY
4	16	OS	WY
5	32	IE	-
6	64	-	WF
7	128	-	RS
8	256	OI	RV
9	512	WA	GS
10	1024	WU	GP
11	2048	-	SC
12	4096	-	RC
13	8192	IN	SE
14	16,384	CC	SD
15	32,768	-	-

The following routines will read the busy instructions status and print out the results.

9825 Controller

```
0:  red 72313,D,E
1:  if D+E=0;prt "no inst busy";gto 29
2:  if D=0;gto "nexchek"
3:  if bit(0,D)=1;prt "OB is busy"
4:  if bit(1,D)=1;prt "IP is busy"
5:  if bit(2,D)=1;prt "OP is busy"
6:  if bit(3,D)=1;prt "II is busy"
7:  if bit(4,D)=1;prt "OS is busy"
8:  if bit(5,D)=1;prt "IE is busy"
9:  if bit(8,D)=1;prt "OI is busy"
10:  if bit(9,D)=1;prt "WA is busy"
11:  if bit(10,D)=1;prt "WU is busy"
12:  if bit(13,D)=1;prt "IN is busy"
13:  if bit(14,D)=1;prt "CC is busy"
14:  "nexchek":if E=0;gto 29
15:  if bit(0,E)=1;prt "RF is busy"
16:  if bit(1,E)=1;prt "DC is busy"
17:  if bit(2,E)=1;prt "AC is busy"
18:  if bit(3,E)=1;prt "CY is busy"
19:  if bit(4,E)=1;prt "WC is busy"
20:  if bit(6,E)=1;prt "WF is busy"
21:  if bit(7,E)=1;prt "RS is busy"
22:  if bit(8,E)=1;prt "RV is busy"
23:  if bit(9,E)=1;prt "GS is busy"
24:  if bit(10,E)=1;prt "GP is busy"
25:  if bit(11,E)=1;prt "SC is busy"
26:  if bit(12,E)=1;prt "RC is busy"
27:  if bit(13,E)=1;prt "SE is busy"
28:  if bit(14,E)=1;prt "SD is busy"
29:  end
```

```
10    ENTER 723,13;D,E
20    IF D+E<>0 THEN Busy
30    PRINT "No instruction busy"
40    GOTO 320
50  Busy:   IF D=0 THEN Nexchek
60    IF BIT(D,0) THEN PRINT "OB is busy"
70    IF BIT(D,1) THEN PRINT "IP is busy"
80    IF BIT(D,2) THEN PRINT "OP is busy"
90    IF BIT(D,3) THEN PRINT "II is busy"
100   IF BIT(D,4) THEN PRINT "OS is busy"
110   IF BIT(D,5) THEN PRINT "IE is busy"
120   IF BIT(D,8) THEN PRINT "OI is busy"
130   IF BIT(D,9) THEN PRINT "WA is busy"
140   IF BIT(D,10) THEN PRINT "WU is busy"
150   IF BIT(D,13) THEN PRINT "IN is busy"
160   IF BIT(D,14) THEN PRINT "CC is busy"
170 Nexchek:   IF E=0 THEN GOTO 320
180   IF BIT(E,0) THEN PRINT "RF is busy"
190   IF BIT(E,1) THEN PRINT "DC is busy"
200   IF BIT(E,2) THEN PRINT "AC is busy"
210   IF BIT(E,3) THEN PRINT "CY is busy"
220   IF BIT(E,4) THEN PRINT "CY is busy"
230   IF BIT(E,6) THEN PRINT "WF is busy"
240   IF BIT(E,7) THEN PRINT "RS is busy"
250   IF BIT(E,8) THEN PRINT "RV is busy"
260   IF BIT(E,9) THEN PRINT "GS is busy"
270   IF BIT(E,10) THEN PRINT "GP is busy"
280   IF BIT(E,11) THEN PRINT "SC is busy"
290   IF BIT(E,12) THEN PRINT "RC is busy"
300   IF BIT(E,13) THEN PRINT "SE is busy"
310   IF BIT(E,14) THEN PRINT "SD is busy"
320   END
```

5-187 INTERRUPT INSTRUCTIONS

5-188 Although the serial mode (instructions are executed one-at-a-time in sequence) frees the user from controlling instruction sequencing, it is restrictive because each instruction must wait until all preceding instructions have completed. The parallel mode cancels this one-instruction-at-a-time restriction, but requires the user to monitor the completion of each instruction in order to control the execution sequence. Interrupt instructions provide a means of programming some parallel operations while still maintaining the automatic sequencing of the serial mode.

5-189 The Output Interrupt (OI) and Input Interrupt (II) instructions are executed in parallel with other instructions. After an interrupt instruction has started executing, it allows other instructions following it in the sequence to begin executing without waiting for the interrupt instruction to complete (see paragraph 5-53). The controller can keep track of which parallel operations have completed because the Multiprogrammer will generate a service request after each

card programmed in an interrupt instruction completes. As soon as any card completes, service request is set and the controller can read back the card address (and card data if instruction is an II) from the applicable extended talk address.

5-190 As described in Table 5-2, the interrupt instructions are the only instructions that generate a service request when they complete in the serial mode. In addition to the OI and II instructions, the Multiprogrammer instruction set also includes the Interrupt Now (IN) instruction which also sets service request and is used to indicate that a group of instructions (e.g. OP's, OS's, IP's, etc.) has completed. Each type of interrupt instruction is described in the following paragraphs.

5-191 The OI and II interrupt instructions are used whenever specific parallel operations are required in an otherwise serial sequence of instruction. For example, an II instruction can be used to monitor an abort button or a series of limit switches, while a control program of sequential instructions is running. With a pure serial sequence, it would be impossible to provide a fast response under all conditions. The parallel II instruction can detect if the button is depressed or if the switch is closed and generate a service request allowing the system to determine the cause of the problem and respond.

5-192 Figure 5-23 illustrates five instructions (I_2 through I_6) executed sequentially to control a unit under test. Although the control is completely serial, the operator has an abort button which he can push at any time to stop the test instantly. I_1 is an II instruction that monitors this abort button. I_2 through I_6 are the five sequential steps of the control algorithm, and I_7 is an IN instruction, that will cause a service request to be generated when the test is over. Once the test has begun, the controller can go inactive simply monitoring or waiting for an interrupt from the SRQ line. SRQ will be set by one of two conditions:

1. The abort button is pushed in which case immediate action is taken by the controller to abort the test.
2. The IN instruction is executed indicating the test is over.

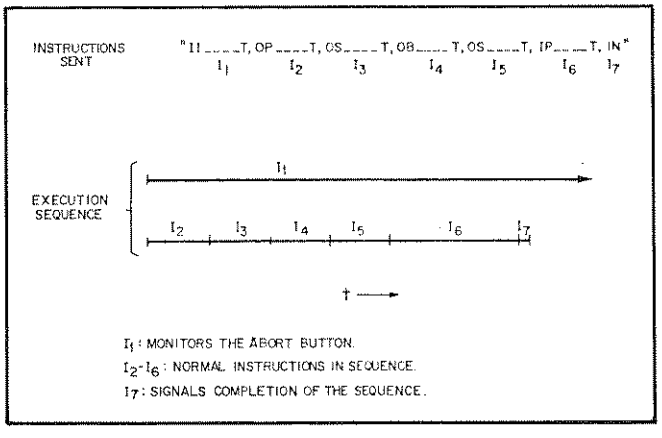


Figure 5-23. Use of Interrupt Instructions

5-193 Output Interrupt (OI) Instruction

5-194 The OI instruction simultaneously outputs data to output cards and allows the addresses of the completed cards to be read back from extended talk address 09. The OI always runs in parallel with other instructions. As each OI card completes, its address is stored in 6942 memory and SRQ is set. The syntax for the OI instruction is as follows:

"OI,A1,D1,A2,D2,...T"

The card addresses and associated data are specified by each address/data pair (A1/D1,A2/D2,etc). If desired, an OI can be used to program one output card.

5-195 As shown in Figure 5-24, the OI first sends the data simultaneously to the specified subaddress (usually subaddress 0, first rank storage) on all cards. The OI then cycles all cards simultaneously. Although the instruction does not complete until all cards addressed in the instruction have completed, once started it does not prevent other instructions from starting in the serial mode. As soon as any card programmed in the instruction completes (EOP) the Multiprogrammer generates a service request and the controller can read back the address of the completed card(s), even though the entire instruction has not yet completed. The controller can also determine the sequence of card completion because the list of completed cards is sent back in the order of completion, the first card to complete is read back first and the last card to complete is read back last.

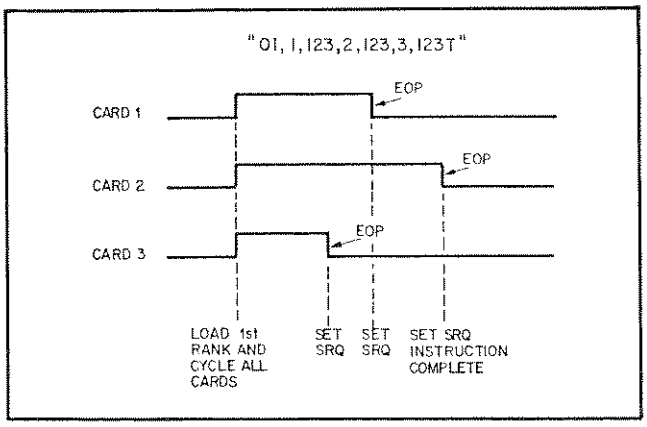


Figure 5-24. Typical OI Instruction

5-196 Processing Modes: The OI instruction is affected by the parallel and serial modes. In both modes, the instruction is started according to the rules of the specified mode, but once started, regardless of mode, the instruction runs in parallel with the rest of the system. In either serial or parallel mode, when any card completes a service request is generated. The OI will also run in the immediate mode. Paragraphs 5-50 through 5-60 outline the affects of the processing modes on each type of instruction.

5-197 Restrictions and Error Checking. The same restrictions and error checks apply to OI instruction as were described for the OP instruction in paragraph 5-67.

5-198 Sample OI Instruction. Example 5-25 uses an OI instruction to send the data value "123" to Digital Output Cards in slots 1, 2, and 3. A particular card can only be programmed once in a single OI instruction. This restriction also applies to different subaddresses on the same card. As each card completes, its address is stored in Multiprogrammer memory and SRQ is set.

Example 5-25. Using An OI Instruciton to Program Three Cards

9825 Controller

```
0: wrt 723,"OI,1,123,2,123,3,,123T"
```

9835/9845 Controllers

```
10   OUTPUT 723;"OI,1,123,2,123,3,123T"
```

5-199 OI Data Readback. When a card associated with an OI instruction completes, a service request is generated and the address of the card that completed can be read back. As mentioned before, the card addresses are sent back in the order that the cards completed. Only the slot number and frame number are returned; the subaddress is not returned. The addresses of cards that completed in an OI instruction can be read back to the controller from extended talk address 09 (red 72309/ENTER 723.09). A single variable containing the card address is returned for each completed card.

5-200 If multiple cards are specified in an OI instruction, it is impossible to tell how many cards have completed when service request is set. Thus, the number of variables required for readback is unknown. This is called a variable length readback and depending upon the specific controller used, requires processing in a special manner.

Variable Length Readback on the 9825: Because it is unknown how many cards in an OI instruction have completed, the readback variables should be initialized to some value which cannot be a card address, such as −1. Normally, a sufficient number of variables are specified to handle all the card addresses programmed, in case all the cards had completed. The following example programs 4 cards with a single OI instruction.

```
wrt 723,"OI,1,123,2,123,3,123,4,123T"
```

If service request was set because of an OI card completion in the above instruction, up to the maximum of 4 variables will be returned. If 2 variables were returned, A and B contain the card addresses and C and D contain −1's. The next example initilizes four variables, reads back the variables, and checks to see how many cards completed and then goes to the appropriate line to handle them.

```
-1→A→B→C→D
red 72309,A,B,C,D
if D#-1;sto "4vars"
if C#-1;sto "3vars"
if B#-1;sto "2vars"
"1vars":
```

When reading more than 10 variables, it is suggested that an array be used. Again, because it is unknown exactly how many addresses will be read back, initialize all the variables to −1 before readback, and check the values after readback. Unlike the other instructions that send data back to the controller, the OI requires that all the data must be taken. If a partial read is performed, the data not taken will be sent back the next time a read from extended talk address 09 is performed, even if other instructions or readbacks are executed before the read from 09 can be continued. The following example sets up an array to read back 100 variables and initializes each variable to −1 before executing the read statement.

```
for I=1 to 100
-1→A[I]
red 72309,A[I]
if A[I]=-1;sto "out"
next I
"out":
```

Variable Length Readback on 9835/9845. The 9835 and 9845 controllers are not designed to handle variable length readbacks. If the Multiprogrammer does not send data for every variable specified in the enter statement list (e.g. ENTER 723.09;A,B,C,D), an I/O error 159 is indicated on the controller and the controller program stops. To avoid this problem, use the "ON ERROR GOSUB" statement and write an error recovery routine exactly as described in paragraph 5-138. This method allows the controllers to correctly process the case where fewer variables were returned than were requested with the ENTER statement. Remember that this method is only for the case where fewer variables were sent back than were requested.

If the Multiprogrammer sends more variables than were requested in the enter list, the 9835/9845 controllers will still accept the additional variables but then throw them away. All the data must be taken with a single ENTER statement (if there are too many variables, use an array). Just as with the 9825, all variables must be initialized to −1 before executing the ENTER statement. The following example utilizes the "ON ERROR GOSUB" in the event that there is less data than variables specified, then sets up an array to read back 100 variables, and initializes each variable to −1 before executing the ENTER statement.

9835/9845 Controllers

```
10   ON ERROR GOSUB Err
20   !
30   !
40   OPTION BASE 1
50   DIM A(100)
60   OUTPUT 723;"OI,1,133,2,123,.....T"
70   FOR I=1 TO 100
80   A(I)=-1
90   NEXT I
100  ENTER 723.09;A(*)
110  !
120  !
130  !
140 Err: IF ERRN=159 THEN RETURN
150  PRINT ERRM$
160  END
```

5-201 Input Interrupt (II) Instruction

5-202 The II instruction simultaneously programs a group of input cards to take readings. The II always runs in parallel with other instructions. As each card specified in the II completes, its data and address are stored in 6942 memory and SRQ is set. The data and addresses can be read back from extended talk address 02. The syntax for the II instruction is as follows:

"II,A1,A2,A3,...T"

Card addresses are specified by A1,A2,A3, etc.

5-203 As shown in Figure 5-25 the II instruction cycles all cards specified in the address list simultaneously. As each card completes, the specified subaddress (usually subaddress 0, input data storage) on all cards is read into the 6942's memory. Although the instruction does not complete until all the cards in the address list have completed, once started the II does not prevent other instruction from starting in the serial mode. As each card completes (EOP), service request is generated and the controller can read back the data and address of the completed card. The card address and card data are sent back to the controller in the order of completion. In the example of Figure 5-25, the order of completion is: card 2, card 3, and 1.

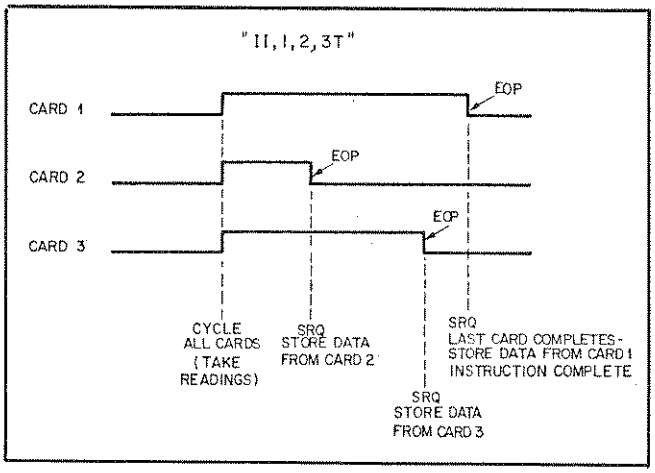


Figure 5-25. Typical II Instruction

5-204 **Processing Modes.** The II is affected by the modes of operation in the same manner as the OI.

5-205 **Restrictions and Error Checking.** The same restrictions and error checks apply to the II instruction as were described for the IP instruction in paragraph 5-111 and the OI in paragraph 5-197.

5-206 **Sample II Instruction.** Example 5-26 uses an II instruciton to take readings from Digital Input cards in slots 1, 2, and 3. As was described for an OI instruction, a particular card can only be programmed once in a single II instruction. As each card completes, its address and data are stored in Multiprogrammer memory, and SRQ is set.

Example 5-26. Using II Instruction to Program Three Cards

9825 Controller

```
0: wrt 723,"II,1,2,3T"
```

9835/9845 Controllers

```
10   OUTPUT 723;"II,1,2,3T"
```

5-207 **II Data Readback.** When card(s) associated with an II instruction completes, service request is generated, and the card addresses and data can be read back from extended talk address 02 (red 72302/ENTER 723.02). The card address followed by the card data are always read back in the order of card completion. Thus, two variables are returned, the first containing the address and the second containing the data, for each card that completed. If multiple cards are specified in an II, it is impossible to determine how many cards have completed when service request was set. For this reason, the same procedures used for the OI must be used for the II instruction. The controller's variables should be preset to −1, an illegal card address value. The −1 value is not necessarily an illegal data value, however. As the values are read in, they should be processed as pairs of data, checking the card address value to, see if it is a −1. If the card address is −1, no data was returned for that particular pair. If the address is not −1, there is valid data in both variables.

5-208 The procedures described previously for reading back data on the 9825 and 9835/9845 controllers in the OI instruction should also be followed for the II instruction. The 9825 allows a readback to be continued, whereas the 9835/9845 controllers do not. With an II (and OI) all data must be taken. However, the 9835/9845 controllers will throw away data values read back if enough variables are not specified in

the ENTER list. The following examples illustrate using arrays to read back data from II instructions on extended talk address 02.

9825 Controller

```
0: wrt 723;"II,1,2,3T"
1: -1→A→B→C→D→E→F
2: red 72302,A,B,C,D,E,F
3: if E#-1;gto "3vars"
4: if C#-1;gto "2vars"
5: "1vars":
```

9835/9845 Controllers

```
10   ON ERROR GOSUB Err
20   !
30   OPTION BASE 1
31   DIM A(6)
40   OUTPUT 723;"II,1,2,3T"
50   FOR I=1 TO 6
60   A(I)=-1
70   NEXT I
80   ENTER 723.02;A(*)
90   IF A(5)<>-1 THEN GOTO Var3
100  IF A(3)<>-1 THEN GOTO Var2
110 Var1: !
```

5-209 Multiple OI and II Interrupt Instructions

5-210 Another feature of Interrupt Instructions is that multiple interrupt instructions of the same type will run in parallel with respect to each other. Thus, multiple independent applications can run in parallel using the interrupt instructions. Figure 5-26 illustrates three OI instructions (I_2, I_4, I_5) running in parallel. Note that at t_2, I_4 will not start until I_3 (an OS) has completed. At t_3, the three OI instructions are all running in parallel.

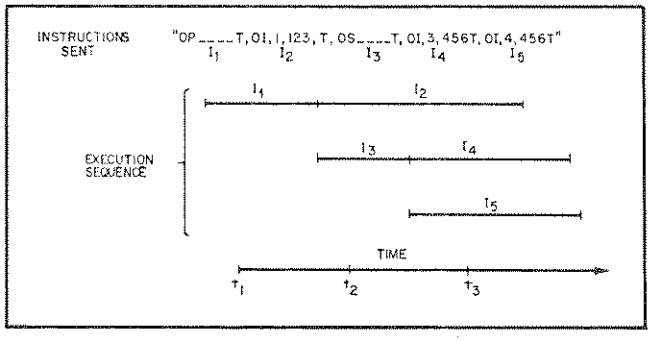


Figure 5-26. Multiple Interrupt Instructions

5-211 **Restrictions.** As noted in Figure 5-26, once the OI instructions get started, they run in parallel. There are a few restrictions:

1. Interrupt Instructions run in parallel only after they have been started. The OI and II instructions are started following the rules of the current mode. In the serial mode, no other instructions can be executing (except other interrupt instructions that were already started) when an OI or II instruction is started. Thus in Figure 5-26, I4 will not start until after I3 has completed.

2. Because it doesn't make sense to have a single card executing in more than one instruction simultaneously, when an OI or II instruction is started, all the cards associated with the instruction are checked to see if they are busy with any other instructions (in the serial mode they could only be busy with other interrupt instructions). If any card is busy, the interrupt instruction must wait until the card completes before the instruction will start. In the serial mode, while the interrupt instruction waits, no later instruction can be started. Figure 5-27 illustrates two II instructions (I_1 and I_2) addressing card 1. Note that instruction I_2 will not start until card 1 in I_1 has completed. Since I_2 hasn't started, no later instructions can start.

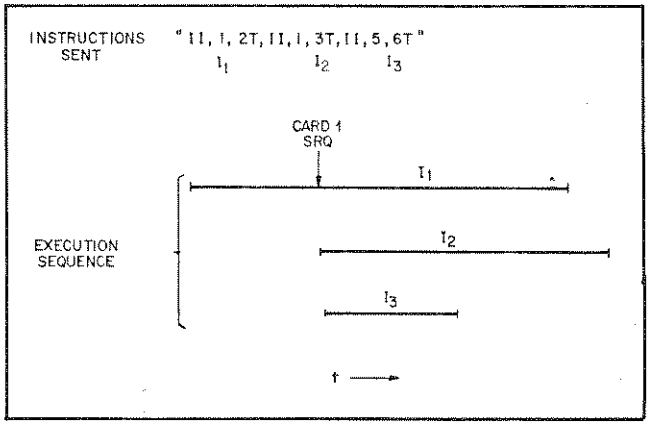


Figure 5-27. Same Card Restriction in Interrupt Instructions

5-212 **Readback of Multiple OI's and II's.** Unlike multiple IP and IE instructions, readback of the data from multiple OI and II instructions is accomplished with a single input command. When multiple's OI's or II's are running in parallel, the Multiprogrammer mergers all the instructions together into one big instruction. thus, all of the data can be read back with one command. As described previously, this requires a variable length readback. Be sure to initialize all of the variables to −1. On the 9825, the user can take the data all at once, or use continuation lines. With the 9835 and 9845, the user must take all the data back with a single ENTER, or else the controller throws the remaining data away.

5-213 When a single OI or II is active, the card addresses are sent back in chronological order. This is not the case with multiple OI's or II's. When the readback is performed, all the cards that have completed in all the instructions are available for readback. The cards inside a specific instruction will still be read back in chronological order, however, the data from the different instructions are sent back in a non-deterministic order with respect to the sequence of the instructions.

Example: Assume all cards complete in the order programmed

"OI,1,123,2,123,3,123T,OI,4,123,5,123,6,123T"

In the above example, the card addresses could be returned (but not necessarily) in the following order from extended talk address 09: address 4,5,6,1,2,3.

5-214 Note also that because of the way that instructions are processed by the firmware, one large OI or II will run somewhat faster than multiple smaller ones.

Example: "II,1,2,3T" will run faster than: "II,1T,II,2T,II,3T"

5-215 Interrupt Now (IN) Instruction

5-216 The IN instruction is used to generate a service request to indicate that all previous instructions have completed. The syntax for the IN instruction is as follows:

"IN or INT"

The terminator "T" is not required but will be accepted if it is sent.

5-217 In the serial mode, most instruction types do not generate service request when they complete. When a group of instructions of this type completes, a completion indicator might be helpful. An IN instruction placed at the end of a routine can be used as a completion indicator. The instructions will be automatically sequenced and executed by the Multiprogrammer while the controller is free to perform other operations. At the completion of the sequence, the IN executes and sets service request indicating that the Multiprogrammer has completed its task and is ready for additional service. The effects are described in paragraphs 5-55.

5-218 Interrupt Instruction Application Program

5-219 Example 5-27 illustrates a program that utilizes an OI and an IN instruction. The OI is the first instruction and will run in parallel with the remaining instructions. Assume that the OI instruction programs a Digtial Output card in slot 1 which in turn arms an external abort detector device. The external gate/flag handhsake from the Digital Output Card is extended to the abort detector device. If the device detects an abort conditions, the flag will be returned to the Digital Output card resulting in a service request and a jump to the "Olint" routine which will abort the test.

5-220 The next five instructions (OP,OS,etc.) will be executed in sequence to control the unit under test. The last instruction in the sequence is an IN which when executed sets service request causing a jump to the "Done" routine indicating that the control sequence was completed. Note that the program includes a serial poll, rds (723)/STATUS 723, to determine if the Multiprogrammer requested service. Extended talk address 10 is read (red 72310,A,B,C/ENTER 723.10;A,B,C)) to detemrine if the OI or IN instruction generated the service request.

Example 5-27. Interrupt Instruction Application Program Using OI and IN Instructions

9825 Controller

```
ami 7;"checkint";eir 7
wrt 723;"OI,1,123T"
wrt 723;"OP,2,5.3,4,0.1T"
wrt 723;"OS,10,20.5,11,22.5,12,22.6T"
wrt 723;"IP,6,7T"
wrt 723;"WC,10,205T"
wrt 723;"IP,6,7,8T"
wrt 723;"IN"



"checkint":if rds(723)#64;sto "OtherInt"
red 72310,A,B,C
if bit(8,A);sto "Olint"
if bit(13,A);sto "done"
```

9835/9845 Controllers

```
10   ON INT #7 GOSUB Checkint
20   CONTROL MASK 7;128
30   CARD ENABLE 7
40   OUTPUT 723;"OI,1,123T"
50   OUTPUT 723;"OP,2,5.3,4,0.1T"
60   OUTPUT 723;"OS,10,20.5,11,22.5,12,22.6T"
70   OUTPUT 723;"IP,6,7T"
80   OUTPUT 723;"WC,10,205T"
90   OUTPUT 723;"IP,6,7,8T"
100  OUTPUT 723;"IN"



150 Checkint: STATUS 723;S
160  IF S<>64 THEN GOTO Otherint
170  ENTER 723.10;A,B,C
180  IF BIT(8,A) THEN GOTO Olint
190  IF BIT(13,A) THEN GOTO Done
```

5-221 Example 5-28 is a sample program that demonstrates how interrupt instructions allow three independent control tasks to run simultaneously. Each task is assigned a sequence of six OI instructions to output data. In each task, instructions are executed sequentially and the next instruction can not start until the previous one has completed. The instructions for each task (A,B, and C) are stored in string arrays A$, B$, and C$ as shown in the example. Note that the output cards for each task are installed in separate frames.

Example 5-28. Using OI Instructions to Control Independent Tasks.

9825 Controller

```
0: dim A$[5,20],B$[6,20],C$[6,20]
1: ldf 5,A$,B$,C$
2: ssb "Cheker"
3: dsp "test running"
4: 2→A→B→C
5: wrt 723,A$[1],B$[1],C[1]
6: "wait":if rds(723)=128;jmp 0
7: -1→D
8: red 723@9,D
9: if D=-1;gto "wait"
10: if D>15;gto "test2"
11: if A<7;wrt 723,A$[A]
12: A+1→A;gto "next"
13: "test2":if D>115;gto "test3"
14: if B<7;wrt 723,B$[B]
15: B+1→B;gto "next"
16: "test3":if C<7;wrt 723,C$[C]
17: C+1→C
18: "next":if A<7 or B<7 or C<7;gto "wait"
19: dsp "test done"
20: stp
```

9835/9845 Controller

```
1     ON ERROR GOSUB Error
2     OPTION BASE 1
10    DIM A$(6,20),B$(6,20),C$(6,20),A(11)
20    ASSIGN #2 TO "OIINSTR"
30    READ #2;A$,B$,C$
40    GOSUB Cheker
50    PRINT "Test running"
60    A=2
70    B=2
80    C=2
90    OUTPUT 723;A$(1,1&LF&1)
100 Wait:  STATUS 723,6
110   IF S=128 THEN Wait
120   D=-1
130   ENTER 723,8;D
140   IF D=-1 THEN Wait
150   IF D>15 THEN Test2
160   IF A<7 THEN OUTPUT 723;A$(A)
170   A=A+1
180   GOTO Next
190 Test2:  IF D>115 THEN Test3
200   IF B<7 THEN OUTPUT 723;B$(B)
210   B=B+1
220   GOTO Next
230 Test3:  IF C<7 THEN OUTPUT 723;C$(C)
240   C=C+1
250 Next:  IF (A<7) OR (B<7) OR (C<7) THEN Wait
260   PRINT "Test done"
270   STOP
```

Instruction Arrays

Task A

```
A$
OI,1,123T
OI,3,567T
OI,2,21.34T
OI,3,3.95T
OI,1,456T
OI,3,-4.92T
```

Task B

```
B$
OI,105,123T
OI,103,.035T
OI,101,-93.2T
OI,103,.111T
OI,105,981T
OI,103,.555T
```

Task C

```
C$
OI,200,73T
OI,202,-49.47T
OI,203,1073T
OI,202,101.4T
OI,203,-2741T
OI,200,0T
```

Explanation:

9825	9835/45	Functional Description
	1	Set up error recovery routine for variable length readback.
	2	Set up Base 1 for arrays.
0	10	Dimension arrays for OI instructions.
1	20,30	Input OI instructions from a data file.
2	40	Call "Cheker" to clear SRQ line and check for any errors before test starts.
3	50	Display message
4	60-80	Initialize the variables that keep track of the next instruction to be sent out for each task.
5	90	Send the first instruction for each task.
6	100,110	Wait for SRQ.
7-9	120-140	See if a card from an OI completed, if not, go back and wait for SRQ again.
10	150	Check if the card that completed is from the first frame (Task A).
11	160	It is Task A. If we have not sent all 6 OI instructions, send the next one.
12	170,180	Point to the next instruction for Task A.
13-15	190-220	Task B operations. Same as described above (lines 10-12/150-180) for Task A.
16,17	230-240	Task C operations. Same as described above for Task A.
18	250	Check if all instructions have been sent (see note below).
19	260	Test completed.

Note: Checks only if all instructions have been sent. Does not check if all instructions have completed. To check if all instructions have completed, change the "7"s to "8"s. If all have not completed, loop back or print the "test done" message and stop.

5-222 I/O CARD FORMAT INSTRUCTIONS

5-223 Chapter 4 introduced the concept of I/O card data formats and described the five format parameters: data type, least significant bit (LSB) value, size (number of bits), an optional programmable limit, and a card identifier. These parameters specify how the Multiprogrammer will process the data it sends to or receives from a particular card. Whenever the system is run through self test, every card in the system is accessed, and a "wake-up" word for each card is read and stored in Multiprogrammer memory. The card "wake-up" word specifies the values of all five data format parameters. Table 4-3 summarizes the "wake-up" values for each type of I/O card. The following paragraphs describe the Read Format (RF) instruction which reads the current value of all five parameters and the Set Format (SF) instruction which can be used to modify one or more of the parameters.

5-224 Read Format (RF) Instruction

5-225 The RF instruction reads the current value of the data format parameters of a card or group of cards. The syntax for the RF instruction is as follows:

"RF,A1,A2,A3,...T"

5-226 The RF Instruction stores five variables of information for each card specified in the address list. If the card is a functional card (no errors detected), it returns, in the following order, the card identifier, data type, LSB, size, and the value of an optional limit. If a hardware error was detected in the self test routine, or if there is no card in the slot, a single variable consisting of an error code, followed by four variables with 0 in them, will be returned. The error code will be between −50 to −61. See Appendix B for a description of the errors.

5-227 Processing Modes. The RF instruction executes immediately as soon as it is decoded by the Multiprogrammer. Thus, it is unaffected by the modes (serial, parallel, or immediate) and can be executed at any time.

5-228 Errors. The RF instruction detects and processes the following errors: sending an illegal character in the middle of the instruction (error code −1831), forgetting to send the "T", or sending an illegal card address (error code −1832). Note that forgetting to send the "T" will be reported as an illegal character (error code −1831) when the first letter of the next instruction is detected. If any errors are detected, the entire instruction is thrown-away (not executed). Card related errors such as "no card in slot" (error code -50) or "card not in system" (error code -51) will be reported when data from the RF instruction is read back.

5-229 Sample RF Instruction. The examples below illustrate sending an RF instruction to read the format parameters of the cards installed in slots 1,2, and 3.

9825 Controller

```
wrt 723,"RF,1,2,3T"
```

9835/9845 Controllers

```
OUTPUT 723;"RF,1,2,3T"
```

5-230 Data Readback. Data from the RF instruction is read back on HP-IB extended talk address 04 (red 72304/ENTER 723.04). Five format parameters are read back for each card in the address list. If no errors are detected, the five parameters are read into five controller variables in the following order:

Variables	Parameter
1	Card identifier Code (0-63)
2	Data type code (1-7)
3	LSB/Range
4	Size (12 or 16)
5	Limit

Each of the parameters is described in detail in Chapter 4.

5-231 Example 5-29 uses an RF instruction to read the parameters of the cards in slots 1,2, and 3. After the RF is executed, the data is read back to the controller from extended talk address 04. Fifteen (5 for each card) controller variables are specified to receive the data. The 9825 example uses r-variables while the 9835/9845 example uses an array to store the data.

Example 5-29. Reading Back Data from an RF Instruction

9825 Controller

```
0: wrt 723,"RF,1,2,3T"
1: for I=1 to 15
2: red 72304,rI
3: next I
```

9835/9845 Controllers

```
10   OPTION BASE 1
20   DIM A(15)
30   OUTPUT 723;"RF,1,2,3T"
40   ENTER 723.04;A(*)
```

5-232 Assume in the example above the slot 1 contains a Digital Output card, slot 2 a Voltage D/A card, and slot 3 is empty. For this card configuration and assuming the standard card wake-up values, the following data would be stored in the controller variable r1-r15 for 9825 and A1-A15 for 9835/9845) specified in the read back from extended talk address 04. Note that the data is read back in the order that the cards were addressed in the RF instruction.

Card 1 parameters read back:
r1/A1 = 42.0 r2/A2 = 3.0 r3/A3 = 1.0 r4/A4 = 16.0
r5/A5 = 0.0

Card 2 parameters read back:
r6/A6 = 48.0 r7/A7 = 1.0 r8/A8 = .005 r9/A9 = 12
r10/A10 = 0

Card 3 parameters read back:
r11/A11 = -51 r12/A12 = 0 r13/A13 = 0 r14/A14 = 0
r15/A15 = 0

The readbacks above indicate that the cards are configured as follows:

Card 1 is a Digital Output card (code 42), with data type 3 (unsigned binary), an LSB = 1, 16-bit size, and no programming limit set (0).

Card 2 is a Voltage D/A card (code 48), with data type 1 (2's complement), an LSB = .005V, 12-bit size, and no programming limit set (0).

Slot 3 is empty (error code -50). The remaining four variables in this case contain zeros.

5-233 If a hardware error is detected on a card during self test, if there is no card in the slot, or if the card slot address is not used in the system a single error code followed by four zeros is returned. Error codes are negative numbers in the range from -50 to -61. The error code -50 in the previous example indicated "no card in slot". Appendix B gives a complete description of all error codes.

5-234 Only one RF instruction can be active in the system at a time. If a second RF is sent before data from the first RF has been read back, the Multiprogrammer will throw away the first instruction's data. Also, when reading back data from an RF, ensure that a sufficient number of variables is specified to avoid losing the data. Readback methods using the 9825 and 9835/9845 controllers are similar to those described for the IP and IE instructions.

5-235 Example 5-30 and 5-31 are programs that print out data returned from an RF instruction. Example 5-30 is for the 9825 controller and Example 5-31 is for the 9835/9845 controllers. Sample printouts are provided with each program. Note that card errors are not printed out and only frame 0 addresses (0 to 15) are read back.

Example 5-30. 9825 Program for Printing Out Data Format Parameters

Program

```
0: clr 723
1: wait 4000
2: for N=0 to 15
3: wrt 723,"RF",N,"T"
4: red 72304,A,B,C,D,E
5: if A<0:gto "nex"
6: prt "Slot #",N
7: prt "Card ID #",A
8: prt "Data Type",B
9: prt "LSB/Range #",C
10: prt "No.of bits",D
11: prt "Limit # is",E
12: prt ""
13: "nex":next N
14: end
```

Sample Printout

```
Slot #          3.00
Card ID #      42.00
Data Type       3.00
LSB/Range #     1.00
No.of bits     16.00
Limit is        0.00

Slot #          4.00
Card ID #      42.00
Data Type       3.00
LSB/Range #     1.00
No.of bits     16.00
Limit is        0.00

Slot #          5.00
Card ID #      48.00
Data Type       1.00
LSB/Range #     0.01
No.of bits     12.00
Limit is        0.00
```

Example 5-31. 9835/9845 Program for Printing Out Data Format Parameters

```
10    ! THIS PROGRAM PRINTS THE DATA RETURNED FROM AN "RF" INSTRUCTION
20    ! IT LOOKS AT ALL THE SLOTS WHICH HAVE CARDS IN THEM IN THE MAINFRAME
30    OPTION BASE 1
31    RESET 723
32    WAIT 4000
40    DIM A(80)
50    OUTPUT 723;"RF,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15T"
60    ENTER 723.04;A(*)
70    FOR N=1 TO 16
80    FOR I=1 TO 5
90    C=1+5*(N-1)
100   IF I<>1 THEN GOTO Var2
110 Var1: IF A(C)<0 THEN GOTO Nex
120   PRINT "SLOT # IS.............................",N-1
130   PRINT "CARD IDENTIFIER # IS.................",A(C)
140 Var2: IF I=2 THEN PRINT "DATA TYPE IS #.....................",A(C)
150 Var3: IF I=3 THEN PRINT "LSB OR RANGE CODE # IS.............",A(C)
160 Var4: IF I=4 THEN PRINT "SIZE IS............................",A(C)
170 Var5:   IF I=5 THEN PRINT "LIMIT # IS.......................",A(C)
180 IF I=5 THEN PRINT ""
190 IF I=5 THEN PRINT ""
200       NEXT I
210 Nex: NEXT N
220   END
```

Sample Printout

```
SLOT # IS.............................        3
CARD IDENTIFIER # IS.................        42
DATA TYPE IS #.......................        3
LSB OR RANGE CODE # IS...............        1
SIZE IS..............................        16
LIMIT IS.............................        0


SLOT # IS.............................        4
CARD IDENTIFIER # IS.................        42
DATA TYPE IS #.......................        3
LSB OR RANGE CODE # IS...............        1
SIZE IS..............................        16
LIMIT IS.............................        0


SLOT # IS.............................        5
CARD IDENTIFIER # IS.................        48
DATA TYPE IS #.......................        1
LSB OR RANGE CODE # IS...............        .01
SIZE IS..............................        12
LIMIT IS.............................        0
```

5-236 Set Format (SF) Instruction

5-237 The SF instruction allows the user to change one or more of the data format parameters on one I/O card or a group of I/O cards. It also provides a means by which the user can respecify certain system parameters. The syntax for the SF instruction is as follows:

"SF,A1,#P,data type, LSB,size,limit,card ID,A2,#P,.....T"

5-238 The SF instruction sets up the data format parameters for each card in the address list as specified by each card's parameters. Multiple cards can be addressed but each card requires its own set of parameters. The #P value indicates how many parameters are being specified for the card. For example, #P = 0 indicates that no parameters will be specified; #P = 2 indicates that two parameters (data type and LSB) will be sent; #P = 4 indicates that four parameters (data type, LSB, size, and limit) will be sent. The actual processing for each I/O card is determined by the #P value. The allowable values are 0 to 4, -1, -2, and -5. The -1, -2, and -5 values indicate that the SF instruction will perform some special processing.

- #P = 0: The I/O card's wake-up word is read and all parameters are returned to their wake-up conditions. No additional parameters are specified.
- #P = 1: With one parameter, only the octal data type (type 7) can be specified.
- #P = 2: With two parameters, a data type and an LSB/Range code can be specified. For data types, 1, 2, 3, and 6, a non-0 LSB must be specified. For data type 4, a range code must be specified. Data type 7, octal, may not be specifed with #P = 2 (since it doesn't have an LSB).
- #P = 3: With three parameters, a data type, LSB, and size are specified. This is processed similar to the #P = 2 except octal data type (7) can be specified, but its LSB value must be programmed = 0.
- #P = 4: With four parameters, the data type, LSB, size and limit are specified. Data types 4 and 7, timer-auto and octal, do not allow limits, so they cannot be specified with #P = 4.
- #P = -1 : Allows the user to specify the maximum rate of interrupts the system will respond to.
- #P = -2: Allows the user to specify the AC line frequency for the real time clock.
- #P = -5: Allows the user to format a defective I/O card specifying all five parameters (the four programmable parameters and the card identifier).

5-239 The data type, LSB, size, limit, and card identifications parameters are the actual values that determine how the Multiprogrammer will process the data it sends to or receives from a specific I/O card. Refer to Chapter 4, paragraphs 4-64 through 4-92 to determine what values to specify for the various parameters. The following is a brief review of the parameters.

5-240 Data Type Codes. These codes specify what conversion routines the data will be processed with. This parameter requires a code from 1 to 7, as follows:

Code	User Programs In	Card Data
1	*Decimal	2's Complement Binary
2	*Decimal	Sign/Magnitude Binary
3	*Decimal (pos. only)	Unsigned Binary
4	*Decimal + Range	Timer-Auto Range
5	Not Used	–
6	*Decimal (pos. only)	Unsigned BCD **
7	Octal integer	Unsigned Binary

*Fixed point decimal number with a maximum of 3 places to the right and 7 places to the left of the decimal point.

**Additional information on using the BCD Data Type is provided in paragraph 5-252.

5-241 LSB Value. When programming data types 1,2,3, and 6 (see list above), a non-zero (positive number between .001 card 65.535) LSB value must be specified. The octal data type (code 7) must not have an LSB value given (specify 0). The timer-auto range data type (code 4) must have a range code specified in place of an LSB value. The range codes are:

CODES	ENGINEERING UNITS
"1" or "U"	Microseconds
"2" or "M"	Milliseconds
"3" or "S"	Seconds

Additional information on using the Timer-Auto Range Data Type is provided in paragraph 5-248.

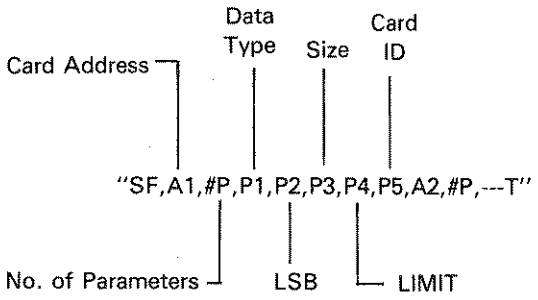
5-242 Size. Any card can be programmed to be a 12 or 16-bit card. The Multiprogrammer must know what size the card is so it can perform the correct data conversion. The only allowable values for this parameter are 12 or 16.

