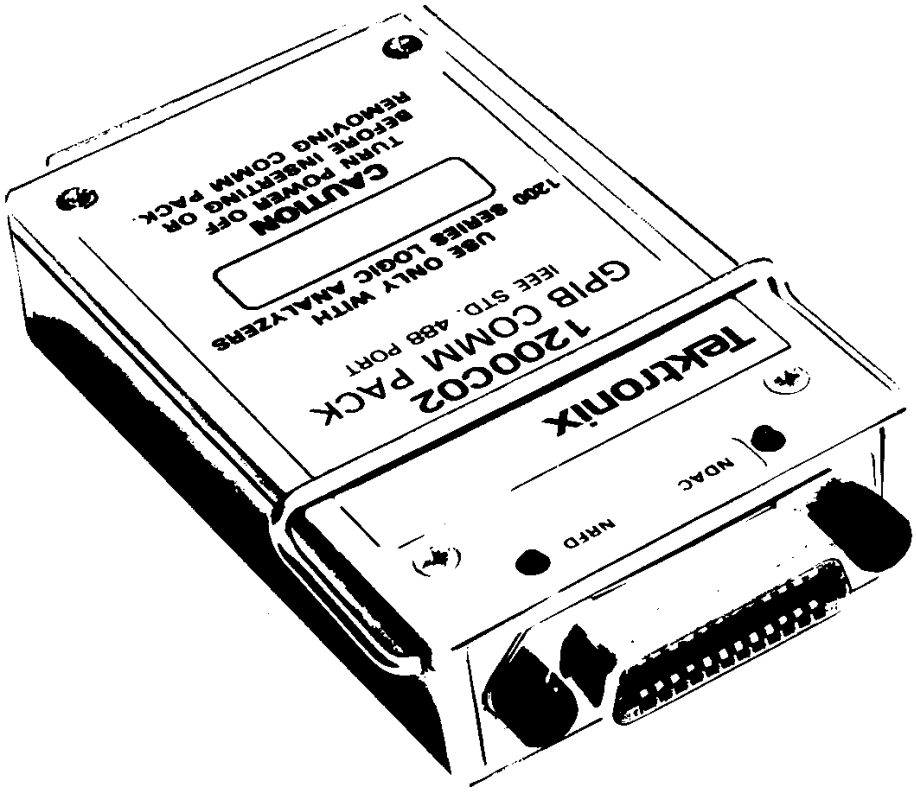
Part No. 070-4344-00
Product Group 57



GPIB COMM Pack 1200C02

FIRST PRINTING MAY 1983
REVISED PRINTING NOVEMBER 1986

Tektronix®
COMMITTED TO EXCELLENCE

Tektronix, Inc.
Walker Road Industrial Park
P.O. Box 4600
Beaverton, Or. 97076

PREFACE

This manual is a supplement to the *1240/1241 Logic Analyzer Operator's Manual.* It provides the additional information necessary to operate the 1240 or 1241 with the 1200C02 GPIB COMMunications Pack. Operation of the 1241 with the 1200C02 is much the same as for the 1240. Unless otherwise noted, it should be assumed that text supporting the 1240 is also true for the 1241. This manual was written for both novice and more experienced GPIB users. It assumes familiarity with both the 1240 Logic Analyzer, and the controller being used.

Users unfamiliar with GPIB should begin by reading *Appendix E,* which provides a GPIB overview.

All service information is located in the 1240 Service Manual.

Sections 1 and 2 provide reference information relating the 1240 to the GPIB.

Section 3 provides generic programming examples to illustrate the use of the 1240 commands. These programming examples are designed to aid the programmer in organizing his thoughts. However, they should not be entered as code in a program.

Section 4 describes formats of 1240 setups, memories, and ROM and RAM packs.

Appendix A provides the necessary information for formatting 1240 setups, memories, and ROM and RAM packs.

Appendix B describes 1240 and 1241 Display Codes.

Appendix C lists 1240 Key Codes.

Appendix D is a table of Error and Event Codes.

This manual describes GPIB only in relation to the 1240.

This instrument complies with Tektronix codes and formats.

For more information about GPIB, refer to the IEEE 488-1978 standard, which is published by:

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47 Street
New York, New York 10017

TABLE OF CONTENTS

		Page
PREFACE		i
LIST OF ILLUSTRATIONS		v
LIST OF TABLES		vi
OPERATOR'S SAFETY SUMMARY		vii
Section 1	**INTRODUCTION**	
	OVERVIEW	1-1
	Capabilities	1-1
	Limitations	1-1
	Optional Accessories	1-1
	Selecting a Controller	1-2
	INSTALLATION	1-2
	PRELIMINARY OPERATION INFORMATION	1-3
	Power-Up Diagnostics	1-3
	Menus	1-3
	COMM Port Control Menu	1-4
	Port Status Display	1-4
	Selectable Parameters	1-5
	1240 Soft Function Keys	1-5
	Error Messages	1-5
	Remote Menu	1-5
	GPIB COMM Pack LEDs	1-6
	THE STATUS BYTE	1-6
Section 2	**PROGRAMMING**	
	DEVICE-DEPENDENT MESSAGES	2-1
	Remote and Local States	2-1
	Device-Dependent 1240 Messages	2-1
	GPIB Data Transfers	2-9
	DATA BLOCK FORMAT	2-9
	DATA TRANSFER—ACQMEM and REFMEM/CONTROLLER	2-10
	Uploading	2-10
	Downloading	2-11
	DATA TRANSFER—INSETUP and RAMPACK	2-13
	Uploading	2-13
	Downloading	2-13
	Interrupting and Aborting Data Transfer in Progress	2-14
	1240 GPIB INTERFACE	2-15
	Interface Control Messages	2-15

TABLE OF CONTENTS (cont.)

Section 3 PROGRAMMING EXAMPLES

RECEIVING A SETUP FROM THE 1240 3-1
SENDING A SETUP TO THE 1240 3-1
GETTING THE ACQMEM FROM THE 1240 3-2
SENDING A REFMEM TO THE 1240 3-2
PERFORMING A REMOTE DATA ACQUISITION 3-2
GETTING KEYSTROKES FROM THE 1240 FRONT PANEL 3-3
WRITING TO THE 1240 DISPLAY 3-4
HANDLING SERVICE REQUESTS (SRQs) 3-4
SUPPORTING THE COMM PORT CONTROL MENU'S FUNCTION KEYS 3-5
PERFORMING A REMOTE AUTO-RUN 3-7
GETTING A COPY OF A 1240 RAM PACK 3-8

Section 4 USER-GENERATED RAM AND ROM PACKS

CONTENTS 4-1
REQUIRED EQUIPMENT 4-1
SERVICE INFORMATION 4-1
THEORY 4-1
- Format 4-1
- Pack Header 4-2
- Directory Format 4-2
- Files 4-3
- Trailer 4-4
- Trailer Format 4-4

CREATING A ROM PACK 4-5
- Upload 4-6
- Header 4-6
- Directory 4-6
- Files 4-6
- Change Directory 4-6
- Trailer 4-6
- Download and Burn 4-7

Appendix A Instrument Setup, Memory Image and Radix Table Formats

INSTRUMENT SETUP A-1
- Variable Descriptions A-4
- Description of a 'PRLWORD' A-14

MEMORY IMAGE A-15
- Variable Descriptions A-15
- General Discussion of the Memory Image Structure A-19
- Overview of Data Correlation A-20
- Locating the Trigger A-21

RADIX TABLE A-22
- Overview A-22
- ROM Radix Table (RTABLE) Format A-22

TABLE OF CONTENTS (cont.)

Appendix B 1240 and 1241 Display Codes

Appendix C 1240 Key Codes

Appendix D Error and Event Codes

CODE TABLES D-1
EVENT CODE EXPLANATIONS D-4

Appendix E Introduction to GPIB

ELECTRICAL ELEMENTS E-1
MECHANICAL ELEMENTS E-1
Connecting GPIB Systems E-1
GPIB Connector E-2
GPIB E-3
FUNCTIONAL ELEMENTS E-4
Typical GPIB System E-4
Controllers, Talkers, and Listeners E-5
Interface Messages E-5
Addressed Commands E-7
Universal Commands E-7
Listen Addresses E-7
Talk Addresses E-7
Secondary Addresses or Commands E-7
GPIB Signal Line Definitions E-7
The Data Bus E-7
The Transfer Bus E-8
Data Valid (DAV) E-9
Not Ready For Data (NRFD) E-9
Not Data Accepted (NDAC) E-9
The Management Bus E-9
Attention (ATN) E-9
End or Identify (EOI) E-9
Interface Clear (IFC) E-9
Remote Enable (REN) E-10
Service Request (SRQ) E-10
Interface Functions E-10
Source and Acceptor Handshake Functions E-11
Talker Function E-11
Listener Function E-11
Service Request Function E-12
Remote-Local Function E-12
Parallel Poll Function E-12
Device Clear Function E-13
Device Trigger Function E-13
Controller Function E-13
TYPICAL ACTIVITY E-13
Remote to Local Changes E-13
Serial Polls E-14
OPERATIONAL ELEMENTS E-14

LIST OF ILLUSTRATIONS

Figure No.	Title	Page
1-1	COMM pack installed in 1240 communication port	1-3
1-2	Typical 1240 screen display in COMM Port Control menu	1-4
2-1	Controller uploads acquisition memory contents with ACQMEM? command	2-10
2-2	Controller downloads data to 1240 compressed memory with ACQMEM command	2-11
2-3	Controller modifies 1240 acquisition memory with LOAD command	2-12
2-4	Controller uploading Setup information with INSETUP? command	2-13
2-5	Controller downloading the Setup memory with the INSETUP command	2-14
4-1	Basic ROM and RAM Pack format	4-1
4-2	How up to four RAM Packs are reorganized for storage in one ROM Pack	4-5
4-3	Relationship between physical location and address contents	4-7
A-1	Data correlation	A-21
E-1	GPIB interface connector plug, with pin locations	E-2
E-2	The GPIB bus lines	E-3
E-3	A typical GPIB system setup with four primary-addressed devices	E-4
E-4	The GPIB code chart	E-6
E-5	An example of data byte traffic on the GPIB	E-8
E-6	A typical handshake timing sequence	E-9

LIST OF TABLES

Table No.	Title	Page
1-1	Operating Specifications	1-2
1-2	The Status Bits	1-6
1-3	Status Byte Interpretation	1-7
2-1	GPIB Interface Functions Supported by the 1240	2-15
2-2	Controller to 1240	2-16
2-3	1240 to Controller	2-17
4-1	Directory Entry Format	4-3
4-2	ROM Pack Trailer Addresses	4-4
A-1	Setup Memory Locations	A-1
B-1	1240 Video Codes	B-1
B-2	1240 Changeable Characters	B-2
B-3	1240 Copycat Characters	B-3
B-4	1241 Video Codes	B-4
B-5	1241 Changeable Characters	B-5
B-6	1241 Changeable Characters (cont.)	B-6
C-1	1240 Key Codes	C-1
D-1	System Events, Priority 1	D-1
D-2	Command Errors, Priority 2	D-1
D-3	Execution Errors, Priority 3 & 4	D-2
D-4	Soft Key Events, Priority 5	D-3
D-5	Operation Complete Events, Priority 6	D-4
D-6	Normal Device-Dependent Status, Priority 7	D-4
E-1	GPIB Interface Functions—General	E-10

OPERATOR'S SAFETY SUMMARY

Specific notes and cautions will be found throughout the manual where they apply, but may not appear in this summary.

TERMS

In This Manual. CAUTION statements identify conditions or practices that could result in damage to the equipment or other property.

As Marked on Equipment. CAUTION indicates a personal injury hazard not immediately accessible as one reads the marking, or a hazard to property including the equipment itself.

Refer to the *Operator's Safety Summary* in the *1240 Operator's Manual* for more information.

INTRODUCTION

OVERVIEW

1240 Operation can be controlled via the IEEE 488 General Purpose Interface Bus (GPIB).

With the GPIB COMM Pack, the 1240 conforms to the IEEE specification 488-1978, *Standard Digital Interface for Programmable Instrumentation.*

Capabilities

The GPIB COMM Pack allows the controller to do:

ACQUISITION CONTROL

The Controller can:
- start a data acquisition
- start the auto-run function
- stop an acquisition or auto-run

The 1240 will notify the controller when:
- a controller-initiated acquisition is over
- a controller initiated auto-run is over

DISPLAY CONTROL
- The controller can write to the 1240 display

FRONT PANEL CONTROL
- The controller can request 1240 keystrokes
- The controller can request an auditory tone from the 1240

DIAGNOSTICS
- The controller can initiate 1240 diagnostics
- The controller can request diagnostic information

BLOCK TRANSFERS - The following structures can be sent to and received from the 1240 as a block of data (not as a sequence of keystrokes):
- instrument Setup
- acquisition Memory
- reference Memory
- RAM Pack contents

Limitations

The 1240/GPIB Interface allows communication only with a controller programmed for this purpose. The 1240 can function as a talker or listener, but not as a controller.

Optional Accessories

The following optional accessories are available for use with the 1200C02 GPIB COMM Pack:
- Cable, 2-meter, part no. 012-0630-01
- Cable, 4-meter, part no. 012-0630-02
- Cable, 1-meter, double-shielded, low EMI, part no. 012-0991-01
- Cable, 2-meter, double-shielded, low EMI, part no. 012-0991-00
- Cable, 4-meter, double-shielded, low EMI, part no. 012-0991-02

Selecting a Controller

The controller for the 1240 must have a GPIB connector and software compatible with the IEEE 488-1978 Standard.

For flexibility and ease of use, the controller should be able to respond to a service request, poll devices serially in any order, evaluate status bytes bit-by-bit, and execute user-defined code.

Table 1-1
OPERATING SPECIFICATIONS

Characteristic	Description
Temperature -	
Operating:	−15°C. to +55°C.
Storage:	−62°C. to +85°C.
Humidity:	95-97% relative humidity for five days cycled from 30°C. to 60°C.
Altitude -	
Operating:	4.5 km (approx. 15,000 ft.)
Non-operating:	15 km (approx. 50,000 ft)
Vibration	0.64 mm (0.025 in.), 10 Hz to 55 Hz, 75 minutes
Shock	50 g (1/2 sine wave), 11 ms; 18 shocks, 3 on each face.

INSTALLATION

To use the interface, power off the 1240 and controller. Plug the COMM Pack into the back panel of the 1240. Secure the bow handle to the spurs on the mainframe to prevent the COMM Pack from accidentally working loose.

Then plug the GPIB interface from the controller into the COMM Pack. Tighten the strain-relief screws to prevent the interface plug from slipping out of the COMM Pack.

See *Figure 1-1.*

CAUTION

Make sure the 1240 and controller are powered OFF before plugging in the COMM Pack.

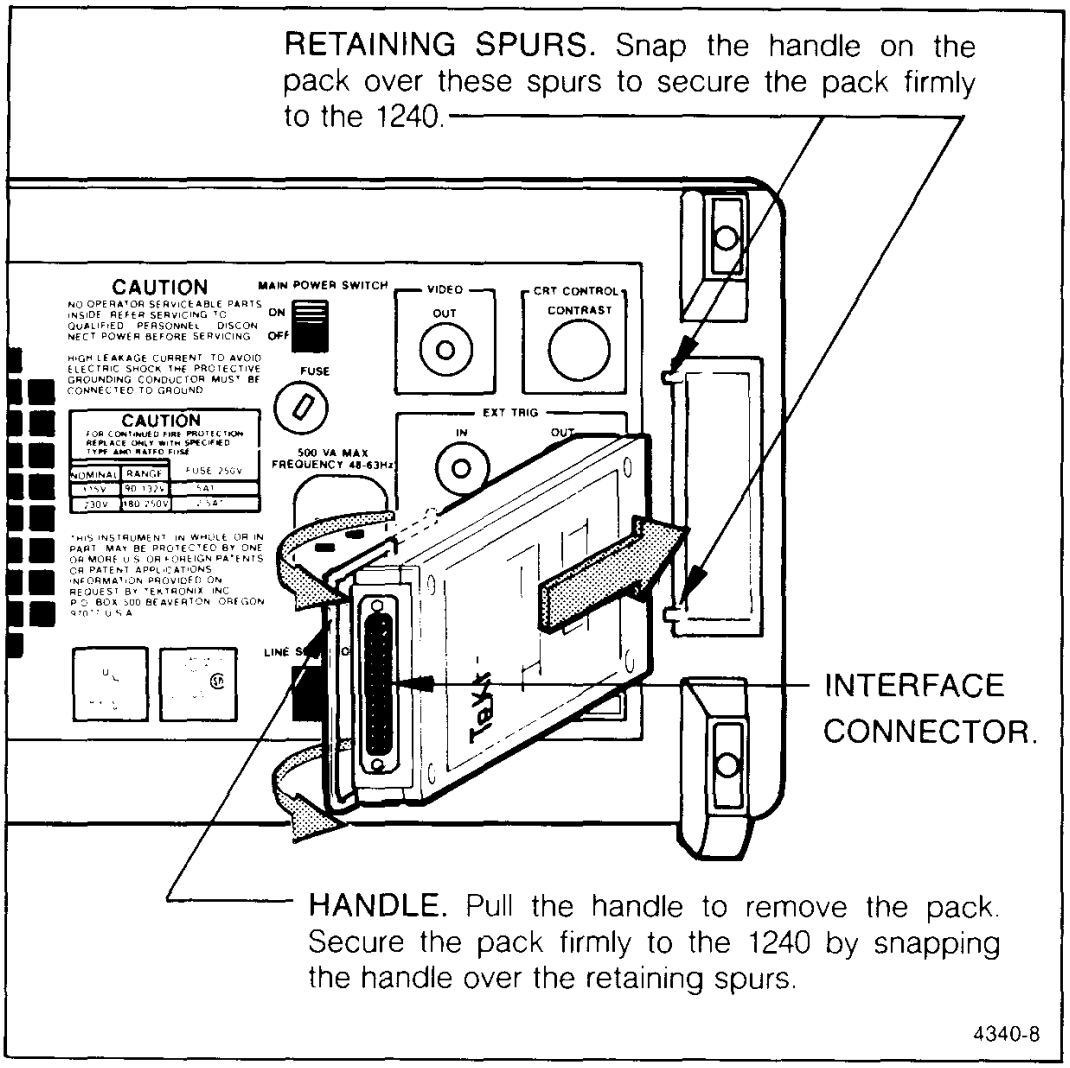


Figure 1-1. COMM pack installed in 1240 communication port.

PRELIMINARY OPERATION INFORMATION

Power-Up Diagnostics

If 1240 power-up diagnostics reveal an error, the 1240 remains under diagnostic monitor control. You must exit the diagnostic monitor before GPIB communication is possible. If no error is present, the 1240 exits the diagnostic monitor automatically.

Menus

Each 1240 menu is designed to perform a specific function. When a menu is called up, it assumes control of the instrument. The two menus discussed in this manual are COMM Port Control, and Remote.

COMM Port Control Menu. Once the COMM Pack has been installed, enter the menu by pressing the UTILITY key, and then the COMM PORT CONTROL soft key. Figure 1-2 shows a typical COMM Port Control menu display.

Use the COMM Port Control menu to select port status, GPIB address, and message termination type. You may also initiate data transfers with the soft keys.

Port Status Display - When in COMM Port Control menu (ONLINE), or Remote menu, the top line of the 1240 screen displays the status of the GPIB port hardware. The display contains three fields: left, center, and right.

Left – Remote/Local State of Interface (see IEEE Std. 488-1978)

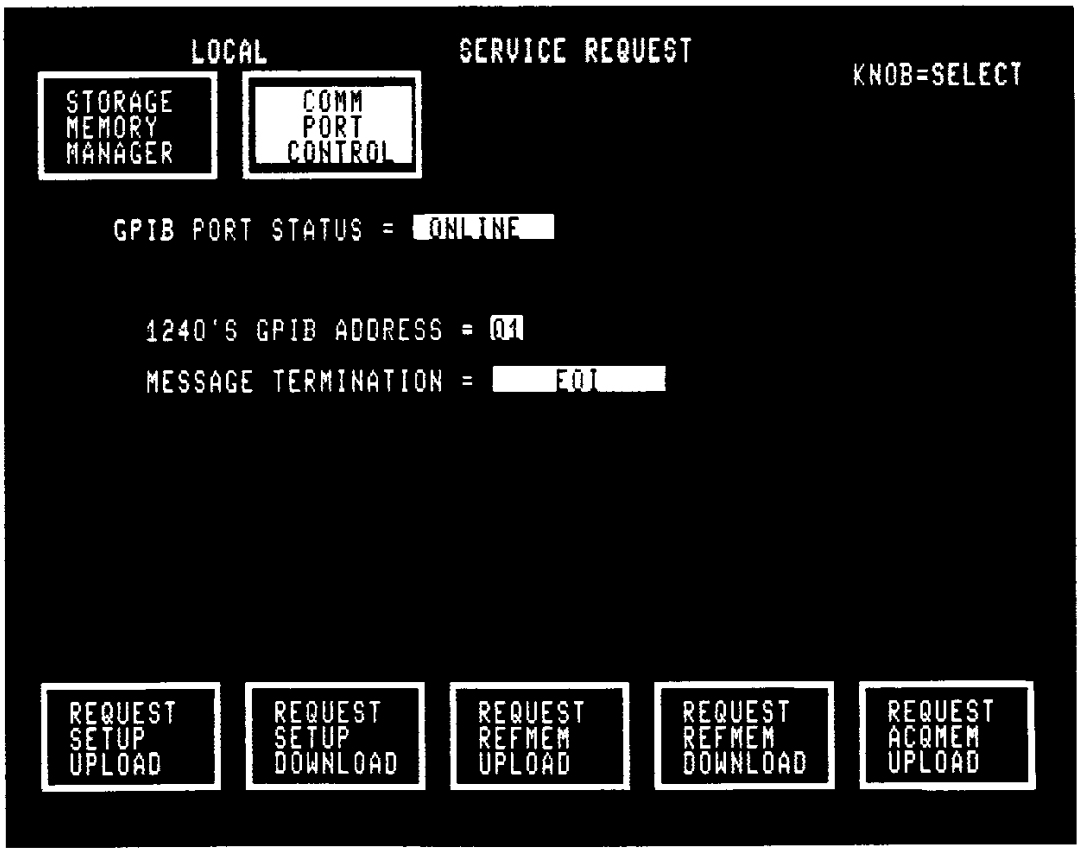
LOCAL	local or local with lockout state
REMOTE (STOP=RTL)	remote state
REMOTE WITH LOCKOUT	remote with lockout

Center – Service Request

SERVICE REQUEST	SRQ being asserted
(blank)	SRQ not being asserted

Right – Listener/Talker

LISTENER	1240 addressed to Listen
TALKER	1240 addressed to Talk
(blank)	not addressed



4344-02

Figure 1-2. Typical 1240 screen display in COMM Port Control menu.

Selectable Parameters. Some GPIB communication parameters are selected through input fields in the 1240 COMM Port Control menu. The 1240 non-volatile memory stores the parameters at power-down. The following parameters are available in the COMM Port Control menu:

- GPIB PORT STATUS. Valid selections are **ONLINE** and **OFFLINE.** In **OFFLINE,** no communication occurs between the controller and 1240. Before changing any other parameters, the 1240 must be **OFFLINE.** When the 1240 goes **ONLINE,** it sends a service request to the controller, notifying the controller of its **ONLINE** status.
- 1240'S GPIB ADDRESS. Valid addresses are **0-30.**
- MESSAGE TERMINATION. Valid types are **EOI,** and **LF OR EOI.**

If **EOI** is selected, messages are terminated by sending EOI concurrent with the last byte of the message. During message reception, receiving an EOI is the only recognized message terminator.

If **LF OR EOI** is selected, a CR followed by LF concurrent with EOI is sent as a message terminator. When receiving, a terminator is LF or EOI.

