IRMA™

Documentation

General Introduction

Terminal Emulator User's Guide

The IRMA product consists of three major components: a Decision Support Interface board named IRMA, software on an IBM Personal Computer compatible floppy disk, and documentation. The documentation is divided into three sections which are: *The Terminal Emulator User's Guide, BASICA Subroutines,* and *Technical Reference.* These sections are formatted for insertion into the IBM supplied ring binders.

The varying needs of different types of users are met by these three sections of documentation. The non-programming user will find all necessary instructions for simple terminal emulation in the *Terminal Emulator* manual. Programmers will find documentation for automatic data transfer and custom application software development in the *BASICA Subroutines* documentation. The *Technical Reference* documentation is intended to provide the necessary additional information that an assembly language programmer would need to access IRMA for complex program development.

The *Terminal Emulator User's Guide* is designed to aid the general user in the operation of the IBM PC as an IBM 3278-2 terminal. It includes step-by-step intructions for operation, a generalized explanation of the operational theory, and a list of the commands and functions that are available. The emulator provides all users with the features of a 3278 model 2 terminal. All keys found on a 3278-2 are available on the PC when using the emulator; however, there are some key position changes due to the format differences of the two keyboards. Normal 3278 screen displays are handled by the emulator including several cursor types, underline, blink, dim characters, and status line. Only the status line will appear different than that of the 3278 because the PC character set lacks some of the special status characters. The emulator offers some additional features, such as attribute display and null field character display which are not available on the 3278 terminals.

If the IBM PC has the color terminal adapter, several additional features become available: 3279 model 2A features, color character, extended field attributes, and extended character attributes are then supported. Simulated color is also supported where protected, unprotected, bright, and dim fields become different colors.

The emulator program is supplied in two forms: executable code and source code, so that the emulator may be used as supplied or custom modified as desired.

All IRMA users and programmers need to be familiar with the Terminal Emulator and its documentation. The *Terminal Emulator User's Guide* is to be inserted into the IBM *DOS* manual.

For additional information refer to:

IBM Personal Computer DOS manual

BASICA Subroutines

The *BASICA Subroutines* section consists of a group of routines which provide keystroke and field access from a BASICA program to the 3270 controller. When processing data to or from the controller, a variety of translations must take place. While the programmer may write this necessary code, the TAC supplied subroutines eliminate this task.

The user documentation for the BASICA subroutine consists of a description of each routine, its entry statement number, the variables in which the programmer should supply input arguments, and the variables which are updates for the user. There are two BASICA programs provided as examples. One program provides a demonstration routine and the other provides an example of a data transfer program.

Any programmer intent on developing a program to do automatic data transfer must become familiar with this manual and its contents. The *BASICA Subroutines* documentation is to be inserted into the IBM supplied *BASIC* manual.

For additional information, refer to:

IBM Personal Computer BASIC manual

Technical Reference

The *Technical Reference* documentation describes in detail the interface specifications for IRMA. Included are the necessary specifications required to handle the IBM 3270 protocol. Detailed descriptions of the commands which access and pass data between IRMA and the 3270 controller are provided. Also included in this section are the Key Scan codes for the 3278-2 terminal, definitions and method of handling the Attribute and Extended Attribute bytes, and installation of the IRMA board into the IBM Personal Computer.

This portion of the documentation is designed to aid the assembly language programmer in the development of specialized software. The *Technical Reference* documentation is to be inserted into the IBM supplied *Technical Reference* manual.

For additional information, refer to:

IBM Personal Computer DOS manual

IBM Personal Computer Technical Reference

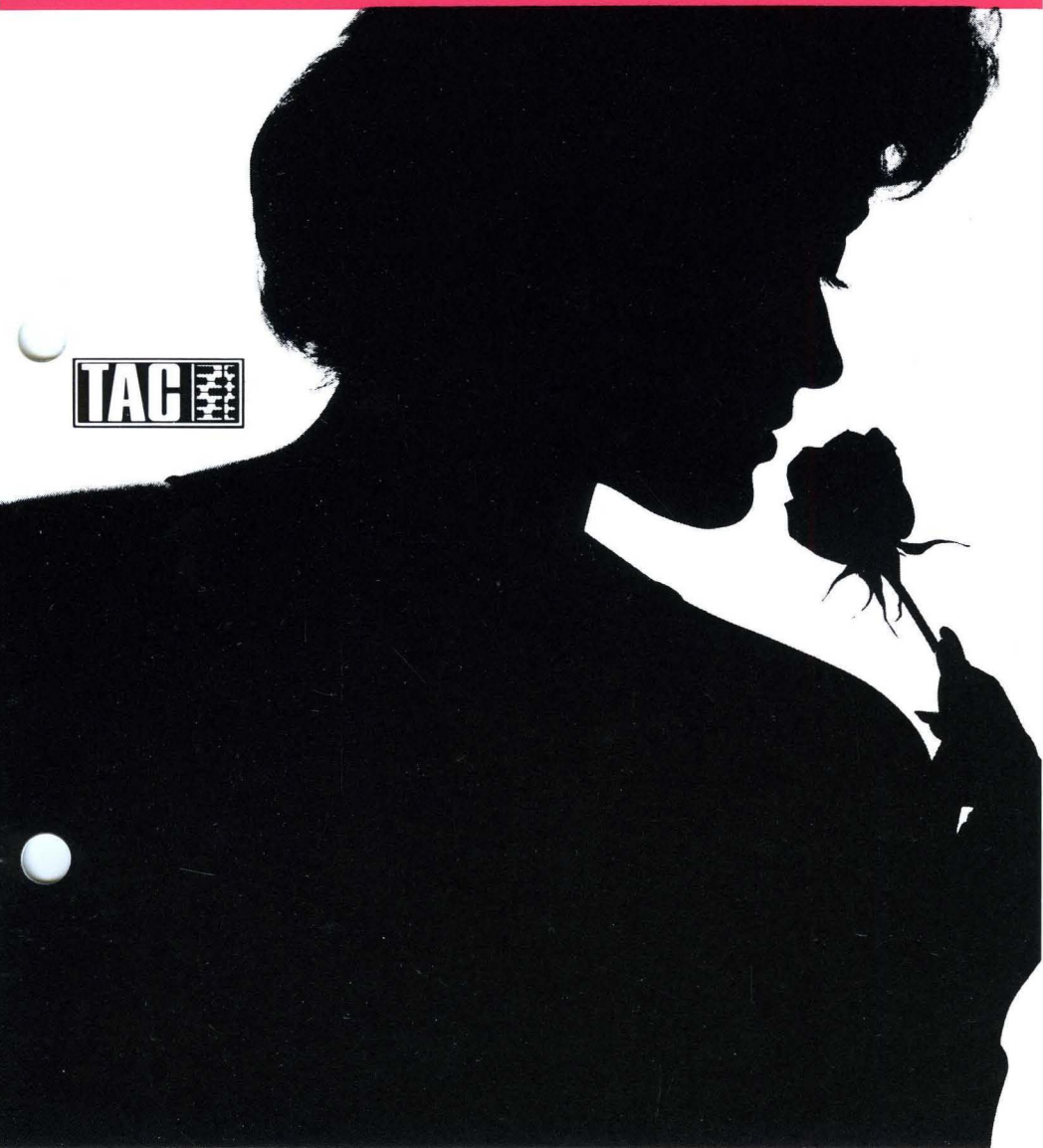
IBM Personal Computer Macro Assembler

IBM GA27-2849 3270 Information Display System Configuration

IBM GA23-0061 3274 Control Unit and Programmer's Guide

IRMA™

Terminal Emulator User's Guide



IRMA TERMINAL EMULATOR

Contents

Document Number: 642-002160-01

Printed in U.S.A.

Introduction

An integral part of IRMA, the Decision Support Interface (DSI), is the terminal emulator program. This program makes it possible for the IBM Personal Computer to emulate an IBM 3278-2 terminal. This allows a PC to serve two functions, as a stand-alone microcomputer and as part of a 327x network accessing the full computing power and data base of the host computer.

IRMA attaches by a coaxial cable to 3274, 3276 or integral type "A" terminal controllers. The DSI is completely compatible with the protocol used by the 327x controller; it functions independently of the Personal Computer's 8088 microprocessor. The programs and operating system in use by the controller are completely accessible when using the PC as a 3278-2 terminal. When the terminal emulator program is active, the PC screen will contain 25 lines of 80 characters with the 25th as the system prompt line. The DSI buffers a full 1920 (80 X 24) characters, just as the 3278-2 does.

The IRMA board operates without dependency upon any program which may be running on the PC. There is no PC software required to handle the 327x system protocol; that protocol is handled completely by IRMA. As soon as IRMA is installed and receiving power, the 327x system protocol is being accepted. When the terminal emulator program is activated, the CRT displays the last screen transmitted over the coax. In other words, the IRMA board, as long as it is powered up, maintains communication with the 327x controller without regard to the operational mode of the PC. However, in order for the user to see the information which has been received and to send keystrokes to the mainframe host over the coaxial cable, the terminal emulator program must be running in the PC. When the PC is operating in the stand-alone mode, IRMA saves data received from the controller and displays the most current screen when the emulator is re-activated.

Features

1. IRMA enables the PC to emulate a 3278 display with the full 1920 character display and the 80 character status indicator line.

2. IRMA keeps a complete screen buffer in memory. This enables the user to alternate between the 3278-2 mode and the stand-alone mode.

3. IRMA's screen buffer is accessible from user programs. This feature allows the user to retrieve data from the mainframe and return the modified screen.

4. Data from the mainframe can be transferred to the diskette or to a printer.

5. Diagnostics are on-board.

6. IRMA is designed for business people. There is no need to learn Assembly Language. Screen Print and Screen Save functions are provided to simplify capturing data.

7. IRMA supports Attribute Characters and Extended Attribute Characters for the field oriented screens.

8. The keyboard of the PC is redefined to functionally correspond to the 3278-2 typewriter type keyboard.

Operational Theory

In order for the PC to perform as a 3278-2 terminal, the functions of the PC keyboard must be redefined to meet the specifications of the 3278-2 terminal. The theory behind the emulator is relatively simple. In the stand-alone mode, each key of the IBM Personal Computer generates a particular code sequence, referred to as BIOS (Basic Input/Output System) key codes. When a key is pressed, the character is translated into the BIOS key code which is processed by the PC's CPU. After the character has been processed, the screen buffer is updated and the new screen is displayed.

When the PC is used as a 3278-2 terminal, the keystrokes are translated twice. For the 327x controller to understand the characters sent from the PC keyboard, the actual keystrokes are first translated into the PC BIOS by the PC's CPU. The emulator program then converts the BIOS codes into the key positions and key scan codes of the 3278-2 terminal. IRMA sends the 3278-2 scan code to the controller where it is processed. The controller then modifies IRMA's screen buffer accordingly and the displayed screen is updated.

In 327x operations, the displayed data is organized into fields. This simplifies data entry for an operator. The key functions that involve printing the display and storage or transmission of data are all field oriented. The characters that define the type of data to be entered in a particular field are called attribute characters. When an attribute character is encountered, all data following that character is considered part of the field. IRMA supports the use of Attribute Characters and Extended Attribute Characters (EAB).