5-243 Limit. As described in Chapter 4, the Multiprogrammer supports the use of programmable limits. When limits are used, the absolute value of every number programmed is compared to the limit. If the programmed value is larger than the limit, a limit exceeded error is reported. The limit is a positive engineering units number that must be within the range allowed for the specified I/O cards' format. Limits cannot be set for cards using the timer auto range or the octal data types (types 4 or 7).

5-244 Processing Modes. The SF instruction is executed immediately. All output instructions given before the SF, even if they have not been executed yet, are processed with the old data format. All output instructions, given after the SF, are processed withing the new format. For input instructions, the numeric conversions are made as the data is read back to the controller, thus even though an input instruction may have completed before the SF was given, if the data is read back afterwards, it will come back in the new format. For these reasons, it is recommended that the SF instruction (and RF) be used as off line instructions, programmed either before processing begins, or after it completes and all the data has been read back. The SF instruction runs immediately in the serial or parallel mode. It also runs in the immediate mode.

5-245 Restrictions and Errors. The SF instruction does extensive error checking for any illegal or ambiguous commands. Only the main address of an I/O card can be formatted. Trying to format a subaddress (other than zero) will cause an error. If an error in the specification of a parameter is detected, it will be reported as a Set Format parameter error, -1640, followed by the card address where the error was made. When this error is detected, the card is always returned to its wake-up condition, just as if it was programmed with a #P = 0. Other errors, such as no card in slot (-1643), or unrecognizable card address (-1632), or illegal character detected (-1631) are also possible. Because the SF is executed immediately, when an error is found, all cards that were processed correctly before the error is encountered, will be correctly updated. The current card (if the error code is -1640), will be reformatted to its wake-up condition. All later cards will be left unchanged. For this reason, when an SF error is detected, it is **strongly** suggested that the entire SF command be retransmitted, with the errors corrected.

5-246 Sample Programs. The examples shown below illustrate using the SF instruction to change various I/O card data format parameters.



Example	Description
"SF,1,0T"	Card one is configured to default.
"SF,1,1,7,2,1,7T"	Cards one and two are configured to octal data code.
"SF,1,3,2,.5,12T"	Card 1 is configured to sign and magnitude (data type 2) with LSB = .5 and size = 12.
"SF,1,4,6,1,16,500 2,2,1,.001T"	Card one is configured to BCD (data type 6) with LSB = 1, size = 16-bit, limit = 500; Card two is configured to two's complement (data type 1) with LSB = .001.

5-247 Special Processing Formats. As stated previously when the #P parameter in an SF instruction has a value of -1, -2, or -5, special processing is indicated. #P = -1 and #P = -5, are advanced features that give the user additional control over the system that is not normally provided or required. The -2 and -5 values should be used with caution. The third value, #P = -2, is used when the system has difficulty determined the AC line frequency.

#P = -1 (Backplane interrupt Rate): Under very heavy system loading when many I/O cards have been armed and interrupts are coming in at a very high rate, it is possible for the system to spend so much time processing backplane interrupts that other processing hangs up. The firmware's backplane interrupt processor runs at a higher priority than the instruction parser. Because of this, under heavy loading, all available microprocessor time could be spent processing backplane interrupts and the current instruction never completes processing. The result is that additional instructions do not get executed. To avoid this situation, the maximum number of backplane interrupts recognized in a 100 millisecond period defaults to 300, which leaves approximately 20% of the microprocessor time available for other processing. In certain applications it may be desirable to change this value. The SF instruction with #P = -1 allows specifying a number between 0 and 400 which becomes the maximum number of interrupts processed in each 100 millisecond period. When using the SF with #P = -1, specify card address 0. The instruction terminator "T" must follow the variables specified.

Example: "SF,0,-1,nT"

where n = the number of interrupts

NOTE

A number greater than 350 allows the system to spend all its time processing backplane interrupts, thus making it possible to lockout all instruction processing. Programming 0 will turn off all backplane interrupts; however, everytime an instruction is started, backplane interrupts are turned on to let one interrupt through. Thus, an occasional interrupt will still occur.

#P = -2 (AC Line Frequency): When self test is invoked, it measures the ac line frequency to determine the time base for the real time clock. If the frequency measured is between 52 and 58Hz, a real time clock frequency error (-16) is reported, and the clock is set to 60Hz. The user can adjust the time base to 50Hz or 60Hz by issuing the SF instruction with #P = -2. The data type variables contain either a 50 or 60 to specify the time base. Card address 0 should be used, and the instruction terminator, "T" must follow the 50 or 60.

Example: "SF,0,-2,nT"

Where n is equal to 50 or 60:

Use the above sequence exactly

#P = -5 (Defective Card): Under certain conditions it may be desirable to try and communicate with a defective I/O card, or even a empty card slot. When an I/O card fails self test, the system will not allow any communication with the card. There are rare circumstances when the user might want to attempt to use the card, even though it is non-functional. (Use of a non-functional card is not recommended.) Another condition is communicating with an empty slot. A user might be attempting to develop software before all of the I/O cards have been received. He might like to simulate the I/O card by formatting an empty card slot. Formatting an empty slot allows the Multiprogrammer to perform dummy reads, writes, and cycles, however an empty slot can never generate an EOP. This feature is only recommended for the very advanced programmer.

The SF instruction with #P = -5 can be used to completely format a card, including the 4 programmable parameters, and the card identifier. Once a card (or an empty slot) has been formatted, the Multiprogrammer will process data for it according to the specifications given, and will not be able to differentiate it from a functional card. The #P = -5 parameter works just like the #P = 4 command, except a card identifier is also required. Data type 7 (octal), and 4 (timer) require the limit value be programmed to 0, while all other card types require that the limit variable be programmed to a non-zero value (a limit must be specified). If the timer auto-range type is used, the card identifier must indicate a timer card (card I/D = 1).

5-248 Using Timer Auto-Range Data Type. The timer auto range data type (Code 4) is designed to work with only one specific card, the 69736A Timer card. The auto ranging format is required on the timer card because the card's programmable range, form one microsecond to 65,535 seconds, is far greater than 16-bits of resolution. The timer card is actually programmed with two values, a 16-bit period value, and a 3-bit multiplier. The actual pulse width is determined by multiplying the period by the power of ten specified by the multiplier.

5-249 The timer auto-range data type shields the user from this calculation, by doing it automatically in the firmware. The only thing the user need be concerned with is the engineering units being used, microseconds, milliseconds, or seconds. The card wakes up being programmed in milliseconds, with the option of overridding that specification for any single output by specifing an "S", "M", or "U". The override only affects the one output, it doesn't change the default.

Example: "OP,4,23ST,OP,4,23UT,OP,4,23T"

The first output is 23 seconds, the second is 23 microseconds, and the third is the default of 23 milliseconds.

5-250 The default can be changed at any time with the SF command, specifying a data type of 4 (timer), and a Range Code. This changes the default until the card is reformatted again, or the system is reset.

Example: "SF,4,2,4,ST,OP,4,23ST,OP4,23UT,OP,4,23T"

The first output is 23 seconds, the second is 23 microseconds, and the third is the new default of 23 seconds.

5-251 The timer card does not have to be operated in the timer data type. An SF can be given that specifies another data type, in which case the timer is processed just like any other card. The timer can be returned to the timer data type by another SF specifying a data type of 4. Trying to program any other type of card to the timer data type will result in a format parameter error, -1640, followed by the card address.

5-252 Using the BCD Data Type. The Multiprogrammer interprets data for BCD formatted I/O cards as three or four decimal digits. Data for a 12 bit card is interpreted as 3 digits (12 ÷ 4) and data for a 16-bit card is interpreted as 4 digits (16 ÷ 4). See Appendix A for discussion of BCD. When processing data sent by the controller to an output card, the Multiprogrammer generates the BCD values; therefore, illegal BCD values will not be generated. However, when processing

BCD data generated by an external device which will be read by the controller, the Multiprogrammer has no control, so it is possible for illegal BCD Codes to be generated and read back.

2-253 Sometimes an external device such as a DVM may generate illegal BCD values intentionally. Certain types of DVM's use illegal BCD values to indicate overflow and other conditions. Because of this, when an illegal BCD value is received, the Multiprogrammer must inform the controller that an error was detected but it must also sends the data back because the illegal value may have been intentional.

5-254 When the Multiprogrammer detects an illegal BCD value, instead of performing the engineering units translation, it will return four alpha-numeric characters representing the hexadecimal value of the data inputted. Those digits that have valid BCD characters will return a decimal digit in that position. Those digits that have illegal BCD values will return a letter from A to F representing the following binary values:

A = 1010
B = 1011
C = 1100
D = 1101
E = 1110
F = 1111
Example: card 1 = 1111 0001 0010 0011
Value returned = F 1 2 3

Remember, engineering units translation (scaling) is not performed on the data, i.e., the four alpha numeric values are returned as shown above.

5-255 When an illegal BCD value is detected, beside returning the data in the above format, the Multiprogrammer will report an error (−5, illegal BCD value, followed by a second variable containing the card address). The determination of whether the value is legal or illegal is not made until the data is converted, which does not occur until the instant before the data is sent back to the controller. Thus, when a read is performed, the controller does not know whether the data will be read back as a number, or as a string of a alpha-numeric characters. If it is possible for the input device to return an illegal value, the controller should read the data back into a string. The following example illustrates using a string array store four BCD digits from a Digtial Input Card in slto 2. If errors are detected (red 72310,A,B/ENTER 723.10;A,B), the program branches to a subroutine labeled "illegal" to process any illegal BCD values. If no errors are detected, the numeric value of the string can be calculated.

9825 Controller

```
0: dim X$[4]
1: wrt 723,"IP,2T"
2: red 72301,X$
3: red 72310,A,B
4: if B#0;gto "illegal"
5: val(X$)→X
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM X$(4)
30   OUTPUT 723;"IP,2T"
40   ENTER 723.01;X$
50   ENTER 723.01;A,B
60   IF B<>0 THEN GOTO Illegal
70   X=VAL(X$)
```

5-256 SYSTEM TIMING INSTRUCTIONS

5-257 The 6942A Multiprogrammer contains a real-time clock that is synchronized to the ac line frequency. The clock has a resolution of 0.1 second and a maximum range of 65,534 days, 23 hours, 59 minutes, and 59.9 seconds (or approximately 179 years). The clock is always running, guaranteeing the accuracy provided by the ac line frequency, regardless of whatever else is happening in the system. Even under heavy loading or when the system is waiting for card(s) to complete, the Multiprogrammer will guarantee the accuracy. Consult your local power company to determine the accuracy of the power system in your area. The following paragraphs describe the instructions that affect or use the real time clock. These instructions are: Set Clock (SC), Read Clock (RC), Wait (WA), Wait Until (WU), and Clear Wait (CW). The four parameters (days, hours, minutes, and seconds) of the real time clock can be set with the SC instruction and read back with the RC instruction. When the Multiprogrammer is switched on, it is run through self test where the ac line frequency is measured to set the clock base frequency at 50 or 60 Hz. If the measured frequency is in the range of 52 Hz to 58Hz, a real-time clock frequency error (-16) is reported and the system defaults to 60Hz. The user can program the clock base frequency to 50 or 60Hz using the SF instruction as described in paragraph 5-247. After a self test, the clock is always reset to 0 days, 0 hours, 0 minutes, and 0.0 seconds.

5-258 Set Clock (SC) Instruction

5-259 The SC instruction can initialize the real time clock to any legal value. The syntax for the SC instruction is as follows:

"SC, days, hours, minutes, seconds, T"

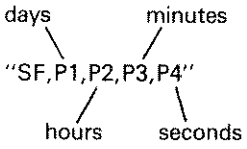
where the legal range for each parameter are:

Parameter	Range
days	0 to 65,534
hours	0 to 23
minutes	0 to 59
seconds	0 to 59.9

5-260 **Processing Modes.** The SC instruction is affected by the serial and parallel modes of operation. In the serial mode, it won't update the clock until all the preceding instructions have completed. In parallel mode, it is executed instantly. The SC instruction is not permitted in the immediate mode.

5-261 **Errors.** All four parameters must be specified in the SC instruction. If all are not specified or if any parameter is outside the legal range, an SC data error (-2934) will be reported. Note for this error, no card address is returned. The SC instruction is examined for standard errors, such as illegal character (-2931).

5-262 **Sample Programs.** The following examples illustrate using the SC instruction to initialize the real time clock to various legal values. Note that four parameters are specified in each example.



Example	Description
"SC,0,0,0,0T"	Resets time to 0.
"SC,1,12,30,1.0T"	Sets time to day 1, 12:30:01:0
"SC,37,4,26,42.5T"	Sets time to day 37, 04:26:42:5

5-263 Because the SC is affected by serial mode, it must wait for a previous sequence of instructions to complete before it can execute. The following example, initializes the real time clock to zero ("SC,0,0,0,0T") after one sequence of instructions has completed but before the second sequence is started.

9825 Controller

```
0: wrt 723,"IP,1,2T,OP,3,123T,IP,5,6T"
1: wrt 723,"SC,0,0,0,0T"
2: wrt 723,"IP,5,2T,OP,9,31T,IP,1T"
```

9835/9845 Controllers

```
10   OUTPUT 723;"IP,1,2T,OP,3,123T,IP,5,6T"
20   OUTPUT 723;"SC,0,0,0,0T"
30   OUTPUT 723;"IP,5,2T,OP,9,31T,IP,1T"
```

5-264 Read Clock (RC) Instruction

5-265 The RC instruction is used to read the real time clock. When it executes it saves the current values (days, hours, minutes, seconds) for readback at a later time on HP-IB extended talk address 14. The values saved are the values at the instant the RC was executed. The syntax for the RC instruction is as follows:

"RC" or "RCT"

Because no parameters are specified, the terminator "T" is not required, but it is allowed.

5-266 **Processing Modes.** The RC instruction is affected by the serial and parallel modes of instruction processing. In the serial mode, RC instruciton run sequentially and can be used to calculate time intervals of instruction sequences (see sample programs that follow). In the parallel mode, an RC executes instantly. An RC is permitted in the immediate mode.

5-267 **Errors.** The only error possible when programming an RC instruction is typing an incorrect opcode.

5-268 **Sample Programs.** The RC instruction reads back four variables from HP-IB extended talk address 14 (red 72314/ENTER723.14) in the following sequence.

Variable	Value
1	days (0 to 65,534)
2	hours (0 to 23)
3	minutes (0 to 59)
4	seconds (0.0 to 59.9)

The following program example illustrates using RC instructions to calculate the interval time of the instruction "IP,1,2T". In this example, the time is calculated by subtracting the time obtained in the second reading (variables E,F,G,H) from the time obtained in the first reading (variables A,B,C,D).

9825 Controller

```
0: wrt 723,"RC"
1: wrt 723,"IP,1,2T"
2: wrt 723,"RC"
3: red 72314,A,B,C,D
4: red 72314,E,F,G,H
```

9835/9845 Controllers

```
10   OUTPUT 723;"RC"
20   OUTPUT 723;"IP,1,2T"
30   OUTPUT 723;"RC"
40   ENTER 723.14;A,B,C,D
50   ENTER 723.14;E,F,G,H
```

5-269 The next example illustrates executing multiple RC's (in the serial mode) to take time interval measurements to determine how much time is spent performing each instruction in the sequence. Assume it won't take more than 1 hour for each instruction to complete. At the completion of the example, variable r1/R1 contains the time it took for the first OP to execute, and r3/R3 contains the IP execution time, and r3/R3 contains the time it took for the second OP to execute. The times are all in seconds with a resolution of 0.1 seconds.

9825 Controller

```
0: wrt 723,"RC,OP,1,123T"
1: wrt 723,"RC,IP,2T"
2: wrt 723,"RC,OP,3,456T,RC,IN,T"
3: if rds(723)#64 Jmp 0
4: red 72314,A,A,r1,r2
5: red 72314,A,A,r3,r4
6: red 72314,A,A,r5,r6
7: red 72314,A,A,r7,r8
8: r3*60+r4-(r1*60+r2)→r1
9: r5*60+r6-(r3*60+r4)→r2
10: r7*60+r8-(r5*60+r6)→r3
```

9835/9845 Controllers

```
10   OUTPUT 723;"RC,OP,1,123T"
20   OUTPUT 723;"RC,IP,2T"
30   OUTPUT 723;"RC,OP,3,456T,RC,IN"
40   STATUS 723;C
50   IF C<>64 THEN GOTO 40
60   ENTER 723.14;A,A,R1,R2
70   ENTER 723.14;A,A,R3,R4
80   ENTER 723.14;A,A,R5,R6
90   ENTER 723.14;A,A,R7,R8
100  R1=R3*60+R4-(R1*60+R2)
110  R2=R5*60+R6-(R3*60+R4)
120  R3=R7*60+R8-(R5*60+R6)
```

5-270 Readback Notes. The data from an RC instruction can be read back (red 72314/ENTER 723.14) immediately or at a later time. Either way, the RC readbacks should follow the same procedures outlined for the IP and IE instructions (see paragraphs 5-104 through 5-160). For delayed readbacks, the data is obtained in chronological order with each read statement returning the data from a single RC instruction. Because four variables are read back for each RC instruction, all four should be specified in a single read statement

5-271 Wait (WA) Instruction

5-272 The WA instruction allows the controller to execute a programmable pause between two instruction sequences. The syntax for the WA instruction is as follows:

"WA,nt"

where n is between 0.0 seconds and 6,553.5 seconds.

5-273 Processing Modes. The execution of a WA instruction is dependent upon the processing (serial or parallel) mode that is in affect. Once the instruction is started, it continues to execute until the programmed wait time elapses. WA is not permitted in the immediate mode.

5-274 In the serial mode, the WA instruction provides a pause between instructions. While it is executing, the other instructions can not run. In the following example, the WA causes a 10.3 second pause after the IP completes and before the OP is started. The WA's resolution as with all instructions that use the real time clock, is 0.1 seconds.

Example: "IP,IT,WA,10.3T,OP,1,123T"

5-275 In the parallel mode, the WA will execute immediately if there are no other WA instructions executing. If multiple WA's are active in the system, they will execute one-at-a time in sequence, the same as other like instructions (except OI ad II) in the parallel mode. Because unlike instructions run in parallel, the WA cannot stop other instruction from starting. At the completion of the instruction, SRQ is set to inform the controller that the WA completed. Thus, in the parallel mode, WA acts as a programmable timer, setting SRQ upon completion.

5-276 Errors. Specifying a value outside the legal range (0.0 to 6,553.5 secs) will generate error code -1034. No card address is specified with this error. The WA is also tested for standard errors such as illegal characters or forgetting to specify the "T".

5-277 Sample Programs. In the following example, a WA is used to make the program pause for 10 seconds between a stimulus and the associated response. In the example the OP instruction programs the stimulus and the IP instruction is used to take the reading. An IN instruction sets SRQ at the completion of the sequence to inform the controller.

9825 Controller

```
0: prt 723;"OP,1,-2.35T,WA,10T,IP,2T,INT"
1: if rds(723)#64;jmp 0
2: red 72301,A
```

9835/9845 Controllers

```
10   OUTPUT 723;"OP,1,-2.35T,WA,10T,IP,2T,IN,T"
20   STATUS 723;C
30   IF C<>64 THEN GOTO 20
40   ENTER 723.01;A
```

5-278 Wait Until (WU) Instruction

5-279 THe WU instruction provides the capability of starting a sequence of instructions at a programmable time. The syntax for the WU instruction is as follows:

"WU, [days,] [hours,] [minutes,] seconds T"

Where the legal ranges for each parameter are:

Parameter	Range
days	0 to 65,534
hours	0 to 23
minutes	0 to 59
seconds	0.0 to 59.9

5-280 All four parameters need not be specified. The parameters are evaluated by the Multiprogrammer from right to left (seconds to days) and any missing parameters default to the current setting of the real time clock. If only two variables are sent, the WU defaults to the current day and hour, and the two variables specify the minutes and seconds.

Example	Description
"WU,1,23T"	Wait until today, this hour, one minute and 23 seconds.
"WU,37.2T"	Wait until today, this hour, this minute, and 37.2 seconds.

5-281 Processing Modes. The exeuction of the WU instruction also depends upon the processing modes (serial or parallel) that is in affect. WU is not permitted in the immediate mode. In the serial mode, no other instructions will start executing until the WU completes. Thus, it can be used to

schedule a sequence of instructions to run at a later time. In the following example, the IP instruction will start executing exactly at midnight on day number 1.

"WU,1,0,0,0T,IP,1,2T,OP,3,456T"

5-282 In the next example (serial mode in effect), the IP is synchronized to start running at completion of the current minute.

"WU,59.9T,IP,1,2T"

5-283 Note that "WU,0T" or "WU,1,0T" will not synchronize to the nearest minute. "WU,0T" will always complete instantly, and "WU,1,0T" will complete the instant the minutes value is greater or equal to 1 (will complete instantly 59 minutes out of an hour).

5-284 In the parallel mode, the WU cannot inhibit other instructions from starting and it will set SRQ upon completion. Thus, WU acts as an alarm clock setting SRQ upon completion. Only one WU can be executing at a time in the parallel mode. In the example below, the WU instruction will complete at midnight on day one. The exeuction of OP and IP will not be delayed, the WU will run normally.

"WU,1,0,0,0T,OP,3,456T,IP,1,2T"

5-285 Errors. Specifying an illegal value for any of the parameters will cause a WU data error, -1134. No card address is specified with this error. The WU is also tested for standard errors such as illegal character or a missing terminator.

5-286 Clear Wait (CW) Instruction

5-287 The CW instruction clears all WU or WA instructions from the system. This instruction could be used if WA or WU instructions with extremely long delays are inadvertently programmed. In the serial mode, this would cause subsequent instructions to wait for the WA or WU to complete before the sequence can continue. The syntax for the CW is as follows:

"CW" or "CWT"

The terminator "T" is not required but it is accepted.

5-288 Processing Modes. The CW executes instantly in all modes: serial, parallel, and immediate. When executed, the CW erases all WA and WU instructions active in the system. This includes those currently running and those that are awaiting execution. For this reason, the CW should be used with caution since **all** WA's and WU's, not just the one that caused the problem, are erased.

5-289 Errors. The only error possible when programming a CW instruction is typing in an incorrect opcode.

Chapter 6
SPECIAL PURPOSE PROGRAMMING INFORMATION

6-1 This chapter provides special purpose programming information that will extend your programming capability beyond that provided by the basic programming information in Chapter 5. The topics and the associated instructions covered in this Chapter include the following:

Topics	Instructions
Group Instructions	Clear Group (CG)
Armed Card Interrupts	Arm Card (AC) Disarm Card (DC)
I/O Card Status	Read Status (RS)
Clearing I/O Cards	Clear Card (CC)
Disabling and Enabling I/O Cards	System Disable (SD) System Enable (SE)
Immediate Mode	Go Immediate (GI) Go Normal (GN)
Memory Instructions	Memory Output (MO) Memory Input (MI)
Multiprogrammer Memory Utilization	—

NOTE

As in Chapter 5, all 9835/9845 examples using arrays assume that the controller is operating in OPTION BASE 1. If you prefer to use OPTION BASE 0, you must subtract 1 from any dimension (DIM) or redimension (REDIM) statements included in 9835/9845 programming examples given in this Chapter.

6-2 GROUP INSTRUCTIONS

6-3 Group instructions can be used to minimize the number of characters sent over the HP-IB and decrease instruction execution time for applications that require executing identical instructions numerous times. When group instructions are used, the original instruction is tagged with a "group number". To re-execute the instruction, you need only specify the opcode, group number, and a terminator. For example, assume that data readings must be taken periodically by 35 input cards. If a group instruction is used you will only have to send the entire instruction containing the card addresses once. Each additional time the readings are required, you will only have to specify the opcode and group number. Group instructions can be used with most of the input and output instructions available with the Multiprogrammer. Group instructions are permitted in the serial and parallel modes but are not allowed in the immediate mode. The following paragraphs describe how to use groups, what the limitations are, and what the advantages are in terms of execution speed and memory utilization.

6-4 As mentioned above, not all instructions can be used as groups. The following tabulation lists the instructions that are and are not permitted. Attempting to define a group using an instruction that is not permitted or a group number other than 0-9 will cause the Multiprogrammer to generate an error message and set SRQ. See instruction error codes -37 and -38 in Appendix B.

Group Instructions

Permitted	Not permitted
AC	CG
CC	CW
CY	GI
DC	GN
IE	GP
II	GS
IP	IN
OI	MI
OP	MO
OS	OB
RS	RC
RV	RF
WC	SC
WF	SD
	SE
	SF
	WA
	WU

6-5 Defining a Group Instruction

6-6 An instruction can be defined as a group instruction by specifying a group number immediately after the opcode the first time the instruction is programmed. Up to 10 group instructions (0-9) are permitted at any given time and are specified as G0-G9) when the group is initially defined. A delimiter (comma or space) is always required after the group number. Example 6-1 illustrates defining an "OP" and an "IP"

as group instructions. Executing example 6-1 will define the "OP" instruction as group number 1 and program output cards in slots 4 and 5 with a data value of 123. The IP instruction will be defined as group number 9 and 100 readings (repeat factor "R100") each will be taken by input cards in slots 12,13, and 14.

Example 6-1. Defining Group Instructions

9825A Controller

```
0: wrt 723,"OPG1,4,123,5,123T"
1: wrt 723,"IPG9,R100,12,13,14T"
```

9835/45 Controllers

```
10 OUTPUT 723;"OPG1,4,123,5,123T"
20 OUTPUT 723;"IPG9,R100,12,13,14T"
```

6-7 Using Group Instructions

6-8 Once an instruction has been defined as a group instruction, it can be rerun by specifying the instruction opcode, the group number receded by a "U", and a terminator. The opcode used to rerun a group instruction must correspond to the opcode used when defining the group instruction or the Multiprogrammer will generate an error message and set SRQ. Example 6-2 illustrates rerunning the group instructions defined in Example 6-1.

Example 6-2. Rerunning Group Instructions

9825 Controller

```
50: wrt 723,"OPU1T,IPU9T"
```

9835/45 Controllers

```
50  OUTPUT 723;"OPU1T,IPU9T"
```

6-9 Clear Group (CG) Instruction

6-10 When a group instruction is no longer required, it can be cleared out of the Multiprogrammer with a Clear Group (CG) instruction. Since a group instruction (even when it is not active ties up a portion of system memory, it is a good idea to clear out group instructions when they are no longer needed. The syntax for the CG instruction is as follows:

"CG,N_1,N_2,N_3,----T"

where N_1,N_2,N_3 are the group numbers to be cleared.

6-11 As indicated above, the CG instruction consists of an opcode, the number(s) of the group(s) to be cleared, and a terminator. CG instructions always execute immediately regardless of whether the Multiprogrammer is in serial or parallel mode. If any group instructions are currently active in the system either executing or waiting to execute, they will not be affected by the clear group instruction except that they will in effect become standard (non-group) instructions. In any case the CG instruction will clear out any group definitions specified in the instruction. Example 6-3 illustrates clearing group instruction numbers 0 and 5.

Example 6-3. Clearing Group Instructions

9825A Controller

```
100: wrt 723,"CG,0,5T"
```

9835/45 Controllers

```
100  OUTPUT 723;"CG,0,5T"
```

6-12 Redefining Group Instructions

6-13 If a group definition is no longer needed but there is another instruction that must be defined as a group, it is not necessary to clear the old group definition before defining the new group. This operation can be accomplished in one step by simply redefining the group. When a group is redefined, the old instruction associated with the group number is replaced with a new one. The new instruction need not be related to the old instruction in any way. Redefining a group is accomplished in exactly the same manner as the initial definition, using the new opcode, card addresses, and data. Example 6-4 illustrates redefining group number 9, previously defined in Example 6-1 as an "IP" instruction using cards in slots 12,13, and 14, to a "WC" instruction which sends data (777) to cards in slots 0 and 1.

Example 6-4. Redefining a Group Instruction

9825A Controller

```
200: wrt 723,"WCG9,0,777,1,777T"
```

9835/45 Controllers

```
200  OUTPUT 723;"WCG9,0,777,1,777T"
```

6-14 Instruction Execution Time and Memory Utilization

6-15 Without using group instructions, every instruction in the system requires a block of memory, called a context block, during the period that the instruction is active (i.e. waiting to execute, executing, or an input instruction that has completed execution but whose data has not been read back by the controller). Normally, the memory reserved for an instruction is released to the system after the instruction goes inactive. When an instruction is defined as a group instruction however, the context block is saved until the group definition is cleared or redefined. Saving a context block containing all the information required to run an instruction increases the speed of the instruction since the system is not required to allocate memory and decode addresses and data sent by the controller before rerunning an instruction.

6-16 In terms of memory utilization, each group instruction reduces the amount of free memory available by an amount equal to what the instruction would require if it were active but not a group instruction. This of course will vary with the size and type of the instruction. As a few examples, an "OP" instruction using 5 cards would require approximately 27 words, and an "IP" using 2 cards and a repeat factor of 10 would require approximately 41 words. Depending on the number of frames in the system, a total of approximately 1300-1400 words of memory are available in the system (see paragraph 6-80).

6-17 ARMED CARD INTERRUPTS

6-18 For certain application's, the cycling of I/O cards will be controlled externally by the user via the EXTERNAL TRIGGER input on the card's edge connector. Other applications may require that the cycling be accomplished internally with the Cycle (CY) or Write and Cycle (WC) "low level" instructions. The application may also require that the controller be informed via the SRQ line when each I/O card is cycled and completes data processing. This can be accomplished if the I/O cards are enabled prior to being cycled. The Arm Card (AC) instruction can be used to enable the I/O cards that will be eventually cycled by an external trigger, a CY instruction, or a WC instruction. As each armed card is cycled and completes processing its data (generates an EOP), it will interrupt the microprocessor and set SRQ. This type of service request is referred to as an "armed card interrupt". Since an understanding of the Multiprogrammer interrupt system as it pertains to I/O cards is required to enable you to visualize the use of armed card interrupts, a brief review follows.

6-19 Multiprogrammer Interrupt System

6-20 Whenever any Multiprogrammer I/O card is cycled and completes its data processing operations, an end-of-process (EOP) signal is generated. If the card is also armed when it is cycled, the resulting EOP signal will generater an interrupt to the microprocessor. Note that all "high level" instructions (OB,IP,OP,II,OS,IE, and OI) automatically arm I/O card(s), consequently each card addressed in the instruction will generate a microprocessor interrupt when it completes. When the micropocessor receives the interrupt, it will determine the instruction that the card is assgned to, disarm the card, and continue the processing according to the type of instruction and the processing mode that is in effect. For example, if the interrupting card is the only card specified in an OS instruction and the serial mode is in effect, the microprocessor will allow the next instruction in the sequence to start. If the parallel mode is in effect and the interrupting card is the only card specified in an OS instruction, the microprocessor will set both the SRQ line and the appropriate Multiprogrammer SRQ status bit.

6-21 Low level instructions however, do not arm I/O cards when they are executed. The WC is a low level instruction that sends data to the card(s) then cycles the card(s). The CY is another low level instruction that simply cycles the card(s). As mentioned previously, I/O cards can also be cycled by applying an external trigger input on the card. The effect of cycling a card will depend on the type of card, but in general cycling an output card will transfer data from first rank storage on the card to the output of the card. Cycling an input card will cause the card to read in and store external data on the card. Although the card will process the data and generate EOP when it completes, since the card is not armed, it will not interrupt the microprocessor and the controller will not be notified that the card has completed. In order to use the microprocessor interrupt system and notify the controller, these I/O cards must be "armed" before they are cycled.

6-22 Arm Card (AC) Instruction

6-23 The AC instruction can be used to enable cards that are being externally triggered or programmed with a "WC" or "CY" instruction to interrupt the microprocessor upon completion of their data processing. The AC instruction will run sequentially with other instructions when the Multiprogrammer is in serial mode or concurrently with other instructions when the Multiprogrammer is in parallel mode. It can also run in the immediate mode. The syntax for the AC instruction is as follows:

"AC,A1,A2,A3,---T"

Card addresses are specified by A1,A2,A3,etc.

Example 6-5 illustrates use of an AC instruction to arm the cards in slots 0 through 5. An armed card interrupt will be generated when each of the cards programmed in this example are cycled and complete (generate EOP). When an armed card interrupt is detected, the third Multiprogrammer SRQ status word (variable C, see Table 5-3) will be incremented and SRQ will be set. Thus, the number of cards that have generated armed card interrupts is stored in the third variable read back from HP-IB extended talk address 10 (red 72310,A,B,C,D,E,F/ENTER 723.10;A,B,C,D,E,F). Refer to the Multiprogrammer SRQ status discussion in paragraphs 5-172 through 5-181.

Example 6-5. Arming Cards

9825A Controller

```
0: wrt 723,"AC,0,1,2,3,4,5T"
```

9835/45 Controllers

```
10   OUTPUT 723;"AC,0,1,2,3,4,5T"
```

6-24 Armed Card Interrupt List Readback

6-25 The list of cards that generated armed card interrupts can be read back from HP-IB extended talk address 12 (red 72312/ENTER 723.12). As mentioned previously, the third variable read back from HP-IB extended talk address (red 72310/ENTER 723.10) indicates the number of cards that interrupted. When more than 10 armed card interrupts are read back, it is suggested that an array be used. The following paragraphs describe reading back armed card interrupts on the applicable controller.

6-26 9825 Controller. If you read back fewer variables than were specified by the third Multiprogrammer SRQ status variable or additional card interrupts occur before reading extended talk address 12, the Multiprogrammer will update the third SRQ status variable and set SRQ again. If you attempt to read more variables than were specified by the third SRQ status variable (from extended address 10), the Multiprogrammer will respond to the additional requests for data by sending carriage return/line feeds which will not change the state of the additional variables requested. The procedures described in paragraph 5-199 for reading back data on the 9825 Controller for the OI instruction should also be followed when reading back the armed card interrupt list.

6-27 9835/9845 Controllers. When using a 9835 or 9845 controller, the third Multiprogrammer SRQ status variable read back from extended talk address 10 can only be used as an indication that armed card interrupts have occurred. If the value of the third SRQ status variable is used to determine how many variables to read from extended talk address 12, it becomes possible to miss interrupts that occurred after the status variable was read. If your program requests less data than is available, due to these additional card interrupts, the controller will throw out the additional data available. For this reason, you should always assume that all cards have completed and read the entire array each time armed card interrutps are detected. By presetting all variables in the array to −1 before reading the array, you will then be able to inspect the contents of the array to determine which cards have interrupted. Positive values in the array after reading extended talk address 12 are the addresses of interrupting cards.

6-28 Attempting to read an entire array of data into a 9835 or 9845 controller from extended talk address 12 will frequently result in a controller error 159 since the Multiprogrammer will not always have enough data available to fill the array. The error trapping subroutine, described in Appendix C, allows the controller to attempt to read more data than is available without the resulting error stopping your program. Assuming that the array has been preset to negative values it can then be inspected to determine the address of the interrupting card. The procedures described in paragraph 5-199 for reading back data on the 9835/9845 controllers for the OI instruction should also be followed when reading back the armed card interrupt list.

6-29 Sample Program. Example 6-6 illustrates arming cards in slots 0,1,2,3,4, and 5 to generate armed card interrupts. This example assumes the cards in slots 0,2, and 4 are Digital Output cards and the cards in slots 1,3, and 5 are Digital Input cards. All cards will be externally triggered. Remember, cycling the cards with a "CY" instruction would have the same results as an external trigger.

6-30 The program in Example 6-6 includes a serial poll, rds (723)/STATUS 723, to determine if the Multiprogrammer set SRQ. If the Multiprogrammer requested service, extended talk address 10 is read (red 72310,A,B,C/ENTER 723.10;A,B,C) and the third variable is checked (variable C) to determine the number of cards that interrupted. The program then reads the addresses of the cards into an array. Refer to the line-by-line description that follows the program listings.

6-31 It is very important to read back the addresses of the interrupting cards when armed card interrupts are detected. If the address of an interrupting card is not read by the controller, subsequent interrupts from the same card will not cause the Multiprogrammer to increment the third status variable or set SRQ. An important point to keep in mind is that the third Multiprogrammer SRQ status variable contains the number of cards that have generated armed card interrupts. It does not contain the number of interrupts that have been generated since one card can interrupt many times.

6-32 As mentioned previously, cards that are assigned to instructions that automatically arm the card also automatically disarm the card once the card has completed data processing and interrupted. Cards that are armed by an arm card instruction and cycled however, are not disarmed after they generate an interrupt. EOP is reset (cleared) and they will continue to interrupt the microprocessor when they are cycled and complete. As long as the addresses of the interrupting cards are being read back by the controller SRQ will be set when one of these cards interrupts.

Example 6-6. Reading the Armed Card Interrupt List

9825A Controller

```
0: dim A[6]
1: red 72310,A,B
2: oni 7,"multi"⁝eir 7
3: wrt 723,"WF,0,123,2,123,4,123T"
4: wrt 723,"AC,0,1,2,3,4,5T"
                    •
                    •
                    •
100: "multi"⁝if rds(723)#64⁝gto "other"
101: red 72310,A,B,C
102: if C=0⁝gto "iret"
103: for J=1 to C⁝red 72312,A[J]⁝next J
104: for J=1 to C
105: if A[J]=0⁝gsb "slot0"
106: if A[J]=1⁝gsb "slot1"
107: if A[J]=2⁝gsb "slot2"
108: if A[J]=3⁝gsb "slot3"
109: if A[J]=4⁝gsb "slot4"
110: if A[J]=5⁝gsb "slot5"
111: next J
112: "iret"⁝eir 7⁝iret
```

9835/35 Controllers

```
10   OPTION BASE 1
20   DIM A(6)
30   ON ERROR GOSUB Ertrap
40   ENTER 723.10;A,B
50   ON INT #7 GOSUB Multi
60   CONTROL MASK 7;128
70   CARD ENABLE 7
80   OUTPUT 723;"WF,0,123,2,123,4,123T"
90   OUTPUT 723;"AC,0,1,2,3,4,5T"
                    •
                    •
                    •
1000 Multi:STATUS 723;C
1010 IF C<>64 THEN Other
1020 ENTER 723.10;A,B,C
1030 IF C=0 THEN Return
1040 A(1)=A(2)=A(3)=A(4)=A(5)=A(6)=-1
1050 ENTER 723.12;A(*)
1060 FOR J=1 TO 6
1070 IF A(J)=0 THEN GOSUB Slot0
1080 IF A(J)=1 THEN GOSUB Slot1
1090 IF A(J)=2 THEN GOSUB Slot2
1100 IF A(J)=3 THEN GOSUB Slot3
1110 IF A(J)=4 THEN GOSUB Slot4
1120 IF A(J)=5 THEN GOSUB Slot5
1130 NEXT J
1140 CARD ENABLE 7
1150 Return:RETURN
                    •
                    •
                    •
8000 Ertrap:IF ERRN=159 THEN RETURN
8010 PRINT ERRM$
8020 STOP
```

Explanation:

9825	9835/45	Functional Description
0	10,20	Dimension a 6 word array for reading armed card interrupts.
	30	Establish linkage to error processing subroutine to allow handling of variable length read-backs.
1	40	Read Multiprogrammer status and clear any outstanding SRQ.
2	50-70	Establish interrupt linkage between HP-IB interface 7 and subroutine labeled "multi"; enable interface 7 for interrupts.
3	80	Write first rank data to output cards.
4	90	Arm cards in slots 0-5.
100	1000, 1010	When an HP-IB SRQ is detected the controller branches to a subroutine named "multi". If a serial poll of the Multiprogrammer determines that the Multiprogrammer was not the device that set SRQ the program branches to another routine where it polls other devices to determine which device set SRQ.
101	1020	If the serial poll determines that the Multiprogrammer was the device that set SRQ the Multiprogrammer status variables are read into controller variables A,B, and C. This also clears the SRQ status bit.
102	1030	If armed card interrupts had not set SRQ the controller returns execution to the main program. Normally, further processing would be done to find out why the Multiprogrammer had set SRQ.

103	1050	The armed card interrupt list is read into array "A".
104	1060	For/next loop is set up to determine which card interrupted and proper processing path to take.
105-110	1070-1120	Tests are made to determine which card(s) interrupted. When controller determines card that interrupted program branches to appropriate subroutine for processing (For example, if an input card interrupted, you may want to use an "RV" instruction to read the card data into the controller).
111	1130	Go and read next address from interrupt list.
112	1140, 1150	Renable HP-IB interface for further interrupts and return.
	8000, 8020	Subroutine to allow variable length readbacks on 9835 and 9845 controllers.

6-33 Disarm Card (DC) Instruction

6-34 The DC instruction can be used to disarm any card in the system, thereby preventing the card from generating a microprocessor interrupt when it is cycled and completes. Although it is intended to complement the arm card instruction, it can also be used to disarm a card programmed by any instruction that normally arms the card. If it is used to disarm a busy card that has been programmed by an instruction that normally arms the card, it will appear as if the card has completed unless the instruction is an "OI" or "II" instruction, in which case, the card will appear to have been removed from the instruction. Use of the DC instruction on cards that have been programmed by "IP" or "IE" instructions that use repeat factors is not recommended. The DC instruction will run sequentially with other instructions when the Multiprogrammer is in the serial mode or concurrently with other instructions when the Multiprogrammer is in the parallel mode. It can also run in the immediate mode. The syntax for the DC instruction is as follows:

"DC,A1,A2,A3,---T"

Card addresses are specifed by A1,A2,A3, etc.

6-35 Example 6-7 illustrates using a DC instruction to disarm the cards in slots 0 through 5. Once disarmed, the cards are prevented from generating a microprocessor interrupt when they are cycled by an external trigger, a CY instruction, or a WC instruction.

Example 6-7. Disarming Cards

9825A Controller

```
200: wrt 723,"DC,0,1,2,3,4,5T"
```

835/45 Controllers

```
200  OUTPUT 723;"DC,0,1,2,3,4,5T"
```

6-36 I/O CARD STATUS

6-37 Knowing the status of an I/O card can be an effective aid in isolating programming or hardware problems that may occur in your system by allowing you to determine which I/O cards in the system are busy. The following coded values indicate whether the card is armed, busy, or has completed processing and generated EOP.

Coded Value	Card Status
0	inactive
1	EOP set
2	Armed
4	Busy

6-38 Card status may appear as any single value or combinations of the values listed above. For example, coded value 6 indicates that the card is armed and is busy. If the card is currently active in high level instruction (OB,IP,OP,II,OS,IE, or OI) that requires an interrupt from the card to complete, the status also includes a coded value representing the instruction that is being used to program the card. The following is a list of coded values of high level instructions that require an interrupt to complete.