1240 Soft Function Keys. Use the soft function keys at the bottom of the COMM Port Control Menu to request a data transfer. Each soft key, when pressed, sends a unique message, a service request (SRQ), to the controller. See the Event Codes for these keys in *Appendix D.*

The soft keys are:

REQUEST SETUP UPLOAD

REQUEST SETUP DOWNLOAD

REQUEST REFMEM UPLOAD

REQUEST REFMEM DOWNLOAD

REQUEST ACQMEM UPLOAD

NOTE

These keys do not perform data transfers automatically. They merely notify the controller that the 1240 user requests an upload to the controller, or a download from the controller.

Error Messages

The following error messages may appear on the 1240 screen in COMM PORT CONTROL menu:

PORT STATUS MUST BE OFFLINE TO CHANGE THIS VALUE

PORT STATUS MUST BE ONLINE TO DO THIS OPERATION

Remote Menu. This menu can only be called by the controller. The 1240 enters this menu when the GPIB interface enters the Remote State (REMS) or the Remote With Lockout State (RWLS). The 1240 returns to the previous menu when the GPIB interface enters the Local State (LOCS) or the Local With Lockout State (LWLS). While in Remote menu, the 1240 screen display, if present, is produced by the controller.

The 1240 does not have to be in Remote menu to communicate with GPIB. Some of its device-dependent messages can be used in either Remote or Local mode. See *Device-Dependent Messages* for more information.

The 1240 STOP key causes a return to local (if the 1240 is not in Remote with Lockout state).

GPIB COMM PACK LED'S

When listening, the two LEDs reflect the 1240's assertion of the NRFD and NDAC interface signal lines. When the 1240 is a talker, the NRFD and NDAC LEDs indicate the current bus status.

THE STATUS BYTE

When the controller serial-polls the 1240 for its status, the 1240 returns an eight-bit status byte.

The status byte bits are numbered DI01 to DI08, (least significant to most significant bit). Following is an illustration of the status byte.

Table 1-2
THE STATUS BITS

Bit	Use
DI01-DI04	Used to specify system or device status, in conjunction with DI08.
DI05	Busy bit 0=ready 1=processing or executing a command
DI06	Error bit 0=normal condition 1=error condition
DI07	RQS bit. 0=not requesting service 1=requesting service
DI08	Device/System Status bit. 0=bits DI01-DI04 contain system status code 1=bits DI01-DI04 contain device status code

Table 1-3
STATUS BYTE INTERPRETATION

Status	8765 4321	Hex	Decimal
power on (online)	010X 0001	41 or 51	65 or 81
command error	011X 0001	61 or 71	97 or 113
execution error	011X 0010	62 or 72	98 or 114
no status, 1240 idle	100X 0000	80 or 90	128 or 144
performing an acquisition	100X 0001	81 or 91	129 or 145
performing an auto-run	100X 0010	82 or 92	130 or 146
performing a key operation	100X 0011	83 or 93	131 or 147
request ACQMEM upload	110X 0000	C0 or D0	192 or 208
request REFMEM upload	110X 0001	C1 or D1	193 or 209
request REFMEM download	110X 0010	C2 or D2	194 or 210
request SETUP upload	110X 0011	C3 or D3	195 or 211
request SETUP download	110X 0100	C4 or D4	196 or 212
acquisition complete	110X 0101	C5 or D5	197 or 213
auto-run complete	110X 0110	C6 or D6	198 or 214
key operation complete	110X 0111	C7 or D7	199 or 215
test complete	110X 1000	C8 or D8	200 or 216
auto-run complete, memories equal	110X 1001	C9 or D9	201 or 217
auto-run complete, memories not equal	110X 1010	CA or DA	202 or 218
input error	111X 0000	E0 or F0	224 or 240

See *Appendix D* for more information about error messages.

PROGRAMMING

This section lists the device-dependent messages the controller may use to operate the 1240 and the GPIB interface functions supported by the 1200C02.

DEVICE DEPENDENT MESSAGES

Following is a list of 1240-specific messages supported by the 1240.

Remote and Local States

Each message has a notation *(Local/Remote)* or *(Remote)*. *Local/Remote* commands are received by the 1240 when it is in either local or remote state. *Remote* commands are executed by the 1240 only when in remote state.

For the purposes of this discussion, local state means any 1240 menu except the Remote menu. The 1240 enters the Remote menu when it receives REN and MLA interface messages from the controller. When this occurs, the 1240 screen is blanked, and the Port Status Display line changes to indicate Remote menu.

In Remote menu, the 1240 keyboard is disabled, with the exception of the STOP key. The STOP key causes a return to local state if pressed -- unless the 1240 is in lockout state.

Lockout state is caused by reception of the LLO interface message from the controller. When the 1240 is in lockout state, no key, including the STOP key, has any effect.

At power-up, the 1240 is in local state.

Device-Dependent 1240 Messages

The following messages are supported by the 1240. Note that some of the characters in each message are shown in bold face. The bold face characters are the abbreviated form of the command. The abbreviated command is the minimum character set that can be entered for the message. For example, the **AC**qmem**?** message will be recognized if only **AC?** is entered.

Here is an alphabetical list of the messages:

ACqmem
ACqmem?
BEll
DAtafmt
DAtafmt?
DIAg?
DISplay
DT
DT?
ERr?
EVent?
HElp?
ID?
INIt
INSetup
KEy
KEy?
LOad
MSgdlm
MSgdlm?
RAmpack
RAmpack?
REfmem
REfmem?
RPHelp?
RQs
RQs?
SEt?
STArt
STOp
TEST

ACqmem <**data block**>, <**data block**> . . . *(Remote)*

Modify a variable number of bytes of the Compressed Memory Image. Arguments are a variable number of data blocks in ASCII Hex, or standard or IEEE 728 Binary Block. For example, this command is used to return the controller's copy of the 1240 Acquisition Memory to the 1240 temporary, compressed, memory image. See *LOAD* and *GPIB Data Transfers*.

ACqmem? *(Remote)*

Compress and send a copy of the current 1240 acquisition memory to the controller. See *GPIB Data Transfers*.

To determine the approximate number of characters in the response to the query, use the following formula:

For ASCII Hex format:

2.4 X (590 + 9 X ((number of 9-channel cards X 65) + (number of 18-channel cards X 130)))

For Binary Block format:

1.2 X (590 + 9 X ((number of 9-channel cards X 65) + (number of 18-channel cards X 130)))

BEll *(Remote)*

Ring the 1240 bell. Note that a string of these commands may not cause the bell to ring once for each command.

DAtafmt { **A**schex / **B**inblk / **I**eee728 } *(Local/Remote)*

Set upload data block format for either ASCII Hex, Binary Block (standard, or IEEE 728 format). The 1240 will accept either format, regardless of DATAFMT setting, when downloading. See *GPIB Data Transfers*. Acceptable arguments are:

Aschex — ASCII Hex format (default state)

Binblk — Standard Binary Block format

Ieee728 — The IEEE 728 Binary Block format

The IEEE 728 standard calls for a #B block preamble.

DAtafmt? *(Local/Remote)*

Send the current data block format status. Response is either

DATAFMT **A**schex -- ASCII Hex

-or-

DATAFMT **B**inblk -- Binary Block

-or-

DATAFMT **I**eee728 -- IEEE 728 Binary Block format

See *GPIB Data Transfers.*

DIAg? *(Local/Remote)*

Send diagnostic results from the 1240.

If any diagnostics have been run, the results are printed in the following format:

DIAG ERRORS NNNNN NNNNN NNNNN NNNNN . . .

or

DIAG "ERRORS NOT FOUND"

The leftmost digit of NNNNN is the test sequence number. If more than one of any type of acquisition module is installed, two 9-channel modules for example, this digit will show which module has the error. Use the other four digits to locate the troubleshooting information in the *Diagnostic Error Indexes* of the *1240 Logic Analyzer Service Manual,* to interpret your diagnostic results.

DISplay <**line #**>, <**column #**>, <**data type**>, <**data**> *(Remote)*

Display data (one line maximum) on the 1240 display. All areas of the display can be written to, except line 1. For 1241 display code, refer to *Appendix B.*

line # — 02-30

column # — 01-64

data type — **A**scii, or **C**ode (1240 display code, see *Appendix B)*

data — ASCII data must be enclosed in double quotes; or must be a #H followed by 1240 display code (ASCII characters converted to two hex digits). 1240 display code data should be sent as an ASCII Hex argument only.

Examples:

DISPLAY 5,7,ASCII, "HELLO, 1240"

DISPLAY 10,1,CODE, #H110E151518262401020400.

DT { **OF**f / **AC**q / **AU**to } *(Local/Remote)*

Sets the 1240 for reception of a GET interface message. Acceptable arguments are:

OFf — Disables the function. Nothing happens when the 1240 receives a GET message (default state).

ACq — Starts a normal data acquisition when the 1240 receives a GET message.

AUto — Starts an auto-run when the 1240 receives a GET message.

DT? *(Local/Remote)*

Send the current Device Trigger setting. The 1240 responds with DT OFF, DT ACQ, or DT AUTO.

ERr? *(Local/Remote)*

Send an error code. The 1240 responds by sending the error code corresponding to the previously read status byte or highest priority pending status, in the following format:

ERR error code

For a list of error codes, see *Appendix D.*

The 1240's response to the error query is identical to EVENT? except for the header.

EVent? *(Local/Remote)*

Send an event code. The 1240 responds by sending the event code corresponding to the previously read status byte or highest priority pending status. The format is:

EVENT event code

For a list of event codes, see *Appendix D.* For a description of the status byte, see *Section 1.*

HElp? *(Local/Remote)*

List valid command headers. The list includes all commands except those handled by routines in an installed ROM pack.

ID? *(Local/Remote)*

Send identity string. The 1240 responds with

ID TEK/1240,VXX.X,SYS:Va.b,COMM:Vd.e,ACQ:f:g:h:i

where:

VXX.X — Code and format version

a.b — System software version

d.e — COMM Pack software version

f — Slot 0 acquisition module

0 — no module installed

1 — 9-channel card

2 — 18-channel card

g — Slot 1 acquisition module

h — Slot 2 acquisition module

i — Slot 3 acquisition module

INIt *(Remote)*

Perform power-up initialization of the 1240 setup. Internal diagnostics are not run, the power-on SRQ is not sent, and the 1240 is not unlistened. See *The 1240 Operator's Manual* for default setup values. Only the 1240 setup is initialized. The communication parameters are not changed.

INSetup <**data block**>, <**data block**> . . . *(Remote)*

Modify the 1240 setup. The setup includes configuration and trigger information.

INSetup**?** *(Remote)*

Upload a copy of the current setup. The 1240 responds with the INSETUP header, followed by a series of data blocks in either ASCII Hex, or standard or IEEE 728 binary block format. The Setup includes configuration and trigger information. The 1240's response to the INSETUP? can be stored by the controller and downloaded to the 1240 later.

Number of characters sent to controller in response to this query is $\leq$2500 ASCII Hex, and $\leq$1250 Binary Block.

KEy *(Remote)*

Notify the controller of the next keystroke.

The 1240 responds by going into a loop waiting for a keystroke. A DCL, GTL, or unassertion of REN interface message, or the 1240 STOP key, terminates the KEY operation. All other remote-only commands will not be executed while the 1240 waits for a key. The STOP command from the interface will terminate the KEY operation. If the 1240 is not in remote with lockout state, the front panel STOP button will also terminate the KEY operation.

When a key is pressed, an SRQ goes to the controller if RQS is on (if RQS is off, the status change is saved when a keystroke occurs).

KEy? *(Remote)*

Send the keycode for the key pressed during the previous KEY operation. Valid keycodes are listed in *Appendix C*. The keycode is initialized to 99 (invalid keycode) when remote state is entered, when INIT is executed, and when the KEY command is sent to the 1240.

LOad { **AC**qmem / **RE**fmem } *(Remote)*

Expand and load temporary compressed memory into Acquisition or Reference Memory.

Following are the valid arguments:

ACqmem — Expands and loads temporary compressed memory into Acquisition Memory.

REfmem — Expands and loads temporary compressed memory into Reference Memory.

See *ACQMEM and REFMEM.* Also see *GPIB Data Transfer.*

MSgdlm { **L**f / **S**emicolon } *(Local/Remote)*

This command sets the message unit separator. The user may select

Lf

-or-

Semicolon (default)

The IEEE-728 standard calls for Lf message unit separators along with EOI message termination.

MSgdlm? *(Local/Remote)*

Request from the controller for current MSGDLM mode.

Answers are:

MSGDLM LF

-or-

MSGDLM SEMICOLON.

RAmpack <**data block**>, <**data block**> *(Remote)*

Modify the RAM Pack contents. Arguments may be in ASCII Hex, or standard or IEEE 728 Binary block. See *GPIB Data Transfers*.

RAmpack? *(Remote)*

Send the entire contents of the RAM Pack to the controller. If no RAM Pack is installed, an error message is returned.

Number of characters sent to controller in response to this query is ≤2.5 times the number of bytes in the pack, ASCII Hex format. For Binary Block format, compute the ASCII Hex format count and divide by two.

REfmem <**data block**>, <**data block**> . . . *(Remote)*

Modify a variable number of bytes of the Compressed Memory Image. Arguments are a variable number of data blocks in ASCII Hex, or standard or IEEE 728 Binary Block. Example: return the controller's copy of the 1240 Reference Memory to the 1240 temporary, compressed, memory image. See *LOAD and GPIB Data Transfers*.

REfmem? *(Remote)*

Compress and send a copy of the current Reference Memory to the controller.

To determine the approximate number of characters in the response to the query, use the following formula:

For ASCII Hex format:

2.4 X (590 + 9 X ((number of 9-channel cards X 65) + (number of 18-channel cards X 130)))

For Binary Block format:

1.2 X (590 + 9 X ((number of 9-channel cards X 65) + (number of 18-channel cards X 130)))

RPHelp? *(Local/Remote)*

List valid command headers in an installed ROM pack.

RQs { **ON** / **OFf** } *(Local/Remote)*

Disable or enable the 1240's ability to generate SRQs. Default state is **ON.**

If RQS is **OFf**, the 1240 stores up all SRQs until the controller sends the **RQS ON** instruction. Then all SRQs are sent to the controller.

While RQS is **OFf**, the controller can poll the 1240 for waiting SRQs with a serial poll.

RQs? *(Local/Remote)*

Send the current 1240 RQS state. The 1240's response is either:

RQS OFF
-or-
RQS ON

SEt? *(Remote)*

Send current settings of RQS, DT, DATAFMT and INSETUP.

Response is:
DAT<data format selection>;

RQS<rqs setting>;

DT<dt setting>;

INS<data block>, <data block>, . . .

STArt { **ACq** / **AUto** } *(Remote)*

Begin data acquisition.

If **RQS** is **ON**, the 1240 will send an SRQ to the controller when the acquisition is complete, or when the Auto-run stop condition has been encountered.

If **RQS** is **OFf**, the 1240 will not send SRQ's, so the **EVent** query should be used to poll for SRQ's.

Valid arguments:
ACq — Normal data acquisition.

AUto — 1240 performs Auto-run.

STOp *(Remote)*

Halt data acquisition, KEY operation, or Auto-run. Pressing the 1240 STOP key has the same effect (if the 1240 is in local state).

TEST *(Remote)*

Execute power-up diagnostics. When the diagnostics are completed, the 1240 is initialized. The 1240 notifies the controller that the diagnostics are completed with the Test Complete SRQ. **TEST** will not be executed if **RQS** is **OFF**.

After receiving **TEST**, but before sending the SRQ, the 1240 ignores all bus activity.

GPIB Data Transfers

Uploads to the controller and downloads to the 1240 are composed of a series of data blocks in a choice of three formats: ASCII Hex, standard binary block, or IEEE 728 Binary Block (see *DATAFMT* message). Each data block is a self-contained message with address, byte count, data, and checksum information. When the 1240 receives or transmits a data block, the location field (aloc or bloc) identifies the source or destination or the data.

Data Block Format. — When the 1240 uploads data to the controller, the response is the query header followed by a series of data blocks separated by commas, like this:

INSETUP

ACQMEM

<data block>(,<data block>,. . .); (EOI)

REFMEM

RAMPACK

<data block> = #H<abc><aloc><ascii data><acs> (ASCII Hex)

-or-

%<bbc><bloc><binary data><bcs> (Binary Block)

-or-

#B<bbc><bloc><binary data><ics> (IEEE 728 Binary Block)

<**abc**> — Two ASCII Hex digits representing the number of characters in the <**aloc**>, <**ascii data**>, and <**acs**> fields, divided by two. The maximum number is 61 *hex*

<**aloc**> — Six ASCII Hex digits representing the starting relative location of <ascii data> within the 1240 port-addressable memory.

<**ascii data**> — This field contains the data itself. Each byte of the data is represented as two ASCII Hex digits.

<**acs**> — Two ASCII Hex digits representing the two's complement of the modulo 256 sum of the hex digit pairs in the <**abc**>, <**aloc**>, and <**ascii data**> fields.

<**bbc**> — A 16-bit binary value specifying the number of bytes in the <**bloc**>, <**binary data**>, and <**bcs**> fields, most significant byte first. The maximum permissible value for <bbc> is 61 *hex*.

<**bloc**> — A 24-bit binary value specifying the starting relative location of <binary data> within the 1240 port-addressable memory.

<**binary data**> — This field contains the binary data.

<bcs> — An 8-bit value containing the two's complement of the modulo 256 sum of the bytes in <bbc>, <bloc>, and <binary data> fields.

<ics> — An 8-bit value containing the two's complement of the modulo 256 sum of the bytes in bloc and binary data fields.

Data downloads to the 1240 (i.e. ACQMEM, REFMEM, RAMPACK, and INSETUP commands) must also conform to the format described above.

Data Transfer, Acqmem and Refmem/Controller. Here is a diagram of Acquisition Memory memory uploading to the controller and downloading to the 1240. The Reference Memory is uploaded and downloaded the same way, only instead of using ACQMEM? and ACQMEM, use REFMEM? and REFMEM.

- Uploading — Controller sends the following message:

```
REN
MLA
ACQMEM? (EOI)
```

The 1240 compresses its acquisition memory and sends it to the controller. See Figure 2-1.

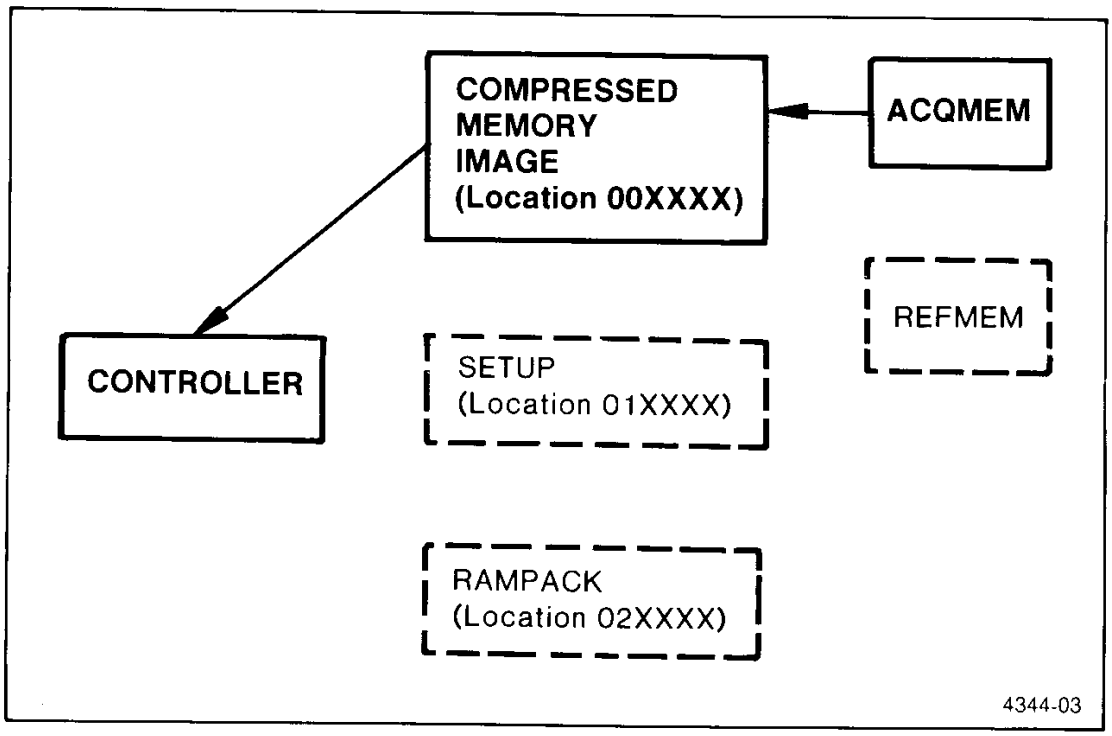


Figure 2-1. Controller uploads Acquisition Memory contents with ACQMEM? command.

The controller receives the data in the format:

MTA
ACQMEM <data block>,
<data block>,
<data block>,. . .
<data block>
EOI

- Downloading — The controller sends the following message:

  REN
  MLA
  ACQMEM
  <data block>,
  <data block>,
  <data block>,. . .
  <data block>
  EOI

The data blocks are sent to the 1240 Temporary Compressed Memory Image. See Figure 2-2.

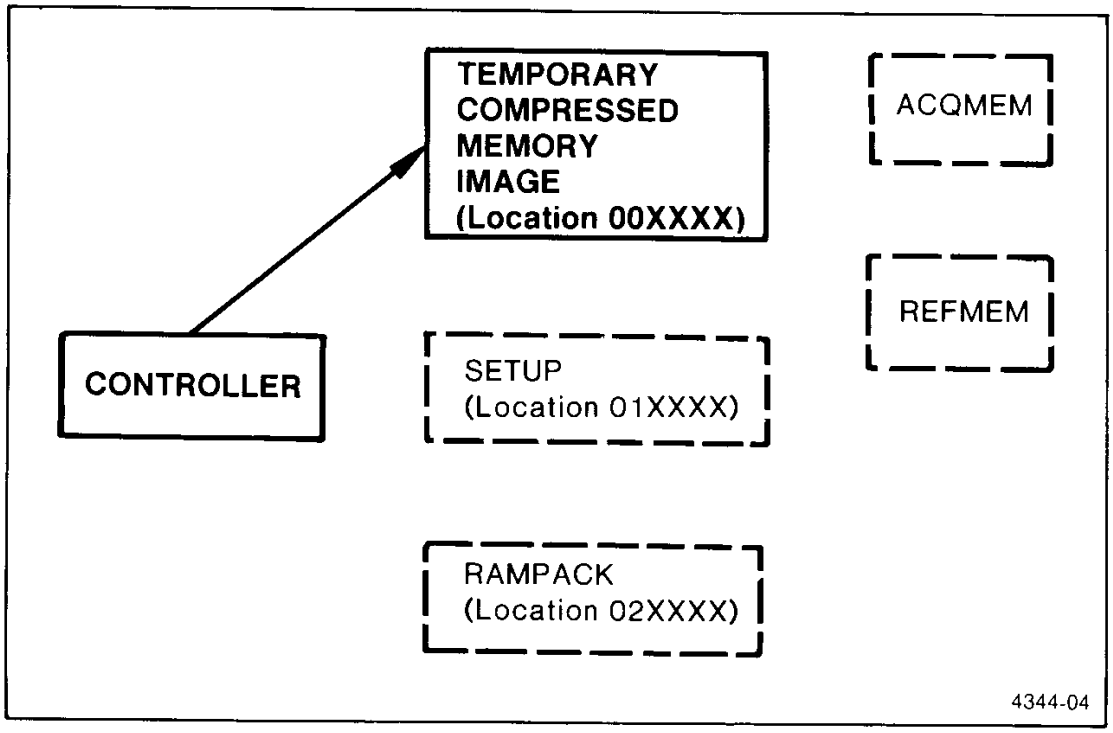


Figure 2-2. Controller downloads data to 1240 compressed memory with ACQMEM command.