Attribute Characters define the following:

1. The start of a field,

2. Whether a field is protected or unprotected (A protected field cannot be modified by the operator. An unprotected field allows for the entry of data.),

3. Whether an input field (unprotected) will accept alphabetic or numeric or both types of data,

4. Whether the current field is to be displayed, not displayed, or intensified,

5. Whether the fields are to be detectable by a light pen,

6. Whether tab stops will correspond to the first character of an unprotected field (auto skip), and

7. Type of Modified Data Tag (MDT) (The controller searches, using bit 9, for modified fields. If the field has been modified, bit 9 will be set to one and the controller updates that field accordingly.).

Extended Attribute Characters define the following:

1. Character type (normal, blink, reverse video, or underlined),

2. Character color, and

3. Character set.

Additional information on the handling of Attribute Characters can be found in the *IRMA Technical Reference.*

The Attribute Characters are normally displayed as blanks. They serve as a signal to the controller and the display that a particular type of field is to follow. If the attribute byte defines a field to be both protected and numeric, the cursor is positioned automatically by the controller to the next unprotected field (auto skip). The user may display the attribute characters by entering Functions F1, F2, or F3. (Function descriptions are found subsequently in the Section on Commands and Functions.)

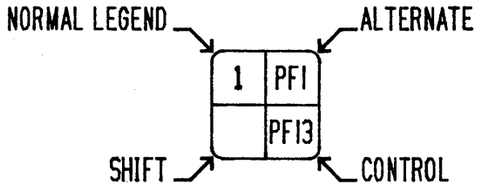
The DSI supports all the programming definitions for the attribute characters and the fields which they define. The keyboard operations, such as CLEAR or ENTER, function in the same manner as with the 3278-2 terminals. While the emulator program handles the translation of the scan codes, IRMA's 8X305 microprocessor handles the 327x controller protocol, such as the actual transfer of data, handshaking, and screen buffer maintenance. The 8X305 acts as a supervisor, assigning the data entered to its proper position and function. It then sends this ordered data to the controller and, on the return, it sends the response to the proper source. (For a complete technical explanation of how the 8X305 handles the protocol and handshaking techniques, refer to the IRMA insert in the PC manual, *IBM Technical Reference.)*

Keyboard

The chart shown here indicates which PC keys perform particular 3278-2 functions.

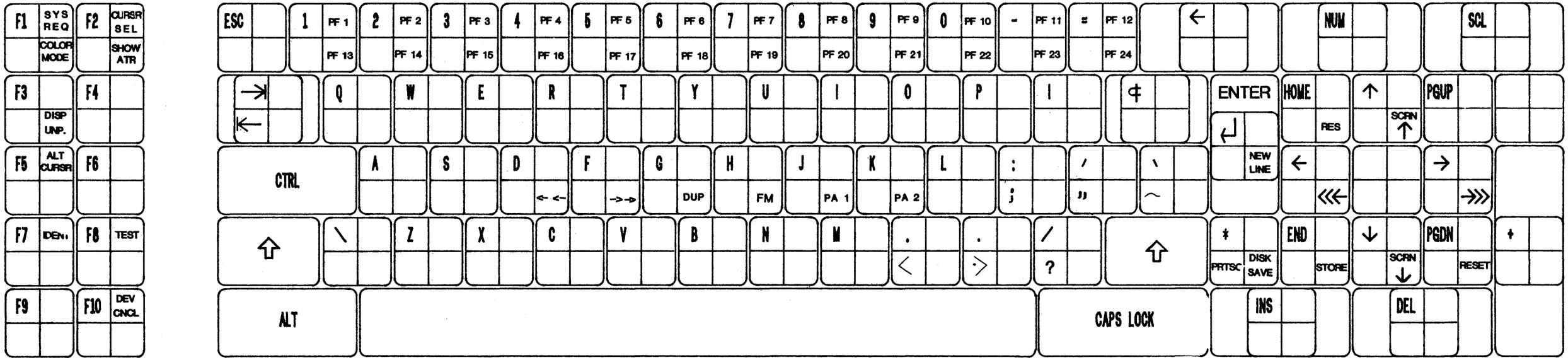
To apply the decals:

1. Refer to the following keyboard chart for the location of each key.
2. Find a sharp tipped item such as a small knife blade, nail file, toothpick, or similar object.
3. Choose a starting point.
4. With the tip of the tool, carefully lift the decal by its top edge and position onto the appropriate key.
5. Smooth the decal onto the key with your fingertip or a smooth blunt object. Avoid touching the adhesive with your hands.
6. Repeat the process for each remaining key. Be sure to match the keyboard chart exactly.



IBM PC KEYBOARD

dca
Digital Communications Associates, Inc.



(FOR SOFTWARE REVISION 1.20 & ABOVE)

How To Use The Terminal Emulator

The first step in converting the PC to a 3278-2 terminal is to attach a coaxial cable from the system controller (3274, 3276, or integral type A) to the BNC connector supplied on the IRMA board. This port must be "sysgen'ed" for a 3278-2 type terminal with a typewriter keyboard. Since the diskette supplied is not a bootable diskette, the operating system must be booted prior to running the emulator program. At this point it is essential, if you have not already done so, to make a duplicate copy of the diskette. Use the duplicate copy and not the original for daily use. Store the original according to the instructions on the protective sleeve. Next, insert the duplicate diskette into either drive. To activate the emulator program, enter:

A:E78< filename > or B:E78 < filename >

depending upon the disk drive into which you inserted the diskette. The filename is optional. It is used to specify a file in which to store 'Screen Saves'. When IRMA is acknowledged by the system controller, the screen will exhibit the same system information that a 3278-2 terminal displays.