Coded Value	Instruction Opcode
100	OB
200	IP
300	OP
400	II
500	OS
600	IE
900	OI

6-39 Read Status (RS) Instruction

6-40 The RS instruction can be used to obtain the current status of an I/O card at any time. It will always execute immediately regardless of whether the Multiprogrammer is operating in the serial or parallel mode. When a read status instruction is executed, the status of the card will be added to a coded instruction value if applicable and stored in the Multiprogrammer. The status of the I/O card may then be read into a controller variable by using HP-IB extended talk address 08 (red 72308/ENTER 723.08). If a second read status instruction is sent to the Multiprogrammer before the data from the first instruction has been read back, data from the first instruction will be thrown away. The syntax for the RS instruction is as follows:

"RS,A1,A2,A3,----T"
etc.
where A1,A2,A3, etc. are the I/O card addresses.

6-41 Sample Program

6-42 Example 6-8 illustrates reading the status of I/O cards in slots 5 and 6 into controller variables A and B. Assume that variables "A" and "B" contained values of 206 and 1, respectively. This would indicate that the card in slot 5 was programmed by an "IP" instruction and is armed and busy. The card in slot 6 is not assigned to any instruction, but EOP is set indicating that an unarmed card was cycled and has completed processing data. The card in slot 6 could have been programmed by a "WC" or "CY" instruction or it could have been externally triggered. If a card is cycled (WC, CY, or external trigger) without first being armed, the EOP signal is set indicating that the card completed its operations. For these conditions, EOP will not be reset until the card is disarmed or a high level instruction addressing the card is programmed.

Example 6-8. Reading Status of I/O Cards

9825A Controller

```
0: wrt 723,"RS,5,6T"
1: red 72308,A,B
```

9835/45 Controllers

```
10   OUTPUT 723;"RS,5,6T"
20   ENTER 723.08;A,B
```

6-43 CLEARING I/O CARDS

6-44 It is possible through improper programming or improper or open connections to the gate/flag lines on some I/O cards to program an I/O card into a busy state under conditions that will either result in the card taking a long time to complete or not completing at all. If the card has been programmed by an instruction which requires card interrupt to allow the instruction to complete, other than an OI or II instruction, the instruction will remain busy and unavailable for programming other cards until the busy card programmed by the instruction completes. If the Multiprogrammer is in serial mode all subsequent instructions, with the exception of instructions that run immediately, will be forced to wait for the busy instruction to complete before they are allowed to run. The clear card instruction (CC) is provided to allow you to terminate the busy state of the card, and therefore the instruction, if desired.

6-45 Clear Card (CC) Instruction

6-46 The CC instruction can be used to clear the arm, busy (timing), and EOP circuits on any I/O card. It will always execute immediately regardless of whether the Multiprogrammer is operating in the serial or parallel mode. The syntax for the CC instruction is as follows:

"CC,A1,A2,A3,---T"

where A1,A2,A3, etc. are card addresses.

6-47 Only the OB, IP, OP, OS, IE, OI, and II instructions are affected by the CC instruction. The Clearing a card affects the instruction that has programmed the card as follows:

1. If a clear card instruction is used to clear a busy card that has been programmed by an OB, IP, OP, OS, or IE instruction, it will appear as if the card has completed normaly. Data from an IP or IE instruction, that contains a card that has been cleared, must still be read back, although data from the card that has been cleared should then be discarded. As a few examples, clearing the only card that is busy in an OP instruction will complete the instruction, thereby allowing subsequent instructions to run if the Multiprogrammer is in serial mode or setting SRQ and the appropriate status bit if the Multiprogrammer is in parallel mode. Clearing the only card that is busy in an IP or IE instruction will have the same effect on the Multiprogrammer and data may then be read back from the appropriate extended talk address.

NOTE

Clearing a card programmed by an "IP" or "IE" instruction that uses a repeat factor and possibly a "wait" can be complicated. To clear the card from an instruction of this type, it must be programmed by a number of clear card instructions equal to the repeat factor. The time between sending each clear card instruction to the card must be greater than the longest time required for each set of readings to be taken by all the cards in the instruction including a "wait" if it is specified.

2. If a clear card instruction is used to clear a busy card that has been programmed by an "OI" or "II" instruction, it will appear as if the card has been completely removed from the instruction. SRQ will not be set. If the card is part of a group instruction it will remain in the group and will be reprogrammed whenever the group instruction is reprogrammed.

6-48 Sample Program

6-49 Example 6-9 illustrates clearing I/O cards installed in slots 5 and 6. The read status (RS) instruction discussed previously can be used to verify that I/O cards have been cleared.

Example 6-9. Clearing I/O Cards

9825A Controller

```
0: wrt 723,"CC,5,6T"
```

9835/45 Controllers

```
10   OUTPUT 723;"CC,5,6T"
```

6-50 IMMEDIATE MODE

6-51 The immediate mode of operation is provided for use whenever an immediate response to an emergency situation is required. Upon entering the immediate mode the system suspends all currently active instructions and allows selected instructions to run immediately. Attempting to execute an instruction that is not permitted in immediate mode will result in an error being reported. Upon returning to the normal mode of operation the system will be restored to the same conditions that existed prior to entry into the immediate mode. The following tabulation lists the instructions that are and are not permitted in the immediate mode.

Immediate Mode Instruction Summary

Permitted*	Not Permitted***
AC	CG**
CC**	GP
CY	GS
CW**	IE
DC	IN
GI**	IP
GN**	MI**
II	MO**
OI	OB
RC	OP
RF**	OS
RS**	SC
RV	WA
SD	WU
SE	
SF**	
WC	
WF	

* Group instructions (see paragraph 6-2) are not permitted in the immediate mode.

**These instructions execute instantly.

***Attempting to execute an instruction that is not permitted in the immediate mode will result in error code -3 being reported.

6-52 Go Immediate (GI) Instruction

6-53 The GI instruction is used to program the Multiprogrammer to the immediate mode of operation. An example of using the immediate mode to respond to an alarm condition is given in paragraph 5-59. The syntax for the GI instruction is as follows:

"GI" or "GIT"

6-54 Since only the opcode is sent, the terminator "T" is not required but will be accepted if it is sent. Example 6-10 illustrates programming the immediate mode.

Example 6-10. Programming the Immediate Mode

9825A Controller

```
0: wrt 723,"GI"
```

9835/45 Controllers

```
10   OUTPUT 723;"GI"
```

6-55 While operating in the immediate mode, the backplane interrupt system is turned off preventing I/O cards from interrupting the microprocessor. However, if an II, OI, CC, or DC instruction is programmed, the backplane interrupt system is temporarily turned on. This will allow any cards which have interrupts pending (arm and EOP set) to interrupt the microprocessor. Suspended instructions that were waiting on these cards to complete will be allowed to complete normally, setting SRQ if the Multiprogrammer is in parallel mode or the interrupting card had been programmed by an II or OI instruction.

6-56 Restrictions

6-57 Although cards will be permitted to interrupt and suspended instruction will be permitted to complete when II, OI, CC, or DC instructions are programmed in immediate mode, data from IP, II, IE, and OI instructions can not be read back from their associated HP-IB extended talk addresses until the Multiprogrammer is returned to normal (serial or parallel) mode. Data from RF, RV, and RS instructions, programmed in the immediate mode, can be read back on the appropriate extended talk address; however, the data must be read immediately. If a second instruction of the same type is executed before data from the first instruction is read back, the first instruction's data is thrown away. Data from these instructions, stored in the Multiprogrammer before the Multiprogrammer was programmed to immediate mode, can not be read back until the normal mode of operation is restored. The data readback restrictions in the immediate mode are summarized below.

Data Readback from HP-IB
Extended Talk Addresses in Immediate Mode

Permitted	Not Permitted**
04 (RF)*	01 (IP)
05 (MI)**	02 (II)
06 (RV)*	03 (IE)
08 (RS)*	09 (OI)
10 (SRQ STATUS)**	
11 (ERROR LIST)**	
12 (ARMED CARD INT. LIST)**	
13 (BUSY INST.)**	
14 (RC) **	

*Data may only be read back from these extended talk addresses in immediate mode if the corresponding instruction has been programmed while in immediate mode.

**Data from these extended talk addresses is unaffected by immediate mode of operation.

***If an attempt is made to read data from an extended talk address that is not permitted the Multiprogrammer will only send back a carriage return/line feed.

6-58 Go Normal (GN) Instruction

6-59 The GN instruction is used to return the Multiprogrammer back to the mode (serial or parallel) of operation that was in effect before the GI was programmed. The syntax for the GN instruction is as follows:

"GN" or "GNT"

6-60 Since only the opcode is sent, the terminator "T" is not required but will be accepted if it is sent. Example 6-11 illustrates programming the GN instruction. When the Multiprogrammer is returned to normal mode, the following actons are taken:

1. All data taken in RV,RS, or RF instructions in the immediate mode, that has not been read back, is thrown away.
2. All suspended instructions are allowed to continue.
3. Backplane interrupts are turned back on.

Example 6-11. Programming the GN Instruction

9825A Controller

```
0: wrt 723,"GN"
```

9835/45 Controllers

```
10   OUTPUT 723;"GN"
```

6-61 DISABLING AND ENABLING OUTPUT CARDS

6-62 When a 6942A/6943A Multiprogrammer system is first turned-on, all output cards with the exception of the 69790A Memory Card are disabled (i.e. preset to a "safe" condition). The system enable (SYE) line that runs throughout the Multiprogrammer system is reset at power turn-on to ensure that all cards are disabled. The first time a card is programmed by an instruction that cycles the card it will also enable the card, allowing the desired output to be programmed. Two instructions are available which will allow you to subsequently disable and then enable all output cards in the system at one time; the system disable (SD) and system enable (SE) instructions. Table 6-1 lists the output cards affected by the SD and SE instructions and the effect of the SD instruction on each cards output.

6-63 System Disable (SD) Instruction

6-64 The system disable instruction is provided to allow you to program an emergency shutdown of outputs from all output cards in the system except the memory card. Executing a system disable instruction will set the output cards in the system to a predetermined (safe) condition as shown in Table 6-1. The syntax for the SD instruction is as follows:

"SD" or "SDT"

6-65 Since only the opcode is sent, the terminator "T" is not required but will be accepted if it is sent. Example 6-12 illustrates programming the SD instruction.

Example 6-12. Programming the SD Instruction

9825A Controller

```
0: wrt 723,"SD"
```

9835/45 Controllers

```
10   OUTPUT 723;"SD"
```

6-66 If the Multiprogrammer is in serial mode an SD instruction will not be executed until all preceding instructions have completed. In parallel mode, the SD instruction will always be executed immediately. An immediate response to a SD instruction can be obtained when the Multiprogrammer is in serial mode by programming the Multiprogrammer into immediate mode before sending the SD instruction:

"GI, SD, GN"

6-67 System Enable (SE) Instruction

6-68 The SE instruction is used to enable outputs of I/O cards that were previously disabled by a SD instruction. When a SE instruction is executed, outputs from the cards listed in Table 6-1 will return to the state that existed prior to executing

Table 6-1. Output Cards Affected By SD and SE Instructions

Model No.	Output Card	Effect of an SD instruction
69700-06A	Resistance Output	Output resistance goes to zero (+ calibration resistance of card).
69720A	D/A Voltage Converter	Output voltage goes to zero.
69721A	D/A Current Converter	Output current goes to zero.
69730A	Relay Output	All relay outputs open
69731A	Digital Output	Output lines go to logical zero.
69735A	Pulse Train Output	Output stops at end of current pulse.
69736A	Programmable Timer	Output stops immediately

a system disable instruction. Timed outputs such as those from the 69735A Pulse Train and 69736A Timer cards will continue where they left off. The SE instruction is affected by the serial, parallel, and immediate modes of operation in the same manner as the SD instruction. As with the SD instruction, inclusion of a terminator is optional. Example 6-13 illustrates programming a system enable instruction.

Exmaple 6-13. Programming the SE Instruction

9825A Controller

```
0: wrt 723,"SE"
```

9835/45 Controllers

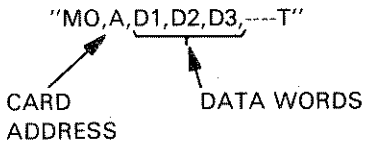
```
10   OUTPUT 723;"SE"
```

6-69 MEMORY INSTRUCTIONS

6-70 Memory instructions are basically intended for sending or reading large quantities of data words to or from 69790A Memory cards. They can, however, be used with other cards providing that you have a thorough understanding of the instructions and feel that they are suited to your requirements. Two memory instructions are available; the memory output (MO) instruction and the memory input (MI) instruction. The MI instruction is used in conjunction with HP-IB extended talk address 05 to read data back into the controller. No user available Multiprogrammer memory is utilized by either the MO or MI instructions. The MO and MI instructions run instantly regardless of serial or parallel modes. They are not permitted in the immediate mode.

6-71 Memory Output (MO) Instruction

6-72 The MO instruction is used to set the Memory card to the FIFO output mode and send a string of data words to the card. The Memory card has various operating modes, including the FIFO output mode, which are described in paragraph 7-132. Data sent with the MO instruction is read from the Multiprogrammer input buffer, translated to an internal format, and sent directly to the card where it is stored. After each data word is sent to the Memory card the card is cycled. Cycling the Memory card in an output mode causes an internal (write) pointer on the card to advance, resulting in incoming data being stored in sucessive locations on the card. If a card other than a Memory card is programmed by a MO instruction, each data word will be transferred to the output of the card as it is received. The syntax for the MO instruction is as follows:



6-73 The MO instruction consists of an opcode, card address, string of data words, and a terminator. Since it is impossible to distinquish between a card address and a data word, only one card address can be programmed at a time in an MO instruction. Example 6-14 illustrates programming a Memory card in slot 6 with an MO instruction. Five data values (1,2,3,4,5) are sent to the card. Since only a small number of fixed data values are sent to the card in this example, the data is sent to the card as constants in a literal field. Executing example 6-14 will result in these data values being stored in successive locations on the card.

Example 6-14. Using a Memory Output Instruction

9825A Controller

```
0: wrt 723,"MO,6,1,2,3,4,5T"
```

9835/45 Controllers

```
10   OUTPUT 723;"MO,6,1,2,3,4,5T"
```

6-74 For many applications you will find it convenient to send large quantities of variable data to the Memory card. Example 6-15 illustrates sending 1000 words of variable data to a Memory card in slot 6. In this example, the data to be sent to the card is contained in an array designated "A".

Example 6-15. Sending Variable Data Values Using a Memory Output Instruction

9825A Controller

```
100: wrt 723,"MO,6,"
101: for J=1 to 1000
102: wrt 723,A[J]
103: next J
104: wrt 723,"T"
```

9835/45 Controllers

```
100  OUTPUT 723;"MO,6",A(*),"T"
```

6-75 Memory Input (MI) Instruction

6-76 The MI instruction is used to set the Memory card to FIFO input mode and specify the slot address and number of data words to be read back from the Memory card (see paragraph 7-132). This instruction consists of an opcode, card address, word count (number of words that will be read back), and a terminator:

"MI,A,Word Count,T"

6-77 Example 6-16 illustrates using an MI instruction to specify a 1000 word readback from a memory card in slot 6. The actual data readback from the card is accomplished by means of HP-IB extended talk address 05 (red 72305/ENTER 723.05). When a read from HP-IB extended talk address 05 is detected, a data word will be read from the Memory card, converted to an ASCII format and sent to the controller. After each data word is accepted by the controller, the card is cycled. Cycling the Memory card in an input mode causes an internal (read) pointer on the card to advance to the next location allowing data to be read from successive locations on the card. If a card other than a Memory card has been programmed by the MI instruction, the card will be cycled after each data word is sent back to the controller, thereby storing fresh data on the card.

Example 6-16. Using a Memory Input Instruction

9825A Controller

```
200: wrt 723,"MI,6,1000T"
```

9835/45 Controllers

```
200  OUTPUT 723;"MI,6,1000T"
```

6-78 Generally it is best to read data from the Memory card into an array within the controller. This array should be equal in size to the number of words that you intend to read from the Memory card. Example 6-17 illustrates reading the data from the Memory card in example 6-16 into a controller array designated "B".

Example 6-17. Reading Data From a Memory Card

9825A Controller

```
300: for J=1 to 997 by 4
301: red 72305,B[J],B[J+1],B[J+2],B[J+3]
302: next J
```

9835/45 Controllers

```
300  ENTER 723.05;B(*)
```

6-79 When using a 9825 controller you should read the maximum number of variables possible with each "red" statement as this will increase the data transfer speed between the Multiprogrammer and the controller. For example, by changing the read loop in example 4 to "for J = 1 to 993 by 8" and reading into 8 array elements instead of 4 with each pass through the loop, the number of times the Multiprogrammer must be addressed to talk is cut in half resulting in a significant increase in data transfer speed.

6-80 MULTIPROGRAMMER MEMORY UTILIZATION

6-81 The following paragraphs describe the way in which the user memory area (RAM) is utilized. With this information, the user can calculate how much memory his program requires so he can take full advantage of the available memory. A brief review of instruction processing is included along with descriptions of how memory is allocated and what happens if memory becomes filled. In most instances, if the memory becomes filled, the system will simply wait until more memory becomes available. However, under certain circumstances, problems will occur if the memory is allowed to fill and overflow. Methods for avoiding and recovering from memory overflow are provided in the following discussion.

6-82 Memory Usage

6-83 The Multiprogrammer contains 2048 words of RAM; 586 are used by the firmware for system functions, leaving 1462 words available to store user instructions and other functions. Out of the 1462 words, the following functions will dynamically take memory as it is needed.

Additional Frames
Special Card Configurations
Instructions
Instruction Wait Blocks
Group Instructions
Immediate Mode
Memory Fragmentation

6-84 Additional Frames. The 1462 words available assumes that there is only a 6942A mainframe in the system. Each additional 6943A frame requires 24 words of memory. The Multiprogrammer determines the number of frames in the system by searching for the highest frame address. It assumes that all frames with a lower frame address are connected. Thus, in a mainframe only system, if the frame address switch is set to 7, the Multiprogrammer will assume there are seven additional frames, and allocate: $7 \times 24 = 168$ words.

6-85 Special Card Configurations. If an I/O card is reformatted, the additional information requires three words of RAM. When a card is returned to its default parameters (either with an SF with 0 parameters, or with a system reset), the additional memory is no longer needed, and it is returned. If 5 cards were reformatted, $5 \times 3 = 15$ words are required.

6-86 Instructions. When an instruction is active in the system, it requires an instruction module, made up of a block of memory. For the purpose of computing memory utilization, there are 5 classes of instructions:

Class	Description
1.	Instructions that are executed immediately out of the buffer, and thus don't require any memory.
2.	Normal instructions, which consists of most of the Multiprogrammer's instructions. These instructions require a 17 word base, plus two words for every card address (regardless of whether its a input or output instruction). Example: an OP with 5 cards, $17 + (5 \times 2) = 27$ words

3 Interrupt instructions (OI and II) require a 17 word base and three words per card. Thus an II with five cards requires:
17 + (5 × 3) = 32 words

4 Instructions with repeat factors (IP with R's and R's + W's, IE with R's and R's + W's) require a base of 17 words, plus two words per card, plus one word per card per reading. Thus, the following: "IP,R20,1,2,3,4,5T"; requires:
17 + (5 + 2) + (5 + 20) = 127 words

5. Special instructions use a fixed amount of memory.

6-87 Table 6-2 specifies which one of the five classes described above pertains to each of the 32 instruction types. Note that for the Class 5 instructions, the amount of fixed memory required for each instruction type is also included.

Table 6-2. Instruction Memory Utilization

Inst. Opcode	Class	Inst. Opcode	Class
AC	2	MI	1
CC	1	MO	1
CG	1	OB	5 (20 words)
CW	1	OI	3
CY	2	OP	2
DC	2	OS	2
GI	1	RC	5 (21 words)
GN	1	RF	2
GP	5 (3 words)	RS	2
GS	5 (3 words)	RV	2
IE	2	SC	5 (21 words)
IE*	4	SD	5 (3 words)
II	3	SE	5 (3 words)
IN	5 (3 words)	SF	1
IP	2	WA	5 (18 words)
IP	4	WC	2
		WF	2
		WU	5 (21 words)

*With Repeat Factors

6-88 Instruction Wait Block. Any instruction which is not started immediately in either the parallel or serial mode, must be placed on an instruction waiting to execute list. Each instruction placed on an instruction waiting to execute list. Each instruction placed on the list requires an additional three word block.

6-89 Group Instructions. As was described earlier, the advantage of group instructions is that their instruction modules are saved for reuse. Thus, when the group is invoked later, the system does not have to regenerate the instruction module, and it saves time. Once defined, group instructions do not relinquish their memory until they are cleared (CG instruction), redefined, or the system is reset.

6-90 Immediate Mode. When a GI instruction is executed, the Multiprogrammer requires 40 words of memory to store the previous state of the system. This memory is released when a GN instruction is executed.

6-91 Memory Fragmentation. The memory requirements described above assume that all of the memory used by the function is contained in ONE contiguous block. In a heavily loaded system, it is possible for memory to become fragmented, broken up into many small blocks of available memory. Although there is still enough memory available for the function, it might be contained in several small blocks. In this situation there is an overhead of two to four words (depending on the function) for each block of memory. Since it is impossible to compute the amount of fragmentation in the system at a given time, when making memory calculations a certain amount of memory should be reserved for fragmentation. Reserving 100 words will probably cover 95% of all fragmentation situations.

6-92 Summary. Taking all of the uses of memory described above, the following formula can be used to compute total memory utilization:

24 × (additional frames) + 3 × (reformatted cards) + Active Instruction Modules: non groups only + Active Instruction Modules: Only groups of the following types: IE,II,IP,OI,RS,RV + All Group Definitions + 3 × (all active instructions: inst wait blocks) + 40 (only if in immediate mode) + 100 (reserved for memory fragmentation) =

Total Number of Words (not to exceed 1,462)

6-93 Memory Full Condition

6-94 Because memory is dynamically allocated, when an instruction is parsed, memory space is allocated, and when the instruction goes inactive the memory is returned, it is possible for all of the Multiprogrammer's memory to become full. When this happens, no additional instructions or further HP-IB extended talk commands will be processed until the memory has been partially emptied. To understand the memory full condition, a quick review of instruction processing follows.

6-95 All data coming from the HP-IB, both instructions and extended talk address commands, go into a 128 byte buffer. The instruction parser, which decodes and parses the instructions, and also detects extended talk commands, pulls data out of this buffer one byte at-a-time. As the parser detects new instruction opcodes, it allocates a block of memory and then parses the instruction. When it detects an extended talk address, it passes control to the appropriate module that will send the data back to the controller. Blocks of memory are returned to free memory as the instructions go inactive. Instructions that do not send data back to the controller (typically output instructions) go inactive when they have completed all communications with the I/O cards. Instructions that send data back to the controller go inactive after all of the data has been sent back, or the Multiprogrammer has thrown it away.

6-96 If, at any time, the input parser tries to allocate a block of memory for an instruction being parsed, and there is no memory available, a memory full condition has occurred. The input parser can not complete processing the current instruction until it gets the required memory, thus it is suspended. With the input parse suspended, no additional isntructions or extended talk commands can be processed. The Multiprogrammer will continue to take bytes from the HP-IB until it has filled up its 128 byte buffer, at which time it will stop accepting data from the HP-IB. In a memory full conditon, the Multiprogrammer is unable to send data back to the controller, because any extended talk commands will remain in the buffer, unseen by the input parser.

6-97 Although memory is full, and the HP-IB suspended, the Multiprogrammer has not stopped executing instructions. As those instructions that were already in the memory go inactive, they will de-allocate their memory, making it available to other instructions. The instruction parser will then wake-up, completing the processing of the instruction it was working on and as bytes are taken out of the buffer, the HP-IB will start operating again. The entire process of memory filling, HP-IB communications stopping, memory unfilling, and HP-IB communications resuming, is normally transparent to the user and the controller.

6-98 Avoiding Memory Overflow

6-99 Because the Multiprogrammer is inhibited from sending back data to the controller in a memory full condition, no instruction that returns data will be able to send back its data and go inactive. Thus, the only way memory will become available is if an output instruction completes. If, for some reason, no output instruction can complete, the Multiprogrammer will permanently hung-up in a memory overflow. The only way to recover from a memory overflow is to reset the Multiprogrammer with a HP-IB Device Clear command, which throws out all the data stored in the Multiprogrammer.

Example: If 100 IP instructions are sent out before trying to read back any of the data, memory will become full before the extended talk command is sent, thus it will not be seen. Since there are no output instructions that can release some memory, a memory overflow has occurred. This could have been avoided if after sending 50 instructions, all of the data from these instructions was readback before sending the next 50 instructions.

6-100 Memory overflow can also occur if an I/O card hangs up. For example, assume that a sequence of 100 OP instructions is sent out. Normally this would not be a problem, because even though memory fills up, as the earlier instructions complete, they will release their memory, making room for the later instructions. However, if the first instruction programmed a card that doesn't complete (a defective card, a defective external device, or just a broken gate/flag handshake line), the next 99 instructions will cause a memory full condition. Eventually the user will detect that everything has stopped, and he will try and read the Multiprogrammer status or possibly send a Clear Card (CC) instruction. Since the memory is already full, no other instructions or readback are allowed, thus there is no way the user can clear out the card, and again, there is a memory overflow.

6-101 The above problem could have been avoided if the program had sent out 50 instructions followed by an IN instruction, and then waited until the SRQ from the IN occurred. SRQ would indicated that the first 50 instruction completed before sending the additional 50 instructions. If a card hung-up, there would always be plenty of memory left to read the card status (RS instruction) and clear the busy card.

Chapter 7
PLUG-IN CARD DESCRIPTIONS AND PROGRAMS

7-1 This chapter provides a brief functional description and programming examples for each type of plug-in I/O card that can be presently used in a 6942/43 Multiprogrammer System. The cards are presented in numerical sequence according to the model number. Block diagrams, connector diagrams and sample test programs are also provided as programming aids. Typical measurement applications using different card types are provided after the individual card programs. All of the procedures in this Chapter assume that you have read Chapters 4 through 6 and have a working knowledge of the Multiprogrammer's instruction set and basic operating modes.

7-2 The sample test programs in this Chapter may yield erroneous results, if card was programmed incorrecty prior to the test. Therefore, it is recommended that the Multiprogrammer be cleared first; by executing a "clr 723" (9825) or "RESET 723" (9835/45). A listing and description of subroutine "cheker", used to read Multiprogrammer status and test for errors in the sample test programs, can be found in Appendix C.

7-3 All 9835/45 program examples in this chapter use OPTION BASE 1 when setting up and redimensioning data arrays. Where required, a description of the program changes required to use OPTION BASE 0 is provided.

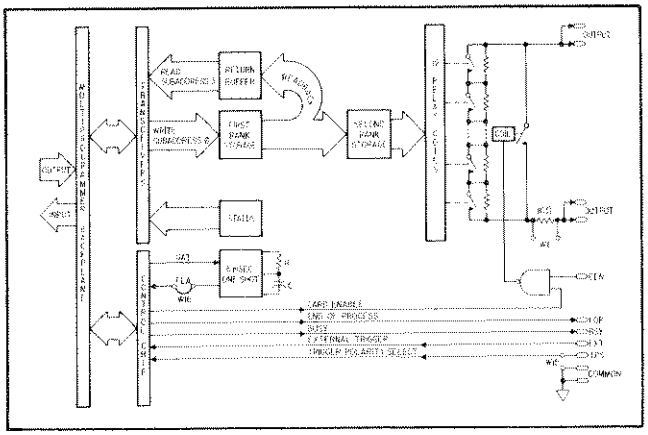
7-4 INDIVIDUAL CARD PROGRAMS

7-5 Resistance Output Cards, Models 69700A-69706A

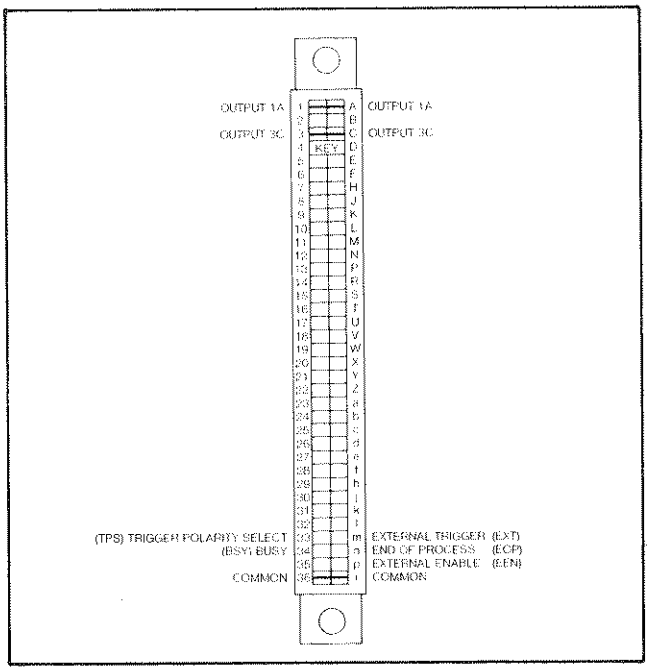
7-6 These cards provide a programmed value of resistance as their output and are generally used for programming option 040 power supplies. Twelve magnetically shielded, mercury-wetted, reed relays select the resistance values by modifying the value of a series string of high-accurary, binary weighted resistors. Data transfer from internal storage circuits on the card to an external device can be initiated by a controller command or by external triggering. Internal timing circuits prevent the card from being reprogrammed until the resistance output has stabilized (approximately 6 msec).

1. Model 69700 is supplied without output resistors so that customers may select and load their own resistors as desired. Jumpers on this card are factory set for decimal programming using integer values in the range of 0-4095 with an LSB of 1.

2. Models 69701, 69704, and 69705 are intended for programming option 040 power supplies which have a maximum output voltage rating of 40 volts. For ease in programming, the power supply and resistance card are considered to be one unit. This allows programming an output in volts rather than resistance. As shipped from the factory jumpers on these cards have been set to allow programming an output voltage from the power supply in the range of 0-40.95 volts with a minimum step change (LSB) of .01 volts (10mV).



69700A-69706A Block Diagram



69700A-69706A Connector

3. Models 69702 and 69706 are intended for programming option 040 power supplies which have a maximum output voltage rating that is greater than 40 Volts. Again, the power supply and resistance card are considered to be one unit for programming purposes, allowing programming in volts rather than resistance. As shipped from the factory, jumpers on these cards have been set to allow programming an output voltage from the power supply in the range of 0-102.375 Volts with a minimum step change of 0.025 Volts (25mV).

7-7 Table 7-1 lists the maximum resistance output and resolution of each card.

Table 7-1. Maximum Output Resistance and Resolution

Model	Maximum Resistance in ohms	Resolution in ohms	Resolution in volts*
69700**	-------	---	----
69701	8190	2	.010
69702	3072.5	7.5	.025
69704	40950	10	.010
69705	81900	20	.010
69706	204750	50	.025

* Resolution in volts is specified for resistance card/power supply combination.

** Resistance values and resolution on the 69700 are determined by the user.

7-8 Example 7-1 illustrates using two Resistance Output Cards, installed in slots 4 and 110, to program two option 040 power supplies to 30.02 and 75.475 volts, respectively. Since addresses and data are both fixed in this example they are sent out as constants in a literal field. When the OP instruction completes, the next instruction (if any) can begin if the Multiprogrammer is in the serial operating mode. If the Multiprogrammer is operating in the parallel mode, SRQ will be set when the OP instruction has completed.

Example 7-1. Programming Two Option 040 Power Supplies

9825A Controller

```
wrt 723,"OP,4,30.02,110,75.475T"
```

9835/45 Controllers

```
OUTPUT 723;"OP,4,30.02,110,75.475T"
```

NOTE

The Output Parallel (OP) instruction used in examples 7-1, 7-2, 7-3, and 7-5 is a typical instruction used to program these cards. The actual instruction to use will be determined by the application. For example, a Write and cycle (WC) instruction can be used if the Multiprogrammer is in parallel mode and you do not request an SRQ from the Multiprogrammer when the programmed card completes its data transfer. Or, as another example, cards that are to be externally triggered would first be programmed by a Write First Rank (WF) instruction to load output data into first rank storage on the card. Then, when an external trigger is subsequently applied to the card, data will be transferered from first rank storage to the output of the card.

7-9 Data values above or below the maximum range of the cards will be rejected and an error messsage generated. There may be instances where you will find it convenient to program these cards using a different data range (and LSB). For instance, you may need current programming of an option 040 power supply rather than voltage programming. In these cases, you may either reformat the card using a set format (SF instruction or permanently select a different format by changing jumpers on the card (see 69700-69706 card manual).

7-10 Another point to consider when programming option 040 power supplies is the maximum output of the power supply. Since the cards are 12 bits it is possible to program 4095 LSB's, thereby making it possible for you to attempt to program more output voltage (or current) than the power supply can provide. If you feel that this is a possible problem for you, you should use an "SF" instruction to reformat the card and specify a limit. If you then attempt to program an output that exceeds the limit, the Multiprogrammer will set SRQ and generate an error message to notify you that a limit has been exceeded. In this case the card will not be programmed.

7-11 Example 7-2 illustrates reformatting a 69701A Resistance Output Card installed in slot 8 to allow programming in ohms, then programming the output of the card to 400 ohms. Reformatting the card to allow programming in ohms is simply a matter of using an "SF" instruction to change the LSB of the card to the resolution of the card in ohms (2). Once the format of the card has been changed it will remain that way until: (1) it is reformatted, (2) a system clear (i.e. clr 723 or RESET 723) is sent to the Multiprogrammer or (3) the Multiprogrammer system is turned off.

Example 7-2. Programming a Resistance Output Card in Ohms

9825A Controller

```
0: wrt 723,"SF,8,2,3,2T"
1: wrt 723,"OP,8,400T"
```

9835/45 Controllers

```
10   OUTPUT 723;"SF,8,2,3,2T"
20   OUTPUT 723;"OP,8,400T"
```

7-12 Example 7-3 illustrates using a 69701A Resistance Output card installed in slot 8 to program a current output of 400 Amps from an HP 6464C option 040 power supply. Since the minimum step change when current programming this supply is 2 Amps, and the maximum current output of the supply is 1000 Amps, a set format instruction is used to change the LSB of the card to 2 Amps and set a limit of 1000 Amps before programming the card.

Example 7-3. Programming a Current Output and Setting a Limit

9825A Controller

```
wrt 723,"SF,8,4,3,2,12,1000T"
wrt 723,"OP,8,400T"
```

9835/45Controllers

```
OUTPUT 723;"SF,8,4,3,2,12,1000T"
OUTPUT 723;"OP,8,400T"
```

7-13 Data contained in first rank storage on the resistance programming card can be read back from sub-address 3 at any time by using a read value (RV) instruction. If the card has been cycled, either by an instruction such as an "OB", "OP", "OS", "OI", "CY", "WC", or by an external trigger, the first rank data will be the same as the data present at the output of the card. Example 7-4 illustrates reading first rank data from a card in slot 4 into variable "A" in the controller.

Example 7-4 Reading First Rank Data

9825A Controller

```
0: wrt 723,"RV,4,3T"
1: red 72306,A
```

9835/45 Controllers

```
10   OUTPUT 723;"RV,4,3T"
20   ENTER 723.06;A
```

7-14 Example 7-5 is a sample test program which illustrates programming with variable data values and is intended to give you some hands-on experience in programming a Resistance Output card. This example assumes the resistance. output card is installed in slot 4. You may either connect an option 040 power supply to your resistance card and measure the output voltage of the supply or you may simply program the card and, using the formula in step 4, Paragraph 7-16, calculate what the output resistance should be and measure it.

7-15 The fixed 3 number format used in the 9825 controller program is required whenever an option 040 power supply is being programmed with a 69702 or 69706 resistance programming card to ensure that voltage output data with 3 decimal places is not truncated. When programming a 69701, 69704, or 69705 resistance programming card with a 9825 controller a fixed 2 number format is required to ensure that voltage output data with 2 decimal places is not truncated. Since fixed 2 is the normal wake-up format of the 9825 it is unnecessary to specify a fixed 2 format unless your program has changed the number format of the controller for other reasons.

7-16 Perform the following steps when using the sample program..

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter voltage" displayed. Enter the output voltage you would like to program and depress CONTINUE.

3. If the program has been correctly entered into the controller, the controller will stop with "test output" displayed. At this point you may either use a voltmeter to check the output voltage of your option 040 power supply or an ohmmeter to check the output resistance of the resistance output card between pins "A" and "C".

4. If a resistance check is made the output resistance of the card may be compared with a nominal value which is determined by the equation "Rnom = Vout/Vlsb × Rlsb", where Rnom is the nominal resistance, Vout is the programmed output voltage, Vlsb is the resolution (LSB) of the card in volts and Rlsb is the resolution (LSB) of the card in ohms. For example, assume that a 20 volt output has been programmed from a power supply connected to a 69701 resistance output card. Since Vout is 20, Vlsb is 0.01, and Rlsb is 2, our equation becomes "Rnom = 20/0.01 × 2" which yields a nominal resistance value of 4000 ohms. When comparing the actual resistance value against the nominal value you must take into account the accuracy specification of the card.