The controller now sends the following message:

REN

MLA

LOAD ACQMEM (EOI)

The Compressed Memory Image is loaded in the Acquisition Memory. See Figure 2-3.

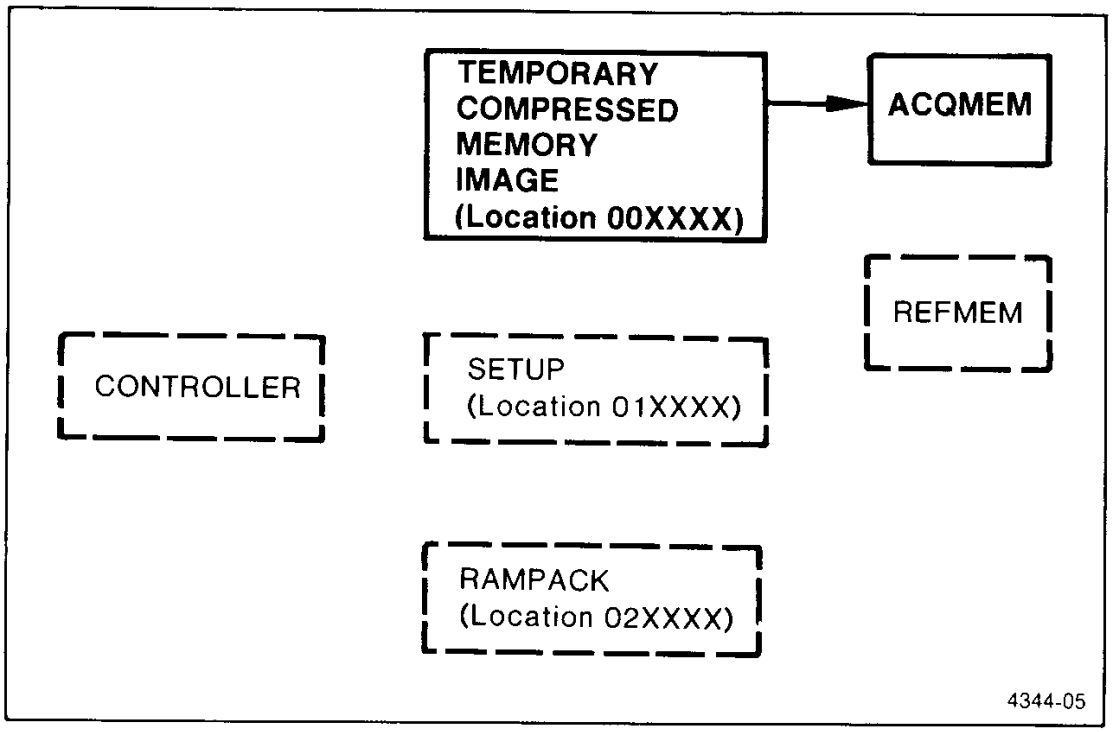


Figure 2-3. Controller modifies 1240 Acquisition Memory with LOAD command.

Data Transfer, INSETUP and RAMPACK. Here is a diagram of SETUP uploading and downloading. The RAMPACK is uploaded and downloaded the same way, only instead of using INSETUP? and INSETUP, use RAMPACK? and RAMPACK.

- Uploading — The controller sends the following message:

  REN
  MLA
  INSETUP? (EOI)
  MTA

The 1240 sends the contents of the SETUP memory in the following format:

INSETUP
<data block>,
<data block>,. . .
<data block>EOI

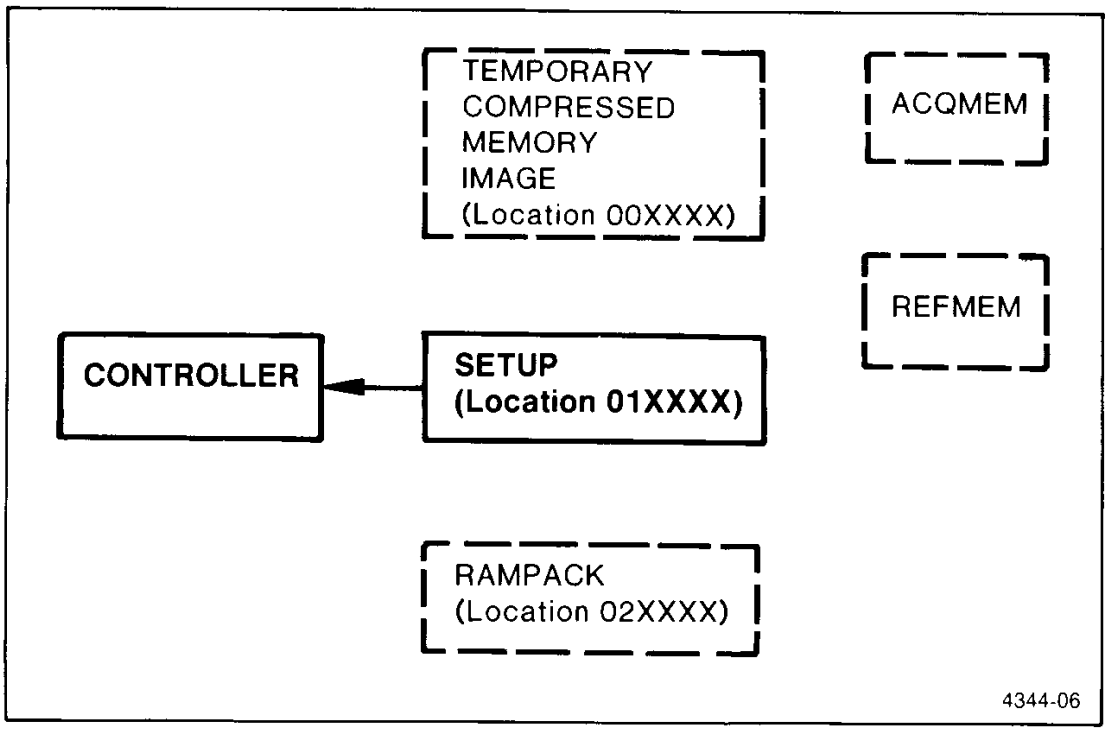


Figure 2-4. Controller uploading setup information with INSETUP? command.

- Downloading — The controller loads the SETUP memory with the following message:

  REN
  MLA
  INSETUP
  <data block>,
  <data block>,. . .
  <data block>EOI

The data blocks are then loaded into the SETUP memory. See Figure 2-5.

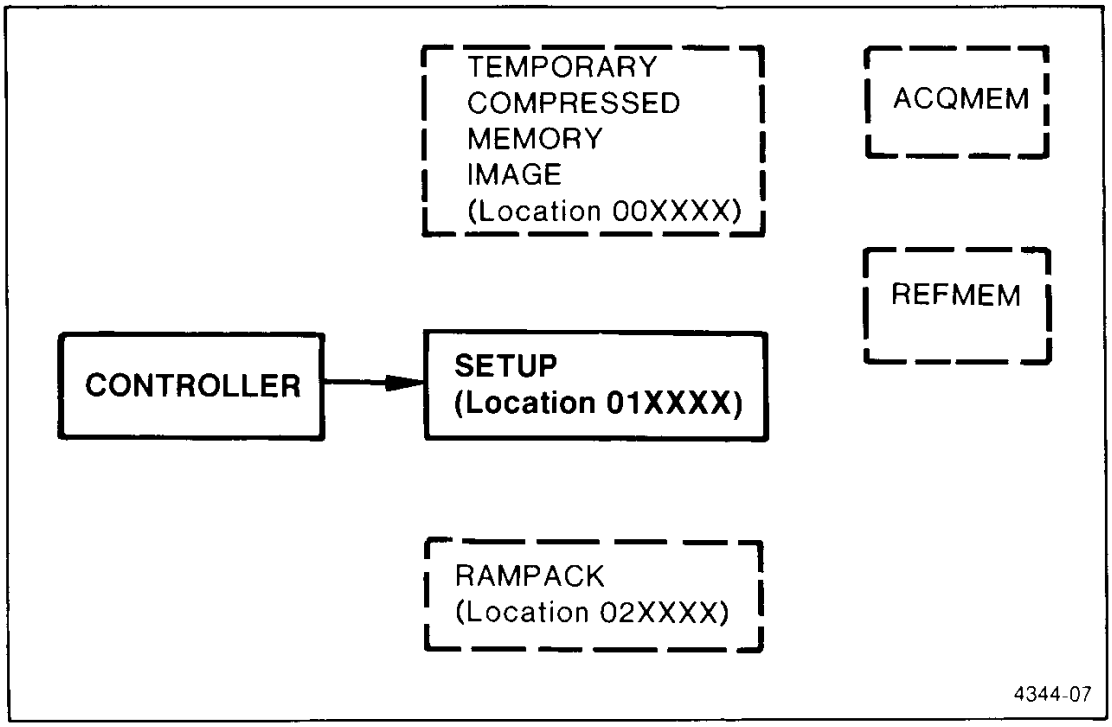


Figure 2-5. Controller downloading the Setup Memory with the INSETUP command.

Interrupting and Aborting Data Transfer in Progress

An IFC interface message received by the 1240 during execution of INSETUP?, ACQMEM?, RAMPACK?, or REFMEM? temporarily interrupts execution. Execution continues when the 1240 is talk addressed.

A DCL or SDC command received by the 1240 will terminate (INSETUP?) execution (upload).

If the user presses the STOP key on the 1240 front panel during INSETUP? execution (upload), the 1240 returns to local state, and sends a warning message to the controller. To prevent this, enter LLO (local lockout) state.

Aborting a data transfer in progress, upload or download, can be done in two ways:

1. If the 1240 is not in an LLO state, press the STOP key on the front panel. This causes a return to local state. The 1240 terminates the transfer in progress by sending an SRQ notifying the controller that data may have been lost by a change to local state. If the 1240 is in an LLO state, the 1240 keyboard is locked out and the 1240 STOP key won't operate.
2. The controller can send DCL (or SDC), or GTL interface messages. Either of these will terminate the transfer.

During uploading or downloading, if a transmission error occurs, or if the transmission is aborted because of a return to local state, you should begin transmission again from the beginning. Any data bytes received after the SRQ should be ignored.

1240 GPIB INTERFACE

Table 2-1 lists the GPIB interface function subsets supported by the 1240:

Table 2-1
GPIB INTERFACE FUNCTIONS SUPPORTED BY THE 1240

IEEE Symbol	Capability
SH1	Full Source Handshake capability.
AH1	Full Acceptor Handshake capability.
T6	Basic Talker, serial poll capability, unaddress if MLA. No Talk Only capability.
L4	Basic Listener, unaddress if MTA, No Listen Only capability.
SR1	Full Service Request capability.
RL1	Full Remote Local capability.
PPO	No Parallel Poll capability.
DC1	Full Device Clear capability.
DT1	Full Device Trigger capability.
C0	No Controller capability.

The 1240 is a tri-state, or E2, device.

NOTE

The above mnemonics are used in the IEEE Standard and identify both the interface function and the subset implemented. For example, T6 means Talker function, subset 6.

Interface Control Messages

Interface control messages are sent with the ATN line asserted, and are of two types: Uniline messages sent on the GPIB signal lines (see *Data, Management, and "Handshake" Buses in Appendix E)*, and Multiline messages sent over the data bus and signal lines.

Interface control messages include the primary talk and listen addresses for instruments on the bus, addressed commands (for previously-addressed instruments), universal commands (for all instruments), and secondary addresses.

The Interface control messages are divided into two categories: messages the 1240 is capable of receiving (controller to 1240), and messages that the 1240 is capable of sending (1240 to controller).

Table 2-2
CONTROLLER TO 1240

Message	Description
ATN	(Attention) tells the 1240 that the accompanying multiline message should be interpreted as an Interface message.
DAB	(Data Byte) is a data byte received by the 1240 from the bus. The data byte will become part of a device dependent message.
DAC	(Data Accepted) is the message received when the NDAC signal line is unasserted. It is used to indicate the condition of acceptance of data by the controller or a device. See *DAV* and *RFD*.
DAV	(Data Valid) is the message received when the DAV signal line is asserted. It is used to indicate to the 1240 that the data on the DI0 signal lines is valid. See *DAC* and *RFD*.
DCL	(Device Clear) is sent to the 1240 by the controller to restart the communication process. None of the 1240's settings are changed, but the 1240 will halt execution of any previously received command, clear the pipeline of input and output buffers, clear SRQ, the status byte, and any pending status.
END	(End) is the interface message sent to the 1240 with the last data byte to indicate End Of Message.
GET	(Group Execute Trigger) is an addressed command. It is used in association with the Device Trigger (DT) commands to start a data acquisition or Auto-run. If the DT is turned off, GET has no effect.
GTL	(Go To Local) causes the 1240 to go to local state from remote if it is listen-addressed.
IFC	(Interface Clear) resets the talk and listen interface functions. Reception of this message does not terminate any device operation.
LLO	(Local Lockout) causes the remote state 1240 keyboard to become inoperative (except for the KEY command). To exit the locked out state, the REN line must be unasserted.
MLA	(My Listening Address) causes the 1240 to listen.
MTA	(My Talk Address) causes the 1240 to talk. If the 1240 has no data to transmit, it sends the "talked with nothing to say" message (FF *hex*).
REN	(Remote Enable) is the message sent to the 1240 when the REN line is asserted. This message, plus the MLA message, causes the 1240 to enter remote state.
RFD	(Ready for Data) is the message the 1240 receives when the NRFD line is unasserted. See *DAV* and *DAC*.
SDC	(Selected Device Clear) has the same function as the DCL, except that it is acted upon only by listen-addressed devices.

Table 2-2 (cont.)
CONTROLLER TO 1240

Message	Description
SPD	(Serial Poll Disable) sets the 1240 for data output rather than for serial poll status bytes when it is talker addressed (power-on default).
SPE	(Serial Poll Enable) sets the 1240 to output serial poll status bytes when it is talker addressed.
UNL	(Unlisten) causes the 1240 to stop listening for data. No data is lost. The MLA message causes the 1240 to start listening again.
UNT	(Untalk) causes transmission from the 1240 to be interrupted. No data is lost. The MTA message will restart the transmission from where it left off.

Table 2-3
1240 TO CONTROLLER

Message	Description
DAB	(Data Byte) 1240 indicates a byte of data sent to the controller.
DAC	(Data Accepted) indicates to the controller that the 1240 has accepted the data.
DAV	(Data Valid) indicates to the controller that the data on the DI0 signal lines is valid.
END	(End) indicates end of message when received with a DAB.
RFD	(Ready for Data) indicates to the controller that the 1240 is ready for data.
RQS	(Request Service) is the message sent when DI07 of the status byte is set, and indicates that the 1240 is requesting service from the controller.
SRQ	(Service Request) is sent via the SRQ interface line to signal the controller. The controller then polls each device on the bus. The device requesting service sends the RQS message bit by setting DI07 in its status byte.
STB	(Status Byte) is sent to the controller during a serial poll operation. It contains summary status information about the 1240.

PROGRAMMING EXAMPLES

The program segments shown in this section are designed to illustrate the GPIB capabilities of the 1240 Logic Analyzer. They are written in generic language. To execute one of these program segments on your controller, translate each statement into your controller's programming language.

Strings written in capital letters and enclosed in double quotes, "INSETUP?" and "START ACQ" for example, are the actual strings that should be sent to the 1240. X$ denotes a string variable. Check your controller's string capability before entering strings.

RECEIVING A SETUP FROM THE 1240

The following program segment extracts the current 1240 setup and stores it on a mass storage device.

```
100  put 1240 in Remote menu
110  output to 1240 "INSETUP?"
120  input from 1240 S$
130  put 1240 back in Local (if desired)
140  output to mass storage S$ (if desired)
```

Line 100 causes the 1240 to change to the Remote menu. This is necessary because the following INSETUP? query is a Remote Only command. An error will occur if this command is sent to the 1240 when it is not in Remote menu. The INSETUP? query causes the 1240 to send a copy of its current setup to the controller. Line 110 sends the query to the 1240, and line 120 receives the 1240's response to the query and places it in string variable S$.

Line 130 gives control of the 1240 back to the 1240 user. This step is not necessary and would, in fact, be undesirable if further communication between the controller and the 1240 were to take place soon. If the controller has some form of mass storage at its disposal it may want to store the 1240 setup just received (line 140).

SENDING A SETUP TO THE 1240

The following program segment sends a previously stored setup to the 1240.

```
100  input from mass storage S$ (the setup)
110  put 1240 in Remote menu
120  output to 1240 S$
130  put 1240 back in Local (if desired)
```

Line 100 loads a string variable (S$ in this case) with a 1240 setup stored on mass storage. The setup retrieved is probably a copy of the 1240's response to an INSETUP? query sent earlier. The 1240 must be in Remote menu for it to accept a setup from the controller (line 110). Line 120 sends the setup to the 1240 and line 130, which may or may not be necessary, gives control of the 1240 back to the local 1240 user.

GETTING THE ACQMEM FROM THE 1240

This program segment could be used to have the 1240 send the controller a copy of its current Acquisition Memory.

```
100  put 1240 in Remote menu
110  output to 1240 "ACQMEM?"
120  input from 1240 A$
130  put 1240 back in Local (if desired)
140  ouput to mass storage A$ (if desired)
```

Line 100 places the 1240 in Remote menu. This is necessary because the ACQMEM? query is a Remote Only command. Line 110 sends the ACQMEM? query to the 1240 and line 120 receives the 1240's response to the query and puts that response in a string variable (A$ in this case). If it is desirable to give control of the 1240 back to the 1240 local user, line 130 can be used to put the 1240 back in a Local menu. Line 140 stores the 1240's response to the ACQMEM? query on the mass storage device.

This same program segment could be used to request a copy of the 1240's Reference Memory, except that the string "REFMEM?" is used in line 110 instead of "ACQMEM?".

SENDING A REFMEM TO THE 1240

The following program segment sends a Reference Memory to the 1240.

```
100  input from mass storage R$
110  put 1240 in Remote menu
120  output to 1240 R$
130  output to 1240 "LOAD REFMEM"
140  put 1240 back in a Local menu (if desired)
```

Line 100 retrieves a memory image previously extracted from the 1240 and stored on the mass storage device. The 1240 must be in Remote menu when a memory image (acquisition or reference) is sent. Line 120 sends the memory image (string variable R$ in this case) to the 1240. After sending the memory image to the 1240, it is necessary to send a LOAD command to the 1240 to cause the compressed memory image to be expanded and placed in the desired memory area (Reference Memory or Acquisition Memory). Line 130 sends the LOAD REFMEM command to the 1240 to cause the Reference Memory to be loaded with the memory image sent. Line 140 gives control of the 1240 back to the local 1240 user.

PERFORMING A REMOTE DATA ACQUISITION

The following program segment sends a setup to the 1240, acquires data using that setup, and then retrieves a copy of the 1240's Acquisition Memory.

```
100  input from mass storage S$ (setup)
110  put 1240 in Remote menu
120  put 1240 in Local Lockout mode
130  output to 1240 S$
140  trigflag = FALSE
150  output to 1240 "START ACQ"
160  if trigflag = TRUE then goto 200
170  goto 160

200  output to 1240 "ACQMEM?"
210  input from 1240 A$
220  put 1240 back in Local (if desired)
230  output to mass storage A$ (if desired)
```

Line 100 gets the desired setup from mass storage and puts it in a string variable (S$ in this case). Lines 110 and 120 cause the 1240 to go to Remote menu with Lockout.

The lockout prevents the local 1240 user from pressing the STOP (return to local) key and terminating the data acquisition. The setup is sent to the 1240 in line 130. The 1240 is now set up and ready to acquire data. The START ACQ command is sent to the 1240 (line 150) to start a data acquisition.

When the acquisition is over (when the 1240 triggers and fills its memory), the 1240 generates an SRQ. Normally, an SRQ causes an interrupt at the controller. In this case, it is assumed that the SRQ interrupt handler will set "trigflag" to TRUE when the "1240 data acquisition over SRQ" is detected. The loop in lines 160 and 170 causes the controller to wait for the occurrence of this "acquisition over SRQ" from the 1240.

When the acquisition is over, execution begins at line 200. Line 200 sends the ACQMEM? query to the 1240 and line 210 gets the response to the query.

NOTE

There are several things the controller can do instead of simply waiting in a loop for the end of the acquisition. See the START *command in* Device-Dependent Messages, Section 2.

GETTING KEYSTROKES FROM THE 1240 FRONT PANEL

The following program segment can be used to allow the controller to be notified of the next 1240 front panel keystroke.

```
100  put 1240 in Remote menu
110  keyflag = FALSE
120  output to 1240 "KEY"
130  if keyflag = TRUE then goto 200
140  goto 130

200  output to 1240 "KEY?"
210  input from 1240 K$
220  put 1240 in Local menu (if desired)
```

Line 100 puts the 1240 in Remote menu. This is necessary because the KEY command and the KEY? query are Remote Only commands. Line 120 sends the KEY command to the 1240 which causes the 1240 to enable its keyboard and wait for the local 1240 user to press a key. When the local 1240 user presses a key, the 1240 asserts SRQ which causes an interrupt at the controller. The controller's SRQ interrupt handler routine (not shown) determines the source of the SRQ (which device on the bus) and the reason for it. If the source is the 1240 executing a KEY command, the interrupt handler should set "keyflag" to TRUE. Control remains in the loop in lines 130 and 140 until this "end of key operation" SRQ occurs causing "keyflag" to be set to TRUE.

After the keystroke has occurred, the KEY? query is sent to the 1240 (line 200) to determine which key was pressed. Line 210 places the response to this query in a string variable (K$ in this case).

What the controller does with this key information is not shown, but a likely application would be for the controller to examine K$ and, based on the value of the key, decide what operation to perform next. See *Appendix C* for a list of the keycodes returned by the KEY? query.

WRITING TO THE 1240 DISPLAY

The following program segment puts the 1240 in Remote menu and then writes a text string to the 1240 display.

```
100   put 1240 in Remote menu
110   output to 1240 "DISPLAY 5,20,ASCII,
      "THIS IS THE REMOTE MENU""
120   output to 1240 "DISPLAY 7,20,ASCII,
      "PLEASE PRESS THE STOP KEY""
130   output to 1240 "DISPLAY 9,20,ASCII,
      "TO EXIT THIS MENU""
```

Line 100 puts the 1240 in Remote menu because the following DISPLAY command is a Remote Only command. Lines 110 through 130 write text strings to the display. Line 110 displays "THIS IS THE REMOTE MENU" on line 5, column 20.

Other formats for the DISPLAY command are available. See the *DISPLAY* command in *Device-Dependent Messages*, *Section 2*.

HANDLING SERVICE REQUESTS (SRQs)

Most controllers can give the programmer the choice of polling for SRQs or having an SRQ cause an interrupt. This is an example of an SRQ interrupt service routine.

A routine similar to this could be used to handle SRQs in a system where an SRQ causes an interrupt.

```
100   on SRQ then call 2000
      .
      .
      .
2000  serial poll the device(s)
2010  output to console "SRQ received", device address, status byte
2020  if SRQ came from 1240 then goto 3000
      .
      .
      .
3000  output to 1240 "EVENT?"
3010  input from 1240 E$
3020  output to console E$ (event code)
3030  if event code indicates a transfer request then goto 4000
3040  if event code indicates end of acquisition then goto 5000
3050  if event code indicates 1240 keystroke then goto 6000
      .
      .
      (other special checks)
      .
      .
3500  return from interrupt
4000  set transfer request flag (transflag) to TRUE
4010  request = eventcode
4020  return from interrupt
5000  trigflag = TRUE
5010  return from interrupt
6000  keyflag = TRUE
6010  return from interrupt
```

Line 100 enables SRQ interrupts and informs the controller that when an SRQ occurs it should call the routine at line 2000 (the SRQ interrupt service routine).

When an SRQ from any device on the bus occurs, execution begins at line 2000. The first thing the service routine should do is determine which of the devices on the bus is causing the SRQ.