After the emulator has been activated, the PC terminal may be used exactly as any other 3278-2 terminal, making it possible for you to access system programs, create new files, or any other function that is part of the 327x network. It is also possible for the user to switch back and forth between the 3278-2 mode and the PC stand-alone mode. A simple press of both shift keys will exit the emulator mode. When you exit the emulator mode, the current screen is saved in memory; this screen is redisplayed when the emulator program is reactivated. To re-enter the emulator mode, type E78 in response to the PC screen prompt and the saved screen will be redisplayed. If the user wishes, the emulator program may become "resident" by entering CONTROL-HOME.

(The PC must be equipped with at least 96K of memory.) Making the emulator "resident" allows the user to exit and re-enter the emulator with greater speed. Both exiting and re-entering the resident emulator are accomplished by pressing both shift keys simultaneously. If using the non-resident emulator, the user must re-activate the emulator by entering the E78 command.

There is a limitation to using the resident emulator which is imposed by the PC's operating system; the Disk Operating System (DOS) does not provide the disk I/O required to save screens from a resident program. The user can have a copy of both forms of the emulator active simultaneously to compensate for this limitation. Refer to the CONTROL & PRINT SCREEN function in the following section for additional information on the uses of the resident and non-resident emulators.

Procedures For Using The Terminal Emulator

1. Install IRMA into the PC. Follow the instructions given in the IBM manual, *Guide to Operations,* Section 5.

2. Be sure that the port to be used is sysgen'ed for a 3278-2 terminal with a typewriter keyboard.

3. Attach the coaxial cable to the BNC connector provided on the IRMA board.

4. Boot the PC operating system.

5. Make a duplicate copy of the IRMA diskette.

6. Insert duplicate diskette into either drive A or B.

7. Enter E78 to activate the emulator program.

8. Enter CONTROL & HOME to make a "resident" copy of the emulator program, providing enough memory (96K) is available.

Commands And Functions

E78	Activates terminal emulator program.

For each of the following functions, both keys listed must be pressed simultaneously.

SHIFT & SHIFT	Exits either the non-resident or resident emulator. This command is also used to access the resident emulator.
CONTROL & HOME	Creates the resident emulator.
CONTROL & PgDn	Causes the system to reset, as if it has just been powered up. The equivalent function on the 3278-2 terminal is the test switch located to the right of the display screen.
CONTROL & 4	≪-- moves the cursor to the left two character positions. This function is also provided as the ALT of 'D'.
CONTROL & 6	--≫ moves the cursor to the right two character positions. This function is also provided as the ALT of 'F'.

CONTROL & F1

Assigns attribute color code for display provided a color display adapter is in use with the PC. There are three levels of color codes. When CONTROL & F1 are first entered, the screen comes up with a black background and white characters. No color or underscoring is attempted. If CONTROL & F1 are entered a second time, monochrome application programs are displayed in the following color scheme:

Unprotected	alpha	dim	white
Unprotected	alpha	bright	red
Unprotected	numeric	dim	yellow
Unprotected	numeric	bright	red
Protected	alpha	dim	green
Protected	alpha	bright	cyan
Protected	numeric	dim	blue
Protected	numeric	bright	blue

If using a 3279 terminal, CONTROL & F1 can be entered a third time to make use of 3279 type color codes. Use of color modes (1,2) with monochrome displays attached to a color display adapter may result in unreadable screens. This situtation is not harmful, just not useful.

CONTROL & F2

Attributes (display buffer codes 0C0H - 0FFH) are displayed as ASCII characters 040H - 07FH.

For example, Attribute 0C0H is displayed as "@". The following chart indicates the displayed characters for each Attribute.

Conversion of Attribute characters to Display symbols

Attr.	Char	Attr.	Char	Attr.	Char	Attr.	Char
0C0H	@	0D0H	P	0E0H	'	0F0H	p
0C1H	A	0D1H	Q	0E1H	a	0F1H	q
0C2H	B	0D2H	R	0E2H	b	0F2H	r
0C3H	C	0D3H	S	0E3H	c	0F3H	s
0C4H	D	0D4H	T	0E4H	d	0F4H	t
0C5H	E	0D5H	U	0E5H	e	0F5H	u
0C6H	F	0D6H	V	0E6H	f	0F6H	v
0C7H	G	0D7H	W	0E7H	g	0F7H	w
0C8H	H	0D8H	X	0E8H	h	0F8H	x
0C9H	I	0D9H	Y	0E9H	i	0F9H	y
0CAH	J	0DAH	Z	0EAH	j	0FAH	z
0CBH	K	0DBH	[	0EBH	k	0FBH	{
0CCH	L	0DCH	\	0ECH	l	0FCH	|
0CDH	M	0DDH	]	0EDH	m	0FDH	}
0CEH	N	0DEH	^	0EEH	n	0FEH	~
0CFH	O	0DFH	_	0EFH	o	0FFH	<

CONTROL & F3

Places dots in unprotected null fields.

SHIFT & PRINT SCREEN	Prints the current screen on the local printer (for both resident and non-resident)
CONTROL & PRINT SCREEN	Copies current screen to diskette. In order to copy to the diskette, you must have specified a filename after the E78 command. (This command works only with the non-resident emulator. If using the resident emulator, you must exit and enter the E78 filename command for a copy of the non-resident emulator. You may have both versions active simultaneouly. After typing E78 the last screen will be displayed and you can then save it on the diskette by entering the CONTROL & PRINT SCREEN command. Pressing both shift keys will exit non-resident copy; pressing both shift keys again will place you back in the resident emulator with the same screen you had upon exit.)