7-17 If errors are detected by the Multiprogrammer, an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-5. 69700-69706 Sample Test Program

9825A Controller

```
0: fxd 3
1: ent "enter voltage",B
2: wrt 723;"OP,4",B,"T"
3: esb "cheker"
4: if V=0;dsp "test output"
5: end
*16762
```

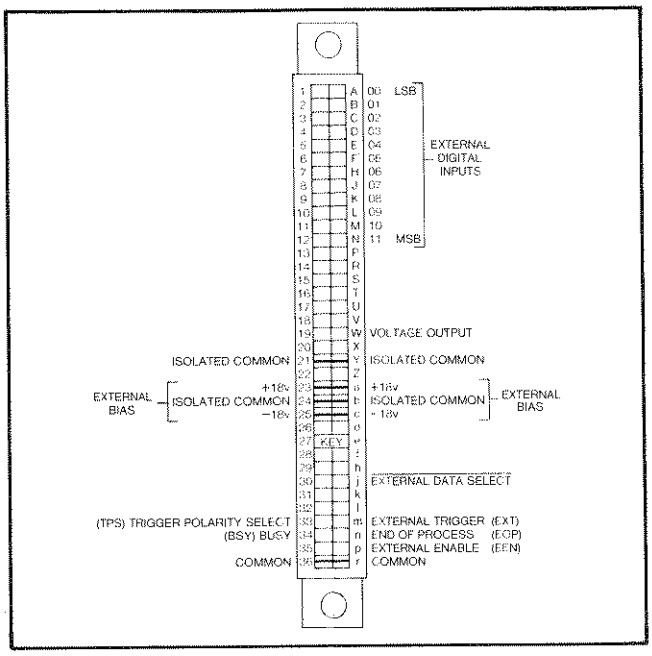
9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(1)
30   INPUT "ENTER VOLTAGE",B
40   OUTPUT 723;"OP,4",B,"T"
50   GOSUB Cheker
60   IF V=0 THEN DISP "TEST OUTPUT"
70   END
```

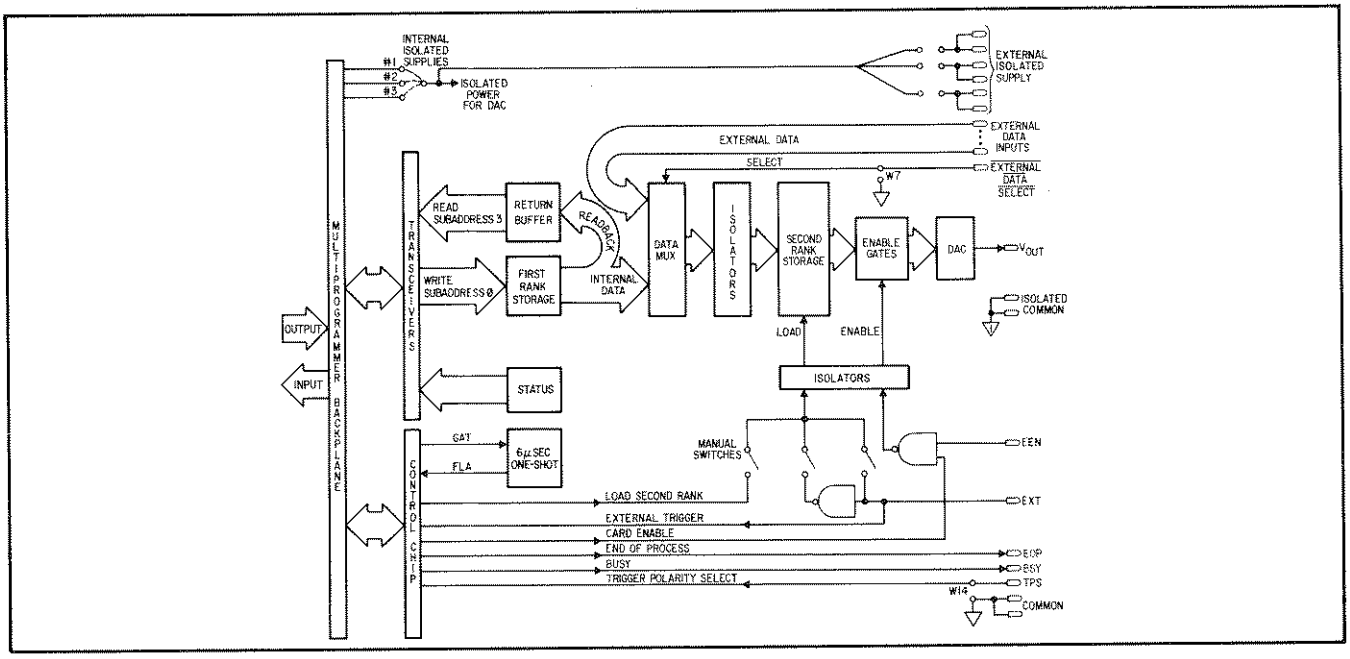
7-18 D/A Voltage Converter Card, 69720A

7-19 This card ia a twelve bit bipolar digital-to-analog converter which may be programmed to output a voltage in the range of −10.24 through +10.235 volts with a minimum step change of 0.005 volts. Data transfer from internal storage circuits on the card to the D/A voltage converter can be initiated by a controller command or by external triggering.

7-20 As shipped from the factory, the data format of the D/A voltage card has been set for programming in volts. Votlage values more negative than −10.24 volts or more positive than +10.235 volts will be rejected and an error message generated. Voltage values that are in range but not an increment of .005 volts (the lsb) will be rounded off to the nearest value that is an increment of .005 volts. If you desire to program this card using a different format you may either use a Set Format (SF) instruction to reformat the card or permanently select a different format by changing jumpers on the card (see 69720A card manual).



69720A Connector



69720A Block Diagram

7-21 Example 7-6 illustrates using an OP instruction to program a D/A voltage card in slot 1 to +5 volts and a D/A voltage card in slot 2 to −5.235 volts. Since addresses and data are both fixed in this example they are sent out as constants in a literal field. After the OP instruction has completed, the next instruction (if any) can start if the Multiprogrammer is operating in the serial mode. If the Multiprogrammer is operating in the parallel mode, SRQ will be set when the OP instruction completes.

NOTE

The Output Parallel (OP) instruction used in examples 6 and 8 is a typical instruction used to program this card. The actual instruction to use will be determined by the application. For example, a Write and Cycle (WC) instruction can be used if the Multiprogrammer is in parallel mode and you do not require an SRQ from the Multiprogrammer when the programmed card completes its data transfer. Or, as another example, cards that are to be externally triggered would first be programmed by a Write First Rank (WF) instruction to load output data into first rank storage on the card. Then, when an external trigger is subsequently applied to the card data will be converted to the programmed output voltage.

Example 7-6. Programming Two D/A Voltage Converter Cards

9825A Controller

```
0: wrt 723,"OP,1,5,2,-5.235T"
```

9825/45 Controllers

```
10  OUTPUT 723;"OP,1,5,2,-5.235T"
```

Example 7-7. Reading First Rank Data

9825A Controller

```
0: wrt 723,"RV,1.3T"
1: red 72306,A
```

9835/45 Controllers

```
10  OUTPUT 723;"RV,1.3T"
20  ENTER 723.06;A
```

7-22 Data contained in first rank storage on the D/A Voltage Converter card can be read back from sub-address 3 at any time by using a read value (RV) instruction. If the card has been cycled, either by an instruction such as an "OB", "OP", "OS", "OI", "CY", "WC", or by an external trigger, the first rank data will be the same as the data present at the output of the card. Example 7-7 illustrates reading first rank data from a card in slot 1 into variable "A" in the controller.

7-23 Example 7-8 is a sample test program which is intended to give you some hands-on experience in programming a D/A voltage card. This example assumes the card is installed in slot 1 and allows you to program any voltage within the range of the card. Voltage output data is sent out in the form of a variable.

Example 7-8. 69720A Sample Test Program

9825A Controller

```
0: fxd 3
1: ent "enter voltage";B
2: wrt 723,"OP,1",B,"T"
3: gsb "cheker"
4: if V=0;dsp "test voltage"
5: end
*9227
```

9835/45 Controllers

```
10  OPTION BASE 1
20  DIM R(11)
30  INPUT "ENTER VOLTAGE",B
40  OUTPUT 723;"OP,1",B,"T"
50  GOSUB Cheker
60  IF V=0 THEN DISP "TEST VOLTAGE"
70  END
```

7-24 The fixed 3 number format used in the 9825 controller program is required whenever the D/A voltage card is being programmed by the 9825 controller to ensure that voltage output data with three decimal places is not truncated.

7-25 Perform the following steps when using the sample test program.

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter voltage" displayed. Enter the output voltage you would like to program and depress CONTINUE.

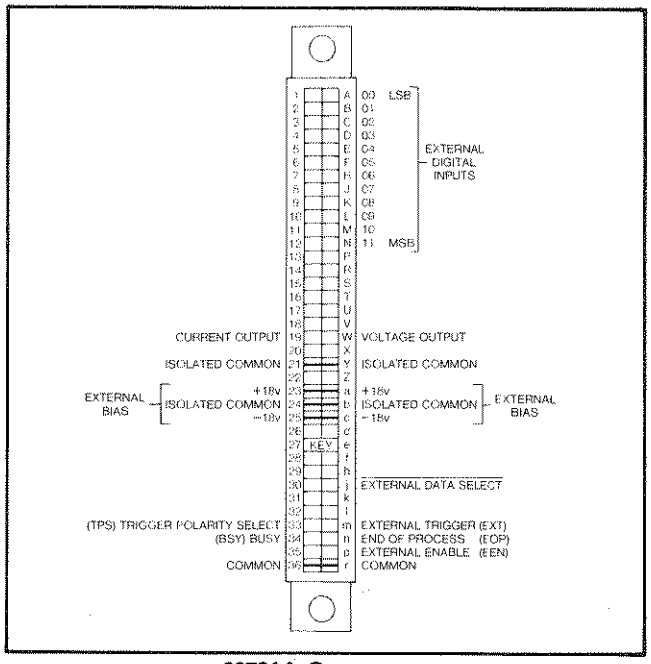
3. If the program has been correctly entered into the controller, the controller will stop with "test voltage" displayed. At this point you may use a voltmeter to check the output voltage between pins W and Y (common).

7-26 If errors are detected by the multiprogrammer an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

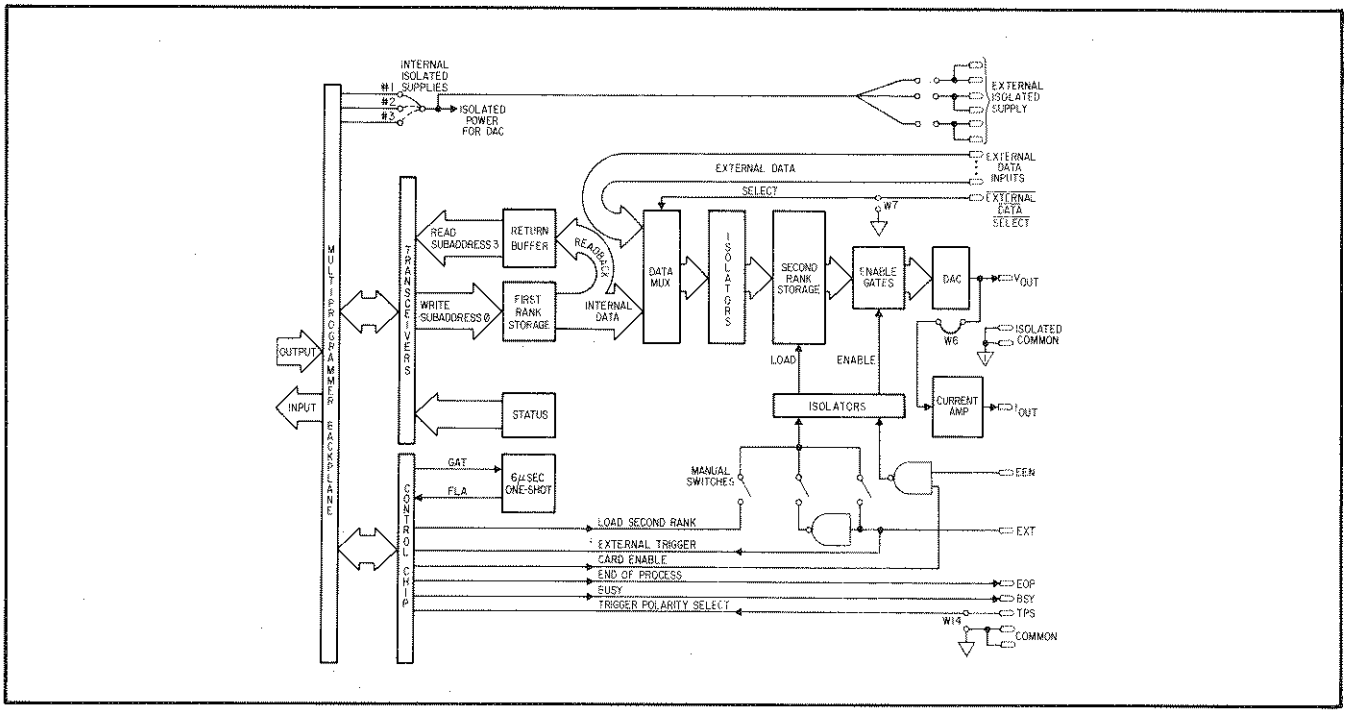
7-27 D/A Current Converter Card, 69721A

7-28 This card is a twelve bit bipolar digital to analog converter which may be programmed to output a current in the range of −20.48 through +20.47 milliamps with a minimum step change of .01 milliamps (10 microamps). Data transfer from internal storage circuits on the card to the D/A current converter can be initiated by a controller command or by external triggering.

7-29 As shipped from the factory the format of the D/A current card has been set for programming in milliamps. Current values more negative than −20.48 milliamps or more positive than +20.47 milliamps will be rejected and an error message generated. Current values that are in range but not an increment of 0.01 milliamps (the lsb) will be rounded off to the nearest value that is an increment of 0.01 milliamps. If you desire to program this card using a different format you may either use a Set Format instruction (SF) to reformat the card or permanently select a different format by changing jumpers on the card (see 69721A card manual).



69721A Connector



69721A Block Diagram

7-30 Example 7-9 illustrates using an OP instruction to program a D/A current card in slot 0 to −20 milliamps and a D/A current card in slot 15 to 10.24 milliamps. Since addresses and data are both fixed in this example they are sent out as constants in a literal field. After the OP instruction has completed, the next instruction (if any) will run if the Multiprogrammer is operating in the serial mode. If the Multiprogrammer is operating in the parallel mode, SRQ will be set when the OP instruction completes.

NOTE

The Output Parallel (OP) instruction used in examples 9 and 11 is a typical instruction used to program this card. The actual instruction to use will be determined by the application. For example, a Write and Cycle (WC) instruction can be used if the Multiprogrammmer is in parallel mode and you do not require an SRQ from the Multiprogrammer when the programmed card completes its data transfer. Or, as another example, cards that are to be externally triggered would first be programmed by a Write First Rank (WF) instruction to load output data into first rank storage on the card. Then when an external trigger is subsequently applied to the card, data will be converted to the programmed output current.

Example 7-9. Programming Two D/A Current Converter Cards

9825A Controller

```
0: wrt 723,"OP,0,-20,15,10.24T"
1: end
```

9835/45 Controllers

```
10 OUTPUT 723;"OP,0,-20,15,10.24T"
20 END
```

7-31 Data contained in first rank storage on the D/A Current converter card can be read back from subaddress 3 at any time by using a read value (RV) instruction. If the card has been cycled, either by an instruction such as an "OB", "OP", "OS", "OI", "CY", "WC", or by an external trigger, the first rank data will be the same as the data present at the output of the card. Example 7-10 illustrates reading first rank data from a card in slot 0 into variable "A" in the controller.

Example 7-10. Reading First Rank Data

9825A Controller

```
0: wrt 723,"RV,0,3T"
1: red 723(6,A
```

9835/45 Controllers

```
10  OUTPUT 723;"RV,0,3T"
20  ENTER 723.06;A
```

7-32 Example 7-11 is a sample test program which is intended to give you some hands-on experience in programming a D/A current card. This example assumes the card is installed in slot 0 and allows you to program any current output within the range of the card. Current output data is sent out in the form of a variable.

7-33 The fixed 2 number format used in the 9825 controller program is required whenever the D/A current card is being programmed by the 9825 controller. Since the 9825 controller wakes-up in fixed 2 format when it is turned on, there is no need to be concerned with line 0 unless your program changed the number format of the controller prior to this.

7-34 Perform the following steps when using the sample program.

1. Connect a 100 ohm load resistor between pins 19 and 21 (common) of the 69721A output connector.

2. Load the program, including subroutine "cheker", into the controller.

3. Depress RUN on the controller. The controller will stop with "enter current" displayed. Enter the output current (in milliamps) you would like to program and depress CONTINUE.

4. If the program has been correctly entered into the controller, the controller will stop with "test current" displayed. At this point you may calculate the output current by using a voltmeter to measure the voltage across the load resistor and dividing the voltage reading by 100. Keep in mind that the result will only be an approximate value unless the exact resistance value is known and used in the calculations.

7-35 If errors are detected by the Multiprogrammer an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-11. 69721A Sample Test Program

9825A Controller

```
0: fxd 2
1: ent "enter current ";B
2: wrt 723;"OP+0";B;"I"
3: esb "cheker"
4: if V=0;dsp "test current"
5: end
*3222
```

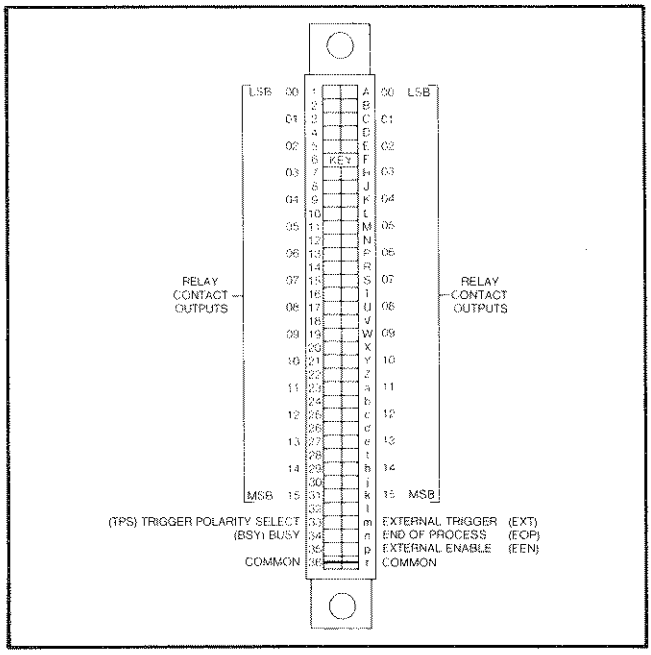
9835/45 Controllers

```
10  OPTION BASE 1
20  DIM R(11)
30  INPUT "ENTER CURRENT",B
40  OUTPUT 723;"OP+0",B;"I"
50  GOSUB Cheker
60  IF V=0 THEN DISP "TEST CURRENT"
70  END
```

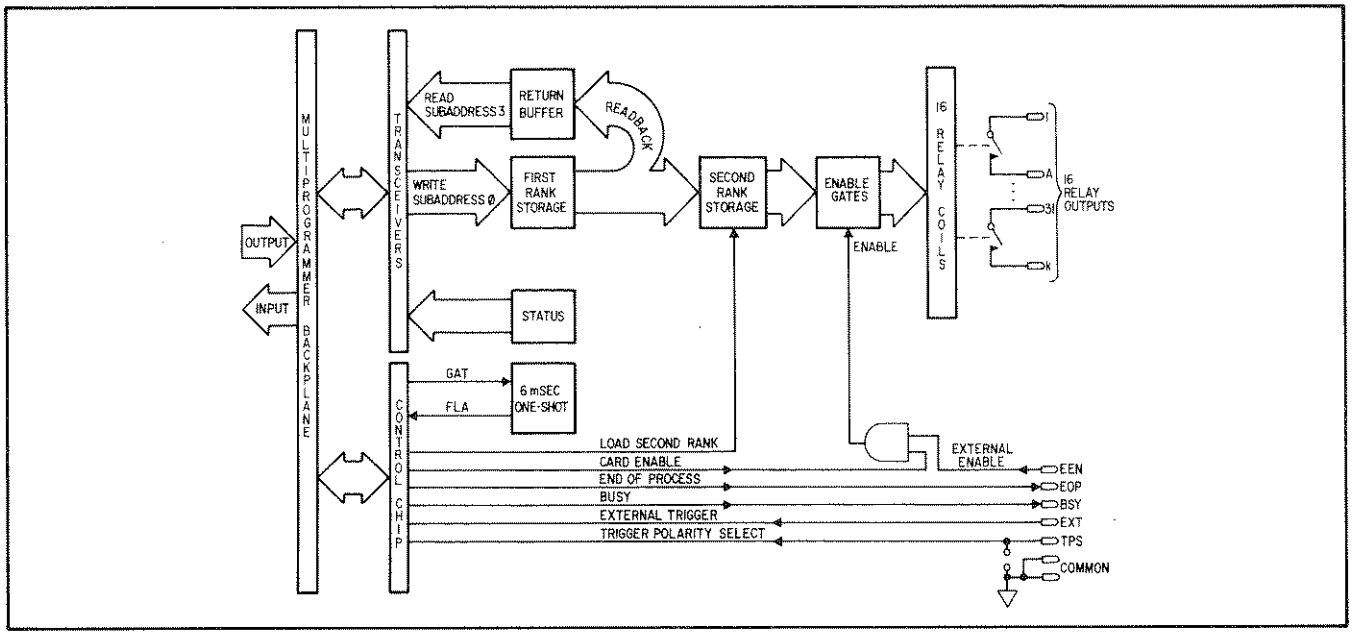
7-36 Relay Output Card, 69730A

7-37 This card provides 16 independent SPST normally open (Form A) relays. Output data will be in the form of contact closures. Data transfer from internal storage circuits on the card to an external device can be initiated by a controller command or by external triggering. Internal timing circuits on the card prevent the card form being reprogrammed until all relays have stabilized (approximately 6 msec).

7-38 As shipped from the factory the data format of the relay output card has been set for decimal programming using integer values in the range of 0-65535 with an LSB of 1. Data values above or below this range will be rejected and an error message generated. If you desire to program this card using a different format or data range you may either use a Set Format instruction (SF) to reformat the card or permanently select a different format by changing jumpers on the card (see 69730A card manual).



69730A Connector



69730A Block Diagram

7-39 Example 7-12 illustrates using an OP instruction to program all the relays closed on a relay output card in slot 7 and programming all the relays open on a relay card in slot 12. Since addresses and data are both fixed in this example they are sent out as constants in a literal field. After the OP instruction has completed, the next instruction (if any) can start provided that the Multiprogrammer is operating in the serial mode. If the Multiprogrammer is operating in the parallel mode, SRQ will be set when the OP instruction completes.

NOTE

The Output Parallel (OP) instruction used in examples 12 and 15 is a typical instruction used to program this card. The actual instruction to use will be determined by the application. For example, a Write and Cycle (WC) instruction can be used if the Multiprogrammer is in parallel mode and you do not require an SRQ from the Multiprogrammer when the programmed card completes its data transfer. Or, as another example, cards that are to be externally triggered would first be programmed by a Write First Rank (WF) instruction to load output data into first rank storage on the card. Then when an external trigger is subsequently applied to the card, data will be transferred from first rank storage to the output of the card closing the appropriate relays.

Example 7-12. Programming Two Relay Output Cards

9825A Controller

```
0: wrt 723,"OP,7,65535,12,0T"
```

9835/45 Controllers

```
10   OUTPUT 723;"OP,7,65535,12,0T"
```

Example 7-13. Programming Individual Relays

9825A Controller

```
0: wrt 723,"OB,7,0,0,15,1T"
```

9835/45 Controllers

```
10   OUTPUT 723;"OB,7,0,0,15,1T"
```

7-40 In some cases you may find that you would prefer to have control over individual relays on the card rather than calculating a data value for the whole card. The Output Bit (OB) instruction provides an excellent means of achieving this by allowing you to open or close any relay(s) desired. Example 7-13 illustrates using an OB instruction to open the LSB relay (bit 0) and close the MSB relay (bit 15) on a card in slot 7.

7-41 Data contained in first rank storage on the relay output card can be read back from sub-address 3 at any time by using a read value (RV) instruction. If the card has been cycled, either by an instruction such as an "OB", "OP", "OS", "OI", "CY", "WC", or by an external trigger, the first rank data will be the same as the data present at the output of the card. Example 7-14 illustrates reading first rank data from a card in slot 7 into variable "A" in the controller.

Example 7-14. Reading First Rank Data

9825A Controller

```
0: wrt 723,"RV,7,3T"
1: red 72306,A
```

9835/45 Controllers

```
10   OUTPUT 723;"RV,7,3T"
20   ENTER 723.06;A
```

7-42 Example 7-15 is a sample test program which is intended to give you some hands-on experience in programming a relay output card. This example assumes the relay card is installed in slot 7 and allows you to specify an output bit number and program a contact closure with the associated relay. Data is sent out in the form of a variable.

7-43 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter bit number" displayed. Enter an output bit number form 0 to 15 and depress CONTINUE.

3. If the program has been correctly entered into the controller, the controller will stop with "test bit" displayed. At this point you can use an ohmmeter to check the contact resistance of the relays. Relay outputs can be measured between odd numbered pins from 1 to 31 and corresponding lettered pins (i.e. 1-A,3-C,31-k) as shown in the 69730A output connector diagram. All relay contacts should be open except for the contacts that correspond to the bit number that was entered in step 2.

7-44 If errors are detected by the Multiprogrammer an error message will be printed (9825) or displayed (9834/45) and the program will terminate.

Example 7-15. 69730A Sample Test Program

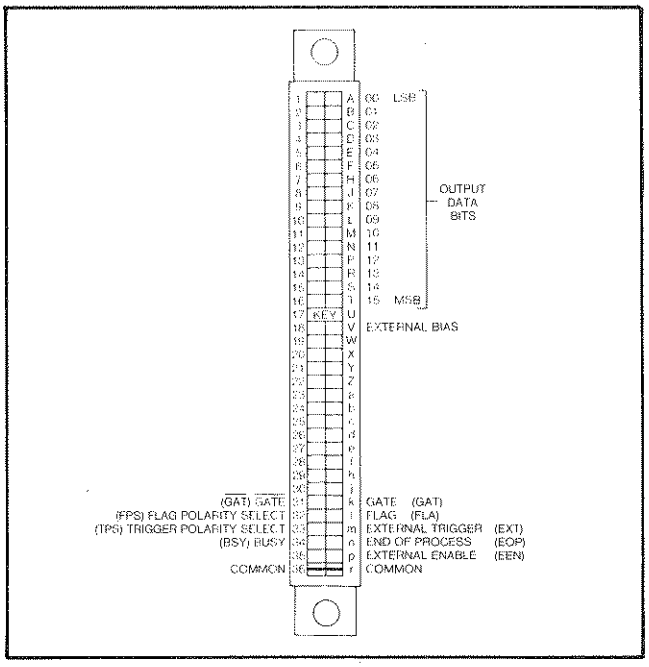
```
0: ent "enter bit number";B
1: wrt 723;"OP;7";21B;"T"
2: asb "cheker"
3: if V=0;dsp "test bit"
4: end
*12513
```

```
10   OPTION BASE 1
20   DIM R(11)
30   INPUT "ENTER BIT NUMBER";B
40   OUTPUT 723;"OP;7";21B;"T"
50   GOSUB Cheker
60   IF V=0 THEN DISP "TEST BIT"
70   END
```

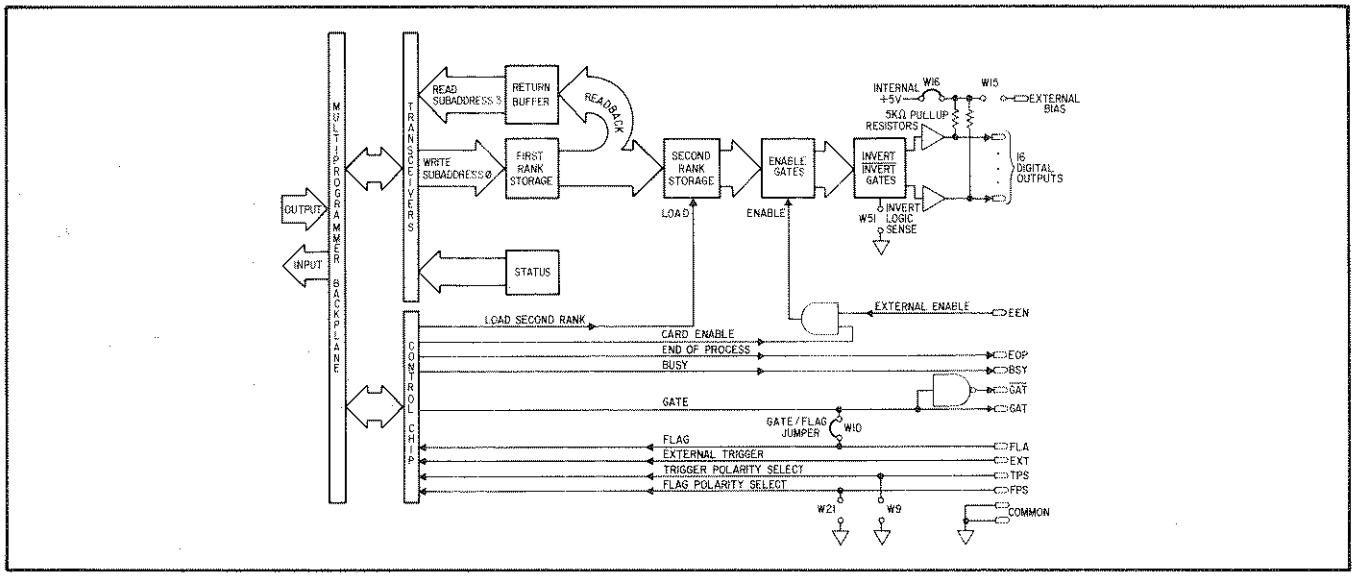
7-45 Digital Output Card, 69731A

7-46 This card provides 16 bits of TTL/DTL/CMOS-compatible logic levels as its output. As shipped from the factory, all output bits are positive true (high = logical 1). Jumper W51 on the card can invert the logic sense of all 16 outputs. Data transfer from internal storage circuits on the card to an external device can be initiated by a controller command or by external triggering. Jumper (W10) on the card is removable to allow an external timing circuit to control the gate/flag sequence of the card.

7-47 As shipped from the factory the data format of the digital output card has been set for decimal programming using integer values in the range of 0-65535 with an LSB of 1. Data values above or below this range will be rejected and an error message generated. If you desire to program this card using a different format or data range you may either use a Set Format (SF) instruction to reformat the card or permanently select a different format by changing jumpers on the card (see 69731A card manual).



69731A Connector



69731A Block Diagram

7-48 Example 7-16 illustrates using an OP instruction to program all bits high on a digital output card in slot 8 and all bits low on digital output card in slot 11. Since addresses and data are both fixed in this example they are sent out as constances in a literal field. After the OP instruction has completed, the next instruction (if any) can start if the Multiprogrammer is operating in the serial mode. If the Multiprogrammer is operating in the parallel mode, SRQ will be set when the OP instruction completes. The time taken for instruction completion is mainly determined by the internal timing circuits on the card if jumper W10 is installed, or by an external device if W10 is removed and the gate/flag lines are connected to the device.

NOTE

The Output Parallel (OP) instruction used in examples 16 and 18 is a typical instruction used to program this card. The actual instruction to use will be determined by the application. For example, a Write and Cycle (WC) instruction can be used if the Multiprogrammer is in serial mode and you do not require subsequent instructions to wait until the programmed card completes its data transfer. Or, as another example, cards that are to be externally triggered would first be programmed by a Write Frist Rank (WF) instruction to load output data into first rank storage on the card. Then when an external trigger is subsequently applied to the card, data will be transferred from first rank storage to the output of the card and a gate/flag sequence will be initiated.

Example 7-16. Programming Two Digital Output Cards

9825A Controller

```
0: wrt 723,"OP,8,65535,11,0T"
```

9835/45 Controllers

```
10   OUTPUT 723;"OP,8,65535,11,0T"
```

7-49 Data contained in first rank storage on the digital output card can be read back from sub-address 3 at any time by using a Read Value instruction. If the card has been cycled, either by an instruction such as an "OB", "OP", "OS", "OI", "CY", "WC", or by an external trigger, the first rank data will be the same as the data present at the output of the card. Examples 7-17 illustrates reading first rank data from a card in slot 8 into variable "A" in the controller

Example 7-17. Reading First Rank Data

9825A Controller

```
0: wrt 723,"RV,8,3T"
1: red 72306,A
```

9835/45 Controllers

```
10   OUTPUT 723;"RV,8,3T"
20   ENTER 723,06;A
```

7-50 Example 7-18 is a sample test program which is intended to give you some hands-on experience in programming a digital output card. This example assumes the digital output card is installed in slot 8 and allows you to specify an output bit number and program a logic level from the associated bit. Jumper W10 should either be installed when running the sample program or you may connect pins "k" and "l" on the output connector. Data is sent out in the form of a variable. Any attempt to program a data value that is out of range will result in an error message being printed (9825) or displayed (9835/45) and the program will be terminated.

7-51 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter bit number" displayed. Enter a bit number from 0 to 15 and depress CONTINUE.

3. If the program has been correctly entered into the controller, the controller will stop with "test bit" displayed. At this point you may use a voltmeter to check the output level of the individual bits. The voltage measured between the lettered pin corresponding to the bit number entered in step 2 (as shown on the 69731A output connector diagram) and pin "r" (common) should be approximately +5 volts. The voltage measured between the other lettered data output pins and pin "r" should be approximately 0.1 volts.

Example 7-18. 69731A Sample Test Program

9825A Controller

```
0: ent "enter bit number";B
1: wrt 723;"OP,8",2*B,"1"
2: ssb "cheker"
3: if V=0;dsp "test bit"
4: end
*12383
```

9835/45 Controllers

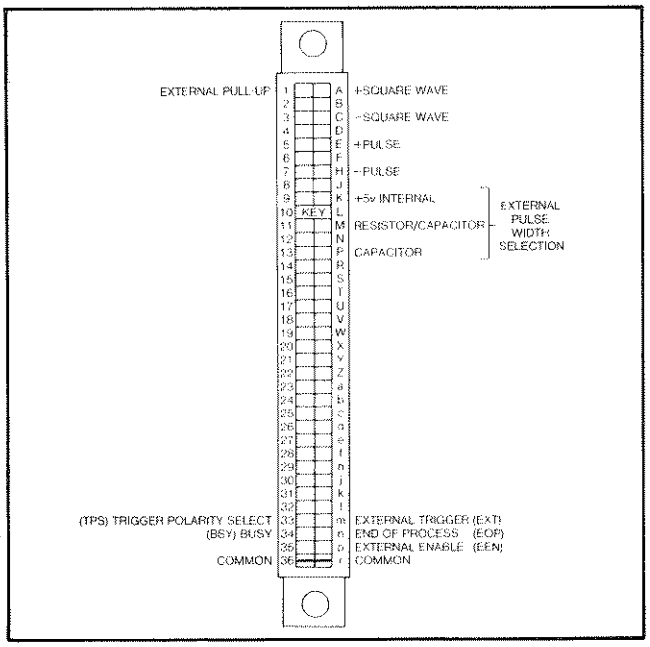
```
10   OPTION BASE 1
20   DIM R(1)
30   INPUT "ENTER BIT NUMBER";B
40   OUTPUT 723;"OP,8",2*B;"1"
50   GOSUB Cheker
60   IF V=0 THEN DISP "TEST BIT"
70   END
```

7-52 Pulse Train Output Card, 69735A

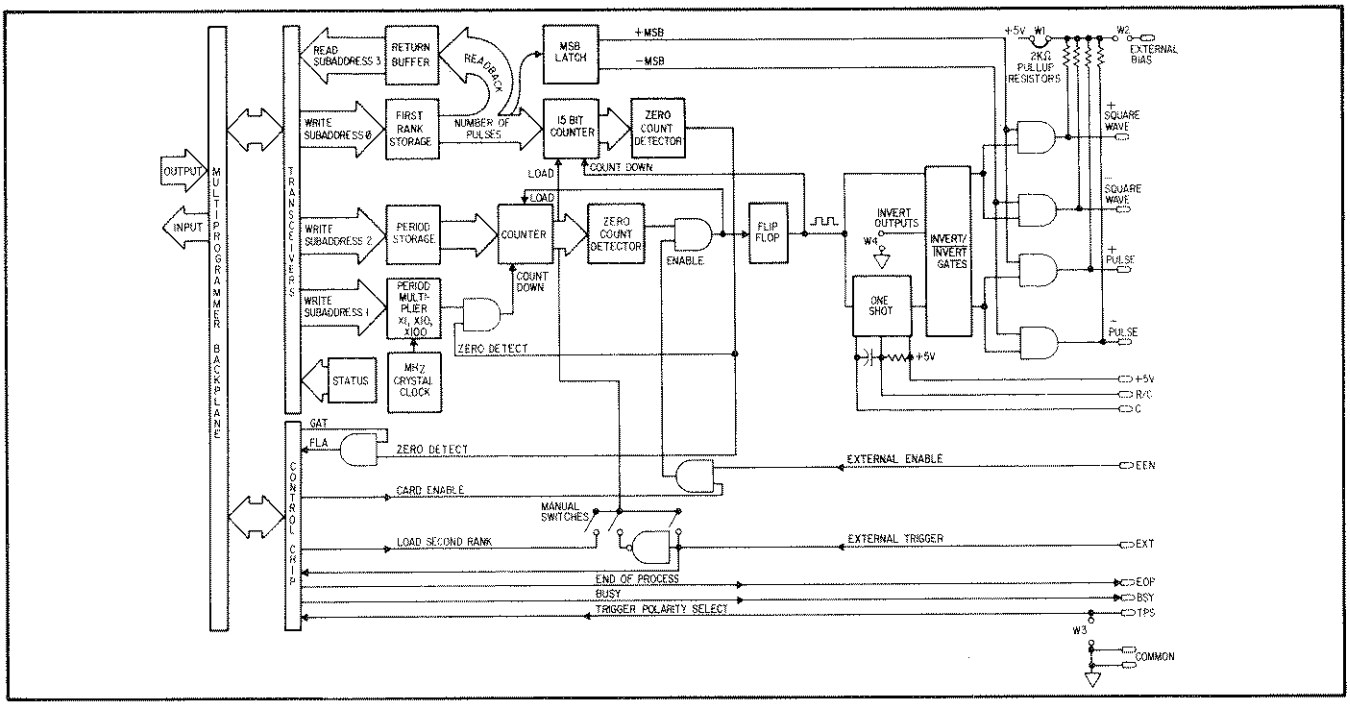
7-53 This card can be programmed to generate from 1 to 32767 square wave cycles of programmable duration and output pulses of fixed (or user selectable widths at programmable intervals. The square waves and pulse outputs are switched between two sets of output terminals as a function of the polarity of the output pulses programmed. The starting time of the output can be determined by a controller command or by external triggering. When applied to a stepping motor translator, these pulses are converted to clockwise and counterclockwise drive pulses for an associated stepping motor. The square wave or pulse outputs can also be used for pulse-train update of supervisory control stations.

7-54 Three sub-addresses are used to program an output from this card.

1. Sub-address 0 (the main address) is used to specify the number of output pulses and select the appropriate output terminals. Decimal values between −32767 and +32767 are accepted.



69735A Connector



69735A Block Diagram

Positive values will generate square wave outputs from pin A and pulse outputs from pin E, while negative values will generate square wave outputs from pin C and pulse outputs from pin H.

2. Sub-address 1 is used to set an internal period multiplier used in conjunction with sub-address 2 to determine the programmable period (from 1usec to 6.5535s) of the output square wave (or time interval between pulses). Data values of 0, 1, or 2 sent to this sub-address will set the period multiplier to 1, 10, or 100, respectively. At system turn-on time, the card will wake up with the period multiplier preset to 1 and sub-address 1 need not be programmed at all unless you require a different multiplier.

3. Sub-address 2 is used to specify the period magnitude of the output square wave in microseconds. Decimal values between 1 and 65535 will be accepted. The actual programmable period is determined by the period magnitude sent to sub-address 2 multiplied by the period multiplier stored in sub-address 1.

7-55 When programming this card, data values above or below the maximum range allowed for the particular sub-address being programmed, except for sub-address 1, will be rejected and an error message generated. Sub-address 1 is an internal octal register which will accept any octal value up to 177777 but should only be programmed with a 0, 1, or 2 as noted previously.

7-56 Example 7-19 illustrates programming square wave and pulse outputs from the positive output terminals of a pulse train card installed in slot 8. The output from pin "A" will be 30000 square wave cycles with a period of 200 microseconds each and the output from pin "E" will be 30000 one microsecond pulses at 200 microsecond intervals. Sub address"1" is not programmed since the period multiplier used for this example is 1. Since the address, period, and number of output pulses are fixed in this example, they are sent out as constants in a literal field.

7-57 Notice that a write first rank instruction, "WF", is used to program sub-address 2 and an "OP" instruction is used to program the main address. Generally, the way to program this card is to use a "WF" instruction to program sub-addresses 1 and 2 and an "OP", "OS", or "WC" instruction to program the main address. The "OP", "OS", or "WC" instructions serve the dual purpose of loading the number of output pulses into sub-address 0 and cycling the card.

7-58 Although the output from the card programmed in example 7-19 will take only 6 seconds to complete, it is possible to program an output from the pulse train card that would take over 59 hours to complete. Remember, if the pulse train card is programmed by an "OP" or an "OS" instruction, and the Multiprogrammer is in serial mode, all subsequent instructions sent to the Multiprogrammer must wait until the pulse train card completes before they are executed. If the Multiprogrammer is operating in parallel mode, subsequent instructions of a different type will be allowed to run concurrently with the instruction that has programmed the card. In parallel mode an "OP" or "OS" instruction will set SRQ to notify the controller when the pulse train output has been completed. In either serial or parallel mode a "WC" instruction may be used to initiate a pulse train output when you do not want to hold up subsequent instructions and you do not require a service request upon completion of the pulse train.

Example 7-19. Programming Outputs From the Positive Terminals

9825A Controller

```
wrt 723,"WF,8,2,200T,OP,8,30000T"
```

9835/45 Controllers

```
OUTPUT 723;"WF,8,2,200T,OP,8,30000T"
```

7-59 Example 7-20 illustrates programming square wave and pulse outputs from the negative output terminals of a pulse train card intstalled in slot 8. The output from pin "C" will be 100 square wave cycles with a period of 5 seconds each and the output from pin "H" will be 100 one microsecond pulses at 5 second intervals. In this example the output period/pulse time interval is determined by multiplying the period value in sub address 2 (50000 microseconds) by the range multiplier stored in sub-address 1 (data value of 2 = internal multiplier of 100).

Example 7-20. Programming Outputs From the Negative Terminals

9825A Controller

```
0: wrt 723,"WF,8,1,2,8,2,50000T"
1: wrt 723,"OP,8,-100T"
```

9835/45 Controllers

```
10 OUTPUT 723;"WF,8,1,2,8,2,50000T"
20 OUTPUT 723;"OP,8,-100T"
```

7-60 After the sub-addresses of the pulse train card have been programmed with the required data the output pulse train may be re-initiated at any time by sending a Cycle instruction to the card (i.e. CY,8T for the previous examples) or applying an external trigger between pins "m" and "r" (common). This is possible because data that has been sent to the main address (sub-address 0), sub-address 1, and sub-address 2 will be stored until (1) the sub-address is reprogrammed, (2) a system clear is sent to the Multiprogrammer, or (3) the system is turned off. The technique of sending data to the sub-addresses then using a "CY" instruction or an external trigger to start the pulse train is similar to using the "WC" in-

struction described earlier in that subsequent system instructions will not be held up in serial mode and SRQ will not be set upon completion of the pulse train in parallel mode. If this is the approach you would like to use, note that the "OP" instructions used in the previous examples can be replaced with "WF" instructions. This will load the sub-address registers without actually starting the pulse train. The pulse train can then be started at any time by the Cycle instruction or by an external trigger.

7-61 One final method of programming the pulse train card is to use an Output Interrupt (OI) instruction. "OI" instructions run concurrently in serial or parallel mode, always set SRQ when a card assigned to an "OI" instruction completes, and never hold up subsequent instructions. Therefore, changing the "OP" to an "OI" instruction in the previous examples will allow you to use the Multiprogrammer in serial mode if desired, run other instructions while the pulse train is being sent out, and receive an SRQ upon completion of the pulse train. When an "OI" instruction sets SRQ you should read back from extended talk address 09 to determine the address of the interrupting card and release the internal memory reserved for the "OI". For example, red 72309,r1,r2,r3,etc (9825) or ENTER 723.09;R1,R2,R3,etc(9835/45).

7-62 Example 7-21 is a sample test program which is intended to give you some hands-on experience in programming a Pulse Train Output card. This example assumes the card is installed in slot 5 and allows you to program from 1 to 32767 ten millisecond output cycles from either the positive or negative output terminals. It will be up to you to provide a means of monitoring the pulse train. Data to program the internal period multiplier and magnitude values is sent out as a constant in a literal field and the number of output pulses is sent out as a variable. Since we will be monitoring the service request line to determine when the "OP" instruction has completed, a Multiprogrammer status read is done to clear SRQ, if necessary, before we program an output from the card.

7-63 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter no. of pulses" displayed. Enter the number of output pulses you would like to see programmed and depress CONTINUE.

3. The Pulse Train Output card will transmit a pulse train approximately +5 volts in amplitude which can be measured as follows.

   a. If a positive value has been entered in step 2, a square wave output with a period of 10 milliseconds (100 Hz) can be measured between pins "A" and "r" (common) and a series of 1 microsecond wide positive pulses at 10 millisecond intervals can be measured between pins "E" and "r".

   b. If a negative value had been entered in step 2, the square wave output can be measured between pins "C" and "r" and the pulse outputs can be measured between pins "H" and "r".

4. In example 7-21 the Multiprogrammer is programmed to parallel mode before programming an output from the pulse train card. This will cause SRQ to be set upon completion of the "OP" instruction. In this example we wait for SRQ to be set then display "done" on the controller. Since the frequency of the output is 100 Hz the time interval between depressing CONTINUE in step 2 and displaying "done" will be approximately 1 second for every 100 pulses entered in step 2 (unless errors are detected).

7-64 If errors are detected; such as, an attempt to program more than 32767 output pulses, an error message will be printed (9825) or displayed (9835/45).

Example 7-21. 69735A Sample Test Program

9825A Controller

```
0: red 72310,r1,r2
1: wrt 723;"CP,WF,5.1,0,5.2,10000T"
2: ent "enter no. of pulses",B
3: wrt 723;"OP,5",B,"T"
4: "wait":if rds(723)#64:jmp 0
5: ssb "cheker"
6: dsp "done"
7: end
*21942
```

9835/45 Controllers