The serial poll performed in line 2000 finds the device causing the SRQ and the status byte returned by the device in response to the serial poll. Line 2010 displays the device and the status byte on the controller's console. If the 1240 is causing the SRQ, control is passed to line 3000 - the section of the interrupt handler which handles 1240 SRQs.

The best way to determine why the 1240 sent an SRQ to the controller is to do an EVENT? query. The status byte (returned by the serial poll) contains some information about the reason for the SRQ, but it may not be specific. The EVENT? query will extract the most detailed information available about the most recent SRQ. Line 3000 sends the EVENT? query to the 1240 and line 3010 puts the response in a string variable (E$ in this case). The 1240's response, which is an event code, is displayed on the controller's console in line 3020.

In most cases control can be returned to the main program at this point. But some SRQs are special and may require some special processing. In this example, transfer requests, the end of acquisition SRQ, and the 1240 keystroke are special.

In the COMM Port Control menu, there are five soft function keys at the bottom of the display. Each of these is a transfer request capable of causing an SRQ to to be sent to the controller. So, there are five different transfer request SRQs. Line 3030 checks to see if the SRQ is a transfer request SRQ. If so, control is passed to a section of code which sets a flag telling the main program that a transfer request has occurred.

Line 3040 checks to see if the SRQ is the SRQ indicating that a remotely-started data acquisition has just completed. If so, "trigflag" is set to TRUE to notify a loop in the main routine that the acquisition is complete.

Line 3050 checks to see if the SRQ is the SRQ indicating that the 1240 front panel keystroke requested by the controller has occurred. If so, "keyflag" is set to TRUE so the main routine will know the keystroke has been found.

SUPPORTING COMM PORT CONTROL MENU'S SOFT FUNCTION KEYS

The following example is a segment of a program that responds to the five soft function keys (transfer requests) at the bottom of the COMM Port Control Menu.

```
100  on SRQ then call 2000
110  put 1240 in Local
120  transflag = FALSE
130  if transflag = TRUE then goto 200
140  goto 130

200  put 1240 in Remote menu
210  output to 1240 "DISPLAY 5,20,ASCII,""REMOTE MENU"""
220  output to 1240 "DISPLAY 7,20,ASCII,""DATA TRANSFER IN
     PROGRESS"""
230  if request = event code for "SETUP UPLOAD" then goto 500
240  if request = event code for "SETUP DOWNLOAD" then goto 600
250  if request = event code for "REFMEM UPLOAD" then goto 700
260  if request = event code for "REFMEM DOWNLOAD" then goto 800
270  if request = event code for "ACQMEM UPLOAD" then goto 900
```

```
500   output to 1240 "INSETUP?"
510   input from 1240 S$
520   output to mass storage S$
530   goto 110

600   input from mass storage S$
610   output to 1240 S$
620   goto 110

700   output to 1240 "REFMEM?"
710   input from 1240 R$
720   output to mass storage R$
730   goto 110

800   input from mass storage R$
810   output to 1240 R$
820   goto 110

900   output to 1240 "ACQMEM?"
910   input from 1240 A$
920   output to mass storage A$
930   goto 110
SRQ INTERRUPT HANDLER

2000  serial poll the device(s) on the bus
2010  if SRQ came from 1240 then goto 3000
        .
        .
        .
3000  output to 1240 "EVENT?"
3010  input from 1240 E$ (event code)
3020  if event code indicates a transfer request SRQ then goto 4000
        .
        .
        .
4000  transflag = TRUE
4010  request = event code
4020  return from interrupt
```

When a local 1240 user presses one of the five soft function keys (transfer requests) at the bottom of the COMM Port Control Menu, an SRQ is sent to the controller.

In most cases the SRQ will interrupt the controller. Line 100 enables these SRQ interrupts. Since the transfer request keys are in the COMM Port Control menu, the 1240 must be in Local for a transfer request to occur. Line 110 places the 1240 in Local.

Transflag is set to FALSE (line 120) to indicate that a transfer request has not yet occurred; and then a loop occurs (lines 130 and 140) waiting for a transfer request (transflag to become TRUE). Transflag is set to TRUE in the SRQ interrupt handler (line 4000) when one of the five possible transfer request SRQs is received from the 1240.

When a transfer request occurs, the 1240 is put into Remote menu (line 200) and a message is written to the 1240 display (lines 210,220). The variable, "Request," contains the event code received from the 1240. This event code indicates which of the five transfers has been requested.

See *Appendix D* for event codes.

Lines 500 through 930 handle the requests. After a request is handled, control returns to line 110 which puts the 1240 back in Local and then waits for the next transfer request.

PERFORMING A REMOTE AUTO-RUN

The following program segment sends a setup and a Reference Memory to the 1240 and then starts a 1240 auto-run. When the auto-run terminates, a copy of the 1240's Acquisition Memory is uploaded and saved on mass storage.

```
100  input from mass storage S$ (setup)
110  put 1240 in Remote menu
120  put 1240 in Local Lockout mode
130  output to 1240 S$
140  input from mass storage R$ (REFMEM)
150  output to 1240 R$
160  arunflag = FALSE
170  output to 1240 "START AUTO"
180  if arunflag = TRUE then goto 200
190  goto 180
200  output to 1240 "ACQMEM?"
210  input from 1240 A$
220  put 1240 back in Local (if desired)
230  output to mass storage A$ (if desired)
```

Line 100 gets the desired setup from mass storage and puts it in a string variable (S$ in this case). Lines 110 and 120 cause the 1240 to go to Remote menu with Lockout. The lockout prevents the local 1240 user from pressing the STOP (return to local) key and terminating the data acquisition. The setup is sent to the 1240 in line 130. An auto-run typically does repeated data acquisitions, comparing the Acquisition Memory with the Reference Memory each time an acquisition is complete. The appropriate Reference Memory is retrieved from mass storage and sent to the 1240 (lines 140 and 150). The 1240 is now set up and ready to do an auto-run. The command is sent to the 1240 (line 170) to start the auto-run.

The auto-run is over when the conditions specified in the Auto-run Spec Menu (part of the setup) have been satisfied. When the auto-run is over the 1240 causes an SRQ. Normally, an SRQ causes an interrupt at the controller. In this case, it is assumed that the SRQ interrupt handler (not shown) will set "arunflag" to TRUE when the "1240 auto-run over SRQ" is detected. The loop in lines 180 and 190 waits for the occurrence of this "auto-run over SRQ" from the 1240.

When the auto-run is over, execution begins at line 200. Line 200 sends the ACQMEM? query to the 1240 and line 210 gets the response to the query.

NOTE

There are several things the controller can do instead of simply waiting in a loop for the end of the auto-run. See the discussion of the START *command in* Device-Dependent Messages, Section 2.

GETTING A COPY OF A 1240 RAM PACK

The following program segment extracts a copy of a 1240's currently installed RAM Pack.

```
100   put 1240 in Remote menu
110   output to 1240 "RAMPACK?"
120   input from 1240 P$
130   put 1240 back in Local (if desired)
140   output to mass storage P$ (if desired)
```

The RAMPACK? query is used to get a copy of the 1240 RAM Pack. This query is a Remote Only command, so the 1240 must be put into Remote menu before sending the query (line 100). The query is sent to the 1240 (line 110) and the response is placed in string variable P$ (line 120).

USER GENERATED RAM AND ROM PACKS

CONTENTS

These instructions contain directions for loading a 32K ROM pack with the contents of up to four full 8K RAM packs. (More than four RAM packs may fit in one ROM pack if they are less than full.) This allows for indelible storage of reference memories and setups. For example, this might be a very useful way for a central service organization or a manufacturing engineering group to provide 1240 reference information to their technicians in the field or on the floor.

REQUIRED EQUIPMENT

The procedure outlined in these instructions requires a 1240 Logic Analyzer, a RAM pack, a COMM pack, a host computer, an EPROM burner, and either an empty ROM pack (12RS11) or a blank ROM pack (12RS12). If you purchased an empty pack, you will need four Motorola 68766s or 68764s or their equivalents.

SERVICE INFORMATION

Service information for the 020-0905-00 and 020-0905-01 ROM packs is contained in the *1240 Logic Analyzer Service Manual.*

THEORY

Format

The same basic format is used by both ROM and RAM packs. Each contains a header, a directory, a number of files, and a trailer. Except for the checksum in the pack trailer, all 16-bit quantities in the header and directory must be stored low-order byte first. The size of the directory and the maximum number of files may vary. Refer to Figure 4-1.

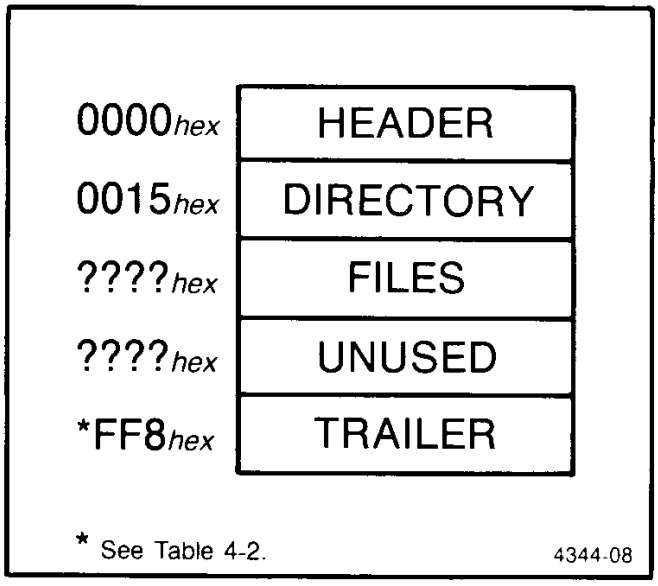


Figure 4-1. Basic ROM and RAM Pack format.

Pack Header

The first 21 bytes of the pack are reserved for the header. The header consists of:

Byte 0: "pack ID" code (01). In a user-generated pack, the "pack ID" byte must always be 01.

Bytes 1-2: length of directory (in bytes). The directory length is variable and indicates the amount of storage allocated to the directory; it does not all have to be used. This entry is stored low-order byte first. For example, a length of 40*decimal* bytes should be stored as 28,00*hex*.

Bytes 3-4: pack trailer address. The pack trailer address (the eighth byte from the end of the pack) allows the 1240 to find a ROM trailer in the pack. The address of this trailer is placed in bytes 3 and 4, low-order byte first. For a 32K pack, this will be F8,7F*hex*.

Bytes 5-20: all zeroes.

Directory Format

The directory immediately follows the pack header. A directory contains a variable number of entries. There must be a directory entry for each file in the pack. The last entry in the directory must always be the UNUSED entry. It acts as the directory terminator. Only one UNUSED entry may appear in the directory.

There are five types of file entries, corresponding to the four file types and the UNUSED entry. Refer to Table 4-1, Directory Entry Format and the *Files* section, both following.

NOTE

All 16-bit values should be stored low-order byte first.

Table 4-1
DIRECTORY ENTRY FORMAT

Byte (decimal)	Description
INSTRUMENT SETUP:	
0	type code (01)
1	length of directory entry (12)
2-3	address of file
4-5	length of file in bytes
6-11	file name (1240 display codes)
MEMORY IMAGE:	
0	type code (02)
1	length of directory entry (12)
2-3	address of file
4-5	length of file in bytes
6-11	file name (1240 display codes)
RADIX TABLE:	
0	type code (03)
1	length of directory entry (6)
2-3	address of file
4-5	length of file in bytes
RESERVED:	
0	type code (16-63)
1	length of directory entry (18)
2-3	address of file
4-5	length of file in bytes
6-11	file name (1240 display codes)
12-17	type label (1240 display codes)
UNUSED:	
0	type code (00)
1	length of directory entry (6)
2-3	address of first byte of unused space
4-5	length of unused area in bytes

Files

Files are placed immediately following the directory. They need not appear in the same order as they appear in the directory or be contiguous, but they must not overlap. The following four types of files may appear in 1200 Series ROM and RAM Packs:

- INSTRUMENT SETUP - 1240 setup
- MEMORY IMAGES - 1240 memory
- RADIX TABLE - Radix table
- RESERVED - Special-purpose internally-generated files

INSTRUMENT SETUP and RADIX TABLE files have fixed (but different) lengths. Only one RADIX TABLE file may appear in a pack. Any RADIX TABLE files after the first one are ignored.

MEMORY IMAGE and RESERVED files may vary in length. RESERVED files are files built and used by some Tektronix-generated ROM packs.

The internal formats of INSTRUMENT SETUP, MEMORY IMAGE, and RADIX TABLE files are described in *Appendix A*.

Trailer

The last eight bytes of a user-generated pack are reserved for the ROM trailer. In a 32K pack these are the bytes from 7FF8*hex* to 7FFF*hex*. The trailer address in a user-generated pack depends on the capacity of the ROM pack.

Table 4-2
ROM PACK TRAILER ADDRESSES

ROM Pack	Trailer Address
8K	1FF8
16K	3FF8
24K	5FF8
32K	7FF8

Trailer Format

In all ROM packs, the least significant bytes of the trailer addresses are F8-FF. The following describes the Trailer format.

F8 — The most significant digit in the Trailer address prefixed by a 1. See *Table 4-2*. For example, 17 for a 32K ROM Pack.

F9 — The first and second most significant digits in the Trailer address. See *Table 4-2*. For example, 7F for a 32K ROM Pack.

FA — 00

FB — 00

FC — 00

FD — FF

FE — High order byte of the checksum.

FF — Low order byte of the checksum.

CREATING A ROM PACK

You may place the contents of up to four RAM packs in one ROM pack. (Even more, if they are not full.) Figure 4-2 shows how the contents of the four RAM packs are reorganized for storage in one ROM pack.

When you put more than one RAM pack into a ROM pack, you must modify the pack header, combine the directories and the files, and compute a new checksum for the trailer.

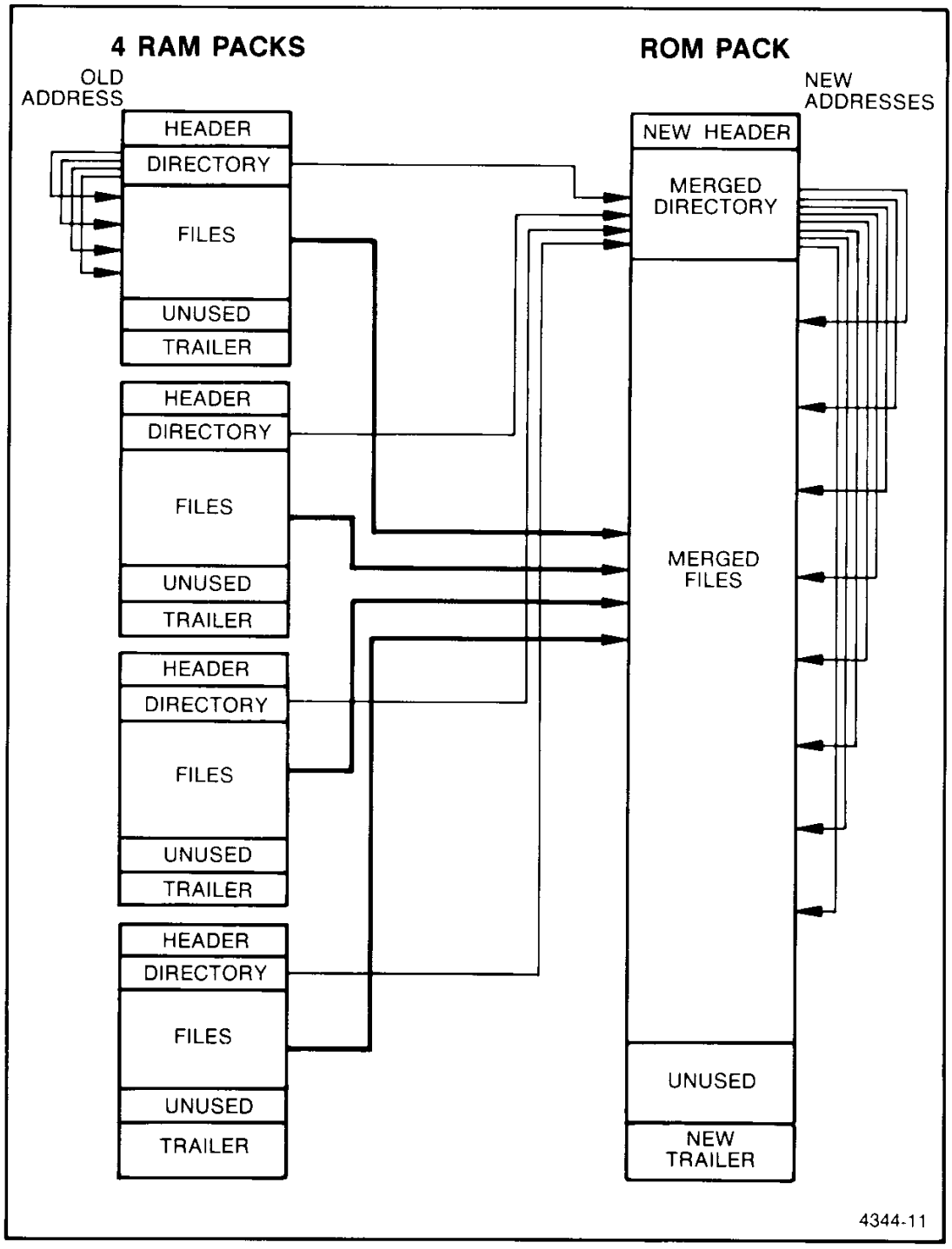


Figure 4-2. How up to four RAM Packs are reorganized for storage in one ROM Pack.

Upload

Upload the contents of the RAM Packs that you wish to put into the ROM pack into the host computer. See the *RAMPACK?* message in *Section 2*.

Header

Place 01 in the "Pack ID" byte of the first header and delete the other headers (first 21 bytes).

Directory

Count the number of files that will be in the new ROM pack. Setup and memory directory entries each require 12 bytes, radix table and unused directory entries 6 bytes, and reserved directory entries 18 bytes. Multiplying the number of each type of file by the length of the directory entry for that type of file and summing the results will determine the size of the whole directory. Enter this directory size in bytes 1 and 2 of the HEADER, low order byte first.

Place the value of the last location in the pack, minus 7, in the pack trailer address of the HEADER, low order byte first. For example, in a 32K pack this will be 7FF8. So byte 3 will contain F8, and byte 4 will contain 7F.

Files

Concatenate all of the files from the RAM packs and place them immediately following the directory.

Change Directory

Put directory entries for each file in the one directory which will be the directory for the ROM pack.

For each directory entry, update the file address. Terminate the directory by adding an UNUSED entry.

Trailer

Compute the new pack checksum. The checksum is a 16-bit value. To calculate the checksum for a user-generated pack, use the following procedure (shown in pseudo code):

- Initialize the variable CHECKSUM to 0.
- Set the variable PACKSIZE to the number of bytes in the pack (8000_{hex} for a 32K ROM Pack).
- Initialize the variable POINTER to the value PACKSIZE - 3 ($7FFD_{hex}$ for an 32K ROM pack).
- While POINTER$\geq$0

  Clear the carry flag.

  Rotate CHECKSUM left through the carry (16-bit rotate; carry = msb, lsb = 0).

  Move the byte pointed to by POINTER to the low end of TEMP and set the high-order byte to all 0s.

  Add-with-carry TEMP to CHECKSUM.

  POINTER = POINTER - 1

- Store CHECKSUM in the last two bytes of the pack (high-order byte followed by low-order byte).

To verify your checksum routine, run it on an 8K (8192*decimal*) block of 55's*hex*. The computed checksum should be 542B*hex*.

Download & Burn

You should now be ready to download the ROM pack image and load your EPROMs. To gain access to the EPROMs, remove the four screws that hold the pack together. The total 32K of ROM contents must be loaded into four 8K EPROMs. The relationship between the locations of the particular EPROM's and the addresses associated with them are shown in Figure 4-3.

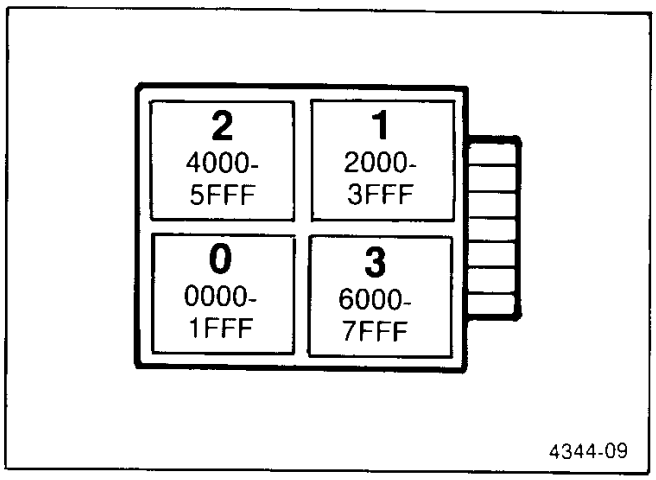


Figure 4-3. Relationship between physical location and address contents.

CAUTION

Observe static precautions to avoid damaging the EPROMs.

APPENDIX A

INSTRUMENT SETUP, MEMORY IMAGE, AND RADIX TABLE FORMATS

The following tables describe the formats of the Instrument Setups, Memory Images, and Radix Tables.

INSTRUMENT SETUP

The following is a description of the Instrument Setup.