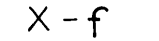
System Message Characters

On any 3278-2 terminal the 25th line is reserved for system messages, for example, the system error codes. Not all of the characters which make up the symbols are available on the PC. The following chart indicates the 3278-2 symbol, the corresponding PC symbol, and where it has been defined, the message that each one represents.

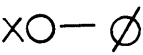
3278-2	PC	Message
	P	Program
	S	Used with X - to indicate symbol keyed is not available
	a	
		Insert mode
	b	
	6	Online to a 3276 controller
		Used in combination with other symbols
		Used in combination with other symbols
		Used in combination with other symbols
		Used to indicate 'not working'
		Shift lock
		Operator
	B	Type B controller
		Alpha lock

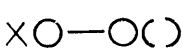
3278-2	PC	Message
		Unknown response
		Operator's program
		Used with X and nn to indicate a communication link error.
		Used in combination with other symbols.
		Used in combination to form 
		Used in combination to form other symbols.
		Left half of clock
		Right half of clock
		System message
		Used in combination to form other symbols.
		Used in combination to form other symbols.
		Right portion of Printer failure message
		Left half of security key
		Right half of security key
		Online to a 3274 controller
		Online to a type A controller
		Symbol for card
		System operator

Combination Symbols


Function unavailable


Security key off

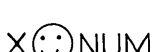

Printer not working

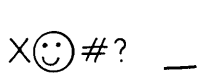

Printer busy

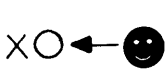

Operator unauthorized for specified printer (Must RESET keyboard)

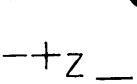

Go elsewhere, action has been attempted which is invalid for field.

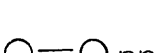

Operator entered too much data into field

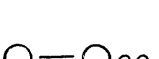

Number lock installed (Must RESET keyboard)


Operator entered invalid number in field


Message received from system operator and rejected (Must RESET)

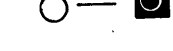

Communication link producing errors


Printer assignment


Printer IDENT has been changed

Printer active


Printer failure


Assign printer

Summary

This section has explained the theory and procedure for using IRMA as a 3278-2 terminal. In the additional documentation supplied with IRMA, you will find the technical information necessary to write data handling programs, to alter the key positions, and to more fully understand the technical details of IRMA.

The *Technical Reference* insert contains the key scan codes, the method by which data is accessed, the component definitions, and the commands used to pass and access data.

The *BASICA Subroutines* documentation lists a description of each of the routines supplied with IRMA. These routines have been designed to handle the more complicated functions required for automatic data transfer via IRMA. For example, they can be inserted into a user-written program to read, modify, or write unprotected fields. The sources for these routines are provided on the diskette as part of the total software package.

IRMA™

BASICA Subroutines

IRMA BASICA SUBROUTINES

Contents

IRMA APPENDIX A

Contents

Document Number: 642-002160-01

Introduction

The *Terminal Emulator User's Guide* has shown how screens of data from the IBM mainframe host computer can be captured in the IRMA screen buffer and displayed by running the Terminal Emulator program. That same Guide also outlined how the PC accepts keystrokes and prepares them for transmission to the host computer which updates the screen buffer accordingly.

The IRMA screen buffer may be accessed by user-written programs as well. Since individual needs are quite different, no single comprehensive PC program is appropriate. There are, however, predictable data handling routines which have been formulated into subroutines for use in BASICA programs.

These BASICA Subroutines provide the foundation for automatic data transfer via IRMA. A programmer can access the appropriate routine to handle many of the more complicated functions of passing data to and from the host. The subroutines offer the means to generate keystrokes and read and write screens from within a user program. These routines also serve as a model for a programmer to write programs which will access IRMA in PASCAL, FORTRAN, and other languages.

Features

1. The IRMA subroutines can be used with BASIC, BASICA, and the IBM BASIC complier.
2. IRMA's subroutines provide access to screen data by row/ column or field number.
3. The subroutines can read, modify, or write unprotected fields.
4. All code conversions between ASCII and 327x display buffer codes are performed automatically.
5. The full ASCII character set as defined in the *IBM Technical Reference* is supported.
6. IRMA's routines check the data type when modifying screen fields.
7. The character translation tables are easily modified to support unique customer applications.
8. Fully commented sources are provided for detailed examination or maintenance. [IRMASUBS.LST]
9. A sparsely commented source is provided for reduced storage requirements in large user applications. [IRMASUBS.BAS]
10. Any screen information, including Attributes and Extended Attribute Buffer (EAB), can be accessed.
11. Easy access to IRMA's status information, including alarm status, cursor position, and line sync indicator, is provided.
12. IRMA subroutines also provide the capability to trigger user application program action on screen updates from the host computer.

Subroutines

Before any use can be made of the subroutines, an initialization call must be made. This call causes all of the parameters to be set to their starting values. The user must make the initialization call prior to using any other routine. Include at the beginning of your program:

GOSUB 50000

Each of the routines are represented in two ways in the following documentation: Name and Statement number. BASICA can only call routines by statement number, but remembering the title of each routine will make its use easier. Note that each routine begins with a REMark statement which includes the proper name.

The description of each subroutine includes those variables to which the user must assign values before calling a routine (Input) and those variables which are assigned values by the routine (Output). All input variables MUST be given values by the user. Reserved variables are listed at the end of this document.

Included as an appendix to this document is a sample BASICA program written to exemplify the implementation of each subroutine. It may be used as a guideline for writing a data handling program to fit your specific needs.

KEYS
Keystroke Send

Statement Number: 50600

Routine Name: KEYS

Input Variables:

I.VST$	String to be sent to host

Output Variables:

I.VER%	Error status (device or key timeout)

Remarks: This routine sends strings as keystrokes to the host computer. The string length is confined to BASICA's limit of 255 characters. Note: See Reserved Names section for specification of error status values.