```
10 OPTION BASE 1
20 DIM R(11)
30 ENTER 723.10;R1,R2
40 OUTPUT 723;"CP,WF,5.1,0,5.2,10000T"
50 INPUT "ENTER NO. OF PULSES";B
60 OUTPUT 723;"OP,5",B,"T"
70 Wait: STATUS 723;C
80 IF C<>64 THEN Wait
90 GOSUB Cheker
100 DISP "DONE"
110 END
```

7-65 Timer/Pacer Card, 69736A

7-66 This card can be programmed to generate crystal-controlled output pulses from 1 microsecond to 65535 seconds (18.2 hrs) in duration. The card operates either in the one-shot mode (single output pulse) or the recirculate mode (continuous train of output pulses). An autoranging capability designed into Mutliprogrammer firmware ensures the best possible resolution for any output pulse width programmed. The starting time of the output pulses(s) can be determined by a controller command or by external triggering.

7-67 As shipped from the factory the format of the timer card has been set for programming in milliseconds. This format may be changed in one of two ways:

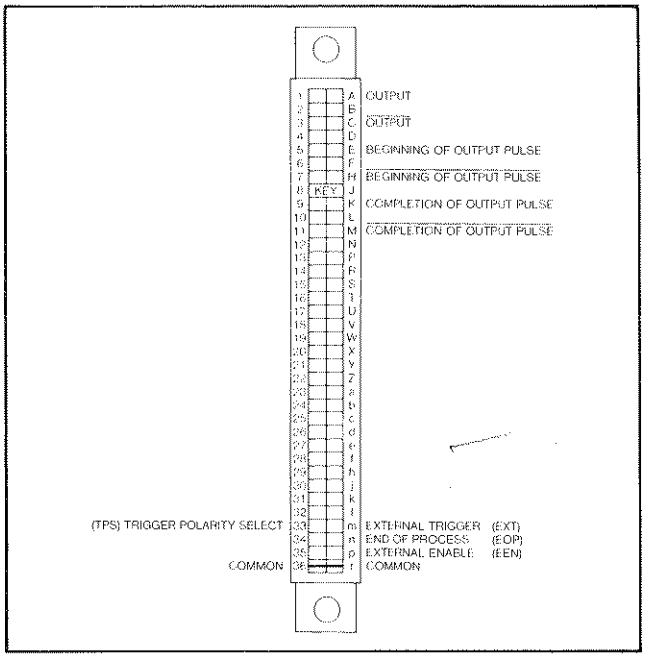
1. Appending on "S", "M", or "U" to the data that is being sent out will override the internal format and cause the data to be interpreted as seconds, milliseconds, or microseconds, respectively.

2. You may use a Set Format (SF) instruction to reformat the card or permanently select a different format by changing jumpers on the card (see 69736A card manual).

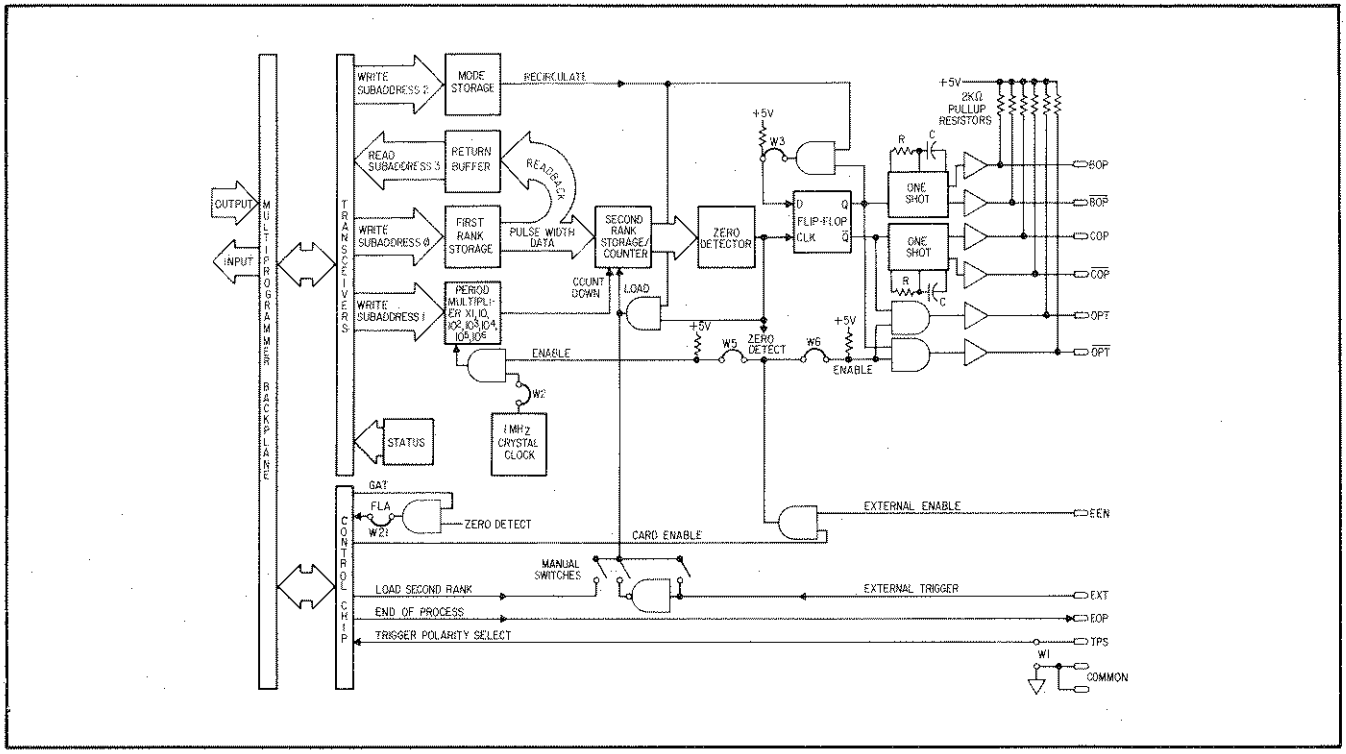
7-68 Table 7-2 is a list of the standard programming time periods, their associated symbols, and the range of data values that may be programmed for each time period.

Table 7-2. Programming Ranges and Symbols

Time Period	Symbol	Programming Range
Seconds	S	0.001-65535
Milliseconds	M	0.001-4294836.225
Microseconds	U	1-4294836



69736A Connector



69736A Block diagram

7-69 One-Shot and Recirculate Modes. This card "wakes-up" in the one-shot mode. Recirculate operation (continuous square wave) can be obtained by using a Write First Rank (WF) instruction to program sub-adress 2 with a data value of 1 before programming the output pulse. If this is done, the period between pulses will be the same length of time as the period of the pulse, thereby enabling you to obtain square wave output frequencies of from 0.00000763Hz to 500kHz. Reprogramming sub-address 2 with a value of 0 will cause the timer card to revert back to the one-shot mode of operation.

7-70 Example 7-22 illustrates programming one, 10-millisecond pulse from a timer card in slot 7. Since both the address and data value in this example are fixed, they are sent out as constants in a literal field. To program a continuous output of 10 millisecond pulses change the literal string in example 7-22 to read "WF,7.2,1T,OP,7,10T".

Example 7-22. Programming a 10 Millisecond Pulse Output

9825A Controller

```
0: wrt 723,"OP,7,10T"
```

9835/45 Controllers

```
10   OUTPUT 723;"OP,7,10T"
```

7-71 Although the output from the card programmed in example 7-22 will take only 10 milliseconds to complete, it is possible to program an output from the timer card that would take up to 18.2 hours to complete. If the timer card is programmed by an "OP" or an "OS" instruction, and the Multiprogrammer is in serial mode, all subsequent instructions sent to the Multiprogrammer must wait until the timer card completes before they are executed. If the Multiprogrammer is operating in parallel mode, however, subsequent instructions of different type will be allowed to run concurrently with the instruction that has programmed the card. In parallel mode an "OP" or "OS" instruction will set SRQ to notify the controller when the pulse output has been completed. In either serial or parallel mode, a "WC" instruction may be used to initiate a pulse output when you do not want to hold up subsequent instructions and you do not require a service request upon completion of the pulse output.

7-72 If the timer card is used in the recirculate mode, programming of the card will be considered to be finished as soon as the card receives output data. Therefore, subsequent instructions will not be held up if the Multiprogrammer is in serial mode. In parallel mode, SRQ will be set immediately after output data is received.

7-73 Example 7-23 illustrates programming one 50 microsecond pulse form a timer card in slot 7 and a 20 second pulse from a timer card in slot 8. This example illustrates programming in microseconds and seconds by using symbols to override the internal format of the card.

Example 7-23. Programming a 50 Microsecond and a 20 Second Pulse

9825A Controller

```
0: wrt 723,"OP,7,50U,8,20ST"
```

9835A Controllers

```
10   OUTPUT 723;"OP,7,50U,8,20ST"
```

7-74 After a Timer/Pacer card has been programmed with the required data, the output pulse may be re-initiated at any time by sending a cycle instruction to the card (i.e. "CY,7T" in example 7-22) or applying an external trigger between pins "m" and "r" (common). This is possible because data sent to the card will be stored until (1) the card is reprogrammed, (2) a system clear is sent to the Multiprogrammer, or (3) the system is turned off. The technique of sending data to the card then using a "CY" instruction or an external trigger to start the output pulse is similar to using the "WC" instruction described earlier in that subsequent system instructions will not be held up in serial mode and SRQ will not be set upon completion of the pulse in parallel mode. If this is the approach you would like to use, note that the "OP" instructions used in the previous examples can be replaced with "WF" instructions. This will load the output data register without actually starting the pulse. The pulse can be started at any time by the cycle instruction or an external trigger.

7-75 One final method of programming the timer card is to use an Output Interrupt (OI) instruction. "OI" instructions run concurrently in serial or parallel mode, always set SRQ when a card assigned to an "OI" instruction completes, and do not hold up subsequent instructions. Therefore, changing the "OP" to an "OI" instruction in the previous examples will allow you to use the Multiprogrammer in serial mode if desired, run other instructions while the pulse is being sent out, and receive an SRQ upon completion of the pulse. When an "OI" instruction sets SRQ, read back from extended talk address 09 to determine the address of the interrupting card and to release the internal memory reserved for the "OI"; (i.e. red 72309,r1,r2,r3,etc (9825) or ENTER 723.09;R1,R2,R3,etc (9835/45).

7-76 Example 7-24 is a sample test program which is intended to give you some hands-on experience in programming a timer card. This example assumes the card is installed in slot 7 and allows you to program a pulse output from one microsecond to 65535 seconds long. It will be up to you to provide a means of monitoring the output pulse. The width of the output pulse will be sent out as a variable. Since we will be monitoring the service request line to determine when the "OP" instruction has completed, a Multiprogrammer status read (i.e. red72310,r1,r2/ENTER 72310;R1,R2) is done at the beginning of the program to clear SRQ, if necessary, before we program an output from the card.

7-77 The fixed 3 number format used in the 9825 controller

program is required to ensure that output pulse data with 3 decimal places is not truncated.

7-78 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.

2. Depress RUN on the controller. The controller will stop with "enter pulse width" displayed. Enter the width of the desired output pulse in milliseconds and depress CONTINUE.

3. The timer card will transmit an output pulse approximately 5 volts in amplitude lasting as long as you specified in step 2. The output pulse may be observed by connecting an oscilloscope between pins "A" and "r" (common) of the timer card output connector.

4. In example 7-24 the Multiprogrammer is programmed to parallel mode before programming an output from the timer card. This will result in SRQ being set upon completion of the "OP" instruction. In this example we wait for SRQ to be set then display "done" on the controller. The time interval between depressing CONTINUE in step 2 and displaying "done" will be approximately one second for every 1000 milliseconds entered in step 2 (unless errors are detected).

7-79 If programming errors are detected by the Multiprogrammer, an error message will be printed (9825) or displayed (9835/45).

Example 7-24. 69736A Sample Test Program

```
0: fxd 3
1: red 72310,r1,r2
2: wrt 723,"GP"
3: ent "enter pulse width",B
4: wrt 723,"OP,7",B,"T"
5: "wait":if rds(723)#64;jmp 0
6: gsb "cheker"
7: dsp "done"
8: end
*19702
```

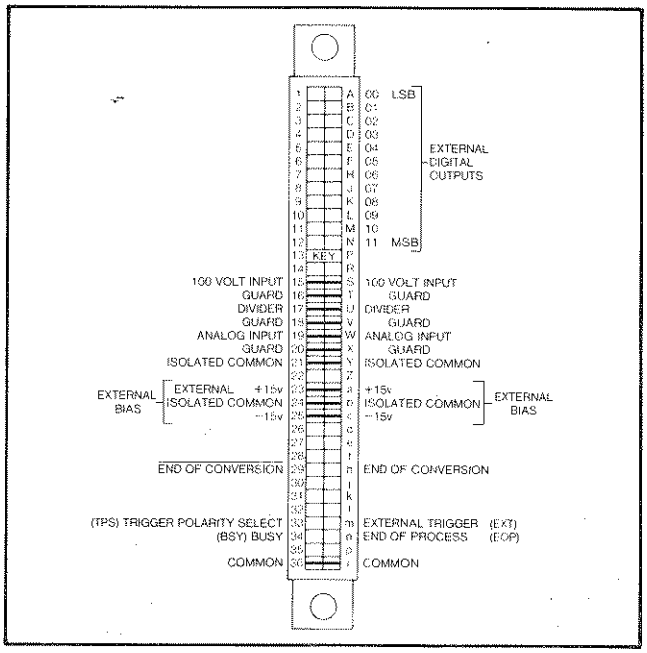
```
10   OPTION BASE 1
20   DIM R(11)
30   ENTER 723,10;R1,R2
40   OUTPUT 723;"GP"
50   INPUT "ENTER PULSE WIDTH",B
60   OUTPUT 723;"OP,7",B,"T"
70 Wait:  STATUS 723;C
80   IF C<>64 THEN Wait
90   GOSUB Cheker
100  DISP "DONE"
110  END
```

7-80 A/D Converter Card, 69751A

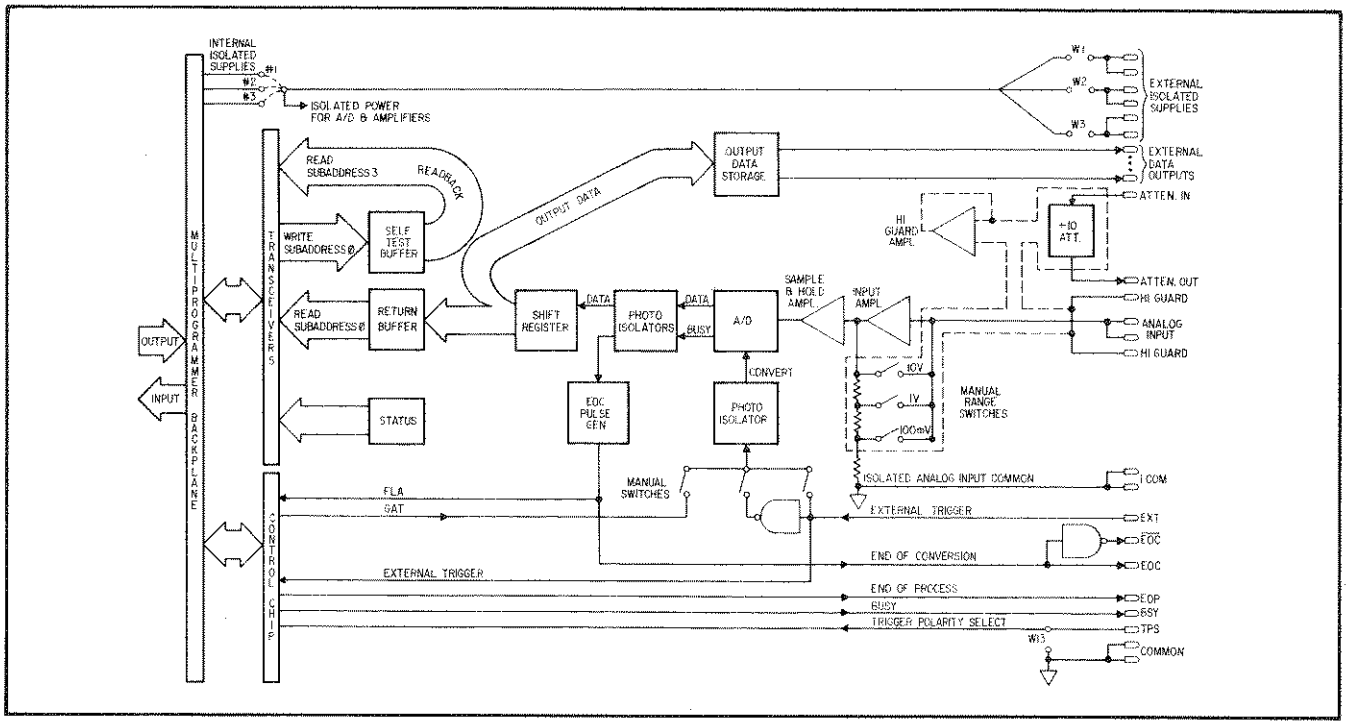
7-81 This card is a 12 bit, analog-to-digital converter used to measure bipolar voltages. Data transfer from the input edge connector to internal storage circuits on the card can be initiated by a controller command or external triggering.

7-82 As shipped from the factory, the A/D card has been set to read −10.24 to +10.235 volts with an LSB of 0.005 volts. Three alternate voltage ranges are also available (±100mV, ±1V, and ±100V). The ±100mV and ±1V ranges are selected by on-card switches while the ±100V range is obtained by connecting the input to an internal divide-by-ten altenuator. If you are using this card in a range other than the standard range (±10V) you should change the LSB of the card to correspond to the range you have set (e.g., 100 volt range requires an LSB of 0.05). This may be done either by using an "SF" instruction or "permanently" by changing jumpers in the card (see 69751A card manual).

7-83 Example 7-25 shows how to use an "IP" instruction to program two A/D converter cards, located in slots 3 and 5, to take readings. Data obtained from the readings is then read



69751A Connector



69751A Block Diagram

back into variables "A" and "B" in the controller. If the Multiprogrammer is in serial mode, the data can be read back immediately after the IP instruction has completed, as shown in the example. Or, another instruction can start and the data can be read back at a later time. If the multiprogrammer is in parallel mode,SRQ will be set when the IP instruction completes.

Example 7-25. Programming Two A/D Cards

9825A Controller

```
0: wrt 723,"IP,3,5T"
1: red 72301,A,B
```

9835A Controllers

```
10   OUTPUT 723;"IP,3,5T"
20   ENTER 723.01;A,B
```

7-84 When an "IP" instruction is used to take a voltage reading, the voltage present at the input of the card is converted to a digital value that is stored on the card. This value is also stored in the memory of the Multiprogrammer and will remain there until read back from the appropriate extended talk address (01). By using a read value (RV) instruction, data stored on the card can be read back at any time without causing the card to take a new voltage reading. This is particularly useful when an external trigger is used to initiate a voltage reading. When an external trigger is applied to the A/D card, or the card is programmed by a Cycle (CY) instruction, it will convert the input voltage to a digital value and store it on the card. It will not, however, store this data in the memory of the Multiprogrammer. The data can be read back from the card by an "RV" instruction as shown in example 7-26, which illustrates reading back data that is stored on an A/D card in slot 5.

Example 7-26. Reading Data without Taking a New Voltage Reading

9825A Controller

```
0: wrt 723,"RV,5T"
1: red 72306,C
```

9835/45 Controllers

```
10   OUTPUT 723;"RV,5T"
20   ENTER 723.06;C
```

7-85 Example 7-27 is a sample test program which is intended to give you some hands-on experience in programming an A/D card. It allows you to apply an external voltage to the input of the card then read in and display the value of the voltage. This example assumes the A/D Card is installed in slot 3.

7-86 For test purposes, subroutine "cheker" is executed immediately after sending the input instruction to check for programming errors before reading back the data.

7-87 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.

2. Apply the voltage to be measured between pins 19 (+) and 21 (common) on the input connector.

3. Depress RUN on the controller. The voltage will be measured and displayed.

7-88 If errors are detected by the Multiprogrammer an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-27. 69751A Sample Test Program

9825A Controller

```
0: wrt 723,"IP,3T"
1: wsb "cheker"
2: if V#0;sto "end"
3: red 72301,A
4: fxd 3;dsp "voltage =",A
5: "end":end
*10847
```

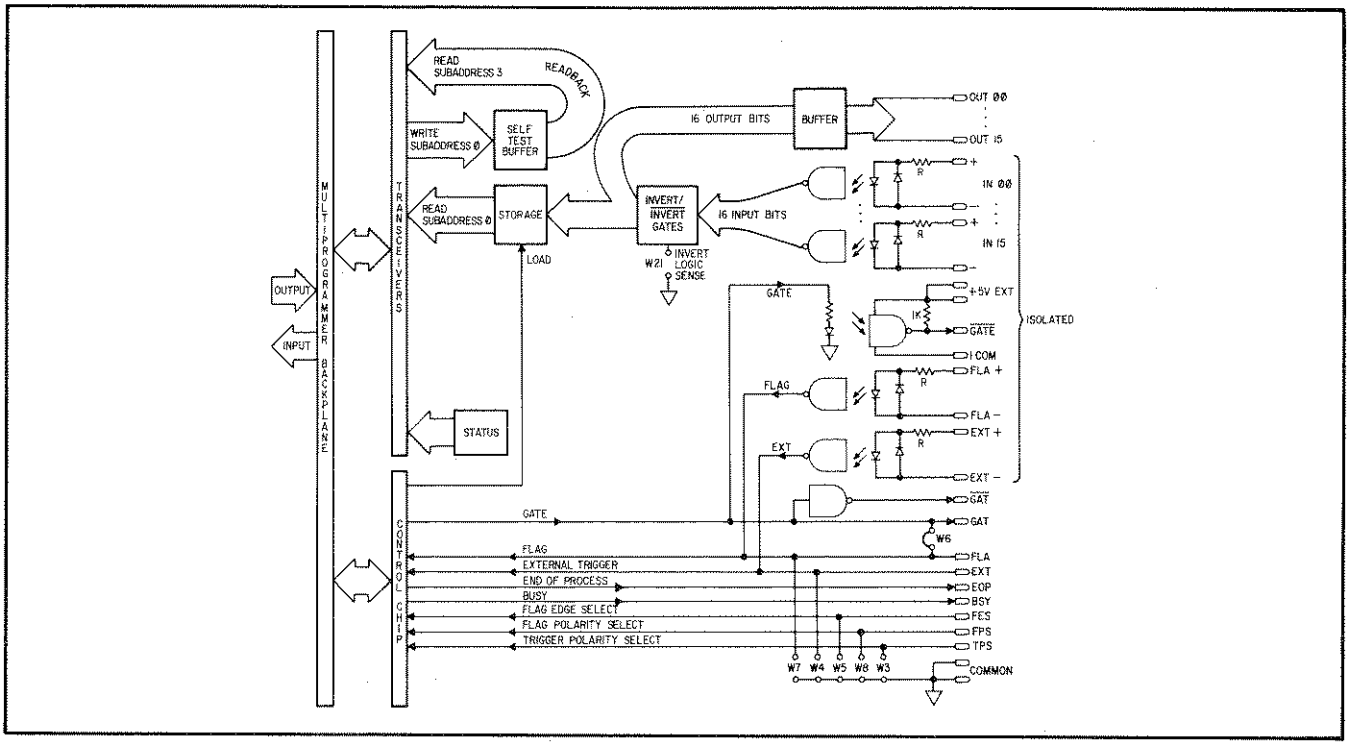
9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(1)
30   OUTPUT 723;"IP,3T"
40   GOSUB Cheker
50   IF V<>0 THEN End
60   ENTER 723.01;A
70   DISP "VOLTAGE =",A
80 End:    END
```

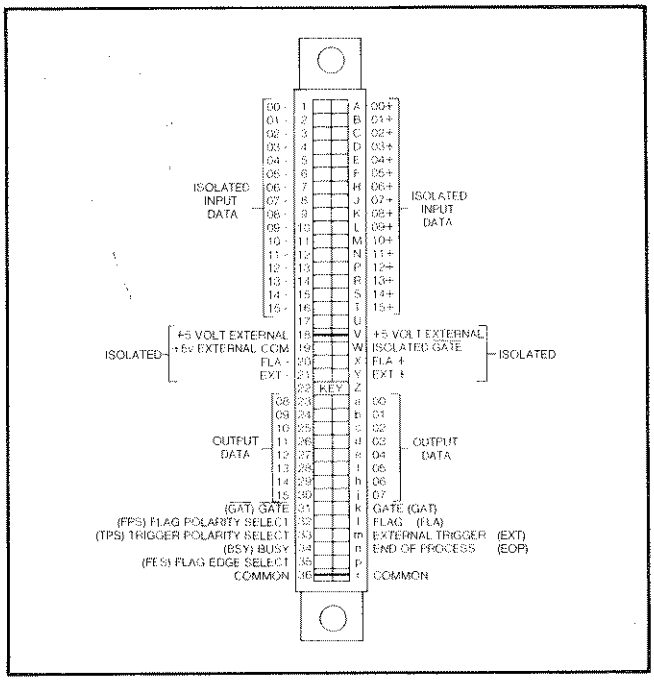
7-89 Isolated Digital Input Card, 69770A

7-90 This card allows the controller to read 16 bits of data that is isolated from ac earth ground, thereby eliminating ground loop problems in automatic test and control systems. Data transfer from the input edge connector to internal storage circuits on the card can be initiated by a controller command or external triggering. Although the gate/flag timing sequence of this card is normally controlled internally by the card, jumper W6 on the card may be removed to allow the external device to control the gate/flag data transfer sequence (see 69770A card manual).

7-91 As shipped from the factory, the format of the Isolated Digital Input card has been set to decimal. Values read from this card will be integers in the range of 0-65535 with an LSB of 1. If you perfer to read this card using a different format or input data range, you may either use a Set Format instruction (SF) to reformat the card or permanently select a different format by changing jumpers on the card (see card manual).



69770A Block Diagram



69770A Connector

7-92 Example 7-28 shows how to use an "IP" instruction to program two 69770A cards, located in slots 7 and 8, to take readings. The data is then read back into variables "A" and "B". If the Multiprogrammer is in serial mode, the data can be read back immediately after the "IP" instruction has completed, as shown in the example. Or, another instruction can be executed and the data can be read back at a later time. If the Multiprogrammer is in parallel mode, SRQ will be set when the "IP" instruction completes.

Example 7-28. Programming Two Isolated Digital Input Cards

9825A Controller

```
0: wrt 723,"IP,7,8T"
1: red 72301,A,B
```

9835/45 Controllers

```
10   OUTPUT 723;"IP,7,8T"
20   ENTER 723.01;A,B
```

7-93 When an "IP" instruction is used to take an input reading, data present at the input of the card is latched into internal storage circuits on the card. This data is also stored in the memory of the Multiprogrammer and will remain there until read back from extended talk address 01. By using a Read Value (RV) instruction, data stored on the card can be read at any time without causing the card to take a new reading. This is a particularly useful when an external trigger is used to initiate a data reading. When an external trigger is applied to the card, or the card is programmed by a cycle (CY) instruction, it will store the input data on the card. It will not, however, store this data in the memory of the Multiprogrammer. The data can be read back from the card with an "RV" instruction as shown in example 7-29 which illustrates reading back data that is stored on a card in slot 7.

Example 7-29. Reading Stored Data from a 69770A Card

9825A Controller

```
0: wrt 723,"RV,7T"
1: red 72306,C
```

9835/45 Controllers

```
10   OUTPUT 723;"RV,7T"
20   ENTER 723.06;C
```

7-94 Example 7-30 is a sample test program which is intended to give you some hands-on experience in programming an Isolated Digital Input card. It allows you to apply an input signal to an input data line, read in the resulting data, then calculate and display the pin number associated with the input signal. This example assumes the isolated digital input card is installed in slot 7. Jumper "W6" (factory connected for internal gate/flag completion) must be installed when running the sample program.

7-95 For test purposes, subroutine "cheker" is executed immediately after sending the input instruction to check for possible errors before reading back the data.

7-96 Perform the following steps when using the sample test program.

1. Load the program, including subroutine "cheker", into the controller.

2. From an external power supply, select a voltage level which corresponds to the high state of the option to be tested as specified in the 69790A manual.

3. Connect an input voltage to the input connector of the isolated input board as follows:
   a. Connect the high (positive) side of the input voltage to any pin, "A" through "T".
   b. Connect the low side of the input voltage to the numbered pin which corresponds to the lettered pin selected in step a (i.e. A-1,B-2,16-T).

4. Depress RUN on the controller. The controller will display the number of the pin connected to the low side of the input voltage. If an input voltage is not connected or is applied to more than one data line the program will end without displaying anything. In this case, you may look at the data that has been read back by displaying variable "A" on the controller.

7-97 If errors are detected by the Multiprogrammer, an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-30. 69770A Sample Test Program

9825A Controller

```
0: wrt 723,"IP,7T"
1: ssb "cheker"
2: if V#0;gto "end"
3: red 72301,A
4: if A=0;gto "end"
5: for J=0 to 15
6: if A#2↑J;next J
7: if J#16;dsp "pin number",J+1
8: "end":end
+19251
```

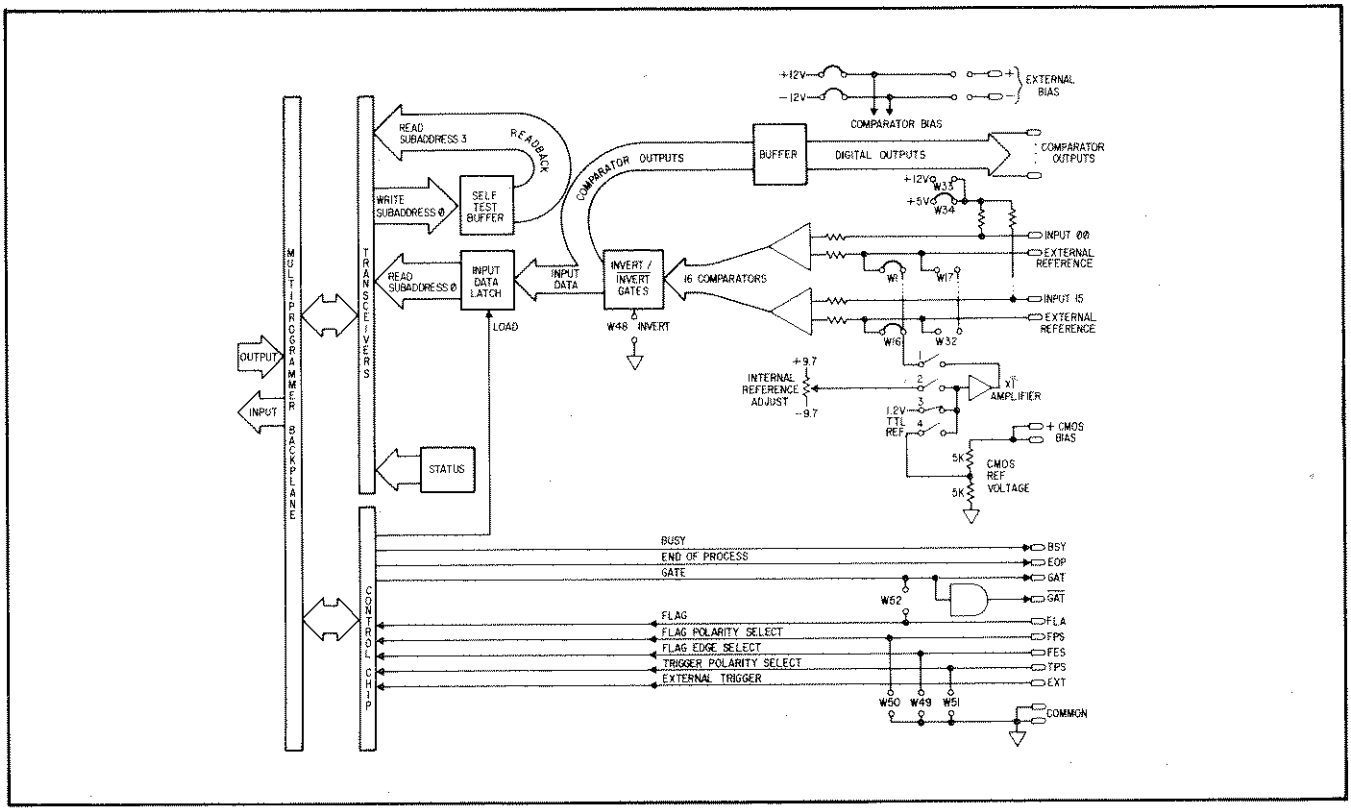
9835/45 Controllers

```
10    OPTION BASE 1
20    DIM R(11)
30    OUTPUT 723;"IP,7T"
40    GOSUB Cheker
50    IF V<>0 THEN End
60    ENTER 723.01;A
70    IF A=0 THEN End
80    FOR J=0 TO 15
90    IF A=2↑J THEN Disp
100   NEXT J
110 Disp: IF J<>16 THEN DISP "PIN NUMBER",J+1
120 End: END
```

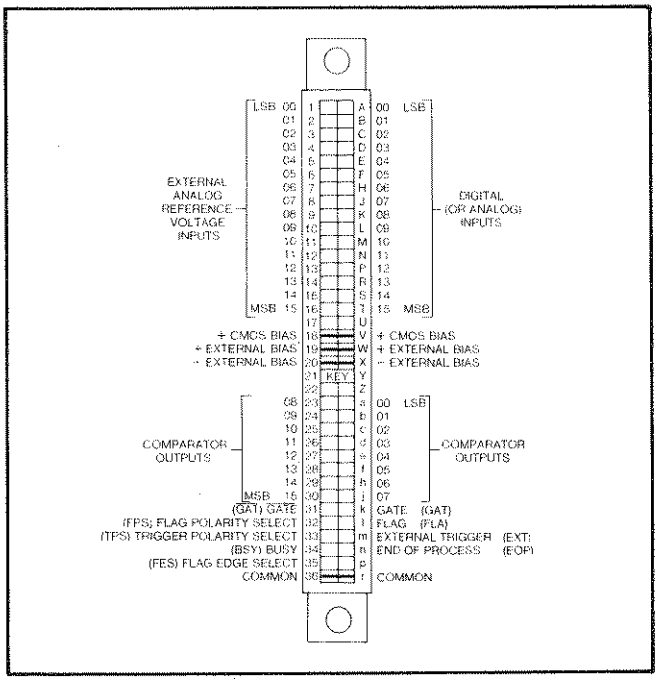
7-98 Digital Input/Analog Comparator Card, 69771A

7-99 This card is used to monitor and read back 16 bits of logic level or contact closure data that is referenced to logic common (ac earth ground). It can also measure 16 analog signals and read back an equivalent 16-bit data word. Data transfer from the input edge connector to internal storage circuits on the card can be initiated by a controller command or external triggering. A factory installed jumper (W52) can be removed to allow an external circuit to control the gate/flag sequence.

7-100 As shipped form the factory the data format of the Digital Input Card has been set to decimal. Values read from this card will be integers in the range of 0-65535 with an LSB of 1. If you prefer to read this card using a different format or input data range you may either use a Set Format (SF) instruction to reformat the card or "permanently" select a different format by changing jumpers on the card (see 69771A card manual).



69771A Block Diagram



69771A Connector

7-101 Example 7-31 shows using an "IP" instruction to program two 69771A cards, located in slots 4 and 6, to take readings. The data is then read back into variables "A" and "B". If the Multiprogrammer is in serial mode, the data can be read back immediately after the "IP" instruction has completed, as shown in the example. Or, another instruction can be executed and the data can be read back at a later time. If the Multiprogrammer is in parallel mode, SRQ will be set when the "IP" instruction completes. The time required to complete the instruction is mainly determined by the internal timing circuits on the card, if jumper W52 is installed, or by an external device if W52 is removed and the gate/flag lines are connected to the device.

Example 7-31. Programming Two 69771A Cards

9825A Controller

```
0: wrt 723,"IP,4,6T"
1: red 72301,A,B
```

9835/45 Controllers

```
10   OUTPUT 723;"IP,4,6T"
20   ENTER 723.01;A,B
```

7-102 When an "IP" instruction is used to take an input reading, data present at the input of the card is latched into internal storage circuits on the card. This data is also stored in the memory of the Multiprogrammer and will remain there until read back from extended talk address 01. By using a Read Value (RV) instruction, data stored on the card can be read at any time without causing the card to take a new reading. This is particularly useful when an external trigger is used to inititate a data reading. When an external trigger is applied to the digital input card, or the card is programmed by a cycle (CY) instruction, it will store the input data on the card. It will not, however, store this data in the memory of the Multiprogrammer. The data can be read back from the card with an "RV" instruction as shown in example 7-32 which illustrates reading back data that is stored in a Digital Input card in slot 4.

Example 7-32. Reading Stored Data From a 69771A Card

9825A Controller

```
0: wrt 723,"RV,4T"
1: red 72306,C
```

9834/45 Controllers

```
10   OUTPUT 723;"RV,4T"
20   ENTER 723.06;C
```

7-103 Example 7-33 is a sample test program which is intended to give you some hands-on experience in programming a Digital Input card. It allows you to short-out an input data line, read in the resulting data (see note), then calculate and display the pin number associated with the shorted line. This example assumes the card is installed in slot 4. Jumper W52 must be installed when running the program.

NOTE

Since leaving all data input lines to the card unterminated results in an input data word of 65535, the data word read in by the card will equal 65535 minus the value of the shorted bit.

7-104 For test purposes, subroutine "cheker" is executed immediately after sending the input instruction to check for possible errors before reading back the data.

7-105 Perform the following steps when using the sample test program.

1. Load the program, including subroutine "cheker", into the controller.

2. Temporarily install a jumper wire between pin "r" and any data input pin (A through T) on the input connector.

3. Depress RUN on the controller. The controller will display the number of the pin physically opposite to the lettered pin shorted to pin "r". If no pin has been shorted to pin "r" or more than 1 pin is shorted to pin "r" the program will end

without displaying anything. In this case you may look at the data that has been read back by displaying variable "A" on the controller.

7-106 If errors are detected by the Multiprogrammer an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-33. 69771A Sample Test Program.