Table A-1
SETUP MEMORY LOCATIONS

Location Within Setup	Variable Name	Menu in which Specified	Field Name in Menu
0:	trigposition	Trigger Spec	TRIGGER POSITION
1:	holdoff	Trigger Spec	LOOK FOR TRIGGER
2:	pwrcmd	Trigger Spec	GLOBAL EVENT
3:	pwrpolarity	Trigger Spec	Global Event ON/ON NOT
4:	pwrctrcmd	Trigger Spec	Global Event Ctr/Timer Action
5-6:	pwrtbfltr	Trigger Spec	Global Event Filter Timebase
7:	pwrcntfltr	Trigger Spec	Global Event Filter
8-13:	pwrctrval	Trigger Spec	IF COUNT/TIMER =
14:	seqdepth	Trigger Spec	Not an input field
15:	seqcmd	Trigger Spec	Final Sequence Action
16:	seqstore	Trigger Spec	Final Sequence Storage
17-128:	seqvalue	Trigger Spec	Sequence Step Fields
17-24	step 1		
25-32	step 2		
33-40	step 3		
41-48	step 4		
49-56	step 5		
57-64	step 6		
65-72	step 7		
73-80	step 8		
81-88	step 9		
89-96	step A		
97-104	step B		
105-112	step C		
113-120	step D		
121-128	step E		

Table A-1 (cont.)
SETUP MEMORY LOCATIONS

Location Within Setup	Variable Name	Menu in which Specified	Field Name in Menu
129-398:	trigwrval	Trigger Spec	Global and Sequence wr's
129-146	global wr		
147-164	step 1 wr		
165-182	step 2 wr		
183-200	step 3 wr		
201-218	step 4 wr		
219-236	step 5 wr		
237-254	step 6 wr		
255-272	step 7 wr		
273-290	step 8 wr		
291-308	step 9 wr		
309-326	step A wr		
327-344	step B wr		
345-362	step C wr		
363-380	step D wr		
381-398	step E wr		
399:	autocondition	Auto-run Spec	AUTO-RUN CONDITION
400:	autopulse	Auto-run Spec	EXTERAL TRIGGER OUT
401:	complimit	Auto-run Spec	COMPARISON LIMITS
402-403:	limit1	Auto-run Spec	LIMITS
404-405:	limit2	Auto-run Spec	LIMITS
406:	autotruecmd	Auto-run Spec	WHEN EQUAL
407:	autofalsecmd	Auto-run Spec	WHEN NOT EQUAL
408:	audiotrig	Auto-run Spec	AUDIBLE TRIGGER
409-426:	automask	Auto-run Spec	MASK
427:	autodelay	Auto-run Spec	DISPLAY DATA AT LEAST
428:	oplevel	Operation Level	OPERATION LEVEL
429:	tpgpat	Operation Level	TPG MODE
430-437:	memstat	Memory Config	n/a
438-441:	datasrc	Memory Config	INPUT POD fields
442:	glitches	Memory Config	GLITCHES ON/OFF
443-450:	threshold	Memory Config	CARD THRESHOLD
451-466:	memtb	Memory Config	TB fields
467-468:	w_vs_d9	Memory Config	9-CHANNEL CARDS
469-470:	w_vs_d18	Memory Config	18-CHANNEL CARDS
471-486:	polarity	Memory Config	POLARITY

Table A-1 (cont.)
SETUP MEMORY LOCATIONS

Location Within Setup	Variable Name	Menu in which Specified	Field Name in Menu
487:	tbactive	Timebase	ACTIVE TIMEBASES
488:	tb1type	Timebase	TIMEBASE 1
489:	tb1async	Timebase	Asynchronous Clock Period
490:	pwrclock	Timebase	GLOBAL EVENT =
491-498:	tb1clock	Timebase	T1 Sync Clock Definition
499-506:	tb1qual	Timebase	T1 Sync Clock Qualification
507:	tb2type	Timebase	TIMEBASE 2
508-515:	tb2clock	Timebase	T2 Clock Definition
516-523:	tb2qual	Timebase	T2 Clock Qualification
524-531:	tb2lclock	Timebase	T2-Last Clock Definition
532-539:	tb2lqual	Timebase	T2-Last Clock Qualification
540-683:	serieslist	Timing Diagram	n/a
683-707:	chansel	Timing Diagram	Timing Trace Labels
708:	curseries	Timing Diagram	PAGE
709:	cardselect	Channel Grouping	CARD TYPE
710-829:	grouplayout	Channel Grouping	NAME, INPUT and DISP
710-721	group 0		
722-733	group 1		
734-745	group 2		
746-757	group 3		
758-769	group 4		
770-781	group 5		
782-793	group 6		
794-805	group 7		
806-817	group 8		
818-829	group 9		
830-901:	channelgroup	Channel Grouping	n/a
902-921:	setupmisc		(reserved)

Variable Descriptions:

audiotrig — Corresponds to the AUDIBLE TRIGGER field in the Auto-run Spec menu. It takes the values 0 and 1, indicating whether or not an audio response to trigger recognition is required

0 = OFF (no audio response)
1 = ON (audio response)

autofalsecmd — Corresponds to the WHEN NOT EQUAL field in the Auto-run Spec menu. It takes the values 0-2, representing the auto-run trigger condition for a false comparison. The mapping is:

0 = DISPLAY AND STOP
1 = DISPLAY AND REACQUIRE
2 = DISCARD AND REACQUIRE

autocondition — Corresponds to the AUTO-RUN CONDITION field in the Auto-run Spec Menu. It takes the values 0-3, representing the auto-run test condition. The mapping is:

0 = COMPARE ACQMEM TO REFMEM
1 = CONTINUOUS TRIGGER OUT
2 = TRIGGER IN
3 = STORE AFTER TRIGGER

autodelay — Corresponds to the DISPLAY DATA AT LEAST field in the Auto-run Spec menu. It takes the values 0-99 (BCD), representing the minimum time in seconds between acquisitions during an auto-run. This variable contains a two-digit BCD value. The most significant four bits and the least significant four bits must each have a value within the range of 0 to 9. For example, to set the field to 13 seconds, the binary value of this variable should be 00010011.

automask — Corresponds to the MASK field in the Auto-run Spec menu. It is a "prlword" structure (see the description of "prlword" at the end of this appendix) indicating which channels are to be considered when doing memory comparisons during autorun. The "val" bit for a channel indicates a value of 0 or 1 unless the corresponding "mask" bit is set. For example:

mask val
0 0 = "0" (don't use the channel in comparisons)
0 1 = "1" (use the channel)
1 1 = "X" (don't use the channel)

autopulse — Corresponds to the EXTERNAL TRIGGER OUT field in the Auto-run Spec menu. It takes the values 0 and 1, representing whether the signal on the EXT TRIG OUT BNC is pulsed or latched during autorun operation.

0 = LATCHED
1 = PULSED

autotruecmd — Corresponds to the WHEN EQUAL field in the Auto-run Spec Menu. It takes the values 0-2, representing the auto-run trigger condition for a true comparison. The mapping is:

0 = DISPLAY AND STOP
1 = DISPLAY AND REACQUIRE
2 = DISCARD AND REACQUIRE

cardselect — Corresponds to the CARD TYPE field in the Channel Grouping menu. It has the following legal values:

0 = 9-channel card groups
1 = 18-channel card groups

channelgroup — This is a 72-element array used in the channel grouping scheme to link the array "grouplayout" with actual data, and to reconstruct the Channel Grouping menu display. All of the channels for a given group are assigned consecutive elements in the array, with the higher-order channels appearing earlier in the array. All channels from 9-channel cards are of a higher order than channels from 18-channel cards. Only the channels currently assigned to a group appear in this array. Unused channels have entries that appear at the end of the array and have a value of FF. For example, if there are five channels not currently assigned to a group, the last five entries in "channelgroup" will contain the value FF. Deletion/insertion of a channel from/into a group implies movement of succeeding elements up/down one postion in the array. Each element of the array contains a value which identifies the memory and channel associated with that bit in the group. The high order three bits (5-7) identify the memory number (0-7) and the low order four bits (0-3) identify the channel number (0-8).

chansel — This is an array of 12 two-byte values. Each element corresponds to a Trace Label field in Timing Diagram menu. For each element, the first byte (byte 0) is the group number and the second byte (byte 1) is the channel number within that group, counted from the LEFT (i.e., the most significant bit in a group is 0). If the group number is 0FFH then the channel is selected as "OFF". Possible values for byte 0 are 0-9 or 0FFH. Possible values for byte 1 are 0-35.

complimit — Corresponds to the COMPARISON LIMITS field in the Auto-run Spec menu. It takes the values 0 and 1, representing the auto-run comparison limit type.

0 = FIXED
1 = BETWEEN CURSORS

curseries — Corresponds to the PAGE field in the Timing Diagram menu. The legal values are 0-5.

datasrc — This is an array with one element for each of the possible acquisition cards. Each element corresponds to an INPUT POD field in the Memory Config Menu. Element N (N=0-3) indicates the data source for the odd memory module of card N.

0 = data source is the even pod
1 = data source is the odd pod

NOTE: IF CARD N IS A 9-CHANNEL CARD, datasrc[N] HAS NO MEANING.

glitches — Corresponds to the GLITCHES ON/OFF field in the Memory Config menu. It takes the values 0 and 1, representing whether glitches are to be stored.

0 = GLITCHES OFF
1 = GLITCHES ON

grouplayout — An array containing information about each of the 10 groups that can be set up in the Channel Grouping menu. It is a 10-element array of "glaelement" structures. Elements 0-4 contain information about the 9 channel groups and elements 5-9 contain information about the 18-channel groups. Each element of grouplayout has the following structure:

bytes 0-3: This corresponds to the NAME field for a group. The name is stored in 1240 display code.

byte 4: The group timebase:

0 = timebase 1
1 = timebase 2
2 = unassigned

byte 5: The number of channels assigned to the group

byte 6: The offset into "channelgroup" to the first (leftmost) channel in the group

byte 7: This corresponds to the INPUT field for a group.

0 = HEX
1 = OCT
2 = BIN

byte 8: The offset from the base horizontal display position for the word recognizer value to the first character of the group when displayed using the input radix.

NOTE

For groups with no channels assigned to them, this value must be equal to this value for the next group which has channels assigned to it.

byte 9: This byte should be set to 1 if there is a ROM or RAM Pack installed which contains a Radix Table. Otherwise, this byte should be set to 0.

byte 10: This corresponds to the DISP field for a group.

0 = HEX
1 = OCT
2 = BIN
3 = OFF
4 = ASC
5 = EBC
6 = ROM

A value of 6 is only valid if byte 9 is 1.

byte 11: The number of characters needed to display the group using the input radix.

holdoff — Corresponds to the LOOK FOR TRIGGER field in the Trigger Spec menu. It takes the values 0 and 1, representing the conditions under which a trigger search is begun.

0 = IMMEDIATELY
1 = AFTER MEMORY FULL

limit1, limit2 — These correspond to the LIMITS fields in the Auto-run Spec menu. They take values in the range 0 - 8190 (decimal), representing the limits of the auto-run comparison. A value of 4095 is the trigger position. To compare from five locations before the trigger to five locations after the trigger limit1 should be set to 4090 and limit 2 should be set to 4100. If "complimit" says that the cursors should be used for comparisons, the valued in "limit1" and "limit2" are assumed to be meaningless. The variables must be stored with the low-order byte first, followed by the high-order byte.

memstat — The values in this array do not directly correspond to fields in a menu, but information in the Memory Config menu affects these values. Entry N in this array shows the current status of memory module N. Possible values are:

0 = missing (no card, or odd module of 9-channel card)
1 = 9-channel card, unchained
2 = 18-channel card, unchained
3 = chained

memtb — Each element in this array corresponds to one of the eight possible TB fields in the Memory Config menu. Element 0 corresponds to memory module 0 and element 7 corresponds to memory module 7. Each element consists of two bytes. The valid values for byte 0 and byte 1 of each element are given in the table below (empty boxes indicate invalid values). Note that a text field is displayed instead of a select in the cases when byte 0 is 0 or 1.

BYTE 0	BYTE 1			
	0	1	2	FF
0 (T1 ONLY)	T1			memory not installed
1 (T2 ONLY, T2=SYNC)		T2		
2 (T2 ONLY, T2=SPLIT)		T2 L	T2 F	
3 (T1 & T2, T2=SYNC)	T1	T2		
4 (T1 & T2, T2=SPLIT)	T1	T2 L	T2 F	

NOTES ON BYTE 1:

The value in a chained memory module should always be current with the value of the head of the chain.

0 ALWAYS indicates T1.

1 indicates EITHER T2 (T2=SYNCH) or T2 L (T2=SPLIT).

2 ALWAYS indicates T2 F.

FF ALWAYS indicates the memory module is not installed.

oplevel — Corresponds to the OPERATION LEVEL field in the Operation Level menu. Valid values are 0-3.

polarity — Each element of this array corresponds to one of the eight possible POLARITY fields. Each element is 16 bits long but only the low-order 9 bits of each value are signficant; each corresponds to one channel in the memory module, with the low-order bit corresponding to channel 0. For each bit:

0 = negative true
1 = positive true

pwrclock — Corresponds to the GLOBAL EVENT = field in the Timebase menu. It takes the following values:

0 = pwr is CLOCKED
1 = pwr is UNCLOCKED

pwrcmd — Corresponds to the GLOBAL EVENT field in the Trigger Spec menu. It takes the values 0-5, representing the five pwr command options. The mapping is:

0 = OFF
1 = TRIGGER
2 = RESET
3 = STORE
4 = START TIMER
5 = INCR CNTR
6 = TIME WHILE

pwrcntfltr — Corresponds to the Global Event FILTER field in the Trigger Spec menu. It takes the values 0-15, representing pwr filter counts of 1-16 in the user interface.

pwrctrcmd — Corresponds to the Global Event Counter/Timer Action field in the Trigger Spec menu. It takes the values 0-2:

0 = TRIGGER
1 = RESET
2 = DO NOTHING

pwrctrval — Corresponds to the IF TIMER/COUNT = field in the Trigger Spec menu. It is an 11-digit bcd value stored in six consecutive bytes. The most significant four bits of the first byte are ignored. Each of the remaining eleven groups of 4 bits should contain one BCD digit (0-9), most significant digit first. A value of 00000000000 is not legal. The valid range for this field is 00000000001 to 99999999999.

pwrpolarity — Corresponds to the ON/ON NOT field in the Trigger Spec menu. It takes on the values 0 and 1:

0 = ON
1 = ON NOT

pwrtbfltr – Two bytes corresponding to the ON field in the Trigger Spec menu (global event filter timebase). The following values are legal:

0 — If "tbactive" is T1 ONLY, this byte should contain 0. If "tbactive" is T2 ONLY, this byte should contain 2. If "tbactive" is T1 & T2, this byte should contain 1.

1 — If byte 0 is 0:
0=IONS
1=T1

If byte 0 is 1:
0=10 NS
1=T1
2=T2

If byte 0 is 2:
0=IONS
1=T2

seqcmd – Corresponds to the Final Sequence Action field in the Trigger Spec menu. It takes the values 0-2:

0 = TRIGGER
1 = RESET
2 = DO NOTHING

seqdepth – This does not correspond to a field. It takes the values 0-14, indicating the number of sequence steps currently programmed.

seqstore – If the Final Sequence Action is DO NOTHING, the value of this variable indicates whether data storage is enabled or disabled while in the Final Sequence action.

0 = storage disabled
1 = storage enabled

seqvalue – This is an array of 14 elements; one element for each of the possible sequence steps. Element 0 corresponds to step 1 and element 13 corresponds to step E (14). Each element contains eight bytes of data and has the same format.

byte 0: This corresponds to the Sequential Event Timebase field in a sequence step.

0 = T1
1 = T2

byte 1: This corresponds to the Sequence Step Action field in a sequence step.

0 = WAIT FOR
1 = WAIT FOR NOT
2 = TRIGGER IF
3 = TRIG IF NOT
4 = RESET IF
5 = RESET IF NOT
6 = JUMP IF
7 = JUMP IF NOT
8 = DELAY

byte 2: This corresponds to the TO LEVEL field in a sequence step. It is only necessary if byte 1 for the same step is JUMP IF or JUMP IF NOT. Legal values are 1 through E (hex) representing the step to be jumped to.

byte 3: This corresponds to the NNNN CLOCKS field in a sequence step. It is not necessary if byte 1 of the same step is DELAY. Legal values are 0 through 15 which translate on the display to filter values of 1 through 16.

bytes 4-6: This corresponds to the TO OCCUR field in a sequence step. It is only necessary if byte 1 is WAIT FOR, WAIT FOR NOT, or DELAY. This is a four-digit BCD value stored in 3 bytes. The first byte should set to 0. Each of the four remaining four-bit quantities is a BCD digit (0-9), most significant digit first. 0000 is an illegal value for this field. The legal range is 0001 to 9999.

byte 7: This corresponds to the WITH STORAGE field in a sequence step. It indicates whether data storage is enabled or disabled in the sequence step.

0 = storage disabled
1 = storage enabled

serieslist – This is an array of arrays defining the 6 possible display sets in the Timing Diagram menu. Each element of "serieslist" is a 12-by-2 byte array defining one display set. The first index defines a particular trace being displayed, there are 12 per display set. The second index defines the particular channel to display: "display set[n][0]" indexes "grouplayout" (a group number) and "display set[n][1]" defines the particular channel in that group.

setupmisc -Ten bytes reserved for future use by Tektronix. Entries to this area could cause the loss of essential code. If the user must enter data in this field, it should be 0's.

tb1async – Corresponds to the Asynchronous Clock Period field in the Timebase menu. Valid only if timebase 1 is asynchronous (tb1type is ASYNC). It takes the values 0-24:

0 = 10 NS
1 = 20 NS
2 = 50 NS
3 = 100 NS
4 = 200 NS
5 = 500 NS
6 = 1 uS
7 = 2 uS
8 = 5 uS
9 = 10 uS
10 = 20 uS
11 = 50 uS
12 = 100 uS
13 = 200 uS
14 = 500 uS
15 = 1 MS
16 = 2 MS
17 = 5 MS
18 = 10 MS
19 = 20 MS
20 = 50 MS
21 = 100 MS
22 = 200 MS
23 = 500 MS
24 = 1 S

A value of 0 (10 NS) is illegal unless glitch storage is turned off ("glitches" is "glitches stored") and there are no 18-channel cards assigned to T1.

tb1clock – This is an array in which each element corresponds to one of the Timebase 1 Synchronous Clock Definition fields in the Timebase menu. Element 0 corresponds to pod 0, element 1 corresponds to pod 1, etc.

0 = falling edge
1 = rising edge
2 = don't care

NOTE: Non-existent lines should be don't-cared. VALID ONLY IF TIMEBASE 1 IS SYNCHRONOUS.

tb1qual – This is an array in which each element corresponds to one of the Timebase 1 Synchronous Clock Qualification fields in the Timebase menu. Element 0 corresponds to pod 0, etc.

0 = 0
1 = 1
2 = X

NOTE: Non-existent lines should be don't-cared.

tb1type – Corresponds to the TIMEBASE 1 field in the Timebase menu. It takes the values 0 and 1:

0 = ASYNC
1 = SYNC

tb2lclock – This is an array in which each element corresponds to one of the T2 Last Clock Definition fields in the Timebase menu. Element 0 corresponds to pod 0 and so on. The following values are valid:

0 = falling edge
1 = rising edge
2 = don't care

NOTE: Non-existent lines should be don't-cared. VALID ONLY IF TIMEBASE 2 IS SPLIT.

tb2lqual – This is an array in which each element corresponds to one of the T2 Last Clock Qualification fields in the Timebase menu. Element 0 corresponds to pod 0 and so on. The following values are valid:

0 = 0
1 = 1
2 = X

NOTE: Non-existent lines should be don't-cared. VALID ONLY IF TIMEBASE 2 IS SPLIT.

tb2clock – This is an array in which each element corresponds to one of the Timebase 2 Clock Definition fields in the Timebase menu. Element 0 corresponds to pod 0 and so on. The following values are valid:

0 = falling edge
1 = rising edge
2 = don't care

NOTE: Non-existent lines should be don't-cared. IF TIMEBASE 2 IS SPLIT, THIS IS ACTUALLY THE DEFINITION OF THE FIRST CLOCK (T2- F).

tb2qual — This is an array in which each element corresponds to one of the Timebase 2 Clock Qualification fields in the Timebase menu. Element 0 corresponds to pod 0 and so on. The following values are valid:

0 = 0
1 = 1
2 = X

NOTE: Non-existent lines should be don't-cared. IF TIMEBASE 2 IS SPLIT, THIS IS ACTUALLY THE QUALIFIER FOR THE FIRST CLOCK (T2- F).

tb2type — Corresponds to the TIMEBASE 2 field in the Timebase menu. It takes the values 0 and 1:

0 = DEMUX
1 = SYNC

tbactive — Corresponds to the ACTIVE TIMEBASES field in the Timebase menu indicating the active timebases, as follows:

0 = T1 ONLY (1-timebase mode)
1 = T2 ONLY (1-timebase mode)
2 = T1 AND T2 (2-timebase mode)

threshold — This is an array of 4 16-bit values in which each element corresponds to one of the four possible CARD THRESHOLD fields in the Memory Config menu. The range of legal values is:

0 = -6.35V
1 = -6.30V
2 = -6.25V
.
.
.
126 = -0.05V
127 = 0.00V
128 = +0.05V
.
.
.
253 = +6.30V
254 = +6.35V
255 = -ECL
256 = TPG
257 = TTL

For all elements except the first, an additional value of 258 is legal. This value means "CARD 0".

tpgpat — Corresponds to the TPG MODE field in the Operation Level menu. Valid values are 0-3.

0 = 12 MHz, no glitches
1 = 6 MHz with glitches
2 = T1, no glitches
3 = T1/2 with glitches

trigposition — Corresponds to the TRIGGER POSITION field in the Trigger Spec menu. It takes the values 0-4, representing the five different trigger positions possible in the acquisition memory.

0 = trigger at 3% point
1 = trigger at 25% point
2 = trigger at 50% point
3 = trigger at 75% point
4 = trigger at 97% point

trigwrval — This is an array of 15 "prlword" structures (see the description of "prlword" in this appendix). The value in "trigwrval[0]" is the global event word recognizer value; the value in "trigwrval[N]" is sequence word recognizer N's value. The "val" bit for a channel indicates the value of the channel (0 or 1) unless the corresponding "mask" bit is set. The value for all unused entries, or channels not in groups, should be "DON'T CARE" (X).

mask	val		
0	0	=	"0"
0	1	=	"1"
1	0	=	"G" (glitch)
1	1	=	"X" (don't care)

For sequence step wr values, only the bits belonging to the wr's timebase need be valid. The bits in the "other" timebase may contain garbage.

w__vs__d9 — Corresponds to the 9 CHANNEL CARDS field in the Memory Config menu. It represents the width and depth configuration of the installed 9-channel cards. This variable consists of two bytes. Byte 0 is THE NUMBER OF 9-CHANNEL CARDS INSTALLED. Byte 1 indicates the width and depth, dependent on byte 0 and whether or not glitches are on, as follows:

BYTE 0 w__vs__d9[0]	BYTE 1 w__vs__d9[1]	WIDTH	DEPTH gl. off	DEPTH gl. on
0	0	0	0	0
1	0	9	512	256
2	0	18	512	256
	1	9	1024	512
3	0	27	512	256
	1	9	1536	768
4	0	36	512	256
	1	18	1024	512
	2	9	2048	1024

w__vs__d18 — Corresponds to the 18 CHANNEL CARDS field in the Memory Config Menu. It represents the width and depth configuration of the installed 18-channel cards. This variable consists of two bytes. Byte 0 is THE NUMBER OF 18-CHANNEL CARDS INSTALLED. Byte 1 indicates the width and depth selection, dependent on byte 0, as follows:

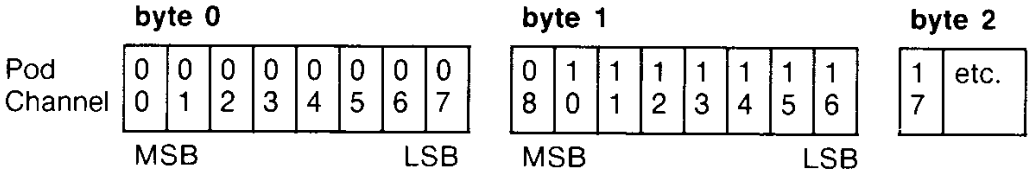
BYTE 0 w__vs__d18[0]	BYTE 1 w__vs__18[1]	WIDTH	DEPTH
0	0	0	0
1	0	18	512
2	0 1	36 18	512 1024
3	0 1	54 18	512 1536
4	0 1 2	72 36 18	512 1024 2048

Description of a 'PRLWORD'

A "prlword" is 18 bytes of data used to store word recognizer values and the Auto-run mask value. A single "prlword" consists of nine bytes (72 bits) of "val" data followed by nine bytes of "mask" data. There is one bit of "val" and one bit of "mask" for each possible 1240 input channel. The "val" and "mask" values for a channel, taken together, specify the word recognizer setting for that channel. The "val" bit for a channel indicates a value of 0 or 1 unless the corresponding "mask" bit is set, as follows:

mask			val
0	0	=	"0"
0	1	=	"1"
1	0	=	"G" (glitch)
1	1	=	"X" (don't care)

Each of the 72 channels is represented, whether or not installed and regardless of chaining. Channel grouping is not taken into account. Within the "val" and "mask" sections of the "prlword", the input channels are ordered as follows:



4344-12

MEMORY IMAGE

The following is a description of the Memory Image.