Reference: In Appendix A, the sample program, SAMPLES.BAS lines 600-620, provides an example of sending keystrokes. Line 602 sets the value for the input variable; line 604 calls the Keystroke Send subroutine. This subroutine occupies lines 50600-50660 of the same sample program. Line 612 sets the input variable to a new value and line 614 calls the Keystroke Send subroutine.

FIND
Find Unprotected Field

Statement Number: 50700

Routine Name: **FIND**

Input Variables:

I.VFL%		Field number to be found

Output Variables:

I.VER%		Error status (device timeout, field error)
I.VCB%		Pointer to leading attribute
I.VCE%		Pointer to trailing attribute
I.VFS%		Current field length
I.VRO%		Row address of field data
I.VCL%		Column address of field data
I.BUF%	(0)	Leading attribute character

Remarks: This routine searches IRMA's internal screen buffer for unprotected fields. As it searches, each unprotected field is counted until the specified count is reached. The absolute address of the selected field's unprotected attribute is returned in I.VCB%. The field contents are then scanned, counting characters and searching for another attribute.

FIND
Find Unprotected Field

I.VCE% is left as an absolute pointer to the trailing attribute. I.VCB% and I.VCE% are used internally for the "Find Next" routine and for several field type consistency checks. The following variables SHOULD NOT be modified by the user: I.VCB%, I.VFS%, I.VCE%, I.BUF%. They must retain their values which were assigned by the FIND and FNEXT subroutines, since they provide the link between these subroutines as well as between successive FNEXT subroutine calls.

Reference: In the sample program, SAMPLES.BAS, provided in the Appendix, note lines 200-240. This portion of the sample program finds and prints each field. Line 202 sets the value for I.VFL% to one to find the first field. Line 210 calls the subroutine, FIND. The next portion of the program reads the found field and prints the specified information. Line 240 calls the subroutine which finds the next field.

FNEXT
Find Next Field

Statement Number: 50800

Routine Name: **FNEXT**

Input Variables:

I.VFL%		Field number to be found

Output Variables:

I.VER%		Error status (device timeout, field error)
I.VCB%		Pointer to leading attribute
I.VCE%		Pointer to trailing attribute
I.VFS%		Current field length
I.VRO%		Row address of field data
I.VCL%		Column address of field data
I.BUF%	(0)	Leading attribute character

Remarks: This routine increments I.VFL%, initializes the internal variables; then goes to the FIND routine.

Reference: For an example of usage see lines 200-240 of SAMPLES.BAS listed in the Appendix. Line 230 calls FNEXT. This routine can be used only after FIND has been used, as it depends on the value of I.VFL% specified in FIND.

RDFLD
Read Field Contents

Statement Number: 50900

Routine Name: **RDFLD**

Input Variables: None

Output Variables:

I.VER%		Error status (device timeout)
I.BUF%	( )	PC buffer is filled with field contents. I.BUF%(0) contains the leading attribute character. I.BUF%(1) through I.BUF%(I.VFS%) contain screen data and EAB information. Screen data is contained in the low order byte and EAB information is in the high order byte.

Remarks: RDFLD transfers IRMA's internal screen memory of screen data and EAB contents to an internal array within the PC. FIND and FNEXT must be called to setup buffer pointers before calling RDFLD.

Reference: RDFLD is also used in the lines 200-240 in SAMPLES.BAS.

WRFLD
Write Field

Statement Number: 51000

Routine Name: **WRFLD**

Input Variables:

I.BUF%	( )	Internal screen buffer
I.VCB%		Initial attribute pointer
I.VFS%		Field length

Output Variables:

I.VER%	Error status (device timeout, illegal character)

Remarks: This routine writes the contents from the PC's internal buffer to IRMA's screen memory. WRFLD transfers data to the screen memory with error checking appropriate to the 3270 terminal system. The field is verified to be non-protected and only I.VFS% characters are written. The routine will abort without modifying the screen memory if a non-numeric character is found in a numeric only field. This routine must be preceded by a FIND.

Reference: SAMPLES.BAS, lines 300-345, gives an example of how one might use this subroutine. This portion of the program finds a field; puts a string within that field into the PC's buffer. WRFLD writes the contents of the PC's buffer into IRMA's internal screen memory.

GTSTR
Get String

Statement Number: 51100

Routine Name: GTSTR

Input Variables:

I.BUF%	( )	Internal screen buffer
I.VFS%		Field length
I.VOO%		Offset within field to begin transfer

Output Variables:

I.VER%	Error status (Offset out of bounds)
I.VST	$ASCII data recovered from buffer
I.VOO%	Offset to REMAINDER of field (If field is longer than 254 characters)

Remarks: The GTSTR routine retrieves the field data from the PC's internal buffer and converts the characters to the extended ASCII used by the PC in its display buffer.

Reference: An example of this subroutine can be found in lines 200-240 of the sample program. GTSTR converts IRMA's buffer code to the extended ASCII characer set used by the PC's display buffer. In order to transfer data from one buffer to another this conversion must take place before the string can be read.

PUSTR
Put String

Statement Number: 51200

Routine Name: **PUSTR**

Input Variables:

I.VST$	String to be placed in buffer
I.VOO%	Offset in buffer at which to begin.

Output Variables:

I.VER%	Error status (device timeout)
I.VOO%	Offset to remainder of field
I.BUF%	Screen format buffer

Remarks: PUSTR writes an ASCII string into the PC's internal display buffer. It moves the ASCII string into the PC's internal buffer and converts it to 3270 type buffer codes. This routine should be called prior to a WRFLD in order to place the data to be written in the internal buffer.