9825A Controller

```
0: wrt 723,"IP,4T"
1: gsb "cheker"
2: if V#0;gto "end"
3: red 72301,A
4: if A=65535;gto "end"
5: for J=0 to 15
6: if A#65535-2↑J;next J
7: if J#16;dsp "pin number",J+1
8: "end":end
*1379
```

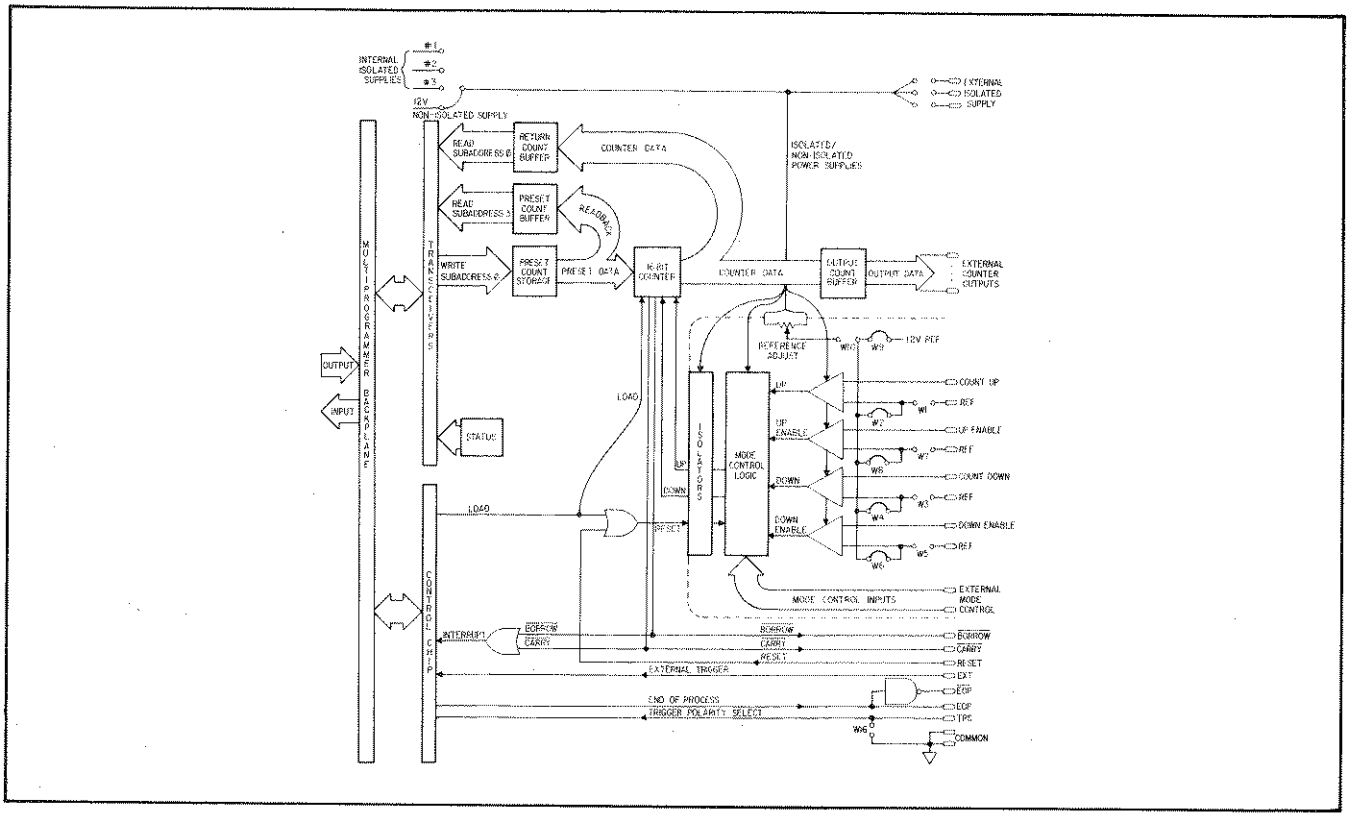
9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"IP,4T"
40   GOSUB Cheker
50   IF V<>0 THEN End
60   ENTER 723,01;A
70   IF A=65535 THEN End
80   FOR J=0 TO 15
90   IF A=65535-2↑J THEN Disp
100  NEXT J
110 Disp: IF J<>16 THEN DISP "PIN NUMBER",J+1
120 End: END
```

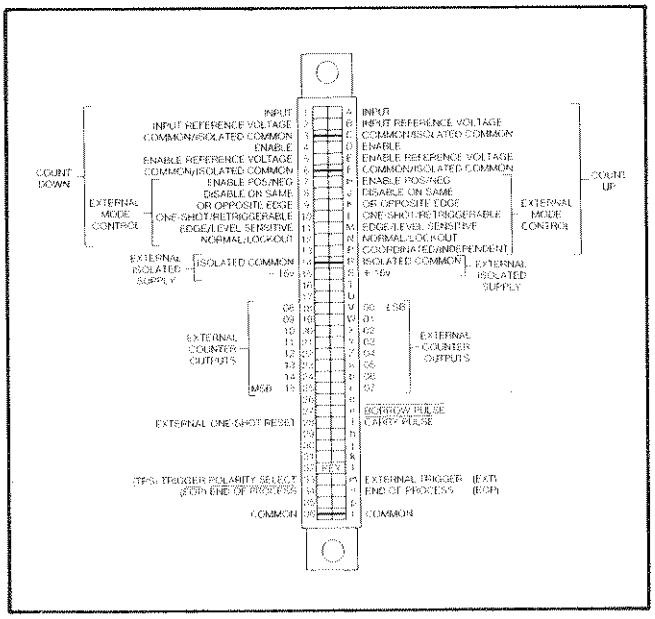
7-107 Counter/Totalizer Card, 69775A

7-108 This card is used to count pulses, contact closures, or analog transistions in the range of 0 to 65535. Up-down counting may be done simultaneously or individually. A carry or borrow pulse is generated as the count goes above 65535 or below 0, allowing multiple Counter/Totalizer cards to be cascaded for greater counting capability.

7-109 As shipped from the factory, the card has been preset for counting positive true input pulses. Both enable lines are also positive true and therefore, pulses will be



69775A Block Diagram



69775A Connector

counted when a positive signal is applied to the appropriate enable input. Through the use of switches on the card many options are available to you. For instance, counting may be disabled in either direction, edge-sensitive operation may be selected and the active polarities of the enable and disable input signals may be reversed. For a complete description of all the options available, see the 69775A card manual. All programming examples and descriptions in this section will assume the card is being used in the mode preset at the factory.

7-110 Programming the Counter/Totalizer card consists of three basic steps, (1) presetting the count, (2) cycling the card, and (3) reading the count. Presetting and cycling the card may be done separately or combined in one instruction.

1. Presetting the count loads a reference count into a preset register on the card. The reference count may be any data value within the range of the card (0 to 65,535).

2. Cycling the card copies the data from the preset register to the main counter. The card may be cycled by an instruction from the controller or by an external trigger.

3. Reading the count retrieves data from the card's counter.

7-111 Example 7-34 illustrates presetting and cycling a counter card in slot 2. A Write First Rank (WF) instruction is used to load a data value of zero into the preset register and a Cycle instruction (CY) is used to cycle the card. The cycle instruction can be omitted if an external trigger is used to cycle the card.

Example 7-34. Presetting and Cycling a Counter Card

9825A Controller

```
0: wrt 723,"WF,2,0T"
1: wrt 723,"CY,2T"
```

9835/45 Controllers

```
10  OUTPUT 723;"WF,2,0T"
20  OUTPUT 723;"CY,2T"
```

7-112 As mentioned previously, presetting and cycling the card may be combined in a single instruction. Example 7-35 illustrates using a Write and Cycle (WC) instruction to preset and cycle a Counter card installed in slot 2.

Example 7-35. Presetting and Cycling With One Instruction

9825A Controller

```
0: wrt 723,"WC,2,0T"
```

9835/45 Controllers

```
10  OUTPUT 723;"WC,2,0T"
```

7-113 Once the card has been preset and cycled, it is ready to begin counting. As shipped from the factory, positive pulses applied to the count up or count down inputs of the counter card will be counted whenever a positive signal is applied to the appropriate count enable input(s). You can read the contents of the counter at any time. Example 7-36 illustrates reading the contents of the counter into variable "A" of the controller.

Example 7-36. Reading the Counter Card

9825A Controller

```
0: wrt 723,"RV,2T"
1: red 72306,A
```

9835/45 Controllers

```
10  OUTPUT 723;"RV,2T"
20  ENTER 723.06;A
```

7-114 In example 7-36 the contents of the counter are obtained by reading from the main address (sub-address 0) of the card. If required, the preset value of the card can also be obtained simply by reading sub-address 3. Changing the literal string in the first line of example 7-36 to "RV,2.3T" would read the preset register instead of the counter. This value would then be stored in the controller by the red/ENTER statement in the second line of the example.

7-115 You have probably noticed that instructions that await card completion were not used in the first three examples. This is because the counter card completes when the counter overflows past 65535 and generates a carry or when it underflows past zero and generates a borrow. Since instructions that require a card to complete will be tied up until the borrow or carry occurs you should avoid using this type of instruction unless you expect the overflow or underflow to occur within an acceptable period of time. There may be occassions when you are operating the Multiprogrammer in serial mode and you would like to have the system wait until the counter card generates a borrow or carry before allowing subsequent instruction to run. There may also be occasions when you are operating the Multiprogrammer in parallel mode and you would like a service request to let you know when the counter card has generated a borrow or carry. In both of these instances you can substitue an "OP" for the WC" shown in Example 7-35.

7-116 Example 7-37 is a sample test program that allows you to feed external pulses to the counter card's count-up input and then reads the counter and displays the number of pulses that were received. You must either provide the external pulses to be counted from a good external source (not switches or mechanical relays) or, if you have other Multiprogrammer cards available, you can use the "BUSY" signal available on pin "34" of the appropriate edge connector. Multiprogrammer cards that can be used to provide a pulse via the "BUSY" signal are the 69700-69706A, 69720A, 69721A, 69730A, 69731A, 69751A, and the 69771A. Programming these cards with a cycle (CY) instruction will cause their "BUSY" signal to toggle, providing a pulse that can be counted. The counter card is installed in slot 2 and if another card is being used to provide a pulse via the "BUSY" signal it should be installed in slot 3.

7-117 For test purposes, subroutine "cheker" is executed immediately after sending the input instruction to check for possible errors before reading back the count.

7-118 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.
2. When using an external source to supply pulses to be counted, connect the output from the pulse source to pin "A" on the counter card edge connector and the pulse source common to pin "C" on the edge connector. The amplitude of the external pulse should be approximately +5 volts.
3. When using the "BUSY" signal from a card in slot 3 to generate the pulses to be counted connect jumper wires from pins "34" and "36" on the edge connector of the card in slot 3 to pins "A" and "C", respectively, on the edge connector of the counter card.

NOTE

The count enable lines normally float high when no input is applied thereby enabling the counter inputs. For this reason, no connections are made to the enable lines when running example 7-37.

4. Depress RUN on the controller. The controller will display "enter pulses". Send from 1 to 65535 pulses from your external source to the counter card, or if using the "BUSY" signal from a card in slot 3, type wrt 723,"CY,3T" on a 9825 controller or OUTPUT 723;"CY,3T" on a 9835/45 controller. Depress EXECUTE, RECALL as many times as desired. Each time EXECUTE is depressed the "BUSY" line on the card in slot 3 will be toggled resulting in a pulse that will be counted. (Although the "BUSY" line goes low when the card goes busy the pulse will be counted when the card completes its cycle and the "BUSY" line returns to its high state.)
5. Depress CONTINUE on the controller. If no errors are detected the controller will display the number of pulses counted in step 4.

7-119 If errors are detected by the Multiprogrammer an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-37. 69775A Sample Test Program

9825A Controller

```
0: wrt 723,"WC,2,0T"
1: dsp "enter pulses":stp
2: wrt 723,"RV,2T"
3: gsb "cheker"
4: if V#0:sto "end"
5: red 72306,A
6: dsp "pulse count =",A
7: "end":end
*10110
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"WC,2,0T"
40   DISP "ENTER PULSES"
50   PAUSE
60   OUTPUT 723;"RV,2T"
70   GOSUB Cheker
80   IF V<>0 THEN End
90   ENTER 723.06;A
100  DISP "PULSE COUNT =",A
110 End: END
```

7-120 Interrupt Card, 69776A

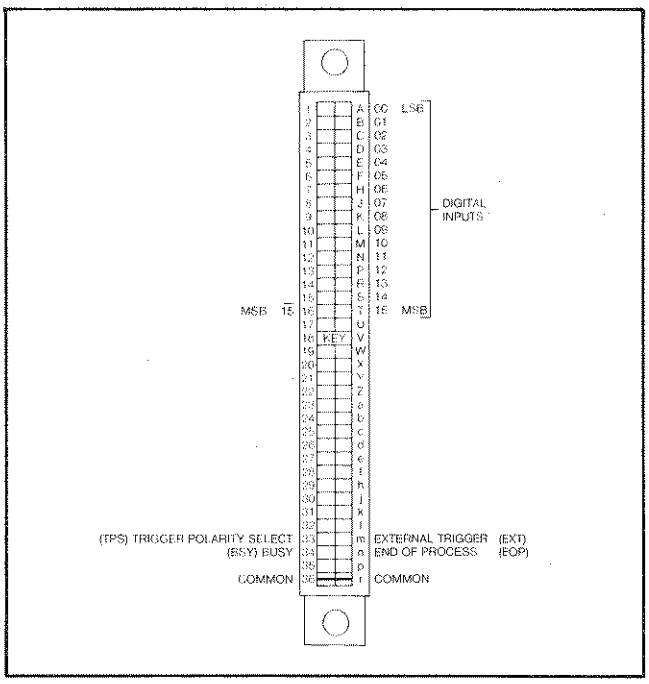
7-121 This card compares an internal 16-bit reference word with an externally applied input word and generates an interrupt when a pre-determined relationship exists. This internal (microprocessor) interrupt can be programmed to occur when the input word is unequal to, equal to, greater than, or less than the reference word. In addition, a mask word may be sent to the card to remove 1 or more bits from the comparison without changing the reference word.

7-122 All four sub-addresses are useful when programming this card. The functions of these sub-addresses will vary depending on whether you are sending data to, or reading data from the card. When sending data to the card:

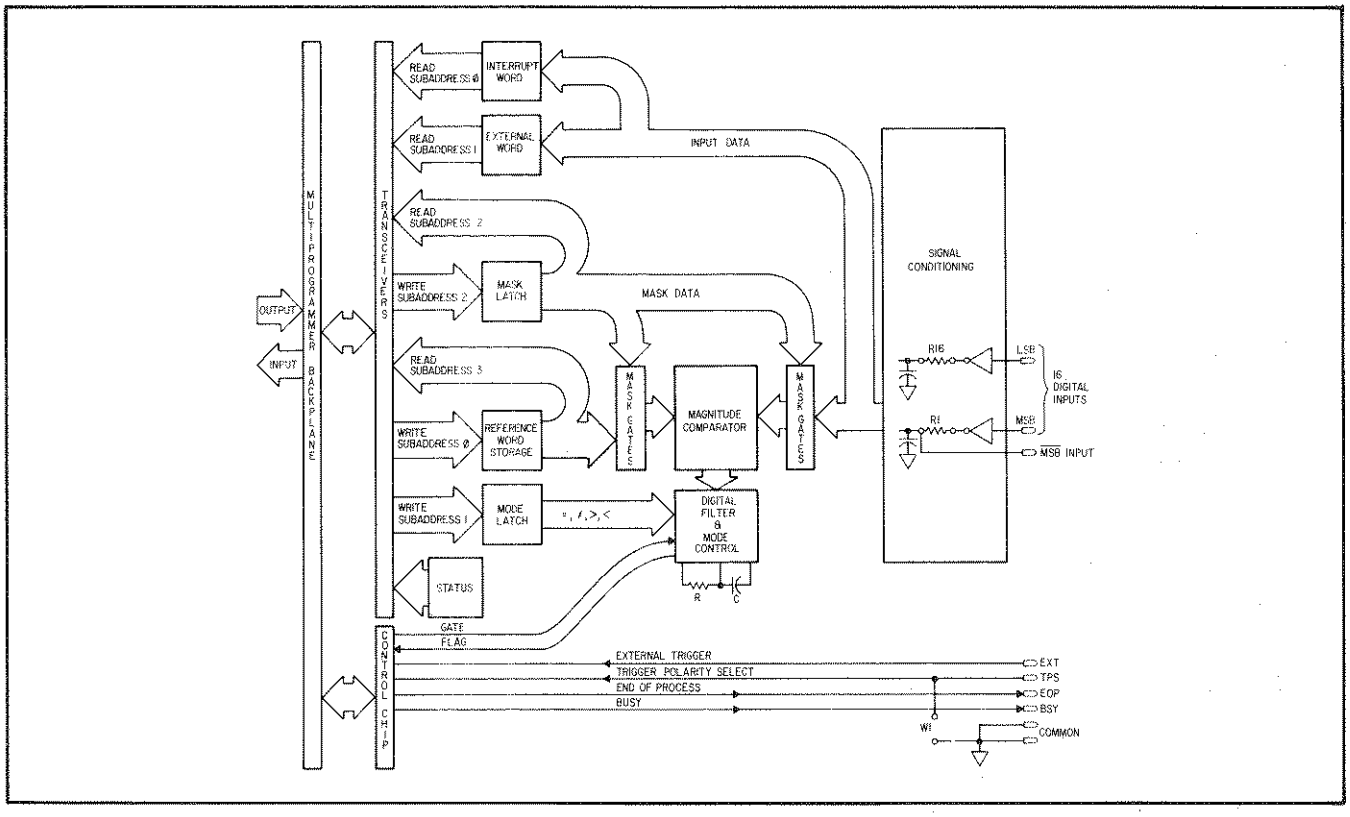
1. Sub-address 0 (the main address) is used to send a reference word to the card. The reference word must be a decimal integer between 0 and 65535.

2. Sub-address 1 is used to program the interrupt mode. Data values of 1, 2, 4, or 10 will select an interrupt mode of "not equal to", equal to", "greater than", or "less than", respectively.

3. Sub-address 2 is used to send a mask value to the card. The mask value is used to remove (or restore) selected bits from the comparison that takes place between the reference word and the external input word. Setting a bit (or bits) to a value of one in the data sent to sub-address 2 will remove the corresponding bit on the card from the comparison. Conversely, setting a bit to a value of zero in the data sent to sub-address 2



69776A Connector



69776A Block Diagram

will restore the corresponding bit on the card to the comparison. At system turn-on, the value in sub-address 2 is preset to zero, thus allowing all input bits to be included in the comparision. Since a mask value will generally not be a calculated value, sub-address 2 is formated to accept octal values to allow you to easily visualize the bit you would like to remove or restore to the comparison. Data sent to sub-address 2 must be an octal integer between 0 and 177777.

4. When sending data to this card, data values above or below the range allowed for the particular sub-address being programmed, except for sub-address 1, will be rejected and an error message generated. Sub-address 1 is an internal octal register which will accept any octal value up to 177777 but should only be programmed with a 1, 2, 4, or 10 as previously noted.

7-123 When reading data from the card:

1. Sub-address 0 is used to read the value of the external word that caused the interrupt. This value is stored on the card at the time the interrupt occurs.

2. Sub-address 1 can be used to read the input word present on the edge connector at any time.

3. Sub-address 2 can be used to read the value currently in the mask register.

4. Sub-address 3 can be used to read the reference word that was previously sent to the card.

7-124 Example 7-38 illustrates programming an interrupt Card in slot 4 to interrupt whenever the value of an external input word being monitored is greater than a reference value of 1000. Notice that a Write First Rank (WF) instruction is used to program the reference word (1000) and the mode of the card (4). Generally, the way to program this card is to use a "WF" instruction to send reference, mode, and mask data to sub-addresses 0,1, and 2, followed by an Input Interrupt (II) instruction to sub-address 0 to allow the card to set SRQ as soon as the selected mode comparison occurs and generates the interrupt. (If you are planning to program two or more cards with an "II" instruction, be sure to read Chapter 5, "Interrupt Instructions".)

7-125 Since the slot address, reference word, and mode are fixed values in this example, they are sent out as constants in a literal field. There will be some instances, however, when you may want to send out one or more of these values as a variable. For example, you may find that your reference word needs to be a variable. In this case, changing the output string in example 7-38 to "WF,4",A,"4.1,4T,II,4T" would allow you to send out a variable reference word value. As indicated the value of the reference word is contained in variable "A".

Example 7-38. Programming an Interrupt Card

9825A Controller

```
0: wrt 723,"WF,4,1000,4,1,4T,II,4T"
```

9835/45 Controllers

```
10   OUTPUT 723;"WF,4,1000,4,1,4T,II,4T"
```

7-126 As mentioned previously, you may remove selected bits from the reference word/input word comparison by programming sub-address 2 with a mask value. Example 7-39 illustrates programming a mask to remove bit 0 (LSB) from the comparison set up in Example 7-38. Since bit 0 will no longer be included in the comparison, the interrupt card programmed in the first example will now interrupt whenever a value of 1001 is exceeded. To restore bit 0 to the comparison it is only necessary to change the data value in Example 7-39 from a one to a zero.

Example 7-39. Programming a Mask Value

9825A Controller

```
0: wrt 723,"WF,4,2,1T"
```

9835/45 Controllers

```
10   OUTPUT 723;"WF,4,2,1T"
```

7-127 Referring back to Example 7-38, when the value of the external input word becomes greater than 1000, the Interrupt card generates an internal interrupt to the Multiprogrammer microprocessor. Because an "II" instruction was used to program the card, the Multiprogrammer will then set SRQ to notify the controller that a card programmed by an "II" instruction has completed. By use of the proper extended talk address (02) you may then read back the data from the "II" instruction, which will consist of the address of the interrupting card followed by the data value of the external word that caused the interrupt. Example 7-40 illustrates this read back procedure. After the SRQ line has been set by the Multiprogrammer, SRQ status is read back to the controller from extended talk address 10 and stored in variables "A" and "B". If bit 3, (decimal value of 8) of variable "A" is set, it indicates that a card in an "II" instruction has interrupted and the address and data associated with the card are read back from extended talk address 02 and stored in variables "C" and "D", respectively.

Example 7-40. Reading Interrupting Data from an Interrupt Card.

9825A Controller

```
0: red 72310,A,B
1: if bit(3,A)=1;red 72302,C,D
```

9835/45 Controllers

```
10   ENTER 723.10;A,B
20   IF BIT(A,3) THEN ENTER 723.02;C,D
```

7-128 By using a Read Value (RV) instruction you may read back data from any sub-address on the card. Example 7-41 illustrates reading the data contained in sub-addresses 0, 1, 2, and 3 into controller variables A, B, C, and D.

Example 7-41. Reading Sub-addressses 0, 1, 2, and 3

9825A Controller

```
0: wrt 723,"RV,4,4.1,4.2,4.3T"
1: red 72306,A,B,C,D
```

9835/45 Controllers

```
10  OUTPUT 723;"RV,4,4.1,4.2,4.3T"
20  ENTER 723.06;A,B,C,D
```

7-129 Example 7-42 is a sample test program which is intended to give you some hands-on experience in programming an Interrupt card installed in slot 4. The data input lines to the card are left unterminated resulting in an input data word of 65535. The card is programmed with a reference word of 65535 and an interrupt mode of 1 (unequal). Momentarily shorting any data input line to common will change the value of the input word and generate an interrupt.

NOTE

In Example 7-42, permanently connecting a data input line to common will result in an immediate interrupt when you run the program. Since subroutine "cheker", used to detect programmming errors, reads Multiprogrammer SRQ status and clears the SRQ line, the Multiprogrammer "completed instruction status" variable returned by "cheker" is tested to see if an "II" instruction has completed. If it has, the program immediately reads and displays the card address and interrupting word.

7-130 Perform the following steps when using the sample program.

1. Load the program, including subroutine "cheker", into the controller.
2. Depress RUN on the controller. The controller will display "waiting for interrupt".
3. Momentarily connect a jumper wire between pin "r" (common) and any input pin (A through T) on the edge connector of the word interrupt card. The controller will display the card address and interrupting word and the program will end.

7-131 If errors are detected by the Multiprogrammer, such as an attempt to program an empty card slot, an error message will be printed (9825) or displayed (9835/45) and the program will terminate.

Example 7-42. 69776A Sample Test Program

9825A Controller

```
0: wrt 723,"WF,4,65535,4.1,1T,II,4T"
1: gsb "cheker"
2: if V#0;gto "end"
3: if bit(3,U)=1;gto "readII"
4: "wait":dsp "waiting for interrupt"
5: if rds(723)#64;gto "wait"
6: red 72310,U,V
7: if bit(3,U)#1;gto "wait"
8: "readII":red 72302,A,B
9: dsp "address=",A,"int. word=",B
10: "end":end
*22613
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"WF,4,65535,4.1,1T,II,4T"
40   GOSUB Cheker
50   IF V<>0 THEN End
60   IF BIT(U,3)=1 THEN Readii
70 Wait:  DISP "WAITING FOR INTERRUPT"
80   STATUS 723;C
90   IF C<>64 THEN Wait
100  ENTER 723.10;U,V
110  IF BIT(U,3)<>1 THEN Wait
120 Readii: ENTER 723.02;A,B
130  DISP "ADDRESS=",A,"INT. WORD=",B
140 End: END
```

7-132 Memory Cards, 69790A

7-133 The 69790A Memory is a bidirectional device used to store a number of 16-bit digital words. The cards can store 1024 16-bit words (standard model), 2048 16-bit words (option 002), or 4096 words (option 004). The cards can operate in the output or input mode. In output mode, data from a controller is stored in the memory cards and then transferred to an external device. In input mode, data from an external device is stored and then read back to the controller. In both output and input modes, data transfer with an external device is controlled by means of 6 handshake lines on the edge connector of the card (3 for output and 3 for input). Data can be transferred to or from an external device at rates up to 125 KHz.

7-134 Physically, the 69790A Memory consists of two separate cards connected together as a card pair. The cards must be installed in adjacent slots and will be referred to as Memory card 1 and Memory card 2 throughout this discussion. Memory card 1 must be installed in the slot with the lowest number (immediately to the left of Memory card 2).

7-135 Memory Card 1 - General Information. Memory and mode control circuits are located on Memory card 1. As noted previously, the size of the memory depends on the card option. Five modes of operation can be programmed via Memory card 1; FIFO input, FIFO output, recirculating input, recirculating output, and external lockout mode. An automatic lockout protection feature is provided in both FIFO modes. This lockout feature will prevent an external device from writing over data that has not been read by the controller in the input mode and will prevent the controller from writing over data that has not been taken by the external device in output mode.

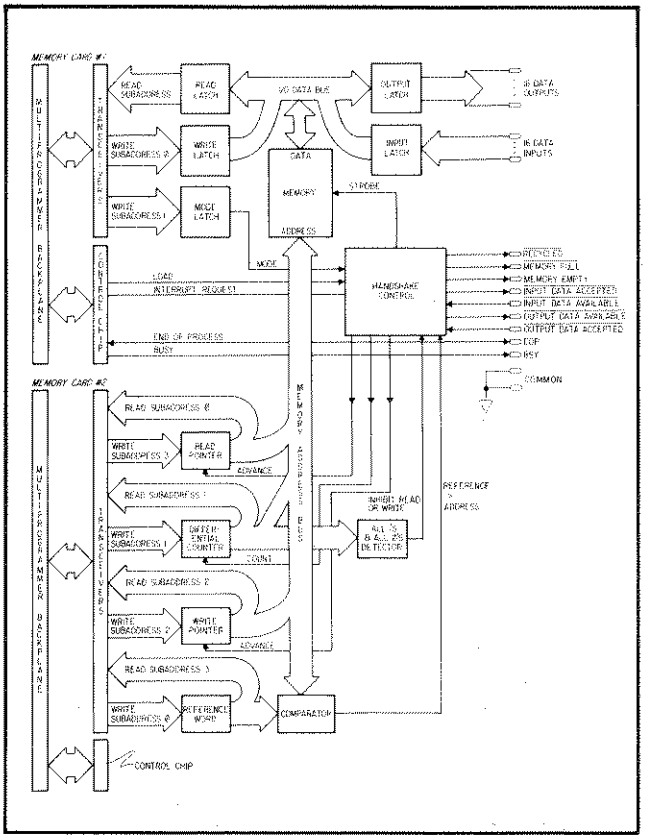
1. FIFO input mode is used to read and store a list of data words from an external device in the order that the data is received and to allow the controller to read this data in the same order.

2. FIFO output mode is used to send a list of data words from the controller to the Memory card and to transfer this list from the Memory cards to an external device in the same order that the data was received.

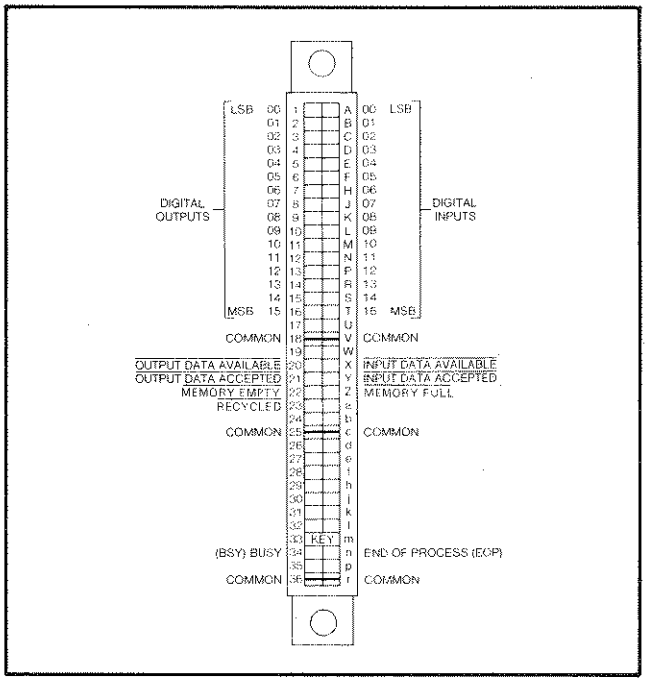
3. Recirculating input mode is used by the controller to read data from selected memory locations on the card. It is also used to read and store data words from an external device into selected locations on the card.

4. Recirculating output mode is used to send data words from the controller to selected memory locations on the card and to transfer data from selected memory locations on the card into an external device.

5. External lockout mode is usually used in conjunction with one of the other four modes to prevent an external device from reading or writing on a memory card.



69790A Block Diagram



69790A Connector

7-136 Memory Card 2 - General Information. Memory card 2 contains; (1) a reference register, (2) a differential counter, (3) a write pointer, and (4) a read pointer, which have the following functions:

1(a). In the FIFO input mode the Memory card will generate an interrupt when the number of words read in and stored is greater than the value in the reference register. This interrupt can be used as an indication to the controller that it is time to read back the data from the Memory card.

(b). In the FIFO output mode the Memory card will generate an interrupt when the number of words left in memory is equal to or less than the value in the reference register. This interrupt can be used as an indication to the controller that it is time to send a new series of data words to the Memory card.

(c). In the recirculating output mode the reference register is used to truncate the size of the memory available to an external device by acting as a pointer to the last memory location that data may be read from. After data has been taken from the memory address that equals the value in the reference register, the read pointer in the Memory card returns to address 0 to continue the output sequence. This allows you to continually send a repetitive sequence of output words. Note: since the first address in the Memory card is 0, the number of words available to the external device will equal the value in the reference register + 1.

2(a). In the FIFO input mode the differential counter contains the number of words read from an external device and stored in memory but not yet read by the controller.

(b). In the FIFO output mode the differential counter contains the number of words stored in the Memory card by the controller but not yet taken by the external device.

3(a). In the recirculating input mode the write pointer is used to specify the address in memory where an external device will begin storing data.

(b). In the recirculating output mode the write pointer is used to specify the address in memory where the controller will begin storing data.

4(a). In the recirculating input mode the read pointer is used to specify the address in memory where the controller will begin reading data.

(b). In the recirculating output mode the read pointer is used to specify the address in memory where an external device will begin reading data.

7-137 Sub-address. Table 7-3 lists the sub-addresses and associated functions that they perform on both Memory cards.

7-138 Example 7-42 illustrates storing variable data values in a Memory card in the FIFO output mode. The Memory cards are assumed to be installed in slots 6 and 7. A Memory Output (MO) instruction is used to load 1000 data words from an array designated "A" into the memory. (The array must have been previously dimensioned and loaded with data). After data has been stored in the memory, a Write First Rank (WF) instruction is used to set the reference register to zero and an Arm Card (AC) instruction is used to "arm" the card to interrupt (see paragraph 7-140). Notice that the Arm Card instruction is sent to Memory card 2. Arm Card, Disarm Card, Clear Card, and Read Status instructions must always be sent to Memory card 2. In this example, the card will interrupt the 6942's microprocessor and set the SRQ line when an external device has taken all of the data (1000 words) from the Memory cards.

7-139 The "MO" instruction used to send data to the Memory card in example 1 is the fastest way to transfer data from the controller to the Memory card in FIFO Output mode. It can not be used in recirculating output or lockout mode because the "MO" instruction automatically sets the memory card to FIFO Output mode.

Example 7-42. Storing Variable Output Data in FIFO Output Mode

9825A Controller

```
50: wrt 723,"MO,6,"
51: for J=1 to 1000
52: wrt 723,A[J]
53: next J
54: wrt 723,"T,WF,7,0T,AC,7T"
```

9835/45 Controllers

```
50   OUTPUT 723;"MO,6",A(*),"T"
60   OUTPUT 723;"WF,7,0T,AC,7T"
```

7-140 The interrupt that will occur when the Memory card in example 7-42 transfers all of its data to an external device is defined as an "Armed Card Interrupt" (see Chapter 6). This type of interrupt does not come from an instruction that is currently active and waiting for an interrupt to complete, but instead comes from a card that has been previously armed by an Arm Card (AC) instruction. Whenever an Armed Card Interrupt occurs, the Multiprogrammer will set SRQ and increment the third Multiprogrammer SRQ status variable from extended talk address 10. When Armed Card Interrupts are detected, you must read from extended talk address 12 to obtain the address of the interrupting card. If this address is not read by the controller, further interrupts from the same card will not set SRQ. Once you have determined the interrupting card is the Memory card you should disarm the card by sending a Disarm Card (DC) instruction. The card can then be reprogrammed. Of course if you do not want to set SRQ after the transfer is complete, you must eliminate the "AC" instruction from Example 42.

7-141 Example 7-43 illustrates a method used to test for an Armed Card Interrupt and disarm the Memory cards if it is determined that they generated the interrupt. This example assumes an SRQ has been detected by the controller and

Table 7-3. Memory Card Sub-addresses and Functions

Memory card	Sub-address	Functions
1	0	Used to send data from controller to Memory card or read data from Memory card to controller. (1)
	1	Used to set operating mode of Memory card. 1 = FIFO input, 2 = FIFO output, 4 = recirculating input 10 = recirculating output, 20 = external lockout
2	0	Used to set the reference register or read the address of the read pointer. (3)
	1	Used to set or read back differential Counter (2).
	2	Used to set or read back the address of the write pointer. (2 and 3)
	3	Used to set the address of the read pointer or read back the reference word. (2)

Notes:

1. When sending data to the Memory cards, the card should be in an output mode and data values must be decimal integers between 0 and 65535. When reading data from the Memory cards, the card should be in an input mode. Data read from the cards, will also be decimal integers between 0 and 65535. If you prefer to use a different data format (e.g., 12 or 16-bit two's complement) you may either use a set format (SF) instruction to reformat Memory card 1 or permanently select a different format by changing the self-ID jumpers on the card (see 69790A card manual).

2. Whenever the address of the write or read pointers is changed, the differential counter must be programmed equal to the difference between the addresses of the write and read pointers before resuming operation in FIFO mode.

3. The address register for the write and read pointers can store up to 4096 addresses. Therefore, an address of 0 may be represented by a value of 0, 1024, 2048, or 3072 on a standard card (1024 words of memory) and "0" or "2048 on an option 002 card (2048 words of memory). To obtain the actual address of the write or read pointer on a Memory card containing 1024 or 2048 words of memory, simply select the highest zero reference value that is below the value of the address and subtract it from the address that has been read back.

subsequent serial poll has identified the Multiprogrammer as the device that has requested service (i.e. rds (723) = 64 on a 9825 controller or STATUS 723 = 64 on a 9835 or 9845 controller.

7-142 In example 7-43, the Multiprogrammer SRQ status is read back from HP-IB extended talk address 10 and evaluated to determine if an Armed Card Interrupt had set SRQ. If an Armed Card Interrupt has occurred, as determined by variable "W" having a value greater than zero, the controller reads HP-IB extended talk address 12 into variable "A". Variable "A" is then tested to see if it contains the address of the Memory card that had been programmed in example 7-42. If this is the address, a Disarm Card instruction is sent to prevent further interrupts from this Memory Card.

NOTE

If you are using a 9835 or 9845 controller and are expecting more than one Armed Card Interrupt to occur, your program should dimension and use an array for reading back the adddresses of the interrupting cards. When using the 9825A controller, the addresses of the interrupting cards may be read into simple variables, "r" variables, or an array. Regardless of the controller type, an array should always be dimensioned equal to, or greater than, the maximum number of armed card interrupts you expect to receive. Please read the description of "Armed Card Interrupts" in chapter 6 before attempting to use more than one card in this manner.

Example 7-43. Testing For an Interrupt and Disarming the Card

9825A Controller

```
100: red 72310,U,V,W
101: if W=0;sto "continue"
102: red 72312,A
103: if A#7;sto "continue"
104: wrt 723,"DC,7T"
```

9835/45 Controllers

```
110  ENTER 723.10;U,V,W
120  IF W=0 THEN Continue
130  ENTER 723.12;A
140  IF A<>7 THEN Continue
150  OUTPUT 723;"DC,7T"
```

7-143 Example 7-44 illustrates storing data values into a Memory card in recirculating Output mode and truncating the Output memory available to an external device to 10 words. In this example the "WF" instruction is used to program the card to recirculating Output mode, set the reference register to 9, the address of the write pointer to 2, and the address of the read pointer to 0. Data from the controller is sent to the card using a write and cycle (WC) instruction and is stored in memory addresses 2 through 6. Since the reference register has been set to 9, an external device reading data from the Memory card will continually read data from memory addresses 0-9 (0,1,2,3,4,5,6,7,8,9,0,1,2,3,etc). To lockout the external device while data is being programmed to the Memory card, the operating mode must be changed from recirculating output (10) to recirculating output with external lockout. (10 + 20 = 30). Hence, you must send a mode value of 30 to address 6.1 to attain lockout. After new data has been programmed, lockout can be removed by sending a Write First Rank instruction to set the mode back to recirculating output without lockout. (i.e. "WF,6.1,10T").

Example 7-44. Storing Data in Recirculating Output

9825A Controller

```
0: wrt 723,"WF,6.1,10,7,9,7.2,2,"
1: wrt 723,"7.3,0T"
2: for J=1 to 5
3: wrt 723,"WC,6",J+1,"T"
4: next J
```

9835/45 Controllers

```
10 OUTPUT 723;"WF,6.1,10,7,9,7.2,2,"
20 OUTPUT 723;"7.3,0T"
30 FOR J=1 TO 5
40 OUTPUT 723;"WC,6",J+1,"T"
50 NEXT J
```

NOTE

When the Memory cards have been programmed to a recirculating output or input mode and data has been read from, or stored on the card, it is necessary to program the differential counter to a value equal to the difference between the addresses of the write and read pointers before resuming operation in the FIFO output or input modes. This can be done in one of two ways. One way is to read the address of the write and read pointers with an "RV" instruction and calculate the difference between the two. The other way is to simply set the write pointer, read pointer, and differential counter to 0 (i.e. "WF,7.1,0,7.2,0,7.3,0T"). This method is useful only when you no longer require the data currently stored on the cards.

7-144 Example 7-45 illustrates programming the Memory cards in FIFO Input mode to store 1000 data words from an external device and to generate an armed card interrupt when all of the data is stored in memory. A Memory Input (MI) instruction is used to specify the address of the Memory card and the number of words that will be read by the controller after the card has interrupted. The "MI" instruction will also automatically set the Memory cards, to FIFO Input mode. A Write First Rank instruction is then used to set the reference register to 999 and an Arm Card instruction is used to arm the card and allow it to generate an interrupt, thus setting SRQ.

Example 7-45. Storing Data from an External Device in FIFO Input Mode

9825A Controller

```
0: wrt 723,"MI,6,1000T"
1: wrt 723,"WF,7,999T,AC,7T"
```

9835/45 Controllers

```
10   OUTPUT 723;"MI,6,1000T"
20   OUTPUT 723;"WF,7,999T,AC,7T"
```

7-145 After the card has generated an interrupt it should be disarmed. The method for detecting the interrupt and disarming the card is the same as that previously described for programming a Memory card in FIFO Output mode. After the card has been disarmed, data can be read back to the controller by means of the appropriate HP-IB extended talk addresses.

7-146 Example 7-46 illustrates a method that can be used to read data from an external device and store it in the Memory cards in recirculating Input mode. A Memory Input instruction is sent first, to specify the address of the Memory card and the number of words that the controller will read back at a later time. A Write First Rank instruction is then used to set the card to recirculating Input mode and the write and read pointers to memory address 100. Data that is stored in memory after the write pointer has been set, will be stored at locations beginning at address 100. Likewise, when the controller begins to read data from the Memory card, it will start reading the data from memory address 100.

Example 7-46. Storing Data From an External Device in Recirculating Input Mode

9825A Controller

```
0: wrt 723,"MI,6,1000T"
1: wrt 723,"WF,6.1,4,7.2,100,7.3,100T"
```

9835/45 Controllers

```
10   OUTPUT 723;"MI,6,1000T"
20   OUTPUT 723;"WF,6.1,4,7.2,100,7.3,100T"
```

7-147 By use of HP-IB extended talk address 05, data can be read back to the controller at any time. In FIFO Input mode, however, it is advisable to wait until the Memory card interrupt has been detected before reading back data. Attempting to read back more data than is available will lock out the card, resulting in data from the memory location immediately above the location containing the last data word stored, being read repeatedly. In the recirculating Input mode data can be read at any time. Generally it is best to read data from the Memory cards into an array within the controller. This array should be equal in size to the number of words that you intend to read as specified by the "MI" instruction. For example, to dimension an array on a 9825 controller to read back the data from example 7-46, you would use the statement "dim A(1000)". To dimension an array on a 9835 or 9845 controller you would use the statement "DIM A(999) or "DIM A(1000)" depending on whether you have specified OPTION BASE 0 or OPTION BASE 1. Note that all 9835/45 examples in this guide use OPTION BASE 1.

7-148 Example 7-47 illustrates reading 1000 words back from the Memory cards that had been programmed in examples 7-45 and 7-46. If desired, you may program the cards to external lockout mode to prevent an external device from storing fresh data in the Memory cards while the data is being read back by the controller. In this case a Write First Rank instruction specifying the appropriate Input mode and lockout would be inserted at the beginning of this example ("WF,6.1,21T" for FIFO input mode with lockout or "WF,6.1,24T" for recirculating input mode with lockout). If lockout mode is programmed, you must remember to reprogram the card to the appropriate input mode without lockout when you require fresh data from the external device.

Example 7-47. Reading Data from a Memory Card into an Array

9825A Controller

```
200: for J=1 to 996 by 5
201: red 72305,A[J],A[J+1],A[J+2],A[J+3],A[J+4]
202: next J
```

9835/45 Controllers

```
200  ENTER 723.05;A(*)
```

7-149 As previously mentioned, the values of the reference register and differential counter, as well as the addresses of the write and read pointers can be read at any time. A Read Value (RV) instruction is used to read these values. In example 7-48 the address of the read pointer, value of the differential counter, address of the write pointer, and value of the reference register are read into controller variables A,B,C, and D, respectively.

Example 7-48. Reading Registers, Counters, and Pointers

9825A Controller

```
0: wrt 723,"RV,7,7.1,7.2,7.3"
1: red 72306,A,B,C,D
```

9835/45 Controllers

```
10  OUTPUT 723;"RV,7,7.1,7.2,7.3"
20  ENTER 723.06;A,B,C,D
```

7-150 A Read Value instruction can also be used to read back the Memory card data word currently addressed by the read pointer without incrementing the pointer to the next word.

7-151 MULTIPLE CARD PROGRAMS

7-152 External Triggering Output and Input Cards

7-153 Example 7-49 shows how to use the "WF" and "AC" instructions in conjunction with external triggers. In the example, an external trigger initiates an output pulse from a 69736A Timer/Pacer card and causes a 69751A A/D card to take a voltage reading. Note that the cards used in these procedures are intended to serve as representative examples only and could be replaced with virtually any other type of output or input card. Example 7-49 makes use of "armed card interrupts" to set SRQ when the card has completed a data transfer as a result of the external trigger. Armed card interrupts will set SRQ in either the serial or parallel operating modes and are described in more detail in Chapter 6.

7-154 When using external triggers with an input card, note that the Input External (IE) instruction can be used rather than the "AC" instruction of example 7-49. The "IE" instruction arms the card and waits for the external trigger to cycle it. It does not, however, set SRQ unless the Multiprogrammer is operating in the parallel mode (refer to Chapter 5 for a complete description of the "IE" instruction).

7-155 Connections to the cards of Example 7-49, are shown in Figure 7-1. As shipped from the factory, both cards will respond to "high true" external triggers having a duration of at least 1μsec. (The trigger polarity can be reversed, if necessary, by connecting the TRIGGER POLARITY SELECT pin, 33, to ground.)

7-156 In Example 7-49, data is first sent top a timer card in slot 2, but no output pulse will be generated until the card receives an external trigger. Next, the timer card, and an A/D card in slot 3, are armed to enable them to generate interrupts. If an external trigger is now applied to the timer card, it will generate an output pulse, five seconds long. The Multiprogrammer will then set SRQ and increment the third SRQ status variable ("W" in this example). If an external trigger is applied to the A/D card, the voltage present at its pins will be converted to a digital value and stored on the card. Again, the Multiprogrammer will set SRQ and increment the third status variable.

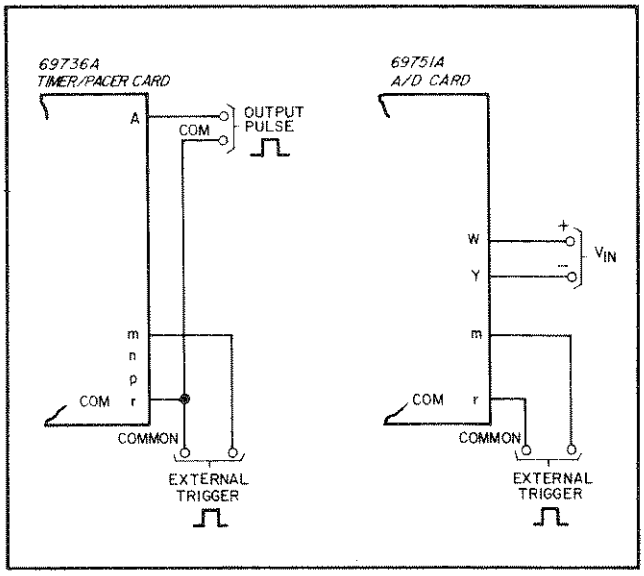


Figure 7-1. External Trigger Connections

7-157 Whenever an SRQ is detected, Multiprogrammer SRQ status (extended talk address 10) should be read back to determine the reason. If an armed card interrupt had set SRQ the third status variable will be greater than 0, and the address(es) of the card(s) that had interrupted can be read back from extended talk address 12 to determine the proper action to be taken. In this example, the controller will merely print "timer output compete" when the timer card output pulse is complete. If the A/D card is externally triggered, the voltage value stored on the card will be read into variable "A" and printed by the controller.

7-158 Additional Consideration. Here are some additional facts about using armed card interrupts:

1. Cards may be externally triggered without being armed. However, SRQ will not be set when the card completes its data transfer.
2. Cards that are armed and then externally triggered are not automatically disarmed after they interrupt. A card can be disarmed at any time by using the Disarm Card (DC) instruction.
3. Once a card generates an armed card interrupt and sets SRQ, it cannot generate another interrupt until its address is read back from extended talk address 12.

Example 7-49. Externally Triggering the 69736A and 69751A Cards

9825A Controller

```
0: wrt 723,"WF2,5ST,AC,2,3T"
1: "wait":if rds(723)#64;gto "wait"
2: gsb "cheker"
3: if V#0;gto "end"
4: if W=0;gto "wait"
5: for J=1 to W;red 723;2,rJ;next J
6: for J=1 to W
7: if rJ=2;gsb "timer"
8: if rJ=3;gsb "A/D"
9: next J
10: gto "wait"
11: "end":end
12: "timer":prt "timer output","complete"
13: ret
14: "A/D":wrt 723,"RV,3T"
15: red 72306,A
16: fxd 3;prt "A/D input =",A
17: ret
*551
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11),A(2)
30   ON ERROR GOSUB Ertrap
40   OUTPUT 723;"WF,2,5ST,AC,2,3T"
50 Wait:  STATUS 723;C
60   IF C<>64 THEN Wait
70   GOSUB Cheker
80   IF V<>0 THEN End
90   IF W=0 THEN Wait
100  A(1)=A(2)=-1
110  ENTER 723,12;A(*)
120  FOR J=1 TO 2
130  IF A(J)=2 THEN GOSUB Timer
140  IF A(J)=3 THEN GOSUB Ad
150  NEXT J
160  GOTO Wait
170 End: END
180 Timer: PRINT "TIMER OUTPUT COMPLETE"
190  RETURN
200 Ad: OUTPUT 723;"RV,3T"
210  ENTER 723,06;A
220  PRINT "A/D INPUT =",A
230  RETURN
240 Ertrap: IF ERRN=159 THEN RETURN
250  PRINT ERRM$
260  STOP
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE 1
	20	Dimension 11 word array to allow subroutine "Cheker" to read any error variables that may occur and a 2 word array for use when reading back addresses of cards that generated armed card interrupts.
	30	Establish linkage to error processing subroutine to allow handling of variable length read-backs.
0	40	Set up timer card to program a 5 second output pulse when externally triggered, then arm timer card and high speed A/D card to allow them to generate armed card interrupts.
1	50,60	Wait for SRQ
2	70	Clear SRQ, put Multiprogrammer status in variables U,V,W, and check for programming errors.
3	80	If programming errors are detected, terminate program.
4	90	If no "armed card interrupts" have been detected, wait for another SRQ.
	100	Initialize array used to read back armed card interrupts to −1.
5	110	Read addresses of cards that have generated armed card interrupts.
6	120	Establish for/next loop with line 9/150 to interpret list of "armed card interrupts" and determine proper processing path.
7	130	If timer card has generated an interrupt process it.
8	140	If A/D card has generated an interrupt process it.
9	150	Get next variable from armed card interrupt list and process it.
10	160	All interrupts are processed. Go wait for another one.
11	170	End of main program: subroutines follow.
12	180	Timer card has interrupted. Controller prints "timer output complete".
13	190	Return and process next variable on armed card interrupt list.
14	200	A/D card has interrupted. Read data from the A/D card into Multiprogrammer memory.
15	210	Read A/D data from the Multiprogrammer into controller variable "A".
16	220	Controller prints voltage measured by A/D card.
17	230	Return and process next variable on armed card interrupt list.
	240-260	Error trapping subroutine used to allow variable length data readbacks on 9835/45 controller. For a description of this subroutine see Appendix C.