Location in Memory Image	Variable Name
0-256:	rawcor1
257-513:	rawcor2
514-529:	rawpodlen
530-545:	rawoldest
546-561:	rawyoungest
562-569:	rawempty
570-571:	rawtpi1
572-573:	rawtpi2
574:	rawtrig
575:	rawc1pod
576:	rawc2pod
577:	rawlast
578:	rawctrunits
579:	rawglitches
580:	rawtb1type
581:	rawtb1asynch
582:	rawtimevalid
583-584:	rawd9
585-586:	rawd18
587-594:	rawtb
595-599:	rawctr
600-601:	rawlength
602-603:	rawpostfig
604-613:	rawmisc
614-5293:	rawdata

Variable Descriptions

rawc1pod, rawc2pod - These variables tell which pods generated the correlation information for timebases 1 and 2, respectively. It shows which entries of "rawoldest" and "rawyoungest" define the bounds of meaningful data in "rawcor1" and "rawcor2". In other words, the youngest location in "rawcor1" is indicated by "rawyoungest[rawc1pod]" and the oldest by "rawoldest[rawc1pod]". It also indirectly shows whether the correlation data is in glitch form or non-glitch form (it is in glitch form if the pod it copies is in glitch form).

rawcor1, rawcor2 - Each of these is a 2056-element bit-array containing the correlation information for TB1 and TB2, respectively. The oldest location in rawcorQ is indicated by "rawoldest[rawcQpod]", and the youngest by "rawyoungest[rawcQpod]". The data is stored in glitch format or non-glitch format, depending on the source pod indicated by "rawcQpod". (For information about "glitch" versus "non-glitch" format, see "rawdata", in this Appendix.)

rawctr - This is five bytes containing the 37-bit value from the ctr/timer when the acquisition was performed. The first byte contains the lowest eight bits, and so on. The three unused bits (in the fifth byte) are set to 0. This item is meaningless if "rawctrunits" (see below) contains 2.

rawctrunits - This variable identifies the units of the ctr/timer value in "rawctr".

0 = events
1 = time
2 = should not be displayed

rawd9 - This is actually two one-byte quantities. The first byte is the number of 9-channel cards installed in the instrument when the acquisition was taken.

The second byte indicates the width and depth, dependent on the first byte and whether or not glitches are on (see "rawglitches"), as follows:

First Byte	Second Byte	WIDTH	DEPTH gl. off	DEPTH gl. on
0	0	0	0	0
1	0	9	513	257
2	0 1	18 9	513 1025	257 513
3	0 1	27 9	513 1537	257 769
4	0 1 2	36 18 9	513 1025 2049	257 513 1025

rawd18 - As with "rawd9", this is actually two one-byte quantities. The first byte is the number of 18-channel cards installed in the instrument when the acquisition was taken.

The second byte indicates the width and depth, dependent on the first byte, as follows:

First Byte	Second Byte	WIDTH	DEPTH
0	0	0	0
1	0	18	513
2	0 1	36 18	513 1025
3	0 1	54 18	513 1537
4	0 1 2	72 36 18	513 1025 2049

rawdata - This is a colossal bit-array which contains the data readout of the memory.

Each acquisition channel occupies a contiguous stream of bits in "rawdata". All of the memory for Pod 0/Channel 0 appears first, followed by all of the memory for Pod 0/Channel 1, and so on up to Pod 7/Channel 8. Only the pods which are currently active input pods are represented in "rawdata".

Its size is allocated to handle the worst case: 4680 = 72 channels x (65 bytes per channel).

For more specifics on the layout, see the more specific discussion in this specification.

rawglitches - This indicates whether glitch storage was enabled or disabled when the acquisition was performed. Glitch storage only applies to 9-channel cards, 18-channel cards are always in non-glitch format.

rawempty - This is an eight element array (one byte per element). For each pod, this entry tells if there was any data collected for that pod. Entries for pods that were not installed or were followers in a chain are meaningless.

0 = nonempty
1 = empty

rawlast - This indicates which of the timebases was the last to store a cycle in the memory. (This information is needed to decode the correlation channels.)

0 = timebase 2 was last to store
1 = timebase 1 was last to store

rawlength - This tells how many locations (in bytes) of "rawdata" contain meaningful information. When the 1240 sends a memory to a controller, only the meaningful portion of "rawdata" is transferred. In other words, only "rawlength" bytes of "rawdata" are transferred.

This variable is a 16-bit quantity, with the low-order byte stored first.

rawmisc -Ten bytes reserved for future use by Tektronix. Entries to this area could cause the loss of essential code. If the user must enter data in this field, it should be 0's.

rawoldest - This is an eight-element array (two bytes per element). Entries in this array correspond to acquisition pods. For each pod that was used, the entry in "rawoldest" indicates the oldest meaningful bit of the memory block in "rawdata" (for channels from that pod).

Entries for pods that were not installed or were followers in a chain are meaningless.

Entries for pods that were empty (see "rawempty") are also meaningless. The convention for "bit offsets" for non-glitch memories is as follows: The MSB of the highest-addressed byte in the channel is considered 0, with the next most significant bit being -1, and so on, until the LSB of that byte is -7. The MSB of the next lower addressed byte is -8, and so on.

For glitch memories, the convention is the same, except that each byte has only four meaningful "bits" in it.

Each element of this array is stored low-order byte first.

rawpodlen - This is an eight-element array (two bytes per element). Each entry in this array corresponds to one of the pods. The entry tells how many bytes of data in "rawdata" is allocated per channel for that pod. If a pod is not installed, or if the pod is a follower in a chain, then the entry in "rawpodlen" will be 0. For a pod that is installed and is the head of a chain, its length will be: (chaindepth) X 64 + 1, where "chaindepth" is the number of memories that were behind the pod (which will always be 1, 2, 3, or 4).

EXCEPTION: If "rawempty" for the pod is true, then no memory is allocated for it even ifit was the head of a chain.

Each element is stored with the low-order byte first.

rawpostfig - This indicates whether or not the data in the memory image has been modified after the acquisition by a data formatting routine in a ROM pack.

0 = not modified
1 = modified

If non-zero, the value of rawpostfig contains the ROM pack id of the ROM pack the modified the memory image.

rawtb - This is an eight-element array (one byte per element). Each entry of this array corresponds to one of the possible pods.

0 = this pod was the head of a chain and was assigned to TB1.
1 = this pod was the head of a chain and was assigned to TB2.
2 = this pod was a follower in a chain.
FF = there is no memory installed behind this pod position.

rawtb1type - This variable takes the values 0 and 1, representing the timebase 1 clock (0-async, 1-sync).

rawtb1async - takes the values 0-24, representing timebase 1 asynchronous clock rates of 10ns, 20ns, 50ns, 100ns, ..., 500ms, 1sec. Valid only if timebase 1 is asynchronous (i.e., "rawtb1async" contains 0).

rawtimevalid - This variable indicates whether or not it is possible to accurately determine the amount of real-time between each successive sample on TB1 (for delta-time readouts). It is possible if TB1 was asynchronous and used no clock qualification, and if the Global Event was not being used as a storage qualifier.

0 = real-time computations not possible
1 = real-time computations are possible

rawtpi1, rawtpi2 - "rawtpiQ" tells how many valid qualified clocks there were on TBQ on or after a trigger occurred. More information about them and how they are used can be found in the OVERVIEW OF DATA CORRELATION later in this appendix. Each of these variables is a 16-bit quantity, with the low-order byte first.

rawtrig - This variable indicates whether or not a trigger occurred during the acquisition.

0 = trigger occurred
1 = trigger did not occur

rawyoungest - This is an eight element array (two bytes per element). It is identical to "rawoldest" except that it indicates the offset of the youngest meaningful bit instead of the oldest.

Each element is stored with the low-order byte first.

General Discussion of The Memory Image Structure

Each channel of each active pod that collected any data will have a block of memory in "rawdata". These channel blocks are in order by channel number within pod number (thus Pod 0 Channel 0 is first, followed by Pod 0 Channel 1, etc.). Note that this refers to active pods, not to active memories. Data from memories that are followers in chains do not have a separate existence in this structure--they appear as part of the data from the pod that was the head of the chain. Each channel block will occupy sufficient memory to hold its worst case amount of valid data, based on its chaining level. Within this block, the values of "rawoldest[podnum]" and "rawyoungest[podnum]" contain the bounds of the valid data. (There is always invalid data somewhere in the block, due to the architecture of the acquisition hardware.)

"rawdata"

pod #	0	0	etc. . .	1	1	1	etc.	7	7	7
chan #	0	1	etc. . .	0	1	2	etc.	6	7	8

Organization of "rawdata" for four 18-channel acquisition cards and no chaining (72 X 513).

"rawdata"

pod #	0	0	. . . .	2	. . . .	6	6	empty
chan #	0	1		0		7	8	

Organization of "rawdata" for four 9-channel acquisition cards and no chaining (36 X 513). Only pods 0, 2, 4, and 6 appear.

The following diagram illustrates a channel's data in "rawdata" when glitches are not stored. 0 is the oldest bit stored.

	MSB							LSB
byte 0	7	6	5	4	3	2	1	0
byte 1	15	14	13	12	11	10	9	8
byte 2	23	22	21	20	19	18	17	16

etc. . . .
. . .
. . .

If glitches were stored:

	MSB							LSB
byte 0	3D	3G	2D	2G	1D	1G	0D	0G
byte 1	7D	7G	6D	6G	5D	5G	4D	4G
byte 2	11D	11G	10D	10G	9D	9G	8D	8G

etc. . . .
. . .
. . .

The glitch-format storage only applies to 9-channel cards. Regardless of the value of "rawglitches", 18-channel cards are stored in the non-glitch format.

If a given pod was assigned to a timebase that had no qualified cycles during the acquisition, then its "rawempty" entry will be TRUE, and no memory will be allocated for it in "rawdata".

The correlation channels are each formatted as a channel from one of the pods. The pod chosen will be the one associated with the proper timebase having the longest chaining. If that pod happens to be in glitch format, then the correlation is in glitch format; otherwise it is not. (In such cases, the correlation information is the "Data" bit, while the "Glitch" bit is unused.)

The correlation channels are allocated memory to handle the worst case. However, not all of that memory is necessarily used. The earliest Q bytes (where Q is the "rawpodlen" value of the pod that the correlation information was read from) will be organized exactly the same as the data channels from that pod. Any other bytes in "rawcorQ" are meaningless. (Indeed, some of the bytes within the first Q could be meaningless as well, but they will be the same as for the data channels.)

Overview of Data Correlation

Briefly, "data correlation" is a term for a proprietary Tektronix hardware feature, and for a way of processing the data received from that hardware in order to bring the data from two timebases into time relationship. This involves inserting cycles of "no data" into one or both data streams where needed.

The terms "oldest", "older", "youngest", and "younger" are used throughout this discussion. They have approximately their normal meanings.

If one cycle is "younger" than another, then that means that it happened later in time than the other. The "youngest" cycle in a memory is the last one to be sampled. The "oldest" cycle in the memory is the earliest one to be sampled.

The information derived from the proprietary hardware is found in "rawcor1", "rawcor2", and "rawlast".

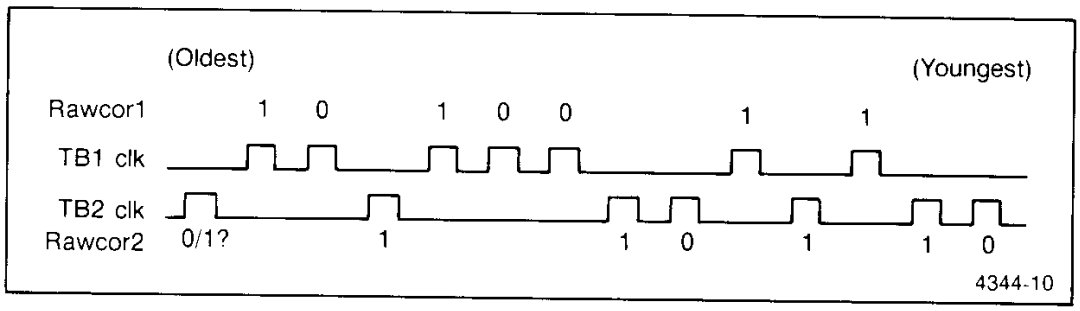


Figure A-1. Data correlation.

The hardware for each correlation channel looks at the other clock. A 1 is stored on "rawcor2" if there was at least one qualified edge on TB1 between the current edge on TB2 and the last edge on TB2. The assumption in Figure A-1 is that there is earlier activity to the left. If so, there is no way to know whether the first bit shown of "rawcor2" is a 0 or a 1. If that bit was the first of the acquisition, it would be 0.

If a clock edge from TB1 happens simultaneously with an edge from TB2, then the hardware indicates that TB1 happened first. For the purposes of correlation, there is no such thing as "simultaneous".

Time relationships are reconstructed beginning from the youngest cycle in the memory, proceeding backwards. It can be seen that the data consists of alternating packets of cycles--first one timebase, then the other, then back to the first. The packets of cycles are a series of zero or more 0's, terminated by a 1 (on the old end of the packet). The reconstruction algorithm starts from the "last stored" timebase and scans backwards until a "1" is encountered on the correlation channel, then switches to the other timebase. It then scans its correlation channel until a "1" is encountered, and then switches back to the original again, and continues in this fashion until all cycles have been used up on one of the timebases. At this point, all the remaining cycles from the other timebase, regardless of their values in the correlation channels, are appended to the oldest end of the memory.

Locating The Trigger

The values "rawtpi1" and "rawtpi2" (tpi means Trigger Position Indicator) are used after the time relationships have been reconstructed in the data to locate the trigger point. Each indicates the number of valid cycles that occurred on that timebase on or after the trigger.

Thus, in a one-timebase case, if the trigger was on the last cycle, then the TPI would contain 1. If on the next-to-last cycle, 2; and so on

In a two-timebase case, each TPI should be used to count from the end of the memory backwards to a certain location in the memory. Only valid cycles for that timebase should be counted. Each of the two TPI's will indicate a location in the reconstructed memory. If they don't agree, the earlier of the two is the trigger. (Note that both TPI's must be searched in the memory. It is not the case that the TPI with the highest value must necessarily be the earlier in the memory.)

There are certain extremely obscure cases in which the trigger location will be one earlier in memory than it should be. These cannot be corrected for in the memory reconstruction, and will not come up often enough to bother the average user.

RADIX TABLE

The following is a description of the Radix Table.

Overview

One of the types of files that can be placed in a user-generated pack (RAM or ROM Pack) is a file containing a Radix Table (RTABLE). Radix Tables allow the user to define his own mnemonics.

For each group defined in the Channel Grouping menu, there is a DISP field. This field is used to select the display radix for the group. The possible selections are HEX, OCT, BIN, OFF, ASC (ASCII) and, EBC (EBCDIC). In addition, if a ROM pack with a radix table is installed, ROM may be selected. If ROM is chosen, the mnemonics in the Radix Table are used to display the data in the group.

ROM Radix Table (RTABLE) Format

A Radix Table can be up to 769 bytes in length. The first byte of the radix table indicates the number of bits that should be considered a "digit" (the number of bits used as an index to the table of mnemonics). The maximum permissible value for this byte is 8.

Following the first byte is an array of up to 256 three-character mnemonics. The mnemonics are specified in 1240 Display codes.

```
    0:  number of bits used for index (8 maximum)
  1-3:  mnemonic for index 0
  4-6:  mnemonic for index 1
  7-9:  mnemonic for index 2
 10-12: mnemonic for index 3
          .
          .
766-768: mnemonic for index 255
```

Take the following Radix Table as an example:

	Index bits used
0: 8	
1-3: "A "	0
4-6: "B "	1
7-9: "C "	2
10-12: "D "	3
766-768: "ZZZ"	255

The following table shows how data would be displayed on the 1240 using HEX, BIN, and ROM display radices, using the ROM radix table above.

HEX	BIN	ROM
01	00000001	B
03	00000011	D
FF	11111111	ZZZ
00	00000000	A

Here's another example:

		Index bits used
0:	4	
1-3:	"S0 "	0
4-6:	"S1 "	1
7-9:	"R0 "	2
10-12:	"W0 "	3
	.	
	.	
	.	
46-48:	"END"	15

The following table shows how data would be displayed on the 1240 using HEX, BIN and ROM display radices.

HEX	BIN	ROM		
01F	0000011111	S0	S1	END
23F	1000111111	R0	W0	END
022	0000100010	S0	R0	R0

It is not necessary to use all 769 bytes of the Radix Table. You may specify four bits as the index to the table. The maximum index is 15, so only 16 elements of the possible 256 are used.

APPENDIX B

1240 DISPLAY CODE

This portion of Appendix B lists the 1240 Display Code characters and values.

Notice that there are two different tables of characters, *Changeable* and *Copycat*.

Changeable characters have values up to 3F, or XX11 1111. This leaves the top two bits free for *video display code*.

Table B-1 shows the video codes.

Table B-1
1240 VIDEO CODES

Video Form	Video Selection Code (bits 8 and 7)	Description
regular video	00	light type on black background
highlighted video	01	light type on grey background
reverse video	10	dark type on white background

To select a video display other than regular video, the first character of any string must be selected from the Changeable Character list.

All other characters in the string may be selected from either the Changeable Character or Copycat Character lists. Copycat characters are displayed in the previously-selected video form.

Example: Suppose you wanted to display the string "ABC" in reverse video on the 1240.

First, you determine the video selection code for reverse video, which, according to Table B-1, is 10.

Next, you obtain the values of your characters, making sure the first character of the string comes from the Changeable Character list:

A=0A (hex), or XX001010 (binary)
B=0B (hex), or XX001011 (binary)
C=0C (hex), or XX001100 (binary)

Then, add the video selection code for reverse video:

A=10001010
B=10001011
C=10001100

Last, convert the values to hex, and enter them in your program.

Table B-2
1240 CHANGEABLE CHARACTERS

value	display	value	display	value	display
00	0	15	L	2A	$
01	1	16	M	2B	open delim.
02	2	17	N	2C	close delim
03	3	18	O	2D	(unused)
04	4	19	P	2E	X
05	5	1A	Q	2F	¿
06	6	1B	R	30	+
07	7	1C	S	31	–
08	8	1D	T	32	]
09	9	1E	U	33	[
0A	A	1F	V	34	*
0B	B	20	W	35	◆
0C	C	21	X	36	▌
0D	D	22	Y	37	◀
0E	E	23	Z	38	▶
0F	F	24	(space)	39	▪ (and)
10	G	25	▪	3A*	alt. space
11	H	26	,	3B	— (cursor 1)
12	I	27	/	3C	-- (cursor 2)
13	J	28	▪	3D	⎍ (rising edge)
14	K	29	^	3E	⎎ (falling edge)
				3F	\

4344-15

* Use the space character 3A for radix tables, instead of the space character 24.

Table B-3
1240 COPYCAT CHARACTERS

value	display	value	display	value	display
C0	!	D5	|	EA	(hist 3)
C1	#	D6	‹	EB	(hist 4)
C2	%	D7	{	EC	(hist 5)
C3	&	D8	}	ED	(hist 6)
C4	'	D9	~	EE	(hist 7)
C5	(	DA	∫	EF	(hist 8)
C6	)	DB	┬	F0	"
C7	=	DC	┤	F1	—
C8	\	DD	|	F2	
C9	(unused)	DE	| (hist 1)	F3	‾
CA	>	DF	(hist 2)	F4	
CB	△	E0		F5	
CC	⊔	E1		F6	
CD	<	E2		F7	
CE	>	E3		F8	
CF	;	E4		F9	
D0	?	E5		FA	
D1	@	E6		FB	
D2	¢	E7		FC	
D3	¬ (not)	E8		FD	(unused)
D4	¦	E9	␣ (space)	FE	
				FF	(switch to tim. diag.)

4344-16

1241 DISPLAY CODE

This portion of Appendix B lists the 1241 Display Code characters and values. Changeable characters have values up to 7F, or X111 1111. This leaves the top bit (bit 8) free for the *video selection code* (Table B-4).

Table B-4
1241 VIDEO CODES

Video Form	Video Selection Code (bit 8)	Description
regular video	0	color type on black background
reverse video	1	color type on color background

To produce a color video display, the first two bytes of every text string that is sent must contain a color identifier and color type. In the text string #H2D01XXXXX.., the color identifier is 2D and the color type is the byte following. In this case, the color type, 01, specifies red characters on a black background. The length of the text string, including color identifier and color type, should be no longer than approximately 58 bytes total. The valid color types are identified as follows:

00 = green characters on black background
01 = red characters on black background
10 = yellow characters on black background
11 = yellow characters on red background

If the text string is to be displayed in the default color (green on black), it is not necessary to specify the color identifier (2D) and color type (00). The reason being that the 1241 string #H2D0001.. and the 1240 string #H01.. will produce the same results, a green 1 character printed on a black background.

The video form operates similarly for the 1240 monochrome strings as for the 1241 color strings. The video form changes between regular characters and reverse characters by manipulation of bit 8 for each character.

Example: Suppose you wanted to display the string "ABC" in reverse red video on the 1241 (black characters on a red background).

First, you determine the video selection code for reverse video, which, according to Table B-1, is 1. Next, you obtain the values of your characters from the Changeable Character list:

A=0A (hex), or 00001010 (binary)
B=0B (hex), or 00001011 (binary)
C=0C (hex), or 00001100 (binary)

Then, add the video selection code for reverse video by making bit 8=1:

A=10001010
B=10001011
C=10001100

Last, convert the values to hex, and enter them in your program. For example: the text string DISPLAY 2,1,CODE #H2D018A8B8C will produce a display showing a reverse red ABC on line 2, starting at column 1.

Table B-5
1241 CHANGEABLE CHARACTERS

value	display	value	display	value	display
00	0	15	L	2A	$
01	1	16	M	2B	open delim.
02	2	17	N	2C	close delim
03	3	18	O	2D	(unused)
04	4	19	P	2E	X
05	5	1A	Q	2F	¿
06	6	1B	R	30	+
07	7	1C	S	31	–
08	8	1D	T	32	]
09	9	1E	U	33	[
0A	A	1F	V	34	*
0B	B	20	W	35	◆
0C	C	21	X	36	▌
0D	D	22	Y	37	◀
0E	E	23	Z	38	▶
0F	F	24	(space)	39	▪ (and)
10	G	25	▪	3A*	alt. space
11	H	26	,	3B	— (cursor 1)
12	I	27	/	3C	-- (cursor 2)
13	J	28	⁝	3D	⎍ (rising edge)
14	K	29	^	3E	⎎ (falling edge)
				3F	\

4344-17

* Use the space character 3A for radix tables, instead of the space character 24.

Table B-6
1241 CHANGEABLE CHARACTERS (cont.)

value	display	value	display	value	display
40		55		6A	(hist 3)
41		56		6B	(hist 4)
42		57		6C	(hist 5)
43		58		6D	(hist 6)
44		59		6E	(hist 7)
45		5A		6F	(hist 8)
46		5B		70	
47		5C		71	
48		5D		72	
49	(unused)	5E	(hist 1)	73	
4A		5F	(hist 2)	74	
4B		60		75	
4C		61		76	
4D		62		77	
4E		63		78	
4F		64		79	
50		65		7A	
51		66		7B	
52		67		7C	
53	¬ (not)	68		7E	
54		69	␣ (space)	FD	graticule character
				FF	(switch to tim. diag.)

4344-18

APPENDIX C

1240 KEY CODES

When the controller sends a KEY command to the 1240, the 1240 stores a key code for the next 1240 key pressed. The controller then requests the key code with a KEY? query. The 1240 responds with the message KEY <keycode>. If no KEY operation was performed, or if it was terminated before completion, the 1240 sends an invalid key code to the controller.