Reference: An example of this subroutine is found in lines 300-345 of SAMPLES.BAS. PUSTR is used here in combination with FIND and FNEXT. FIND locates the field and PUSTR translates the ASCII string, I.VST$, into IRMA's buffer code and moves that string to the internal buffer (I.BUF%). WRFLD takes I.BUF% and writes it in the screen buffer.

RDABS
Read Absolute Screen

Statement Number: 51300

Routine Name: **RDABS**

Input Variables:

I.VRO%	Row number of starting character
I.VCL%	Column number of starting character
I.VRR%	Length of area to read

Output Variables:

I.VER%	Error status (device timeout)
I.VRO%	Row position after last character read
I.VCL%	Column position after last character read
I.VST$	ASCII form of screen data
I.VS0$	EAB data

Remarks: This routine moves IRMA's screen memory characters and EAB data into user strings from a given row and column screen position for a given length. Screen characters are translated into ASCII. The EAB data is unmodified.

Reference: The example for this subroutine is found in lines 400-412 in SAMPLES.BAS. The variables are set for the row and column at which to begin the read and the number of columns (characters) to be read before calling the subroutine in line 406. In this particular example the procedure will be performed five times, such that the first 40 characters (I.VRR%) of the top five lines (I.VRO%) of the 3278 screen are displayed. (Value for I.VRO% is determined by the FOR/NEXT loop, lines 404 and 412.)

GTCP
Get Cursor Position

Statement Number: 51400

Routine Name: **GTCP**

Input Variables: None

Output Variables:

I.VER%	Error status (device timeout)
I.VRO%	Cursor row position
I.VCL%	Cursor column position

Remarks: GTCP reads the 3270 screen cursor position from IRMA. It should be noted that the row address may not be within the confines of the normally displayed screen.

Reference: Lines 500-520 of SAMPLES.BAS provides an example of GTCP. This portion of the program retrieves the cursor position and prints the location of the buffer pointers. It also calls a subroutine listed in lines 900-980 which retrieves and displays the state of the main and aux status words.

XPOR
Execute Power-on-reset

Statement Number: 50100

Routine Name: **XPOR**

Input Variables: None

Output Variables: I.VER% Error status (device timeout)

Remarks: The XPOR routine causes the controller to set the terminal as though it has just been powered up. This call can be useful in clearing some controller errors, especially if the coaxial cable has been disconnected.

Reference: This subroutine is not used in the sample program; however, the subroutine is listed in lines 50100-50108. Its use should be limited to particular conditions, such as during data transfer if the controller has sent data which does not appear to be that which was requested. When all other attempts to remedy the problem have been exhausted inserting this subroutine into the program will reset the terminal. The controller will acknowledge the terminal as if it has just been powered-on.

GSTAT
Get Status

Statement Number: 50200

Routine Name: GSTAT

Input Variables: None

Output Variables:

I.VST%	Main device status
I.VAX%	Auxiliary device status
I.VER%	Error code

Remarks: GSTAT reads the current main and aux status from IRMA. Each bit in these status words has a specific meaning. Parameters are provided (See the list at the end of this section) to allow the user to 'and' mask the status with a parameter to test for a specific condition. For example, to test for the buffer modified status (I.MDI% from table), the program would read:

```
GOSUB     50200
IF I.VST% AND I.MDI% THEN GOTO . . .
```

Bit mask parameters for both I.VST% and I.VAX% are provided.

Reference: Lines 700-714 contain an example of this subroutine. The routine reads the status, displays the current status, clears each of the status bits, and then reads and displays the status after the clear.

RSTAT
Reset Status

Statement Number: 50300

Routine Name: RSTAT

Input Variables:

I.VST%	Bits in status word to be reset

Output Variables:

I.VER%	Error status (device timeout)

Remarks: RSTAT resets status bits in the main status word. If a status bit such as I.MPR% (controller reset) is read in a GSTAT, the user program must take any action needed, then reset the bit. This can be done by calling RSTAT with I.VST% as set by GSTAT. Note that any bit set in I.VST% when RSTAT is called will be reset.

Reference: Lines 700-714 include an example of this subroutine. After displaying the status, the status bits are cleared or reset.

STDNM
Set Trigger Data AND Mask

Statement Number: 50400

Routine Name: STDNM

Input Variables:

I.VRO%	Row for trigger test
I.VCL%	Column for trigger test
I.VMS%	Trigger mask value
I.VVL%	Trigger test value

Output Variables:

I.VER%	Error code on exit

Remarks: STDNM allows the program to declare a location on the screen, for a delay until a specific value is written into that location. WTRIG will not return until that value is found in the requested location. The location=value test is made only on those bits which have value of one in the mask. Thus, to make an exact match test, the value of I.VMS% must be set to 255 (decimal) which is all one bits. The condition of I.VMS%=0 is special. If the mask is zero, then any CHANGE in the location requested will result in WTRIG returning. In this case, I.VVL% is unused.

Reference: An example of this subroutine can be found in lines 800-899. Lines 802 and 804 set the input variables. The row and column are set to (one,one) which positions the pointer to the upper left corner. The mask and text values are both set to zero which makes a trigger occur upon any change in row 1, column 1.

WTRIG
Wait Trigger

Statement Number: 50500

Routine Name: **WTRIG**

Input Variables:

I.VTO%	Time constant (in seconds)

Output Variables:

I.VER%	Error code
I.VTO%	Time remaining

Remarks: WTRIG waits until a specific trigger event occurs (see STDNM). The time constant, I.VTO allows the programmer to select how long WTRIG will wait for the event to become true (in seconds).