7-159 Resistance Measurement

7-160 Example 7-50 illustrates use of a 69721A D/A current Converter card in conjunction with a 69751A A/D Converter card to measure the value of a resistor. This example assumes that the 69721A D/A card is installed in slot 2 and the 69751A A/D card is installed in slot 3. Connections between the cards and the resistor being measured (RX) are shown in Figure 7-2.

7-161 In the example, the current output from the D/A card is programmed in two steps, 1 milliamp, and 10 milliamps. After each current step, the A/D card is programmed to take a voltage reading across the resistor. If the voltage measured across the resistor is between 1 and 10.235 volts, or both readings have been taken, the resistance is calculated and displayed on the controller. The maximum resistance value that can be measured by this program is approximately 10,000 ohms. If an attempt is made to measure a resistance value that is too high the controller will display "resistance too high".

Example 7-50. Measuring Resistance

9825A Controller

```
0: wrt 723,"GS"
1: .1→A
2: "loop":A*10→A
3: wrt 723,"OP,2",A,"T,1P,3T"
4: esb "cheker"
5: if V#0;gto "end"
6: red 72301,B
7: if B<1 and A<10;gto "loop"
8: if B=10.235;gto "toohi"
9: dsp "R=",B/A*1000,"ohms"
10: gto "end"
11: "toohi":dsp "resistance too high"
12: "end":end
*1003
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"GS"
40   A=.1
50 Loop:  A=A*10
60   OUTPUT 723;"OP,2",A,"T,1P,3T"
70   GOSUB Cheker
80   IF V<>0 THEN End
90   ENTER 723.01;B
100  IF (B<1) AND (A<10) THEN Loop
110  IF B=10.235 THEN Toohi
120  DISP "R=",B/A*1000,"OHMS"
130  GOTO End
140 Toohi: DISP "RESISTANCE TOO HIGH"
150 End: END
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE
	20	Dimension 11 word array for use by subroutine "Cheker"
0	30	Multiprogrammer is programmed to serial mode. This statement is not required unless the Multiprogrammer had been previously programmed to parallel mode.
1	40	Value used to calculate current output from D/A current converter card is initialized.
2	50	Value of output current is calculated.
3	60	Current output is programmed from D/A card and A/D card is programmed to read the voltage across resistor being measured.
4,5*	70,80*	Check for programming errors. If programming errors are detected, terminate program.
6	90	Read the voltage value measured across the resistor into controller variable B.
7	100	If measured voltage is less than 1 volt and current is less than 10 milliamps, increase current by a factor of 10 and take another voltage reading.
8	110	If voltage value is maximum of A/D card, resistance is too high too measure with this program.
9	120	Calculate and display resistance.
10	130	Program is done, go to end
11	140	This statement is only executed if resistance is too high to measure with this program.
12	150	End of program

*Program statements are optional.

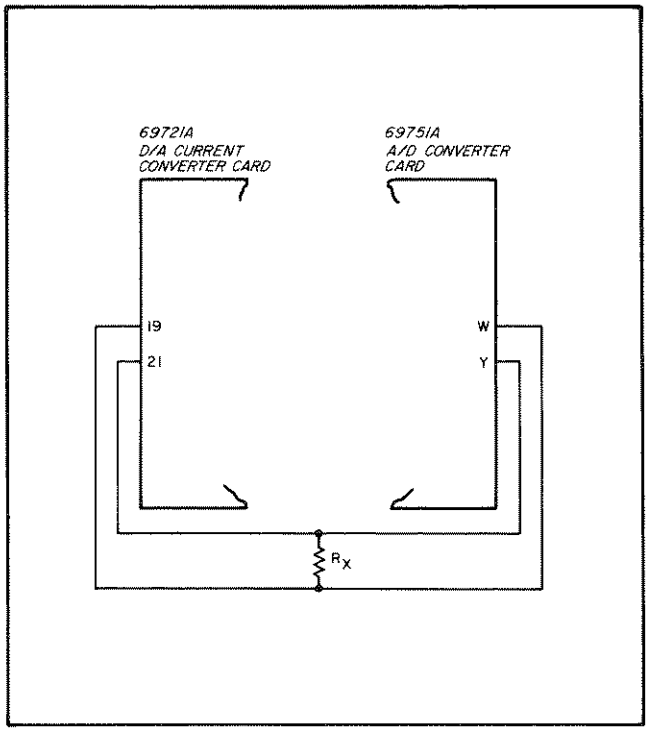


Figure 7-2. Resistance Measurement Connections

7-162 Voltage Scanning and Measurement Using the Relay Card

7-163 Example 7-51 illustrates using a 69730A Relay Output card in conjunction with a 69751A A/D card to measure the voltages on 16 channels. The 16 voltages all share a common reference. All voltage values are stored in an array. When all 16 voltages have been measured the controller will read the array and print the voltage values. By appropriate modifications to the hardware connections and the program, this same card combination can be used to measure up to 8 voltages that are isolated from each other. If desired, additional 69730A Relay Output cards (and program modifications) can be used to expand the scanner array.

7-164 Example 7-51 assumes that the 69751A A/D card is installed in slot 5 and the 69730A Relay Output card is installed in slot 6. It is further assumed that the voltages to be measured are in the range of −10.240 to +10.235 volts, permitting the use of the standard A/D card. Connections between the cards and the voltages to be measured are shown in Figure 7-3.

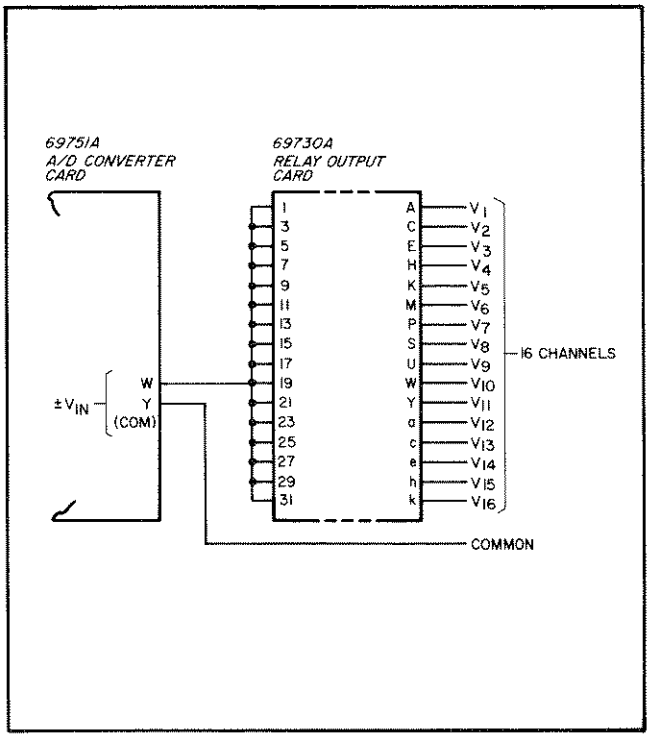


Figure 7-3. Voltage Scanning Connections

Example 7-51. Voltage Scanning and Measurement Using Relay and A/D Cards

9825A Controller

```
0: dim A[16];fxd 3
1: wrt 723,"GS"
2: for J=0 to 15
3: wrt 723,"OP,6,0T,OP,6",2↑J,"T"
4: wrt 723,"IP,5T"
5: gsb "cheker"
6: if V#0;gto "end"
7: red 72301,A[J+1]
8: next J
9: for J=1 to 16
10: prt A[J]
11: next J
12: "end":end
*3746
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM A(16),R(11)
30   OUTPUT 723;"GS"
40   FOR J=0 TO 15
50   OUTPUT 723;"OP,6,0T,OP,6",2↑J,"T"
60   OUTPUT 723;"IP,5T"
70   GOSUB Cheker
80   IF V<>0 THEN End
90   ENTER 723.01;A(J+1)
100  NEXT J
110  FOR J=1 TO 16
120  PRINT A(J)
130  NEXT J
140 End: END
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE 1
0	20	Dimension 16 word array to store voltage measurements, change number format of 9825A controller to fixed 3 to allow printing values to 3 decimal places, and dimension 11 word array for subroutine "Cheker" in 9835/45 program.
1	30	Multiprogrammer is programmed to serial mode. This statement is not required unless the Multiprogrammer had been previously programmed to parallel mode.
2	40	Establish a for/next loop with line 8/90 in order to calculate and program the relay closures required to take the 16 voltage measurements and store them in the proper array locations.
3	50	Program all relays open to prevent possibility of shorting voltages together. Then program appropriate relay closure for voltage measurement.
4	60	Program A/D card to take a voltage reading.
5,6*	70,80*	Check for programming errors. If programming errors are detected, terminate program.
7	90	Read measured voltage value into appropriate array location.
8	100	Repeat sequence for next voltage measurement.
9	110	Establish another for/next loop with line 11/120 to read and print array variables in order.
10	120	Print array variable.
11	130	Read and print next array variable.
12	140	End of program

*Program statements are optional.

7-165 Frequency Measurement

7-166 Example 7-52 illustrates using a 69775A Counter/Totalizer card in conjunction with a 69736A Timer/Pacer to measure the frequency of a square wave or pulse train which is then displayed on the controller. This example assumes that the 69775A counter card is installed in slot 8 and the 69736A timer card is installed in slot 9. The frequency of the signal to be measured should not exceed 1 MHz.

7-167 In Example 7-52 the timer card is used to provide a gate to the "up enable" line on the counter card. The frequency of the signal being measured can then be determined by multiplying the number of pulses counted by the gate time in seconds. Connections between the cards and the signal being measured are shown in Figure 7-4.

7-168 A primary consideration when measuring frequency with this technique is the capacity of the counter card. Care must be taken to select a time period for the gate that will not allow the maximum capacity of the counter card (65535 counts) to be exceeded during the measurement period. In Ex-Example 7-46 the program automatically selects gate times as follows:

1. The initial gate time is 10 milliseconds which produces 10000 counts at the maximum input frequency of 1 MHz.
2. Whenever the total count is less than 6500 and the gate time is less than one second, the gate time is increased by a factor of 10 and a new measurement is made. The maximum gate time in this program is one second.

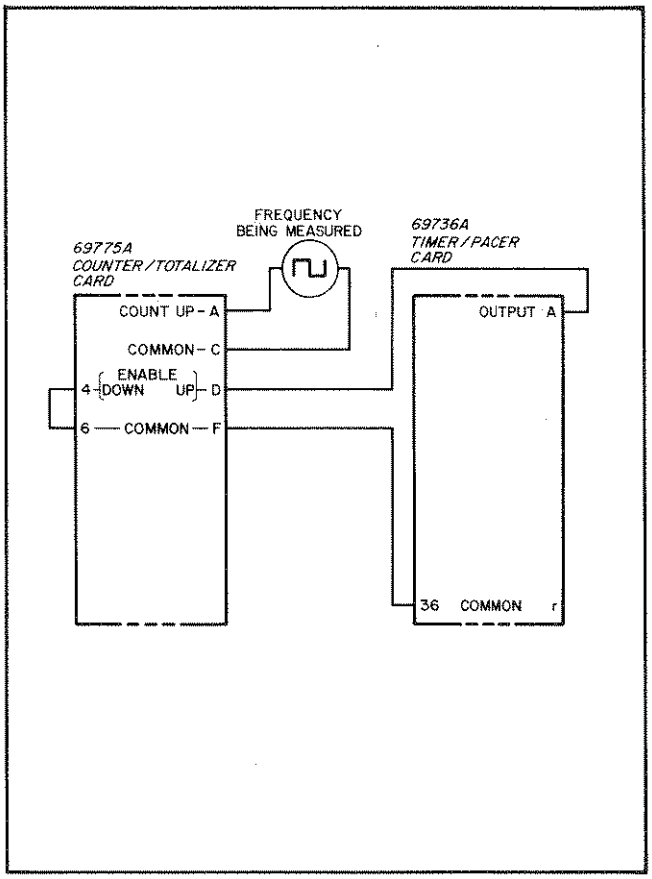


Figure 7-4. Frequency Measurement Connections

Example 7-52. Frequency Measurement Using 69736A and 69775A Cards

9825A Controller

```
0: wrt 723,"GS"
1: "init":1→A
2: "loop":A*10→A
3: wrt 723,"WC,8,0T,OP,9",A,"I,RV,8T"
4: esb "cheker"
5: if V#0;gto "end"
6: red 72306,B
7: if B<6500 and A<1000;gto "loop"
8: fxd 0;dsp "frequency =",B*1000/A,"Hz"
9: gto "init"
10: "end":end
*25199
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"GS"
40 Init:  A=1
50 Loop:  A=A*10
60   OUTPUT 723;"WC,0,0T,OP,9",A,"I,RV,8T"
70   GOSUB Cheker
80   IF V<>0 THEN End
90   ENTER 723.06;B
100  IF (B<6500) AND (A<1000) THEN Loop
110  DISP "FREQUENCY =",B*1000/A,"Hz"
120  GOTO Init
130 End: END
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE 1
	20	Dimension 11 word array for use by subroutine "Cheker".
0	30	Multiprogrammer is programmed to serial mode. This statement is not required unless the Multiprogrammer had been previously programmed to parallel mode.
1	40	Value used to calculate gate time is initialized.
2	50	Duration of gate in milliseconds is calculated.
3	60	The Counter card is preset to 0, timer card is programmed to output a gate pulse, and, upon completion of the gate pulse, the number of pulses counted by the counter card is read into the Multiprogrammer memory.
4,5*	70,80*	Check for programming errors. If programming errors are detected, terminate program.
6	90	Read number of pulses counted into controller variable "B".
7	100	If less than 6500 pulses have been counted and the gate time is less than 1 second (1000 ms.) increase gate time and take another reading.
8	110	Display frequency of signal being measured.
9	120	Go back and measure frequency again
10	130	End of program.

*Program statements are optional.

7-169 Synthesizing a Waveform

7-170 Example 7-53 shows a program for synthesizing a waveform in the shape of a staircase. The amplitude of the waveform ranges from −10 to +10 volts and consists of twenty one 1-volt steps lasting 50 microseconds each. A 69736A timer card, a 69720A D/A Voltage Converter card, and the 69790A Memory cards installed in slots 4, 5, 6, and 7 (the Memory cards use two slots) are used in this example. Connections between the cards are shown in Figure 7-5.

7-171 In Example 7-53 the Memory cards are loaded with the data necessary to program the D/A card to the required output voltage. The timer card is then programmed to the recirculating (continuous output) mode and to a pulse width of 25 microseconds. As a result, the timer card produces a square wave output with a 50 microsecond total period. As each output pulse from the timer card pulls the data accepted (DAC) line low on the Memory card, the memory card will set the data available (DAV) line high. Upon completion of each output pulse from the timer card, the Memory card transfers the next available word in memory to the output of the card and sets its data available line low. The data available signal serves as an external trigger ot the D/A card which then converts the digital value supplied by the Memory card to an analog (voltage) output. This output sequence continues for as long as the timer card is in the recirculate mode.

7-172 By changing the values and amount of data loaded into the Memory card, it is possible to generate an unlimited variety of output waveforms by the method used here. The frequency of the output waves will be determined by the period of the pulses programmed from the timer card because each pulse from the timer card will cause the next output word from the Memory card to be transferred to the D/A card. The D/A card then translates this word into an analog output.

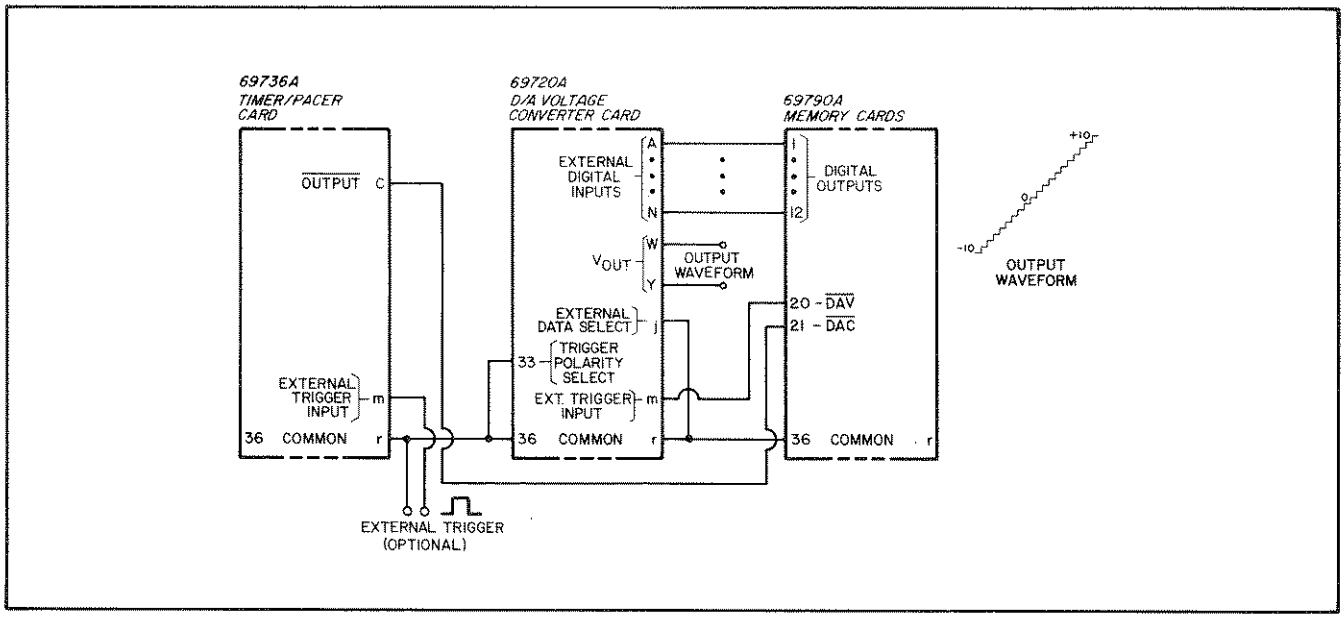


Figure 7-5. Connections for Waveform Synthesizing

Example 7-53. Synthesizing a Waveform

9825A Controller

```
0: wrt 723,"SF,6,3,1,,.005,12T"
1: wrt 723,"WF,6.1,10,7,20,7.2,0,"
2: wrt 723,"7.3,0T"
3: for J=-10 to 10
4: wrt 723,"WC,6",J,"T"
5: next J
6: wrt 723,"WF,4.2,1T,WC,4,25UT"
7: ssb "cheker"
8: end
*5727
```

9835/45 Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"SF,6,3,1,,.005,12T"
40   OUTPUT 723;"WF,6.1,10,7,20,7.2,0,"
50   OUTPUT 723;"7.3,0T"
60   FOR J=-10 TO 10
70   OUTPUT 723;"WC,6",J,"T"
80   NEXT J
90   OUTPUT 723;"WF,4.2,1T,WC,4,25UT"
100  GOSUB Cheker
110  END
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE 1
	20	Dimension 11 word array for used by subroutine "Cheker"
0	30	Reformat Memory card to agree with format of D/A card (data range = −10.24 to 10.235, LSB = .005, number of bits = 12).
1,2	40,50	Program Memory card to recirculating output mode, set reference register to truncate memory to 21 words, set write and read pointers to 0.
3-5	60-80	Load output data into Memory card.
6	90	Program timer card to recirculating mode (continous output), then program a 25 microsecond output pulse.
7*	100*	Check for programming errors.
8	110	End of program

*Program statement is optional.

7-173 High-Speed Analog Data Acquisition

7-174 Example 7-54 illustrates using an external trigger to initiate a sequence which will take 1000 voltage readings at 60 microsecond intervals and store the data in the Memory cards. The controller will then read the data from the Memory cards into an array. A 69735A Pulse Train Output card, 69751A A/D Converter card, and the 69790A Memory cards installed in slots 4, 5, 6, and 7 (the Memory cards require two slots) are used in this example. If desired, a cycle instruction can be used to initiate the measurement sequence instead of an external trigger by adding the cycle instruction to the end of the literal string in line 3/40 (i.e. "AC,7T,WF,4,1000,4.2,60T,CY,4T"). Connections to the cards are shown in Figure 7-6.

NOTE

As shipped from the factory, the data conversion time of the 69751A A/D card is approximately 60 microseconds. This includes an A/D conversion time and control circuitry delays of approximately 30 microseconds each. When using this card in an externally triggered mode, data conversion times of approximately 30 microseconds can be achieved by proper switch settings on the card. (switch S2-2 should be on, and S2-1,3, and 4, should be off; see manual). Instructions that would normally cycle the A/D card (such as, the "IP" or "CY" instructions) will not do so when the switches are set for conversion times of 30μsec.

7-175 When an external trigger is applied to the Pulse Train Output card of Example 7-54 the card will begin a series of 1000 pulses at 60 microsecond intervals which serve as external triggers for the A/D card. Externally triggering the A/D card will cause the card to convert a voltage present at the input of the card to a digital value. This value is sent to the Memory cards via the external digital output connections on the A/D card. An end-of-cycle pulse (EOC) generated by the A/D card when it completes its conversion process, is used as a data available signal to the Memory card. When the Memory cards receive a data available signal they read in the data present on the digital input lines and increment their write pointer to the next highest address. When 1000 words have been read in by the Memory cards, an "armed card interrupt" will be generated by the Memory cards and the Multiprogrammer will set SRQ.

7-176 Since only one armed card interrupt can occur as a result of running Example 7-54 the interrupt must have been generated by the Memory card and it is not necessary to interpret the armed card interrupt list to detemrine the address of the interrupting card. The address of the interrupting card must still be read back, however, to allow future interrupts from the Memory card to set SRQ. Armed card interrupts and external triggers are discussed in Paragraph 7-152 and in Chapters 5 and 6.

7-177 With slight modifications to the program, the Pulse Output card used in this example could be replaced with a 69736A timer card. If this is done, the timer card would be progrmamed to the recirculate (continous output) mode and externally triggered. Then, when the Memory card interrupted, the recirculate mode would be programmed off while the data was being read back from the Memory card.

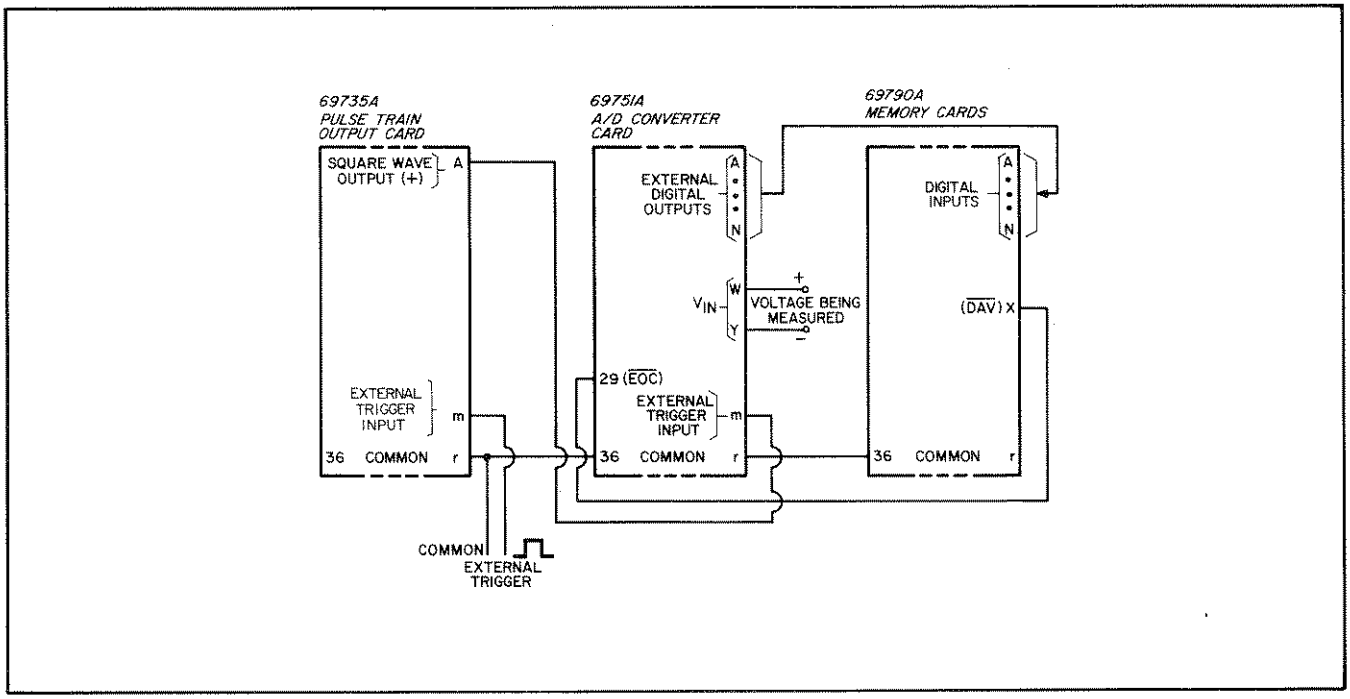


Figure 7-6. Data Acquisition Connections

Example 7-54. High Speed Analog Data Acquisition

9825A Controller

```
0: dim AL1000]
1: wrt 723,"SF,6,3,1,,005,12T"
2: wrt 723,"MI,6,1000T,WF,7,999["
3: wrt 723,"AC,7T,WF,4,1000,4.2,60T"
4: "wait":if rds(723)#64;sto "wait"
5: gsb "cheker"
6: if V#0;sto "end"
7: if W=0;sto "wait"
8: red 72312,A
9: wrt 723,"DC,7T"
10: for J=1 to 1000
11: red 72305,AL J]
12: next J
13: dsp "data is ready"
14: "end":end
*10745
```

9835/45 Controllers

```
10    OPTION BASE 1
20    DIM AC1000),R(11)
30    OUTPUT 723;"SF,6,3,1,,005,12T"
40    OUTPUT 723;"MI,6,1000T,WF,7,999["
50    OUTPUT 723;"AC,7T,WF,4,1000,4.2,60T"
60 Wait:  STATUS 723;C
70    IF C<>64 THEN Wait
80    GOSUB Cheker
90    IF V<>0 THEN End
100   IF W=0 THEN Wait
110   ENTER 723.12;A
120   OUTPUT 723;"DC,7T"
130   ENTER 723,05;AC*)
140   DISP "DATA IS READY"
150 End: END
```

Explanation:

9825	9835/45	Description
	10	Set 9835/45 to OPTION BASE 1
0	20	Dimension 1000 word array to store data that will be read from Memory card; Dimension 11 word array for use by subroutine "Cheker" in 9835/45 program.
1	30	Reformat Memory card to agree with format of A/D card (data range = −10.24 to 10.235, LSB = .005, number of bits = 12).
2	40	Program Memory card in slot 6 to FIFO Input mode and specify 1000 words to be read back; Set Memory card to interrupt when high speed A/D card has stored 1000 words in Memory card.
3	50	Arm Memory card for interrupts; Set up Pulse Train output card to send out 1000 pulses at 60 microsecond intervals after it is externally triggered (or cycled).
4	60,70	Wait for SRQ.
5	80	Clear SRQ, put Multiprogrammer status in variables U,V,W, and check for programming errors.
6	90	If programming errors are detected, terminate program.
7	100	If an "armed card interrupt" was not detected, wait for another SRQ.
8	110	Read address of card that has generated an "armed card interrupt".
9	120	Disarm Memory card
10-12	130	Read 1000 data words from Memory card into array.
13	140	Controller displays "data is ready"
14	150	End of program.

Appendix A
NUMBER THEORY

A-1 This appendix provides a brief description of number theory as it relates to the data types used by the Multiprogrammer System. The decimal, the octal, and the binary number systems are all used to define, in numerical fashion, the data transferred and converted within the Multiprogrammer System.

A-2 DECIMAL NUMBERS

A-3 Most of us are so familiar with the decimal number system that we have become largely unconscious of the underlying principles; at least so far as they relate the decimal system to other systems such as binary or octal. Any number system has what is called a "base", which is the number of unique symbols used in the particular system. Since it has ten unique symbols (0,1,2,---9), the decimal system is the base 10 number system. Although it is often omitted since the number system is usually implied by the context, the base may be specified by a subscript:

$$235_{10} \qquad (1)$$

A-4 This subscript simply tells us that the preceding number is a unique quantitative value presented in decimal form. If we examine the individual digits that make up a decimal number, it becomes clear that the number can be represented as follows:

$$\begin{aligned} 235_{10} &= 2\times10^2 = 2\times100 = 200 \\ &\quad 3\times10^1 = 3\times\ 10 = \ 30 \\ &\quad 5\times10^0 = 5\times\ \ 1 = \ \ 5 \\ &\qquad\qquad\qquad\quad \overline{235_{10}} \end{aligned} \qquad (2)$$

A-5 From (2) it may be seen that each digit is weighted by a power of the base (in this case 10), and that the power increases by one for each successive digit to the left. this relationship holds true for number systems to any base, and provides a longhand method for converting to base 10 from any other base.

A-6 BINARY NUMBERS

The binary system, which is the system of machine language, is the base 2 number system, and has two unique symbols; "0" and "1".

$$11101011_2 \qquad (3)$$

A-8 If we change (3) to the form used in (2), we get:

$$\begin{aligned} 11101011_2 &= 1\times2^7 = 1\times128 = 128 \\ &\quad 1\times2^6 = 1\times\ 64 = \ 65 \\ &\quad 1\times2^5 = 1\times\ 32 = \ 32 \\ &\quad 0\times2^4 = 0\times\ 16 = \ \ 0 \\ &\quad 1\times2^3 = 1\times\ \ 8 = \ \ 8 \\ &\quad 0\times2^2 = 0\times\ \ 4 = \ \ 0 \\ &\quad 1\times2^1 = 1\times\ \ 2 = \ \ 2 \\ &\quad 1\times2^0 = 1\times\ \ 1 = \ \ 1 \\ &\qquad\qquad\qquad\quad \overline{235_{10}} \end{aligned} \qquad (4)$$

A-9 OCTAL NUMBERS

A-10 The octal number system has eights unique symbols (0 through 7) and functions in a similar fashion:

$$353_8 \qquad (5)$$

and:

$$\begin{aligned} 353_8 &= 3\times8^2 = 3\times64 = 192 \\ &\quad 5\times8^1 = 5\times\ 8 = \ 40 \\ &\quad 3\times8^0 = 3\times\ 1 = \ \ 3 \\ &\qquad\qquad\qquad\quad \overline{235_{10}} \end{aligned} \qquad (6)$$

A-11 The octal number system has another characteristic which makes it particularly useful to the programmer. Returning to binary numbers for a moment, we see that three binary digits allow us to represent eight unique values:

$$\begin{aligned} 111 &= (1\times2^2)+(1\times2^1)+(1\times2^0) &&= 7 \\ 110 &= (1\times2^2)+(1\times2^1)+0 &&= 6 \\ 101 &= (1\times2^2)+0+(1\times2^0) &&= 5 \\ 100 &= (1\times2^2)+0+0 &&= 4 \\ 011 &= 0+(1\times2^1)+(1\times2^0) &&= 3 \\ 010 &= 0+(1\times2^1)+0 &&= 2 \\ 001 &= 0+0+0+(1\times2^0) &&= 1 \\ 000 &= 0+0+0 &&= 0 \end{aligned} \qquad (7)$$

A-12 Since a single octal digit may also represent eight unique values, we can use octal digits to represent binary triads for any binary number regardless of its size. Conversion is by direct inspection:

$$\begin{array}{ccccccc} 1 & \underbrace{111} & \underbrace{101} & \underbrace{111} & \underbrace{100} & \underbrace{011}_2 \\ | & | & | & | & | & | \\ 1 & 7 & 5 & 7 & 4 & 3\ _8 \end{array} \qquad (8)$$

The reverse process is just as simple:

$$\begin{array}{cccccc} 0 & 2 & 7 & 6 & 1 & 5 \\ | & | & | & | & | & | \\ 0 & 010 & 111 & 110 & 001 & 101 \end{array}_2 \quad 8 \qquad (9)$$

A-13 The octal number is clearly easier to remember than its binary equivalent, yet is also readily converted to or derived from binary and is therefore more closely related to machine language than decimal.

A-14 Conversion from binary to octal by inspection is a particularly useful technique when relay output cards are used as switches in a calculator based mutliprogrammer system. The octal pattern is easily converted to the desired switch (relay contact) closures.

A-15 DECIMAL CONVERSION ALGORITHMS

A-16 Integer conversions between the decimal and either the binary or the octal systems are less obvious and considerably more tedious. Direct conversions from decimal to binary, and vice-versa, are not usually performed, but instead conversions between decimal and octal are used in conjunction with the conversion by inspection techniques shown in (8) and (9) to achieve the same results indirectly.

A-17 Octal To Decimal. Conversion from octal to decimal may be performed as follows:

1. Multiply the most significant octal digit by 8.
2. Add the next most significant octal digit and multiply the sum by 8.
3. Repeat step 2 until the least significant digit is reached.
4. Add the least significant octal digit but do no multiply the sum by 8.

For example:

175743_8
$\times 8$
8
$+7$
15
$\times 8$
120
$+5$
125
$\times 8$
1000
$+7$
1007
$\times 8$
8056
$+4$
8060
$\times 8$
$64480 + 3 = 64483_{10}$

(10)

A-18 IF we put (10) in a more general form, its relationship to (6) becomes apparent:

$A\,B\,C\,D\,E\,F_Y$
$\times Y$
AY
$+B$
$AY + B$
XY
$AY^2 + BY$
$+C$
$AY^2 + BY + C$
XY
$AY^3 + BY^2 + CY$
$+D$
$AY^3 + BY^2 + CY + D$
XY
$AY^4 + BY^3CY^2 + DY$
$+E$
$AY^4 + BY^3 + CY^2 + DY + E$
XY
$AY^5 + BY^4 + CY^3 + DY^2 + EY + F_{10}$

$= (A \times Y^5 + (B \times Y^4) + (C \times Y^3) +$
$(DXY^2) + (EXY^1) + (FXY^0) =$
$AY^5 + BY^4 + CY^3 + DY^2 + EY^{10} + F_{10}$

(11)

A-19 Decimal To Octal. Conversion from decimal to octal may also be reduced to a simple algorithm:

1. Divide the decimal number by 8, and write down the remainder.
2. Divide the quotient of step 1 by 8 and write down the remainder.
3. Repeat step 2 until the quotient is zero (the last remainder is also retained).

A typical example:

$8\overline{)173981}$ r. 5
$8\overline{)21747}$ r. 3
$8\overline{)2718}$ r. 6
$8\overline{)339}$ r. 3
$8\overline{)42}$ r. 2
$8\overline{)5}$ r. 5
0

$5\,2\,3\,6\,3\,5_8$

(12)

A-20 Several operations are implied in (12) and if shown would appear as follows:

$8\overline{)173981}$	10		
$8\overline{)21747}$	$5 \times 10^0 =$	5	
$8\overline{)2718}$	$3 \times 10^1 =$	30	
$8\overline{)339}$	$6 \times 10^2 =$	600	
$8\overline{)42}$	$3 \times 10^3 =$	3000	
$8\overline{)5}$	$2 \times 10^4 =$	20000	
0	5×10^5	500000	
		523635_8	

(13)

A-21 When used in the form given in (13) this algorithm is fully reversible:

$$
\begin{array}{lll}
10\overline{)523635} & & \\
10\overline{)52363} & 5\times 8_0 & 5 \\
10\overline{)5236} & 3\times 8_1 & 24 \\
10\overline{)523} & 6\times 8_2 & 384 \\
10\overline{)52} & 3\times 8_3 & 1536 \\
10\overline{)5} & 2\times 8_4 & 8192 \\
0 & 5\times 8_5 & 163840 \\
 & & \overline{173981_{10}}
\end{array}
\qquad (14)
$$

A-22 NEGATIVE NUMBERS

A-23 A number system is not really complete unless it makes some provision for negative values. The decimal system does so through the use of two additional symbols: the plus (+) and the minus (−) signs. Obviously, these two symbols serve a dual purpose since they not only indicate polarity, but also act as arithmetic operators. Any value within the range of $-\infty < n < +\infty$ may be more or less conveniently represented by this system of twelve symbols (i.e., the digits 0-9 and the "+" and "−" signs).

A-24 The relationship between the polarity indicating and the operative functions of the "+" and "−" signs is implicit in subtractions:

$$
\begin{array}{r}
10 \\
-\ 5 \\
\hline
5
\end{array}
\quad \text{or} \quad 10-5=5 \qquad (15)
$$

is the same as:

$$10 + (-5) = 5 \qquad (16)$$

The sum of any number and its true negative is, of course, zero.

A-25 Two's Complement Numbers. The binary number system is useful precisely because it has only two symbols, and as a consequence, polarity cannot be indicated as it is in the decimal system without destroying the very property that permits implementation of binary codes in hardware. This reality together with the fact that from a hardware standpoint, it is generally easier to add than to subtract directly has encourage use of the two's complement system for representing negative numbers in binary. The two's complement of any binary number is formed by:

1. Complementing each digit (changing all "0's" to "1's" and vice-versa).

2. Adding one.

Thus:

$$
\begin{array}{l}
011010011101 \text{ —— Original Number} \\
\quad\downarrow \qquad\qquad\qquad\qquad (17) \\
100101100010 \text{ —— One's Complement} \\
+\qquad\quad 1 \\
\hline
100101100011 \text{ —— Two's Complement}
\end{array}
$$

A-26 Subtraction using two's complement numbers is as follows:

$$
\begin{array}{ll}
12_{10} & 1100 \text{ (Two's complement} \\
-\ 9_{10} & +0111 \text{ of } 1001_2) \\
\hline
3_{10} & 10011 \qquad (18) \\
 & \quad\downarrow \\
\text{(Last carry is ignored)} & 0011_2 = 3_{10}
\end{array}
$$

A-27 Returning to (17), we see that adding the original number and its two's complement produces:

$$
\begin{array}{r}
011010011101 \\
+\ 100101100011 \\
\hline
1000000000000
\end{array}
\qquad (19)
$$

$$000000000000_2 = \text{zero}$$

A-28 The result is zero and demonstrates the validity of treating the two's complement as a true negative of its root binary number.

A-29 Sign and Magnitude. The sign and magnitude code provides another means of indicating polarity. Using this code, the most significant but (MSB) specifies the sign (polarity) and the remaining bits represent the magnitude of the number. For positive numbers, the sign bit is a "0", and for negative nubmers, it is a "1". Examples of positive and negative values using the sign and magnitude code and a 16-bit binary number are:

$$0111000000000010 = +28{,}672_{10} \quad \text{(pos. sign)}$$

$$1111000000000010 = -28{,}672_{10} \quad \text{(neg. sign)}$$

$$0011000000000001 = +12{,}289_{10} \quad \text{(pos. sign)}$$

$$1011000000000001 = -12{,}289_{10} \quad \text{(neg. sign)}$$

Note with sign/magnitude there are two representations for 0:

$$0000000000000000 = 0$$

and

$$1000000000000000 = 0$$

A-30 BINARY CODED DECIMAL (BCD)

A-31 Certain external devices require data to be in binary coded decimal (BCD) format. BCD uses four binary digits to represent ten unique values:

$1001 = (1 \times 2^3) + 0 + 0 + (1 \times 2^0) = 9$
$1000 = (1 \times 2^3) + 0 + 0 + 0 = 8$
$0111 = 0 +)1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 7$
$0110 = 0 + (1 \times 2^2) + (1 \times 2^1) + 0 = 6$
$0101 = 0 + (1 \times 2^2) + 0 + (1 \times 2^0) = 5$
$0100 = 0 + (1 \times 2^2) + 0 + 0 = 4$
$0011 = 0 + 0 + (1 \times 2^1) + 1 \times 2^0) = 3$
$0010 = 0 + 0 + (1 \times 2^1) + 0 = 2$
$0001 = 0 + 0 + 0 + (1 \times 2^0) = 1$
$0000 = 0 + 0 + 0 + 0 = 0$

A-32 In BCD, the value of any four binary digits can not exceed 1001(9). Values 1010(10), 1011(11), 1100(12), 1101(13), 1110(14), 1111(15) are all illegal. Examples of BCD numbers in a 16-bit binary word and their decimal equivalents are:

$1001\ 0110\ 0011\ 0000 = 9630_{10}$
$0001\ 0100\ 0111\ 1000 = 1478_{10}$

Appendix B
ERROR CODES

B-1 Error conditions that can be detected and reported by the Multiprogrammer can be divided into four categories; general programming errors, hardware errors, I/O card errors, and instruction errors. A brief description of each category, followed by a list and description of the associated errors is given in this appendix.

B-2 GENERAL PROGRAMMING ERRORS

B-3 These error codes indicate programming errors that are not associated with any particular instruction. If an asterisk follows an error code it indicates that a second error variable containing the address of a card associated with the error code will be returned.

B-4 HARDWARE ERRORS

B-5 These error codes indicate hardware problems have been detected in a 6942A mainframe or 6943A extender unit(s). Descriptions of errors are provided whenever it is felt that the user may be able to correct the error condition without a service call. No descriptions are provided in the case of error codes that indicate an internal failure requiring a service call to correct.

General Programming Error Codes

Code	Description
-1	Illegal Extended Talk Address - An attempt was made to read back data from a non-existent extended talk address. (i.e. a talk address other than 00-14 was received by the Multiprogrammer).
-2	Illegal Opcode - The instruction sent to the Multiprogrammer contains an opcode that is undefined. This is generally caused by a typographical error.
-3	Illegal Operation in Immediate Mode - The Multiprogrammer was in immediate mode and an attempt was made to execute an instruction that is not permitted in immediate mode. If this error is encountered, return to normal mode or use a different combination of instructions to perform the same function.
-5*	Illegal BCD Code Returned - The card whose address is contained in the following error variable is formatted as a BCD card but it has read back an illegal BCD value. The value sent to the controller is a 4 character alpha-numeric string representing the hexidecimal value of the data. All digits that are legal BCD digits are returned as numbers while all digits that are illegal digits are returned as the letters "A" through "F" indicating the value of the digit (1010 = A, 1011 = B, 1100 = C, 1101 = D, 1110 = E, 1111 = F). In the event this type of data is considered useful by the program it must be read back into a string variable. Reading alpha characters back into a standard variable will yield erroneous data.
-6	Diagnostic Error - This error can occur during procedures used by the service center to diagnose I/O card problems or in the unlikely event a typographical error results in the controller sending a diagnostic opcode to the controller during normal programming.
-7	Diagnostic Error - This error can only occur during procedures used by the service center to diagnose I/O card problems.

B-6 I /O CARD ERROR CODES

B-7 When a defective I/O card is detected by self test (hardware error -15), an error code indicating the type of error is stored in the Multiprogrammer. These error codes can be read into the controller using the Read Format (RF) instruction. The Self Test Error Detection and Card Identifier utility program provided in Appendix C illustrates use of the RF instruction to read the specific I/O card error codes. Error codes -52 through -60 indicate failures in specific circuits on I/O cards. When one of these errors is reported, it indicates what the first error detected was. Note that there could be additional errors in the I/O card but only the first error detected is reported. The Multiprogrammer will not allow communication with the associated I/O card when one of these error codes is detected. If an I/O card hardware error is reported, substitute a new card in the slot and repeat the self test. If the new I/O card passes self test, the problem has been isolated to the replaced I/O card. Note that in rare circumstances the user may desire to communicate with a non-functional card. The Set Format (SF) instruction described in Chapter 5 can be used to change the format of a non-functional card so it can communicate with the system. This feature is **not recommended** but may be used by the advanced programmer under very special circumstances.

B-8 Codes -50 through -60 indicate I/O card errors. As stated previously codes -52 through -60 indicate hardware errors in specific circuits on an I/O card. Codes -50 and -51 are reported if an RF instruction is programmed to an empty card slot or to a card address that is not used.

Hardware Error Codes

Code	Description
-11	Self Test Error
-12	Transmission System Error In Self Test - This error generally indicates either a defective or loose transmission cable between any two frames in a system, a transmission system board not plugged in all the way in a 6943A extender, or a 6943A extender that is not turned on or was turned on after the 6942A.
-13	Self Test Error
-14	Self Test Error
-15	Card Errors Detected In Self Test - One or more I/O cards were found to be defective when Self Test was executed. This is sometimes caused by I/O cards not being fully inserted in Multiprogrammer sockets. Running the Self Test and Card Identifier utility program in Appendix C will identify the addresses of the bad cards as well as the error code associated with the type of failure detected.
-16	Real Time Clock Frequency Error - The measured value of the AC line frequency was out of tolerance. If this error is detected, the multiprogrammer sets the real time clock base to 60 Hz. Since the real time clock base does not agree with the line frequency, however, the clock will not be accurate.
-21	Spurious Interrupt Detected - This error is generally an indication of noise in the system although it can also be caused by a defective I/O card or improper programming of a 69790A Memory Card.
-22	Extender Power Failure - After turning on the system power normally, one of the 6943A extender units in the system had its power removed. This error can also be caused by a transmission cable that has been pulled off. The Multiprogrammer will not allow any further operations, other than a status or error list readback, until the power is restored, at which time a Device Clear must be executed before resuming normal use of the system.
-23	Transmission System Cable Fault - A failure in the transmission system cable was detected after the system had been powered up normally and completed self test. The Multiprogrammer will not allow any further operations, other than a status or error list readback, until the cable fault is corrected, at which time a Device Clear must be executed before resuming normal use of the system.
-24	Mainframe Error Detected

I/O Card Error Codes

Code	Description
-50	No Card in Slot - This code will be reported back to the controller if an RF instruction is programmed to an empty slot.
-51	Card Address not in System - This code indicates that there is no frame connected at the address specified in an RF instruction. For example, reading the format of a card in slot address 102 when frame 0 is the only frame in the system would result in a code of -51 being returned.
-52	Bad Data Readback - The Multi wrote a test pattern to the card and could not read it back correctly.
-53	Arm and EOP do not Clear - Problem with the card's interrupt system.
-54	Data Bus Does Not Float High - Problem with the data paths, possibly a shorted line.
-55	Bad Self-ID - The Multi could not read the card's wake-up data format parameters, probably a problem with the data paths.
-56	Failed Memory Card Test - ID indicates its a Memory Card, and it failed a custom memory card test.
-57	Failed Pattern Sensitivity Test - ID indicates its a Memory Card, and the Multi detected a RAM failure.
-58	Data Readback Error - Error in reading back data from the card, probably a data path error.
-59	Frame Interrupt Register Bad - Problem with frame's interrupt system. This error would be detected when checking a card's interrupt system.
-60	Arm or EOP Error - Problem with the card's interrupt system.
-61	Bad Card Interrupt Register - Problem with the interrupt system (I/O card or frame).

B-9 INSTRUCTION ERROR CODES

B-10 These error codes indicate that programming errors have been detected inside instructions sent to the Multiprogrammer. Each instruction error code consists of two parts, an instruction identification code and an error code. The instruction identification code identifies the instruction that contained the error and the error code identifies the type of error detected. The instruction error code reported by the Multiprogrammer is the addition of the instruction identification code and the error code. For example, an error code of -331 consists of an instruction identification code of -300 and an error code of -31. The instruction identifcation codes and corresponding instruction opcodes are listed first followed by a list and description of the instruction error codes.

B-11 If an asterisk follows an error code it indicates that the complete error message consists of two variables. The instruction error code is returned in the first variable and a positive value representing the address of the card associated with the error is returned in the second variable.

Instruction Identification Codes

ID CODE	OP CODE	ID CODE	OP CODE	ID CODE	OP CODE
−100	OB	−1200	Not Used	−2300	GN
−200	IP	−1300	GI	−2400	WF
−300	OP	−1400	IN	−2500	RS
−400	II	−1500	CC	−2600	RV
−500	OS	−1600	SF	−2700	GS
−600	IE	−1700	CG	−2800	GP
−700	MI	−1800	RF	−2900	SC
−800	MO	−1900	DC	−3000	RC
−900	OI	−2000	AC	−3100	SE
−1000	WA	−2100	CY	−3200	SD
−1100	WU	−2200	WC	−3300	CW

Instruction Error Codes

Error	Description
-30	Number of Cards Incorrect
-31	Illegal Character in the Instruction - An illegal character was detected in the processing of an instruction. This is generally caused by a typographical error.
-32	Unrecognizable Card Address - A frame address other than 0-7, slot address other than 0-15 or sub-address other than 0-3 was sent to the Multiprogrammer.
-33*	Illegal Use of Card Address - This error is caused by attempting to program the same card twice in an instruction that only permits a card to be programmed once. It can also be caused by attempting to program Memory Card 2 with an MI, MO, or SF instruction.
-34*	Data Error - This error indicates that the data sent to a card was either out of range or illegal for the type of card being programmed.
-35*	Limit Exceeded - The programmable bipolar limit specified for the card was exceeded.
-36	Illegal Repeat Factor or Wait in IP/IE - The repeat factor specified was either a negative number or non-integer, or the wait time specified was out of the legal range (0.0 to 6553.5).
-37	Illegal Use of Group Number - A group number was specifed for an instruction that does not allow groups, the opcode and group number did not match when re-programming a group instruction, or the program attempted to use a group that had not been previously defined.
-38	Illegal Group Number - A group number other than 0-9 was specified.
-39	Illegal OI or II in Immediate Mode - An OI or II instruction, using an I/O card that was already active in another instruction was programmed in immediate mode.
-40*	Set Format Parameter Error - An error was detected with one of the parameters in an SF instruction.
-41*	Card Address not in Multiprogrammer System - There is no frame connected at the address specified in the instruction. For example, attempting to program a card in slot address 102 when frame 0 is the only frame in the system.
42*	Card Address Used has a Faulty Card - The slot address being programmed contains a faulty card and can not be programmed.
43*	Card Address Used has no Card in Slot - There is no card installed in the slot address specified.
-99	Miscellaneous - This error means that the Multiprogrammer can not figure out what the user wants.

Appendix C
UTILITY PROGRAMS

C-1 This appendix provides brief descriptions and listings of the utility programs contained in the 9825 and 9835/9845 controller tape cartridges included with this User's Guide. Each cartridge contains the following programs:

Program Name	9825	9835/9845
Error Checking	File 0	CHEKER
Error Trapping	N/A	ERTRAP
Self Test Error Detection and Card Identifier	File 1	CARDID
Multiprogrammer Status	File 2	MPSTAT

C-2 CHECKING SUBROUTINE

C-3 Subroutine "cheker" is used extensively throughout this guide as a programming aid. Inclusion of this subroutine in your programs will provide you with an indication of programming errors that occur when you run your program. "Cheker" will clear SRQ, read the status of the Multiprogrammer, and print (9825) or display (9835/45) any errors detected. It returns to the mainline program with the Multiprogrammer status variables contained in variables U,V,W,X,Y,and Z. (The value contained in variable V is the number of error words detected before the Multiprogrammer error buffer was emptied by "cheker"). The call to the subroutine in the program can be located wherever you require Multiprogrammer status information (such as after an SRQ has been detected) or wherever you would like to check to see if you have made any programming errors.

C-4 To use "Cheker" with a 9835 or 9845 controller your program must first dimension an 11 word array, designated "R". Since all 9835/45 examples in this guide use OPTION BASE 1, the dimension statement used in this manual is "DIM R(11)". This array will be redimensioned by subroutine "Cheker" whenever errors are detected, thereby ensuring that the correct number of variables is read from the error list. Note: if you prefer to use OPTION BASE 0 on your controller, lines 9020 and 9060 of subroutine "Cheker" must be rewritten as follows