Table C-1
1240 KEY CODES

1240 Key	Key Code
Hard keys:	
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
A	10
B	11
C	12
D	13
E	14
F	15
DON'T CARE	16
GLITCH	17
(up arrow)	18
(down arrow)	19
(left arrow)	20
(right arrow)	21
SELECT UP	22
SELECT DOWN	23
NEXT	24
TRIGGER	25
CONFIG	26
DATA	27
EDIT	28
UTILITY	29
START	30
STOP	(not monitored)
AUTO	31

Table C-1 (cont.)
1240 KEY CODES

1240 Key	Key Code
Soft Keys:	
Top left	70
Top 2nd	71
Top Middle	72
Top 4th	73
Top right	74
Bottom left	75
Bottom 2nd	76
Bottom Middle	77
Bottom 4th	78
Bottom right	79
Invalid Key:	99

APPENDIX D

ERROR AND EVENT CODES

Table D-1
SYSTEM EVENTS, PRIORITY 1

Hex	Decimal	Event	Name
41 or 51	65 or 81	401	ONLINE (Power On)

Table D-2
COMMAND ERRORS, PRIORITY 2

Hex	Decimal	Event	Name
61 or 71	97 or 113	101	Command header error
61 or 71	97 or 113	102	Header delimiter error
61 or 71	97 or 113	103	Command argument error
61 or 71	97 or 113	104	Argument delimiter error
61 or 71	97 or 113	105	Non-numeric Argument (numeric expected)
61 or 71	97 or 113	106	Missing argument
61 or 71	97 or 113	107	Invalid message unit delimiter
61 or 71	97 or 113	108	Binary block checksum error
61 or 71	97 or 113	109	Binary block byte count error
61 or 71	97 or 113	121	Illegal Hex Character
61 or 71	97 or 113	122	Unrecognized argument type
61 or 71	97 or 113	123	Argument is too large
61 or 71	97 or 113	124	Non-binary Argument (binary or hex expected)

Table D-3
EXECUTION ERRORS, PRIORITY 3 and 4

Hex	Decimal	Event	Name
Priority 3			
62 or 72	98 or 114	201	Remote Only command received while in local mode
62 or 72	98 or 114	202	Command aborted - change to to local
62 or 72	98 or 114	203	I/O Deadlock detected
62 or 72	98 or 114	205	Argument out of range
62 or 72	98 or 114	206	Group execute trigger ignored
62 or 72	98 or 114	251	Header/Location conflict in ACQMEM, REFMEM, INSETUP, or RAMPACK
62 or 72	98 or 114	252	System error (illegal command)
62 or 72	98 or 114	253	Integer overflow (range 0 - 65535)
62 or 72	98 or 114	254	RAM pack not installed.
62 or 72	98 or 114	255	Illegal ROM pack command
62 or 72	98 or 114	256	REFMEM not compatible with ACQMEM
62 or 72	98 or 114	257	TEST command cannot be executed when RQS is off

Table D-3 (cont.)
EXECUTION ERRORS, PRIORITY 3 and 4

Hex	Decimal	Event	Name
Priority 4			
62 or 72	98 or 114	261	Possible loss of data - change to local during upload
62 or 72	98 or 114	262	Acquisition terminated - change to local
62 or 72	98 or 114	263	Auto-run terminated - change to local
62 or 72	98 or 114	264	Key operation terminated - change to local
62 or 72	98 or 114	265	Conflict in SETUP memory
62 or 72	98 or 114	266	Data block location out of range
E0 or F0	224 or 240	271	Output buffer full
E0 or F0	224 or 240	272	Command too long

Table D-4
SOFT KEY EVENTS, PRIORITY 5

The following events may be generated by pushing one of the soft keys in the COMM Port Control menu. This is a mechanism for the 1240 to tell the controller to perform one of the following.

Hex	Decimal	Event	Name
C0 or D0	192 or 208	711	Request ACQMEM upload
C1 or D1	193 or 209	712	Request REFMEM upload
C2 or D2	194 or 210	713	Request REFMEM download
C3 or D3	195 or 211	714	Request SETUP upload
C4 or D4	196 or 212	715	Request SETUP download

Table D-5
OPERATION COMPLETE EVENTS, PRIORITY 6

Hex	Decimal	Event	Name
C5 or D5	197 or 213	721	End of acquisition
C6 or D6	198 or 214	722	End of auto-run
C7 or D7	199 or 215	723	End of KEY
C9 or D9	201 or 217	724	End of auto-run, memories equal
CA or DA	202 or 218	725	End of auto-run, memories not equal
C8 or D8	200 or 216	731	Diagnostics test complete

Table D-6
NORMAL DEVICE-DEPENDENT STATUS, PRIORITY 7

The following are normal device-dependent status and do not generate SRQ's.

Hex	Decimal	Event	Name
80 or 90	128 or 144	000	Idle (No status to report)
81 or 91	129 or 145	000	Acquisition in process
82 or 92	130 or 146	000	Auto-run in process
83 or 93	131 or 147	000	Waiting for key press.

EVENT CODE EXPLANATIONS

101—Command Header Error. The command header received was not a valid command header. The command could have been misspelled or was perhaps too abbreviated. The HELP? query returns a list of valid headers.

102—Header Delimiter Error. The 1240 was expecting a space, comma or legal argument.

103—Command Argument Error. The argument contains an illegal character or missing character.

104—Argument Delimiter Error. The comma is missing.

105—Non-numeric Argument.

106—Missing Argument.

107—Invalid Message Unit Delimiter.

108—Binary Block Checksum Error. The 1240 received a data block in Binary Block format. It computed the checksum and compared it with the checksum in the data block. The two did not match.

109—Binary Block Byte Count Error. Data block byte count exceeds maximum. For data block format, see *Data Block Format, Section 2.*

121—Illegal Hex Character. A non-hexadecimal character is present. Legal hexadecimal characters include 0-9 and A-F or a-f. An odd number of hexadecimal characters in an argument can cause this error.

122—Unrecognized Argument Type.

123—Argument is Too Large. Byte count maximum is 97.

124—Non-binary Argument. The 1240 was expecting a binary argument.

201—Remote Only command received while in Local. A Remote Only command was received by the 1240 while it was in Local state.

202—Command Aborted—Change to Local. A transition to Local state caused a previously received but unexecuted Remote Only command to be aborted.

203—I/O Deadlock Detected. Both input and output buffers are full. The Output buffer is flushed.

205—Argument Out of Range. The range of legal values has been exceeded.

206—Group Execute Trigger Ignored. A Device Trigger is received while the 1240 is in Local State. Also sent when the DT is off.

251—Header/Location Conflict. During a download (controller to 1240), the location field in a data block did not agree with the command header sent at the beginning of the message. For example: INSETUP #H3400000001FF75C4 states that the data blocks are going to the instrument setup. The first two digits of the 6-digit location field of the data block also indicate the destination of the data block (Setup, Memory Image, or RAM Pack). In this case, the location field indicates that the destination is the Memory Image. This conflicts with the header. (Memory Image address is 00XXXX, SETUP address is 01XXXX, and RAM Pack address is 02XXXX.)

252—System Error. Illegal command.

253—Integer Overflow. The integers have exceeded 0 to 65535.

254—RAM Pack Not Installed. The RAMPACK? query was received by a 1240 without an installed RAM Pack.

255—Illegal ROM Pack Command. A command dedicated to a ROM Pack was sent to a 1240 without an installed ROM Pack.

256—REFMEM not Compatible with ACQMEM. The controller attempted to perform an Auto-run comparing Reference Memory to Acquisition Memory. The memories were acquired with different 1240 SETUP configurations and cannot be compared.

257—RQS off; cannot execute command.

261—Possible Loss of Data— Change to Local. A transition from Remote to Local occurred while an upload was in progress. This terminated the upload at that point. It is likely that the upload is incomplete.

262—Acquisition Terminated—Change to Local. An acquisition was terminated when the 1240 returned to Local state.

263—Auto-run Terminated—Change to Local. An auto-run was terminated when the 1240 returned to Local state.

264—Key Operation Terminated—Change to Local. A KEY operation was terminated prematurely when the 1240 returned to Local state.

265—Conflict in Setup. An instrument setup error exists. Acquisition or auto-run cannot begin until it is corrected.

266—Data Block Location Out of Range. The location field in a data block consists of 6 digits (or three bytes in Binary Block format). The first two digits indicate type of memory (Setup or Memory Image). The last four digits indicate the address within the memory. This error indicates that the address digits exceeded the memory range limit.

271—Output Buffer Full. Too many queries have been received in a particular command.

272—Input Buffer Full. The command was too long to fit into the input buffer.

401—This SRQ informs the controller that the 1240 is now ONLINE and can function as an active member of the system. This event code is sent when the 1240 user causes a transition from OFFLINE to ONLINE, and when the 1240 is powered on after having been powered off when ONLINE.

711—REQUEST ACQMEM UPLOAD 1240 soft key was pressed.

712—REQUEST REFMEM UPLOAD 1240 soft key was pressed.

713—REQUEST REFMEM DOWNLOAD 1240 soft key was pressed.

714—REQUEST SETUP UPLOAD 1240 soft key was pressed.

715—REQUEST SETUP DOWNLOAD 1240 soft key was pressed.

721—End of Acquisition.

722—End of Auto-Run. An auto-run not involving comparison of the Acquisition Memory to the Reference Memory is complete.

723—End of Key Operation. A KEY operation is complete. A subsequent KEY? query will reveal the keycode of the 1240 key pressed.

724—End of Auto-Run, memories equal. An auto-run involving comparison of Acquisition and Reference memories is complete. The most recent comparison shows the two memories to be equal, using the limits defined in the Auto-Run Spec menu (in the SETUP).

725—End of Auto-Run, memories not equal. Similar to 724, only memories are not equal.

731—Diagnostics Test Complete. The TEST command sequence is complete.

APPENDIX E

INTRODUCTION TO GPIB

"GPIB" and "General Purpose Interface Bus" are terms applied to interfaces conforming to IEEE Standard 488-1978. This standard describes a digital interface that allows efficient communications between instruments interconnected in a system, regardless of its purpose.

The IEEE 488 standard defines three parts of the interface: *electrical* elements, *mechanical* elements, and *functional* elements.

In a working GPIB system, additional *operational* elements are required to define the *device-dependent messages* that control each instrument's operations. Device--dependent messages are not defined by IEEE, since they are different for each instrument. See *Device-Dependent Messages* in *Section 2* for a list of the messages supported by the 1240.

ELECTRICAL ELEMENTS

The GPIB is TTL-compatible. The power source for bus drivers and receivers cannot exceed +5.25 V, referenced to logic ground. Bus drivers are typically open-collector devices, but may be tri-state under some circumstances. The standard defines logic levels as follows:

Logic	Electrical Properties	Other References
0	2.0 V - 5.2 V	High State, unasserted, false
1	0.0 V - 0.8 V	Low State, asserted, true

MECHANICAL ELEMENTS

Mechanical elements are the bus connector and the cable.

A 24-pin bus connector and cable are used to connect one instrument to another.

Connecting GPIB Systems

1. A maximum of 15 primary-addressed devices, including the controller, can be connected to the bus cable at one time.
2. At least one device must be connected for every two meters of cable, average. (For example, an 8-meter cable must have at least 4, but no more than 15, devices connected to it.)
3. The combined bus cable length cannot exceed 20 meters.
4. Any primary-addressed device may have *secondary-addressed* devices attached to it. For example, a primary-addressed mainframe may contain secondary-addressed plug-ins.
5. At least 2/3 of the primary devices must be powered-on.

NOTE

The 1240 can only be a primary-addressed device.

GPIB Connector

The 24-pin GPIB Connector Plug Interconnecting Cables include both plug- and receptacle-connector types at each end, to allow either a star or linear bus structure. Connectors may be rigidly stacked, using standard counter-bored captive screws. A Connector Plug is shown in Figure E-1.

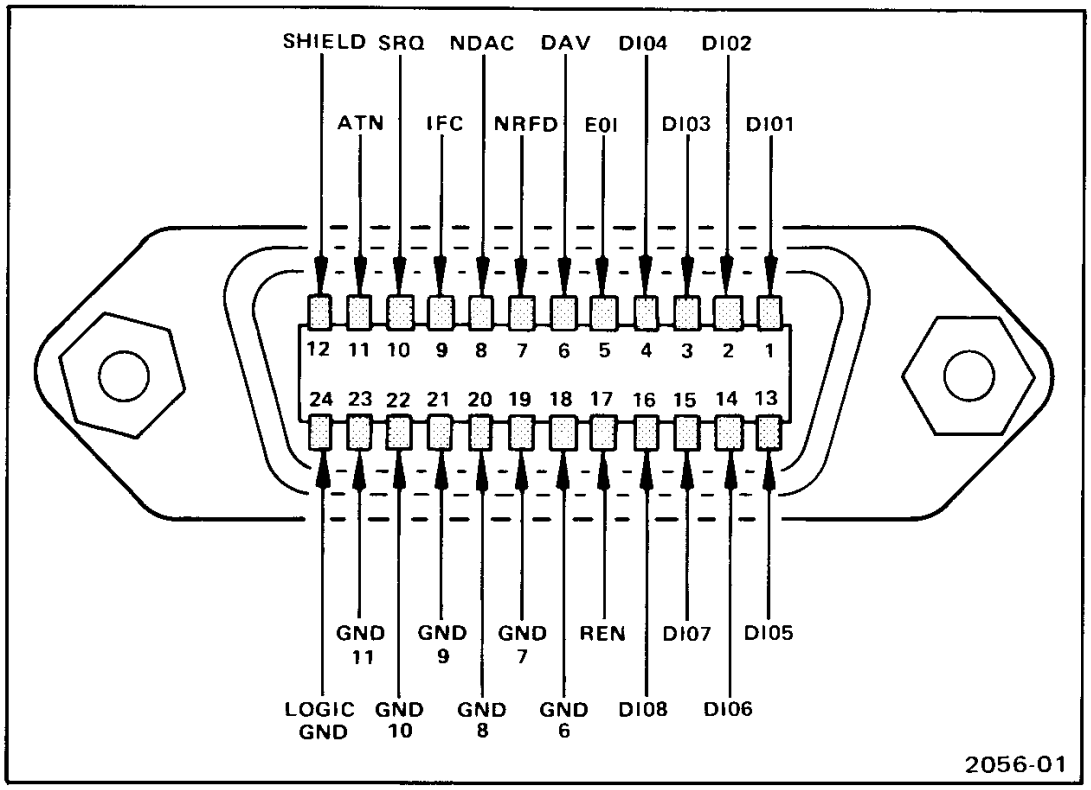


Figure E-1. GPIB Interface Connector Plug, with pin locations.

GPIB

The GPIB cable system consists of eight data I/O lines, five control lines for bus management, and three data handshake lines. Figure E-2 shows a breakdown of the cable system.

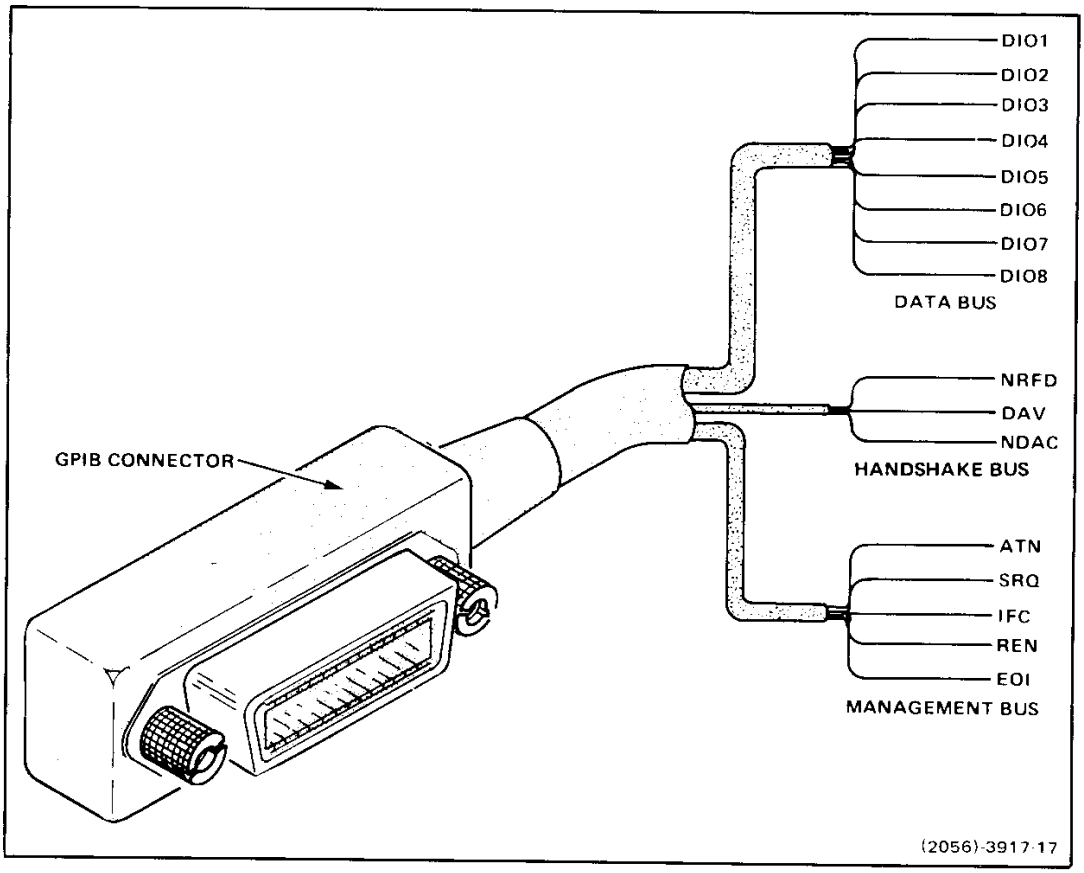


Figure E-2. The GPIB Bus Lines.

FUNCTIONAL ELEMENTS

Typical GPIB System

Only four instruments are shown, but the GPIB can support up to 15 instruments connected directly to the bus. More than 15 devices can be connected if they are interfaced through a primary device rather than connected directly to the bus. These become secondary-addressed devices.

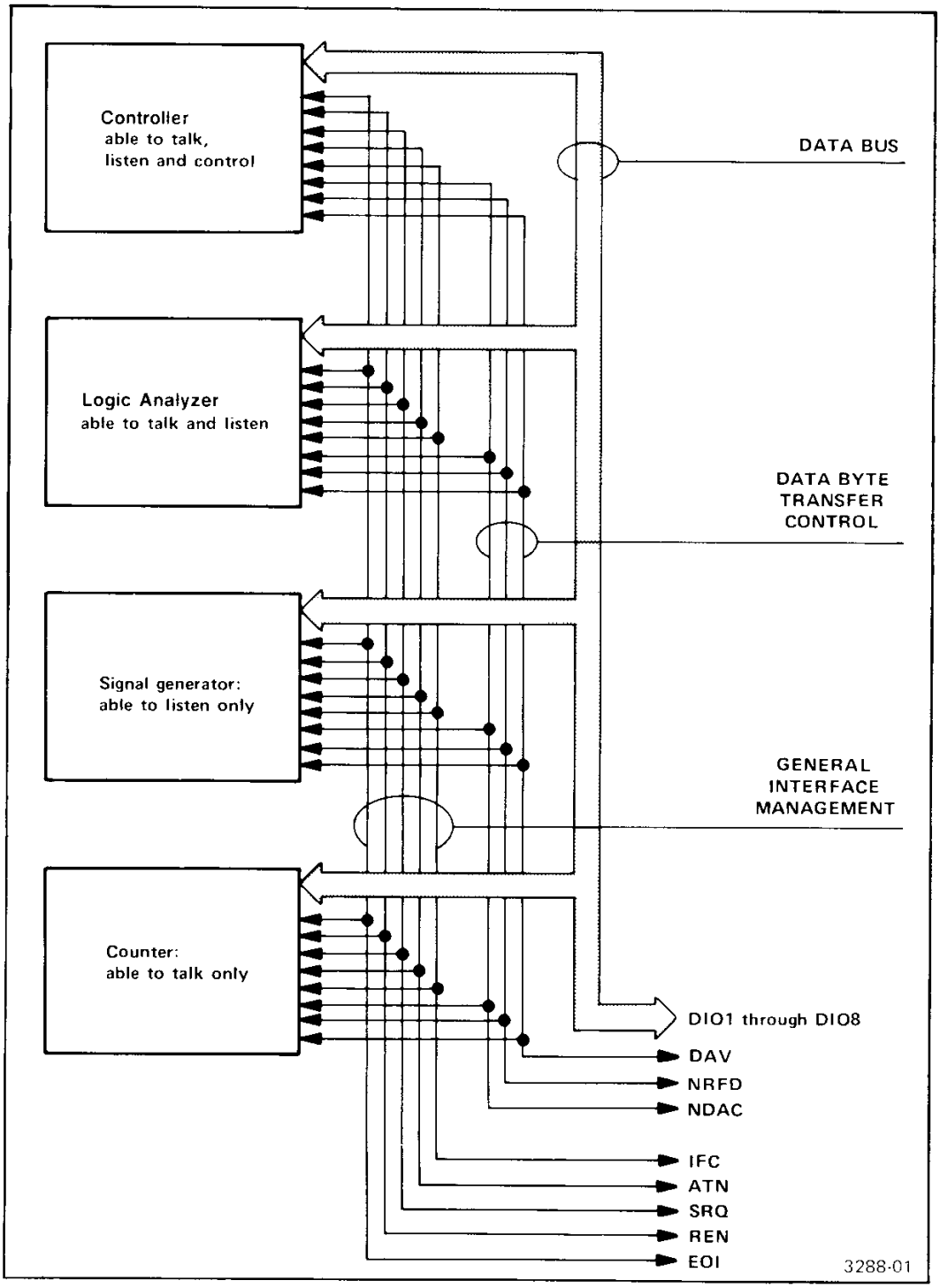


Figure E-3. A typical GPIB system setup with four primary-addressed devices.

Controllers, Talkers, and Listeners

A GPIB system device may be a *controller*, a *talker*, a *listener*, or a combination of all three. The 1240, for example, is a talker and listener, but not a controller.

A *talker* is an instrument that can be addressed with *interface messages* to send data over the data bus. Only one instrument at a time in a system can be addressed as a talker.

A *listener* is an instrument that can be addressed with interface messages to accept data from the data bus. Any number of instruments at a time can be addressed as listeners.

The *controller*, using interface messages, addresses instruments as talkers and listeners. The controller also has the ability to address itself as a talker or listener whenever the need arises. In addition to designating the current talker and listeners for a particular communication sequence, the controller has the task of sending other interface messages to any or all of the instruments on the bus.

More than one controller may exist on the bus at one time. If more than one controller is on the bus, the following rules apply: Only one controller may be designated as the *system controller*. The system controller is distinguished as the only controller that:

1. may take control of the bus by clearing it with the *IFC* (Interface Clear) interface message.
2. controls the REN line.

Otherwise, controllers and system controllers are identical. One controller may pass control to another with the *TCT* (Take Control) interface message. Only one controller may act as a controller at any one time. It is called the *controller-in-charge*. The other controllers may act as talkers or listeners, if desired.

Interface Messages

Interface messages are commands sent by the controller-in-charge to perform certain operations on the bus. Only the controller-in-charge may send interface messages. Interface messages are of two types:

- Uniline messages (sent over a single line of the data transfer control or general interface management bus). Uniline messages are ATN, SRQ, IFC, REN and EOI.
- Multiline messages (sent over the data bus with the ATN line asserted). Most messages are multiline.

All interface messages are listed in *Section 2*.

Figure E-4 is a chart that relates multiline message coding to various formats. This chart provides binary, octal, decimal, hexadecimal, and ASCII formats for the interface messages.