```
9020  REDIM R(V-1)
9060  FOR J=0 TO V-1
```

In this case, the Dimension statement would be written "DIM R(10)".

Program Listings

9825A Controller (File 0)

```
0: "cheker":red 7231@,U,V,W,X,Y,Z
1: if V=0;ret
2: prt "error list:";V
3: for J=1 to V;red 7231;prt r[1];next J
4: spc 1;ret
*17853
```

9835/45 Controllers (CHEKER)

```
9000 Cheker:ENTER 723,10;U,V,W,X,Y,Z
9010 IF V=0 THEN RETURN
9020 REDIM R(V)
9030 ENTER 723,11;R(*)
9040 PRINT "ERROR LIST:"
9050 PRINT
9060 FOR J=1 TO V
9070 PRINT R(J)
9080 NEXT J
9090 PRINT
9100 RETURN
```

C-5 ERROR TRAPPING SUBROUTINE

C-6 Subroutine "Ertrap" is an error trapping subroutine used when writing programs on the 9835/45 controllers. This subroutine will allow a return back to the main program whenever an attempt is made to read more data than the Multiprogrammer has available. The mainline program must execute an "ON ERROR GOSUB Ertrap" statement to utilize this subroutine.

Program Listing

9835/45 Controllers (ERTRAP)

```
9200 Ertrap:IF ERRN=159 THEN RETURN
9210 PRINT ERRM$
9220 STOP
```

C-7 SELF TEST ERROR DETECTION AND CARD IDENTIFIER PROGRAM

C-8 This program is a "stand alone" program used to verify that the system is functioning properly and to list the card types that are installed in the card slots. The program first sends an HP-IB "Device Clear" command to the Multiprogrammer System to reset the system and initiate the system self test. The program waits 4 seconds for the self test to complete and then checks if any mainframe hardware or I/O card errors were detected during self test. If no errors were detected, it prints out (9825) or displays (9834/45) the slot numbers, the card types, and the card data format parameters (data type, LSB, no. of bits). If a card error was detected, the error code is listed along with the applicable slot no. If a mainframe hardware error was detected, the error code is printed out (9825) or displayed (9835/45) but the card identification information is not listed.

```
0: fxd 0;prt "SELF TEST and","card identifier";spc ;clr 723;wait 4000
1: if rds(723)#64;prt "no SRQ after","self test","test aborted";sto "end"
2: red 72310,A,B;if B=0;prt "passed self test";spc ;sto "loop"
3: 0→r1→r2;red 72311,r1,r2
4: prt "error detected","error code =",r1;spc
5: prt "failed self test";spc
6: if r1=-15;sto "loop"
7: if r1=-16 and r2=-15;sto "loop"
8: if r1=-21 and r2=-15;sto "loop"
9: sto "end"
10: "loop":for I=0 to 7;I*100→F
11: for H=0 to 15;0→A→G
12: wrt 723,"RF",H+F,"T";ssb "cheker"
13: red 72304,A,B,C,D,E
14: if A=-50;sto "nextH"
15: if A=-51;sto "nextI"
16: prt "slot",H+F
17: if A>-1;sto "cardID"
18: prt "error code =",A;sto "Hloop"
19: "cardID":if A=1;prt "timer card";1→G
20: if A=4;prt "memory card 2";1→G
21: if A=6;prt "memory card 1";1→G
22: if A=12;prt "interrupt card";1→G
23: if A=42;prt "digital output","card";1→G
24: if A=44;prt "digital input","card";1→G
25: if A=45;prt "isolated digital","input card";1→G
26: if A=46;prt "relay output","card";1→G
27: if A=48;prt "D/A card";1→G
28: if A=52;prt "A/D card";1→G
29: if A=56;prt "resistance","output card";1→G
30: if A=58;prt "pulse train","output card";1→G
31: if A=63;prt "counter card";1→G
32: if G=0;prt "card ID code =",A
33: prt "data type =",B
34: if A#1;sto "prtlsb"
35: if C=1;prt "range code =   U"
36: if C=2;prt "range code =   M"
37: if C=3;prt "range code =   S"
38: sto "size"
39: "prtlsb":if B=7;sto "size"
40: fxd 3;prt "lsb =",C
41: "size":fxd 0;prt "# of bits =",D
42: if B=4 or B=7 or E=0;sto "Hloop"
43: fxd 3;prt "limit =",E
44: "Hloop":spc
45: "nextH":next H
46: "nextI":next I
47: "end":spc ;prt "test complete";spc ;spc ;end
48: "cheker":red 72310,U,V,W,X,Y,Z
49: if V=0;ret
50: prt "error list:";spc
51: for J=1 to V;red 72311,r1;prt r1;next J
52: spc ;ret
*2710
```

```
10    OPTION BASE 1
20    DIM R(11)
30    PRINT "SELF TEST AND CARD IDENTIFIER"
40    PRINT
50    RESET 723
60    WAIT 4000
70    STATUS 723;A
80    IF A=64 THEN Mstat
90    PRINT "NO SRQ AFTER SELF TEST, TEST ABORTED"
100   GOTO End
110 Mstat:  ENTER 723.10;A,B
120   IF B<>0 THEN Bad
130   PRINT "PASSED SELF TEST"
140   PRINT
150   GOTO Loop
160 Bad: R1=R2=0
170   IF B=1 THEN ENTER 723.11;R1
180   IF B=2 THEN ENTER 723.11;R1,R2
190   PRINT "ERROR DETECTED, ERROR CODE =",R1
200   PRINT
210   PRINT "FAILED SELF TEST"
220   PRINT
230   IF R1=-15 THEN Loop
240   IF (R1=-16) AND (R2=-15) THEN Loop
250   IF (R1=-21) AND (R2=-15) THEN Loop
260   GOTO End
270 Loop:  FOR I=0 TO 7
280   F=I*100
290   FOR H=0 TO 15
300   A=G=0
310   OUTPUT 723;"RF",H+F,"T"
320   GOSUB Cheker
330   ENTER 723.04;A,B,C,D,E
340   IF A=-50 THEN Nexth
350   IF A=-51 THEN Nexti
360   PRINT "SLOT",H+F
370   IF A>-1 THEN Cardid
380   PRINT "ERROR CODE =",A
390   GOTO Hloop
400 Cardid:  IF A=1 THEN PRINT "TIMER CARD"
410   IF A=1 THEN G=1
420   IF A=4 THEN PRINT "MEMORY CARD 2"
430   IF A=4 THEN G=1
440   IF A=6 THEN PRINT "MEMORY CARD 1"
450   IF A=6 THEN G=1
460   IF A=12 THEN PRINT "INTERRUPT CARD"
470   IF A=12 THEN G=1
480   IF A=42 THEN PRINT "DIGITAL OUTPUT CARD"
490   IF A=42 THEN G=1
500   IF A=44 THEN PRINT "DIGITAL INPUT CARD"
510   IF A=44 THEN G=1
520   IF A=45 THEN PRINT "ISOLATED DIGITAL INPUT CARD"
```

```
530   IF A=45 THEN G=1
540   IF A=46 THEN PRINT "RELAY OUTPUT CARD"
550   IF A=46 THEN G=1
560   IF A=48 THEN PRINT "D/A CARD"
570   IF A=48 THEN G=1
580   IF A=52 THEN PRINT "A/D CARD"
590   IF A=52 THEN G=1
600   IF A=56 THEN PRINT "RESISTANCE OUTPUT CARD"
610   IF A=56 THEN G=1
620   IF A=58 THEN PRINT "PULSE TRAIN OUTPUT CARD"
630   IF A=58 THEN G=1
640   IF A=63 THEN PRINT "COUNTER CARD"
650   IF A=63 THEN G=1
660   IF G=0 THEN PRINT "CARD ID CODE =",A
670   PRINT "DATA TYPE =",B
680   IF A<>1 THEN Prtlsb
690   IF C=1 THEN PRINT "RANGE CODE = U"
700   IF C=2 THEN PRINT "RANGE CODE = M"
710   IF C=3 THEN PRINT "RANGE CODE = S"
720   GOTO Size
730 Prtlsb:   IF B=7 THEN Size
740   PRINT "LSB =",C
750 Size:   PRINT "NUMBER OF BITS =",D
760   IF (B=4) OR (B=7) OR (E=0) THEN Hloop
770   PRINT "LIMIT =",E
780 Hloop:   PRINT
790 Nexth: NEXT H
800 Nexti:   NEXT I
810 End:   PRINT
820   PRINT "TEST COMPLETE"
830   PRINT
840   END
850 Cheker: ENTER 723.10;U,V,W,X,Y,Z
860   IF V=0 THEN RETURN
870   REDIM R(V)
880   ENTER 723.11;R(*)
890   PRINT "ERROR LIST:"
900   PRINT
910   FOR J=1 TO V
920   PRINT R(J)
930   NEXT J
940   PRINT
950   RETURN
```

C-9 MULTIPROGRAMMER STATUS PROGRAM

C-10 This program is a "stand-alone" program that can be used when debugging the system and when writing programs. The program first checks the Multiprogrammer SRQ Status. If SRQ is set, it prints out the instruction or condition (self test) that set SRQ. The program then reads the instruction busy status, checks for any hardware or programming errors, and finally checks for any "armed card" interrupts.

Multiprogrammer Status

Program Listings

9825 Controller (File 2)

```
0: fxd 0;red 72310,A,B,C
1: if A=0;prt "no inst set SRQ";gto "Bstat"
2: if bit(0,A)=1;prt "OB set SRQ"
3: if bit(1,A)=1;prt "IP set SRQ"
4: if bit(2,A)=1;prt "OP set SRQ"
5: if bit(3,A)=1;prt "II set SRQ"
6: if bit(4,A)=1;prt "OS set SRQ"
7: if bit(5,A)=1;prt "IE set SRQ"
8: if bit(8,A)=1;prt "OI set SRQ"
9: if bit(9,A)=1;prt "WA set SRQ"
10: if bit(10,A)=1;prt "WU set SRQ"
11: if bit(13,A)=1;prt "IN set SRQ"
12: if bit(14,A)=1;prt "self tst set SRQ"
13: "Bstat":spc ;red 72313,D,E
14: if D+E=0;prt "no inst busy";gto "erchek"
15: if D=0;gto "nexchek"
16: if bit(0,D)=1;prt "OB is busy"
17: if bit(1,D)=1;prt "IP is busy"
18: if bit(2,D)=1;prt "OP is busy"
19: if bit(3,D)=1;prt "II is busy"
20: if bit(4,D)=1;prt "OS is busy"
21: if bit(5,D)=1;prt "IE is busy"
22: if bit(8,D)=1;prt "OI is busy"
23: if bit(9,D)=1;prt "WA is busy"
24: if bit(10,D)=1;prt "WU is busy"
25: if bit(13,D)=1;prt "IN is busy"
26: if bit(14,D)=1;prt "CC is busy"
27: "nexchek":if E=0;gto "erchek"
28: if bit(0,E)=1;prt "RF is busy"
29: if bit(1,E)=1;prt "DC is busy"
30: if bit(2,E)=1;prt "AC is busy"
31: if bit(3,E)=1;prt "CY is busy"
32: if bit(4,E)=1;prt "WC is busy"
33: if bit(6,E)=1;prt "WF is busy"
34: if bit(7,E)=1;prt "RS is busy"
35: if bit(8,E)=1;prt "RV is busy"
36: if bit(9,E)=1;prt "GS is busy"
37: if bit(10,E)=1;prt "GP is busy"
38: if bit(11,E)=1;prt "SC is busy"
39: if bit(12,E)=1;prt "RC is busy"
40: if bit(13,E)=1;prt "SE is busy"
41: if bit(14,E)=1;prt "SD is busy"
42: "erchek":spc ;if B=0;prt "no errors";gto "chkint"
43: prt "error list:";spc
44: for J=1 to B;red 72311,r1;prt r1;next J
45: "chkint":spc ;if C=0;prt "no armed card","interrupts";gto "end"
46: prt "number of armed","card interrupts=",C;spc
47: for J=1 to C;red 72312,r1;prt "slot number",r1;next J
48: "end":spc ;spc ;end
*6451
```

```
10   OPTION BASE 1
20   DIM A(128),F(11)
30   ON ERROR GOSUB Errtrap
40   ENTER 723.10;A,B,C
50   IF A<>0 THEN Complete
60   PRINT "No instruction set SRQ"
70   GOTO Bstat
80 Complete: IF BIT(A,0) THEN PRINT "CE set SRQ"
90   IF BIT(A,1) THEN PRINT "IF set SRQ"
100  IF BIT(A,2) THEN PRINT "OP set SRQ"
110  IF BIT(A,3) THEN PRINT "II set SRQ"
120  IF BIT(A,4) THEN PRINT "OS set SRQ"
130  IF BIT(A,5) THEN PRINT "IE set SRQ"
140  IF BIT(A,8) THEN PRINT "OI set SRQ"
150  IF BIT(A,9) THEN PRINT "WA set SRQ"
160  IF BIT(A,10) THEN PRINT "WU set SRQ"
170  IF BIT(A,13) THEN PRINT "IN set SRQ"
180  IF BIT(A,14) THEN PRINT "SELF TEST set SRQ"
190 Bstat: PRINT
200  ENTER 723.13;D,E
210  IF D+E<>0 THEN Busy
220  PRINT "No instruction busy"
230  GOTO Erchek
240 Busy: IF D=0 THEN Nexchek
250  IF BIT(D,0) THEN PRINT "CE is busy"
260  IF BIT(D,1) THEN PRINT "IF is busy"
270  IF BIT(D,2) THEN PRINT "OP is busy"
280  IF BIT(D,3) THEN PRINT "II is busy"
290  IF BIT(D,4) THEN PRINT "OS is busy"
300  IF BIT(D,5) THEN PRINT "IE is busy"
310  IF BIT(D,8) THEN PRINT "OI is busy"
320  IF BIT(D,9) THEN PRINT "WA is busy"
330  IF BIT(D,10) THEN PRINT "WU is busy"
340  IF BIT(D,13) THEN PRINT "IN is busy"
350  IF BIT(D,14) THEN PRINT "CC is busy"
360 Nexchek:  IF E=0 THEN Erchek
370  IF BIT(E,0) THEN PRINT "RF is busy"
380  IF BIT(E,1) THEN PRINT "DC is busy"
390  IF BIT(E,2) THEN PRINT "AC is busy"
400  IF BIT(E,3) THEN PRINT "CY is busy"
410  IF BIT(E,4) THEN PRINT "CY is busy"
420  IF BIT(E,6) THEN PRINT "WF is busy"
430  IF BIT(E,7) THEN PRINT "RS is busy"
440  IF BIT(E,8) THEN PRINT "RV is busy"
450  IF BIT(E,9) THEN PRINT "GS is busy"
460  IF BIT(E,10) THEN PRINT "GP is busy"
470  IF BIT(E,11) THEN PRINT "SC is busy"
480  IF BIT(E,12) THEN PRINT "RC is busy"
490  IF BIT(E,13) THEN PRINT "SE is busy"
500  IF BIT(E,14) THEN PRINT "SD is busy"
510 Erchek: PRINT
520  IF B<>0 THEN List
530  PRINT "No errors detected"
540  GOTO Chkint
550 List:  REDIM R(B)
560  ENTER 723.11;R(*)
570  PRINT "Error list:"
580  PRINT
590  FOR J=1 TO B
600  PRINT R(J)
610  NEXT J
620 Chkint: PRINT
630  IF C>0 THEN Armed
640  PRINT "No armed card interrupts"
650  GOTO End
660 Armed: C=0
670  FOR J=1 TO 128
680  A(J)=-1
690  NEXT J
700  ENTER 723.12;A(*)
710  FOR J=1 TO 128
720  IF A(J)<>-1 THEN C=C+1
730  NEXT J
740 PRINT "Number of armed card interrupts =";C
750  PRINT
760  FOR J=1 TO C
770  PRINT "Slot number";A(J)
780  NEXT J
790 End: PRINT
800  END
810 Errtrap: IF ERRN=159 THEN RETURN
820  PRINT ERRM$
830  STOP
```

Appendix D
DEBUGGING THE SYSTEM

D-1 This appendix contains information which will help you avoid problems when you are first beginning to program your system as well as information that will help you isolate and correct problems if they should occur.

D-2 When the system is initially turned on or is reset (clr 723 or RESET 723, depending on the controller) a self test is automatically performed on the mainframe, extenders (if any), and all I/O cards. On completion of the self test the Multiprogrammer will set SRQ to notify the controller that self test is complete. If any hardware errors are detected during self test, the errors will be stored within the Multiprogrammer and may be read back from HP-IB extended talk address 11. In the case of I/O cards, specific error codes identifying the type of failure will also be stored and may be read back by means of the RF instruction. The Self Test and Card Identifier utility program described in Appendix C will reset the Multiprogrammer system, read the status of the Multiprogrammer after it completes self test, and print out any errors detected. When initially turning on a new system, you should run this program to be certain that the system is operating properly before starting to write programs. Any errors reported by the program should be corrected before continuing.

D-3 Once you have determined that your system hardware is operational (it passes the Self Test program) the next step is to begin writing your program. A person totally unfamiliar with a 6942/43 Multiprogrammer system is likely to make a few mistakes when first attempting to write programs. These mistakes will result in error conditions in the Multiprogrammer. Error conditions in the Multiprogrammer can be grouped into two general categories: errors that are reported by the Multiprogrammer by means of a service request and Multiprogrammer status variable (reportable errors) and errors that the Multiprogrammer will be unable to report (non-reportable errors).

D-4 REPORTABLE ERRORS

D-5 Error conditions that result in SRQ being set are generally very easy to troubleshoot since coded, well defined error messages (see Appendix B) can be read back from extended talk address 11. The Self Test and Card Identifier program mentioned previously is an example of a program that monitors service request and Multiprogrammer status to determine if errors have been detected and, if they have, it reads extended talk address 11 and prints the error codes on the controller.

D-6 Another example of a program that monitors reportable errors is subroutine "checker" in Appendix C. Subroutine "cheker" is a powerful debugging tool that should be included in any program you write to help you during the debugging phase of program development. The "cheker" routine reads the status of the Multiprogrammer and prints out any error messages that have been detected. Once the program is operational, "cheker" can either be removed from the program or left in and used whenever a status read of the Multiprogrammer system is required.

NOTE

Since subroutine "cheker" reads the Multiprogrammer status and clears SRQ it should not be called at random in a program that normally requires an SRQ to notify the controller that a process is complete. In this case it can be included in the routine you write to process SRQ once it has been detected.

D-7 Example D-1 illustrates use of subroutine "cheker" in debuggging a program. This example assumes that output cards are installed in slots 1 and 2 and an input card is installed in slot 5. All other slots are empty. Subroutine "cheker" is not shown in this example but is listed in Appendix C.

Example D-1. Debugging a Program With Subroutine "cheker"

9825A Controller

```
0: wrt 723,"Op,1,5T"
1: wrt 723,"WC,14,7T"
2: wrt 723,"IP,5T"
3: wrt 723,"OS,2,T"
4: ssb "cheker"
5: red 72301,A
6: end
```

9835A Controllers

```
10   OPTION BASE 1
20   DIM R(11)
30   OUTPUT 723;"Op,1,5T"
40   OUTPUT 723;"WC,14,7T"
50   OUTPUT 723;"IP,5T"
60   OUTPUT 723;"OS,2,T"
70   GOSUB Cheker
80   ENTER 723.01;A
90   END
```

Explanation (Example D-1):

9825	9835/45	Description
	10	Set Controller to Option Base 1.
	20	Dimension an 11 word array for use by subroutine "cheker".
0	30	Illegal opcode sent out (Op instead of OP)
1	40	WC instruction used to program slot 14 which is empty.
2	50	Good IP instruction sent out.
3	60	Data has been left out of an OS instruction used to program a card in slot address 2.
4	70	Check for errors in the program.
5	80	Read back data from input card programmed by IP instruction.
6	90	End of program.

D-8 Executing example D-1 will result in the following error message being printed on the 9825 controller or displayed on 9835 or 9845 controllers.

Error Message	Description
ERROR LIST:	
–2	Opcode Error
–2243 14	There is no card in the slot address (14) specified in a WC instruction.
–534 2	A data error has been detected in an "OS"instruction when programming slot address 2 (data word has been left out in this example).

D-9 Since the error codes reported in example D-1 are fairly self-explanatory, correcting the errors become a simple matter of correcting the opcode in the OP instruction, installing an output card in slot 14 (Caution: turn off the power first), and adding a data value to the OS instruction. A description of all error codes is provided in Appendix B.

D-10 NON-REPORTABLE ERRORS

D-11 Error conditions that can not be reported by the Multiprogrammer are more difficult to interpret than those that can be reported. It is possible, due to improper programming or improper I/O card connections, to set the Multiprogrammer to a state where it is either not ready to respond or unable to respond to the controller, thereby stalling the controller. This condition, where the controller is waiting for a response from the Multiprogrammer but the Multiprogrammer can not respond will be referred to as a hang-up condition. Hang-up conditions can be detected by using the timeout feature of the controller.

D-12 Hang-up conditions result from three specific problems: an unterminated gate on an I/O card, an I/O card that takes a long time to complete, or a memory overflow condition. Hang-ups due to a card with an unterminated gate or an I/O card that takes a long time to complete will not prevent additional instructions from being sent to the Multiprogrammer or Multiprogrammer status from being read by the controller (red 72310,A,B/ENTER 723.10;A,B). However, hang-ups due to a memory overflow condition will prevent further communication of any kind with the Multiprogrammer until the condition has been corrected.

D-13 An Unterminated Gate on an I/O Card

D-14 By removing a jumper on a 69731A, 69770A, or 69771A I/O card, an external device can be used to control the data transfer time of the card. If the jumper has been removed but the card is not connected to an external device, or the external device is not turned on or not working properly, the card will not be able to complete a data transfer. Programming a card with an unterminated gate using an OB,IP,O,II,OS,IE, or OI instruction will cause the instruction that has programmed the card to remain busy indefinitely, waiting for the card to complete. If the Multiprogrammer is in serial mode and the I/O card has been programmed by an OB,IP,OP,OS, or IE instruction all subsequent instructions (other than CC,-CG,CW,GI,MI,MO,RF,RS, or SF instructions which run immediately) will be forced to wait until the busy instruction completes before they are started. In parallel mode, all subsequent instructions of the same type as the busy instruction, other than an II or OI, will be forced to wait until the busy instruction completes before they are started. In either mode, any attempt to read data from an instruction in the system (other than an II or OI) that is busy or waiting to run will force the controller to wait for the busy card to complete before being allowed to read the data. Since a card with an unterminated gate will never complete, the controller will be hung-up.

NOTE

Since multiple II and/or OI instructions can be running concurrently, a hung-up card in an II or OI instruction will tie up that particular instruction but allow other II or OI instructions to run as long as the other instructions do not attempt to use the same card.

D-15 In the event that a hang-up has occurred due to an I/O card with an unterminated gate, you can stop the program and use a Read Status instruction to determine which card in the system has caused the hang-up ("RS, card address,T"). The card found to be busy when a hang-up occurs is the cause of the hang-up.

D-16 After determining which card has caused the hang-up condition you have four choices:

1. If you do not need the data stored in the Multiprogrammer up to the time of the hang-up, turn off the Multiprogrammer, correct the problem, turn on the Multiprogrammer, and rerun the program.

2. If you do need the data already stored in the Multiprogrammer, it can be read back in a normal fashion. The problem that caused the hang-up should then be corrected before the program is rerun.

3. If the instruction that contains the hung-up card is an input instruction and you would like to read the data obtained by other cards in the instruction, or you would like other instructions stored in the Multiprogrammer at the time of the hang-up to complete before you correct the problem, send a Clear Card instruction ("CC,card address,T") to the card that is hung-up. This will allow the instruction containing the hung-up card to finish as if the card had completed normally. Any instructions in the Multiprogrammer that are waiting to run will then be started. Data from the hung-up instructions (assuming it is an input instruction) as well as data from the instructions that were waiting to run can then be read back, although data from the card that was cleared should be discarded. The problem can then be corrected and the program rerun.

4. If you would like to continue execution of the program without correcting the problem, simply send a Clear Card instruction to the hung-up card. The program may then be continued at the line that had caused the hang-up.

D-17 Example D-2 is a sample program which illustrates programming two 69771A Digital Input cards, in slots 1 and 2, to take a reading, and then, print the results on the controller. The timeout feature of the controller is used to monitor any timeouts that may occur during execution of the program (the timeout feature of the controller is described in the appropriate controller manual). As shipped from the factory, these cards will not cause a timeout when example D-2 is executed. However, disconnecting jumper W52 on a Digital Input card without connecting the gate/flag lines to an external device, will cause a timeout to occur when the program is executed. In this case, the controller prints "timeout detected" and stops. If CONTINUE is depressed on the controller the status of both cards will be printed out and the program will again stop. No corrective action is taken in this example since you will normally determine the course of action to take as described above.

Example D-2. Detecting a Hang-up Caused By an I/O Card

9825A Controller

```
0: time 10000
1: on err "timout"
2: wrt 723,"IP,1,2T"
3: red 72301,A,B
4: prt A,B
5: end
6: "timout":if rom#69 or ern#4;ret
7: prt "timeout detected"
8: stp
9: wrt 723,"RS,1,2T"
10: red 72308,C,D
11: fxd 0;prt "slot #1 status =",C
12: prt "slot #2 status =",D
13: stp
```

9835/45 Controllers

```
10   SET TIMEOUT 7;10000
20   ON INT #7 GOSUB Timout
30   OUTPUT 723;"IP,1,2T"
40   ENTER 723.01;A,B
50   PRINT A,B
60   END
70 Timout:  IF TIME OUT(7)<>1 THEN RETURN
80   PRINT "TIMEOUT DETECTED"
90   PAUSE
100  OUTPUT 723;"RS,1,2T"
110  ENTER 723.08;C,D
120  PRINT "SLOT #1 STATUS =",C
130  PRINT "SLOT #2 STATUS =",D
140  STOP
```

Explanation:

9825	9835/45	Description
0	10	Set 10 second timeout.
1	20	Establish linkage to timeout subroutine.
2	30	Take a reading from two input cards using an IP instruction.
3	40	Read back data obtained by IP instruction.
4	50	Print data.
5	60	End of Program.
6	70	9825: Error has been detected. If error is not caused by a timeout on Extended I/O ROM return to main program (in your programs you would probably want to display the error message). 9835/45: An interrupt has been detected on HP-IB interface. If interrupt was not caused by a timeout, return to main program (in your programs you would continue processing the interrupt).
7	80	A timeout has been detected. Controller prints "timeout detected".
8	90	Program pauses.
9	100	Read the status of the cards that have been programmed into the Multiprogrammer.
10	110	Read the status of the cards from the Multiprogrammer into controller variables C and D.
11	120	Print status of first card.
12	130	Print status of second card.
13	140	Program stops.

D-18 An I/O Card That Takes A Long Time To Complete

D-19 By their very nature, some cards can take a long time to complete. For example, a 69736A Timer/Pacer card can be programmed to output a timed pulse 10 hours long. Or a 10-hour long pulse train can be programmed from a 69735A Pulse Train card. In addition, some cards such as the 69731A Digital Output card or the 69771A Digital Input card can be connected to allow an external device to control the gate/flag data transfer time of the card. In some cases the external device may require a substantial amount of time before completing a data transfer, tying up the card and associated instruction within the Multiprogrammer until the data transfer completes. It is perfectly acceptable to program cards in this manner. However, keep in mind that attempting to read data from an instruction in the system (other than an II or OI) that contains a busy card will force the controller to wait for the busy card to complete before being allowed to read the data. In serial mode, attempting to execute succeeding instructions will also force the controller to wait until the busy card completes because succeeding instructions will not run until the busy instruction completes.

D-20 After attempting to read data from an instruction containing a busy card, you may decide to stop the readback and come back for the data at a later time. In this case, you can stop the program and either manually execute instruction from the keyboard of the controller , or restart the program at a different line.

D-21 Memory Overflow

D-22 A 6942A Multiprogrammer without extenders contains 1462 words of memory available for running instructions. Each 6943A Extender unit that is used (up to 7) requires 24 words of memory. Therefore, a full system (one 6942A and seven 6943A's) has 1294 words available for the user's program. Memory utilization information provided in Chapter 6 allows the user to calculate how much memory his program requires, so he can take full advantage of the available memory and avoid memory overflow. If the memory overflows, no instructions can complete and the system will hang-up. Memory overflow will result from any of the following conditions:

1. An input instruction is programmed to take more readings than the Multiprogrammer can store.
2. An unterminated gate exists on an I/O card and the controller continues to send instructions to the Multiprogrammer after the card with the unterminated gate was programmed.
3. Data is not being read back from the Multiprogrammer after input instructions are programmed.
4. The immediate mode (GI instruction) is programmed when the memory is nearly full. Note that when a GI instruction is executed, the Multiprogrammer requires 40 words of memory to store the previous state of the system.

D-23 If a memory overflow condition occurs, no additional instructions can be processed and no data from an extended talk address can be read back. The only possible way to recover is to reset the Multiprogrammer by sending a clear message (clr 723/RESET 723) or turning the 6942A off and on When the Multiprogrammer is reset, all data stored within the Multiprogrammer will be lost. Example D-3 illustrates memory overflow condition 1 listed above by attempting to take 300 readings from each of 5 input cards located in slots 1 through 5, respectively. Since this will require 1527 words of memory (1500 words of data + 27 words of overhead) a memory overflow condition will result. The next output or input of any type from the controller to the Multiprogrammer will cause the controller to hang-up. In this example, the Multiprogrammer Status Read (red 72310/ENTER 723.10) which follows the IP instruction will cause the hang-up. The timeout feature of the controller is used to detect the hang-up condition.

Example D-3. Causing a Memory Overflow Condition

9825A Controller

```
0: time 10000
1: on err "timout"
2: wrt 723,"IP,R300,1,2,3,4,5T"
3: red 72310,A,B,C
4: end
5: "timout":if rom#69 or ern#4;ret
6: prt "timeoutdetected"
7: stp
```

9835/45 Controllers

```
10   SET TIMEOUT 7;10000
20   ON INT #7 GOSUB Timout
30   OUTPUT 723;"IP,R300,1,2,3,4,5T"
40   ENTER 723.10;A,B,C
50   END
60 Timout:  IF TIME OUT(7)<>1 THEN RETURN
70   PRINT "TIMEOUT DETECTED"
80   STOP
```

Explanation:

9825	9835/45	Description
0	10	Set 10 second timeout.
1	20	Establish linkage to timeout subroutine.
2	30	Take 300 readings from each of 5 input cards.
3	40	Read Multiprogrammer status.
4	50	End of program.
5	60	9825: Error has been detected. If error is not caused by a timeout on extended I/O ROM return to main program (in your program you would probably want to display the error message). 9835/45: An interrupt has been detected on HP-IB interface. If interrupt was not caused by a timeout return to main program. (In your program you would continue processing the interrupt).
6	70	A timeout has been detected. Controller prints "timeout detected".
7	80	Program stops.