ASCII & IEEE 488 (GPIB) CODE CHART

DIO 4 3 2 1	7 6 5 = 0 0 0 CONTROL	0 0 1 CONTROL	0 1 0 NUMBERS SYMBOLS	0 1 1 NUMBERS SYMBOLS	1 0 0 UPPER CASE	1 0 1 UPPER CASE	1 1 0 LOWER CASE	1 1 1 LOWER CASE
0 0 0 0	0 NUL 0 0	20 DLE 10 16	40 SP 20 32	60 0 30 48	100 @ 40 64	120 P 50 80	140 ` 60 96	160 p 70 112
0 0 0 1	1 GTL SOH 1 1	21 LLO DC1 11 17	41 ! 21 33	61 1 31 49	101 A 41 65	121 Q 51 81	141 a 61 97	161 q 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 " 22 34	62 2 32 50	102 B 42 66	122 R 52 82	142 b 62 98	162 r 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 # 23 35	63 3 33 51	103 C 43 67	123 S 53 83	143 c 63 99	163 s 73 115
0 1 0 0	4 SDC EOT 4 4	24 DCL DC4 14 20	44 $ 24 36	64 4 34 52	104 D 44 68	124 T 54 84	144 d 64 100	164 t 74 116
0 1 0 1	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 % 25 37	65 5 35 53	105 E 45 69	125 U 55 85	145 e 65 101	165 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 & 26 38	66 6 36 54	106 F 46 70	126 V 56 86	146 f 66 102	166 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' 27 39	67 7 37 55	107 G 47 71	127 W 57 87	147 g 67 103	167 w 77 119
1 0 0 0	10 GET BS 8 8	30 SPE CAN 18 24	50 (28 40	70 8 38 56	110 H 48 72	130 X 58 88	150 h 68 104	170 x 78 120
1 0 0 1	11 TCT HT 9 9	31 SPD EM 19 25	51) 29 41	71 9 39 57	111 I 49 73	131 Y 59 89	151 i 69 105	171 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 * 2A 42	72 : 3A 58	112 J 4A 74	132 Z 5A 90	152 j 6A 106	172 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 + 2B 43	73 ; 3B 59	113 K 4B 75	133 [5B 91	153 k 6B 107	173 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 , 2C 44	74 < 3C 60	114 L 4C 76	134 \ 5C 92	154 l 6C 108	174 ¦ 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 − 2D 45	75 = 3D 61	115 M 4D 77	135] 5D 93	155 m 6D 109	175 } 7D 125
1 1 1 0	16 SO E 14	36 RS 1E 30	56 . 2E 46	76 > 3E 62	116 N 4E 78	136 ^ 5E 94	156 n 6E 110	176 ~ 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 2F 47	77 UNL ? 3F 63	117 O 4F 79	137 UNT _ 5F 95	157 o 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY

octal 25 PPU — GPIB code
NAK — ASCII character
hex 15 21 — decimal

2690-10

Figure E-4. The GPIB Code Chart.

Interface messages are divided into five categories.

Addressed Commands. Only instruments on the bus that are addressed to listen receive addressed commands. (The exception is the command TCT, which requires that the instrument receiving it be addressed to talk.)

Universal Commands. All instruments on the bus will receive universal commands regardless of whether the instrument is addressed or unaddressed.

Listen Addresses. The available range of listen addresses is 0 to 30. An address of 31 will unlisten (UNL) all instruments on the bus.

Talk Addresses. The available range of talk addresses is 0 to 30. A talk address of 31 will untalk (UNT) all instruments on the bus.

Secondary Addresses or Commands. The range of secondary addresses is 0 to 30. This is useful with instruments that have a mainframe and several plug-ins. The mainframe may be set to a primary talk or listen address and each plug-in set to a secondary address.

GPIB Signal Line Definitions

Figure E-3 illustrates the 16 signal lines of the GPIB functionally divided into three component buses:

1. Eight-bit data I/O bus
2. Three-line transfer (handshake) bus
3. Five-line management bus

The Data Bus. The data bus has eight bidirectional signal lines. It carries all multiline interface messages and the device dependent messages. A handshake sequence between the source device and the acceptor device transfers one data byte at a time. Since the GPIB handshake sequence is an asynchronous operation, the data transfer rate is only as fast as the slowest instrument involved in a data byte transfer at any one time. A talker cannot place data bytes on the bus faster than any one listener can accept them.

Figure E-5 illustrates the flow of data bytes when a typical controller sends ASCII data to an assigned listener on the bus. The first data byte, $44_{decimal}$, enables primary-addressed device 12 as a listener. The second data byte, $108_{decimal}$, enables a secondary-addressed plug-in (number 12) as the final destination of the data. The data consists of two ASCII characters A and B ($65_{decimal}$ and $66_{decimal}$).

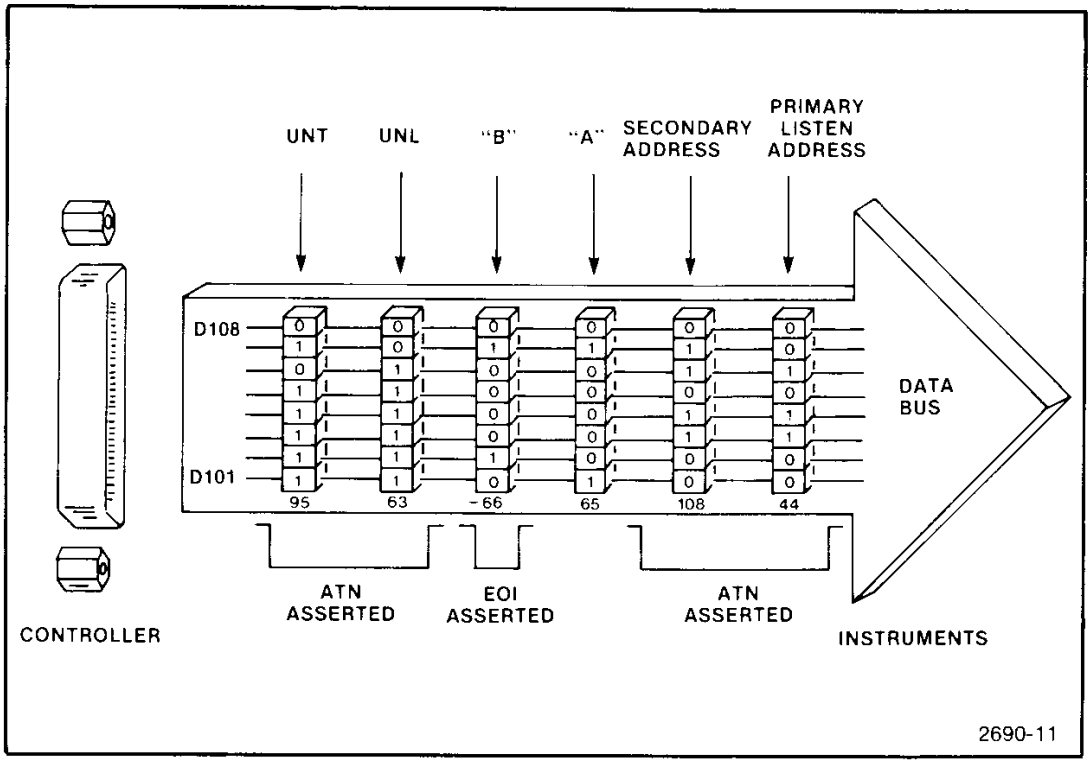


Figure E-5. An example of data byte traffic on the GPIB.

The EOI line is asserted along with data byte B to signify the end of the device-dependent message. The controller activates the ATN line again and sends the UNL (unlisten) and UNT (untalk) commands to clear the bus. In this case the UNT command was not necessary, but many controllers, as a matter of practice, send UNT and UNL after a transaction. Six handshake cycles on the transfer bus are required to send the six data bytes.

Notice that the most significant bit on the data bus, DI08, is not used for interface messages; the chart in Figure E-4 shows only DI01-DI07.

The Transfer Bus. Each time a data byte is sent over the data bus the source device and the acceptor device execute a handshake sequence via the transfer bus. Figure E-6 illustrates the basic timing relationship between the three transfer bus lines, DAV, NRFD, and NDAC. The ATN line is included to illustrate its role in the process.

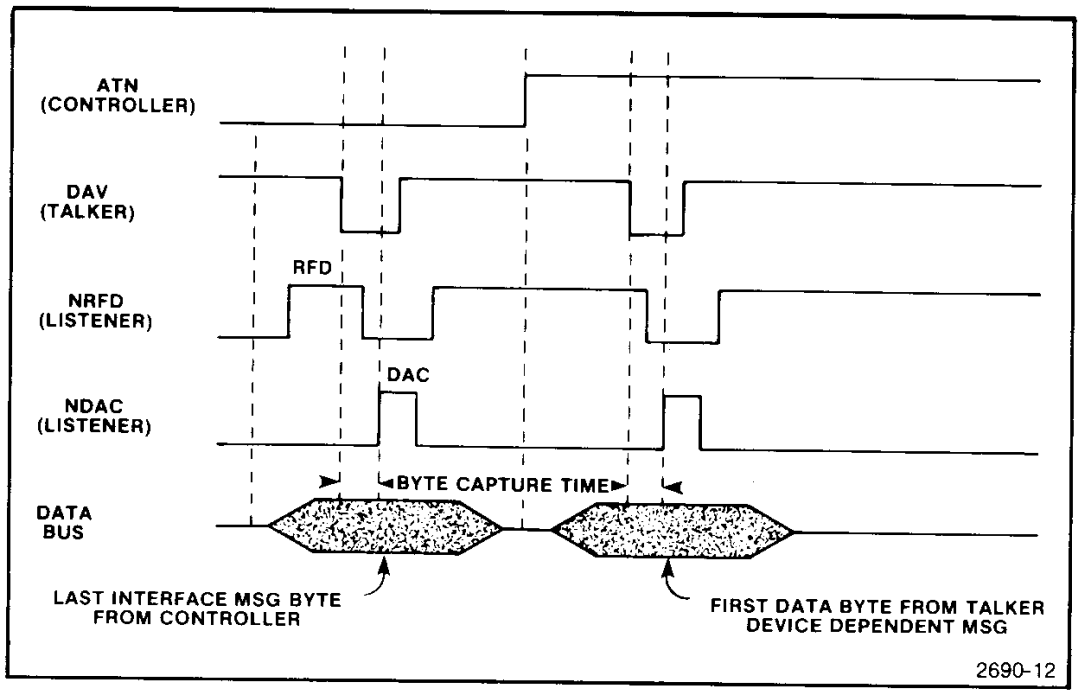


Figure E-6. A typical handshake timing sequence.

- **Data Valid (DAV)** - The DAV signal is asserted (low) by the talker after the talker places a data byte on the data bus. When asserted, DAV tells each assigned listener that a new data byte is on the data bus. The talker is inhibited from asserting DAV as long as any listener holds the NRFD signal asserted.

- **Not Ready For Data (NRFD)** - An asserted (low) NRFD signal indicates that one or more assigned listeners are not ready to receive the next data byte from the talker.

- **Not Data Accepted (NDAC)** - Each assigned listener holds the NDAC signal low-true until the listener accepts the data byte currently on the data bus. When all assigned listeners accept the current data byte, the NDAC line becomes unasserted (high), telling the talker that all assigned listeners accepted the current data byte, and to remove the data byte from the bus.

The Management Bus. The management bus is a group of five lines which are used to send uniline interface messages.

- **Attention (ATN)** - A controller asserts the ATN signal line when a controller wants to send a multiline interface message to one or more instruments on the bus. Only a controller may assert the ATN line.

- **End or Identify (EOI)** - A talker can use the EOI signal line to indicate the end of a data transfer sequence. The talker asserts the EOI signal line along with the last byte of data transmitted. The EOI line is also used when conducting a parallel poll. A controller may assert ATN with EOI when conducting a parallel poll sequence.

- **Interface Clear (IFC)** - The interface clear line untalks and unlistens all of the instruments on the bus. Only the system controller may assert IFC. Only three messages may be recognized while IFC is asserted: Device Clear (DCL), Local Lockout (LLO), and Parallel Poll Unconfigure (PPU). Interface clear is usually necessary only when there is more than one controller on the bus and the system controller wants to regain control.

- **Remote Enable (REN)** - The system controller asserts the REN signal line whenever the interface system operates under Remote program control. The REN line, in conjunction with other interface messages, causes an instrument on the bus to respond to program control (Remote) when true, and to its front panel controls (Local) when false.

- **Service Request (SRQ)** - Any instrument connected to the bus can request the controller's attention by asserting the SRQ line. The controller may respond by asserting ATN and executing a serial poll of the *status byte* of each instrument to determine which instrument is requesting service. After the instrument requesting service is found, program control is transferred to a service routine for that instrument. When the service routine is completed, program control returns to the main program. When polled, the instrument requesting service unasserts the SRQ line.

Interface Functions

The following discussion provides general information about the 10 GPIB interface functions, and their relationship to various interface messages. The 10 interface functions provide a variety of capabilities for each device within a system. These functions may be implemented within an instrument by either hardware or software. Only those functions necessary for an instrument's purpose are implemented by the instrument's designer. It is unlikely that a single instrument will have all ten interface functions. For example, the 1240 does not support the PP or C functions.

For information about the interface function subset supported by the 1240, see *Section 2.*

Table E-1
GPIB INTERFACE FUNCTIONS -- GENERAL

Symbol	Interface Function
SH	Source Handshake
AH	Acceptor Handshake
T	Talker
L	Listener
SR	Service Request
RL	Remote Local
PP	Parallel Poll
DC	Device Clear
DT	Device Trigger
C	Controller

Source and Acceptor Handshake Functions. These functions control the data transfer control bus handshake. The Source Handshake function guarantees proper transmission of data, while the Acceptor Handshake function guarantees proper reception of data.

The SH function must wait for at least 2 μs after receiving the RFD message before sending DAV. The additional time allows data to settle on the data bus. If tri-state drivers are used, the settling time is reduced to 1.1 microseconds after RFD is true.

The time it takes for the AH function to accept a message (signified by sending the DAC message) depends only on how the device implements the function.

Talker Function. The T function provides an instrument with the ability to send device-dependent data (including status bytes during *serial polls*) over the GPIB. The T function becomes active when a talker receives a one-byte primary address code called MTA (My Talk Address). Only one device may be addressed to talk at any one time.

- ATN - The T function suspends any current activity when the ATN line is true, and watches the data bus for interface messages that may affect the function's operation. If the device is addressed as a talker, it begins to transmit data when ATN becomes false.
- END - The talker may send the END message (EOI asserted) while sending a data byte to indicate that the byte is the last of a device-dependent message.
- IFC - The T function becomes unaddressed when the IFC line is pulsed for at least 100 microseconds.
- MLA - If a device contains both talker and listener functions, the talker function is usually disabled when the MLA message is true. Typically, a device's listen address is the same as its talk address.
- MTA - The T function within a device is enabled when the MTA message is sent.
- SPD - The serial poll response capability of a T function is disabled when SPD is received.
- SPE - The serial poll response capability of the T function is enabled when SPE is received. If a device is talk-addressed after it receives SPE, it sends its status byte after ATN becomes false.
- STB - The STB message represents a device's status during a serial poll. The devices's status also contains the RQS message.
- UNT - The T function is disabled when the controller sends the UNT message.

Only one instrument in a GPIB system can be the active talker at any given time.

Listener Function. The L (listener) functions provide an instrument with the ability to receive interface and device-dependent messages from the GPIB. The L function becomes active when a listener receives a one-byte primary address code, MLA. Any number of devices on the bus may be enabled as listeners at one time.

- ATN - The L function receives device-dependent data when the ATN line is high, if the device has been previously addressed as a listener. When ATN is low, the device monitors the bus for interface messages.

- IFC - The T function becomes unaddressed when the IFC line is pulsed for at least 100 microseconds.
- MLA - The L function becomes addressed after receiving the MLA message.
- MTA - If a device contains both talker and listener functions, the listener function is usually unaddressed when the MTA message is sent.
- UNL - All listeners are unaddressed when the UNL message is sent.

Service Request Function. The Service Request function provides a device with the ability to asynchronously interrupt the controller. The device requests service by sending the SRQ message. Typical reasons for requesting service include reporting the results of an operation or reporting an error.

Several devices can request service at the same time. Since the SRQ line is driven by open-collector devices, the line represents the logical OR of all pending SRQ messages.

The controller may choose to respond or not respond to an SRQ message, depending upon the specific controller program. When a controller responds to the SRQ message, the controller performs a *serial poll* sequence (described later in this section) to service interrupting devices.

Remote-Local Function. The RL function enables an instrument to select between two sources of input information. The function indicates to the instrument that its internal operation should be controlled from the front panel (Local), or over the GPIB (Remote).

- REN - The Remote Enable line indicates that instruments must go to the Remote state if the controller addresses them to listen. When the REN line goes high, all instruments return to the Local state.
- MLA - The device's listen address. After a device is listen-addressed, Remote-to-Local transitions are generated.
- GTL - Go To Local returns an addressed listener to the Local state, even if the REN line is still low.
- LLO - The Local Lockout message tells all instruments to ignore the operator control that usually forces the instrument into the Local state. If an instrument is in the Local state when it receives LLO, the effects of the LLO message begin once the instrument goes to the Remote state.

Parallel Poll Function. Parallel Poll capability allows an instrument to send one bit of status information to the controller without being previously addressed to talk. The parallel poll function will allow up to 8 instruments to be polled by the controller simultaneously. (If the Data Bus lines are shared, up to 15 instruments can be polled simultaneously.)

The 1240 does not support the Parallel Poll function.

Device Clear Function. The DC function enables the controller to put a device into a predefined state.

There are two interface messages associated with the DC function:

- DCL - When the controller sends the Device Clear message, the DC function is activated in all instruments that implement the function.
- SDC - When the controller sends the Selected Device Clear message, the DC function is activated in all addressed listeners that implement the function.

The IEEE 488 standard does not specify the settings an instrument goes to as a result of receiving the DCL or SDC commands. Tektronix instruments use these messages only to clear the GPIB communication circuits within an instrument. Instruments from other manufacturers often reset to a power-up state after receiving these messages.

Device Trigger Function. The DT function allows the controller to have a device execute a basic operation. The particular operation is device-dependent. The GET message activates the DT function within an addressed listener. If several devices have been addressed as listeners, the operation of the system can be synchronized with the GET command.

Once an instrument starts operating in response to the GET message, the instrument will not respond to subsequent triggers until after the current operation is complete. The duration of an instrument's operation thus determines how fast an instrument can be repeatedly triggered by commands from the bus.

Controller Function. The controller-in-charge may relinquish control to any other instrument in the system with controller capability. (The 1240 does not have controller capability.)

TYPICAL ACTIVITY

Remote to Local Changes

Suppose your system consists of a controller and three instruments. The instruments have addresses of 03, 09, and 12.

When the system is powered up, all instruments are in the Local state, and may be adjusted by the operator. When the operator starts the controller's program running, the controller asserts the REN line (causing all instruments to go to the Remote state when they are listen-addressed). The controller may then address each instrument and send data.

During program operation, Devices 03 and 09 return to Local after operating in the Remote state. The following example shows two possible ways for the return-to-local transition to occur:

1. The operator pushes Device 03's "Return-to-Local" button. When this occurs, the instrument obeys commands from (or displays readings on) its front panel, and ignores any Remote commands on the GPIB.
2. The system program requires an operator to intervene at some point, and sends the GTL message to Device 09. This message has the same effect as pushing the "Return-to-Local" button. The GTL message only affects instruments that have first been addressed to listen.

In the next case, the system program addresses the instruments, putting them in the Remote state. The controller then issues the LLO command, causing all devices in the system to invoke Local lock-out. The results of this command can be seen after the system program has performed several tasks. When an operator pushes a device's Return-to-Local button, nothing happens. The operator cannot manually return an instrument to the Local state for front panel operations. All instruments remain in the Remote state, and the system program continues its tasks. As this example shows, the LLO message is used to prevent undesirable alterations of instrument settings by manual operation.

Serial Polls

The poll may be initiated at any time by the system program, or it may occur in response to the SRQ (service request) message.

The controller first unlistens all instruments to clear the bus, and then issues the Serial Poll Enable message. The SPE message tells all devices with serial poll capability to place their status byte on the bus when the controller addresses them to talk. The controller then addresses each device to talk in turn, and the device responds with its status byte. After the poll the controller issues the SPD (Serial Poll Disable) message, and optionally issues UNL (Unlisten) and UNT (Untalk) as a safeguard that the bus will be clear for subsequent activity.

Depending on the controller type and the system program, the controller may modify this sequence. The controller typically polls instruments until SRQ is no longer asserted. Controllers generally examine the RQS bit of each device's status byte to determine if the device was requesting service (asserting the SRQ line). Individual service requests may be handled either during or after the poll occurs, depending on the controller and the system program.

OPERATIONAL ELEMENTS

Device-dependent messages are used in system programs to operate the various devices. Device-dependent messages are sent over the GPIB data bus with the ATN line unasserted (high).

For a list of the device-dependent messages supported by the 1240, see *Section 2*.

INDEX

Page

Aborting a Data Transfer 2-13
Accessories 1-1
ACQMEM 2-2, 3-2
ACQMEM? message 2-2, 3-5
ASCII Hex data block format 2-9
ATN interface message 2-15, E-5, E-9, E-14

BELL message 2-2
Binary Block data block format, (Standard, or IEEE 728) 2-9

Compressed Memory Image 2-11
Controllers 1-1, 1-2, E-5

DAB interface message 2-16, 2-17
DAC interface message 2-16, 2-17, E-11
Data Block Format, ASCII HEX and Binary Block 2-9
DATAFMT message 2-2
DATAFMT? message 2-3
Data Transfer, Interrupting 2-14
DAV interface message 2-16, 2-17, E-8, E-11
DCL interface message 2-14, 2-16, E-9, E-13
Device Address 1-5, E-1
DIAG? message 2-3
Directory, ROM or RAM Pack 4-2
DISPLAY message 2-3
Display code, 1240 . . . . . 2-3, B-1
Downloading . . . . . 2-9, example of, 3-5
DT message 2-4
DT? message 2-4

END interface message 2-16, 2-17, E-11
ERR? message 2-4
Error Messages . . . . . 1-5, Appendix D
Event Codes 3-4, 3-6, D-1
EVENT? message 1-5, 2-4, 3-6

GET interface message 2-16, E-13
GTL interface message 2-16

Header, ROM or RAM Pack 4-1
HELP? message 2-4

ID? message 2-5
IEEE Standard 1-1, 1-2, 1-4, E-1
IFC interface message 2-16, E-11
INIT message 2-5
INSETUP message 2-5
INSETUP? message 2-5, 3-1, 3-5

INDEX (Cont.)

Page

Installation, about 1-2, E-1
INSTRUMENT SETUP 4-3, A-1
Interface functions 2-15, E-5, E-10
Interrupting a Data Transfer 2-14
Key Codes C-1
KEY message 2-6, 3-3, C-1
KEY? message 2-6, 3-3, C-1
LEDs 1-6
LLO interface message 2-16, E-9
LLO (Local Lockout state) 2-14
LOAD message 2-6, 3-2
Local commands state 1-4, 1-5, 2-1, 3-1, 3-2, 3-5, 3-8, E-10, E-12

MEMORY IMAGE 4-3, A-15
Menu, COMM PORT CONTROL 1-3, 1-4, 1-5
Menu, Remote 1-3, 1-4, 1-5
Message Termination 1-4, 1-5
MLA interface message 2-16, E-11
MTA interface message 2-16, E-11

Online, Offline Status . . . . . 1-4, 1-5, status bytes 1-7

Port Status Display 1-4

RADIX Tables 4-3, A-22
RAM Packs 1-1, 3-8, 4-1
RAMPACK message 2-7
RAMPACK? message 2-7, 3-8
REFMEM message 2-7, 3-2
REFMEM? message 2-7, 3-2, 3-6
Remote commands, state 1-5, 2-1, E-12
Remote menu 1-5, 3-1
REN interface message 2-16, E-5, E-10
RFD interface message 2-16, E-11
ROM Packs 4-1
RPHelp 2-7
RQS interface message 2-17
RQS message 2-8
RQS? message 2-8

SDC interface message 2-16, E-13
Serial Polls 3-4, E-11, E-12, E-14
Service Requests Port Status Display of, ... 1-4, 1-5
SET? message 2-8
Soft Keys, 1240 . . . . . 1-4, 1-5, 3-5
Soft Key codes C-2
SPD interface message 2-17, E-11
SPE interface message 2-17, E-11
Specifications, instrument operating 1-2
SRQ interface message 2-17, E-5, E-10
START key, code C-1,
START message 2-8, 3-2, 3-7

INDEX (Cont.)

Page

Status Byte 1-6, 3-4, E-10
STB interface message 2-17, E-11
STOP key 2-8, 3-3, 3-7
STOP message 2-8

TEST message 2-9
Test Complete SRQ 2-9
Trailer, ROM or RAM Pack 4-4
Tri-State 2-15

UNL interface message 2-17, E-8, E-12
UNT interface message 2-17, E-8, E-11
Uploading 1-5, 2-10, 3-5
UTILITY key, 1240 ..... 1-4, code C-1