Reference: This subroutine is used in conjunction with STDNM. The WTRIG routine is called in line 824. It allows a program to wait for a trigger condition specified by STDNM to occur. If I.VTO% returns with a value of zero, no trigger event has occurred and the call to WTRIG has timed out.

Reserved Names

The BASICA subroutines occupy statement numbers from 50000 to 59999. No user statements should be included in this range. The variables, flags, and parameters used by these routines are named so that user routines will not conflict. The following list shows this usage:

Name	Use	Type
	IRMA commands	
I.CRD%	Slave read data	Parameter
I.CWR%	Slave write data	Parameter
I.CAC%	Slave aux status and cursor update	Parameter
I.CCL%	Slave main status bit clear	Parameter
I.CKY%	Slave send keystroke	Parameter
I.CSP%	Slave selector pen strike	Parameter
I.CXP%	Slave execute power-on-reset	Parameter
I.CMD%	Slave load trigger mask and data	Parameter
I.CTA%	Slave load trigger address	Parameter
I.CIM%	Slave set attention request mask	Parameter
	Main status word (I.VST%) bit mask	
I.MAX%	Auxiliary status change	Mask
I.MTG%	Trigger event occurred	Mask
I.MKY%	Slave key scan code buffer empty	Mask
I.MXX% *	Unused, reserved for future use	Mask
I.MPR%	Controller requested reset occurred	Mask
I.MCC%	Last command complete flag	Mask
I.MDI%	IRMA buffer modified by controller	Mask
I.MCM%	Cursor position modified by controller	Mask
	Auxiliary status word (I.VAX%) bit mask	
I.MXX% *	Unused, reserved for future use	Mask
I.MPO%	IRMA polled since last status read	Mask
I.MAL%	Sound alarm request	Mask
I.MDD%	Display disabled (inhibit) request	Mask
I.MCI%	Cursor inhibited	Mask
I.MRC%	Reverse block cursor select	Mask
I.MBC%	Blinking cursor select	Mask
I.MCK%	Keyboard clicker enabled	Mask
	*Known to be duplicates — Reserved	

Reserved Names

Name	Use	Type
	IRMA device code parameters	
I.RG0%	Device code of register 0	Parameters
I.RG1%	Device code of register 1	Parameters
I.RG2%	Device code of register 2	Parameters
I.RG3%	Device code of register 3	Parameters
I.RST%	Handshake slave to start select	Parameters
I.RAK%	Handshake ATTN acknowledge	Parameters
I.RAF%	Handshake flag read select	Parameters
	IRMA handshake flag bit masks	
I.MAT%	IRMA requests System Unit attention	Mask
I.MBS%	IRMA busy with System Unit request	Mask

BASICA internal use temporary variables

I.VT0%	I.VT1%	I.VT2%	I.VT3%	I.VT4%	Temp. variable
I.VT5%			I.VT8%	I.VT9%	Temp. variable
I.VS0$	I.VS1$	I.VS2$	I.VT3$	I.VS4$	Temp. variable
I.VT5$	I.VS6$	I.VS7$	I.VS8$	I.VS9$	Temp. variable

BASICA subroutine I/O variables

I.VER% Error code returned to user — Variable

0 — No error occurred
1 — IRMA did not respond to command
2 — Row number out of range
3 — Column number out of range
4 — Byte value out of range
5 — Invalid field type for operation
6 — Invalid character in NUMERIC only field
7 — Field number out of range
8 — Invalid extended key code
9 — Timeout on key send attempt
10 — Timeout on trigger wait event
11 — Illegal internal buffer pointer
12 — Field or string too long
13 — Found field does not match internal buffer type
14 — Buffer offset out of range of internal buffer
15 — Bad key scan code

Reserved Names

Name	Use	Type
I.VST%	Main status word	Variable
I.VRO%	Screen row number (0-24 [0 is the status line, row 1 starts at the top of the screen])	Variable
I.VCL%	Screen column number (1-80)	Variable
I.VMS%	Trigger mask byte	Variable
I.VVL%	Trigger value byte	Variable
I.VFG%	General Boolean flag	Variable
I.VST$	General string variable	Variable
I.VFL%	Current field length	Variable
I.VAX%	Aux status value	Variable
I.VOO%	Buffer offset pointer for string I/O	Variable
I.VRR%	Raw screen read data length	Variable
I.VCB%	Internal pointer to beginning of field	Variable
I.VCE%	Internal pointer to end of field	Variable
I.VFS%	Internal field length (size)	Variable
I.VTO%	Timeout constant	Variable
I.TAB%	Code conversion tables	Variable
I.BUF%	Screen format buffer	Variable

The parameters above are constants. BASICA does not provide for any parametric declarations, so all of the basic subroutines use one set of variables which are initialized to the correct value. Parameters NEVER change during a program execution.

The temporary variables listed above are used by these subroutines to hold values needed during execution. No data is guaranteed to be left in any of these variables.

The argument-passing variables are used to pass data to and/or from the BASICA subroutines. In the description of each routine, those variables listed as INPUT must be set prior to the GOSUB. Those variables listed as output are updated during routine execution. Note that some variables are both input and output.

THE USER IS WARNED THAT THESE ROUTINES PROVIDE THE PROTECTION NEEDED TO PREVENT THE SENDING OF ILLEGAL DATA TO THE 3270 NETWORK. THIS PROTECTION IS ONLY AVAILABLE IF THE ROUTINES REMAIN UNMODIFIED.

IRMA™

Technical Reference

IRMA TECHNICAL REFERENCE

Contents

APPENDIX B

Contents

Document Number: 642-002160-01

Printed in U.S.A.

Introduction

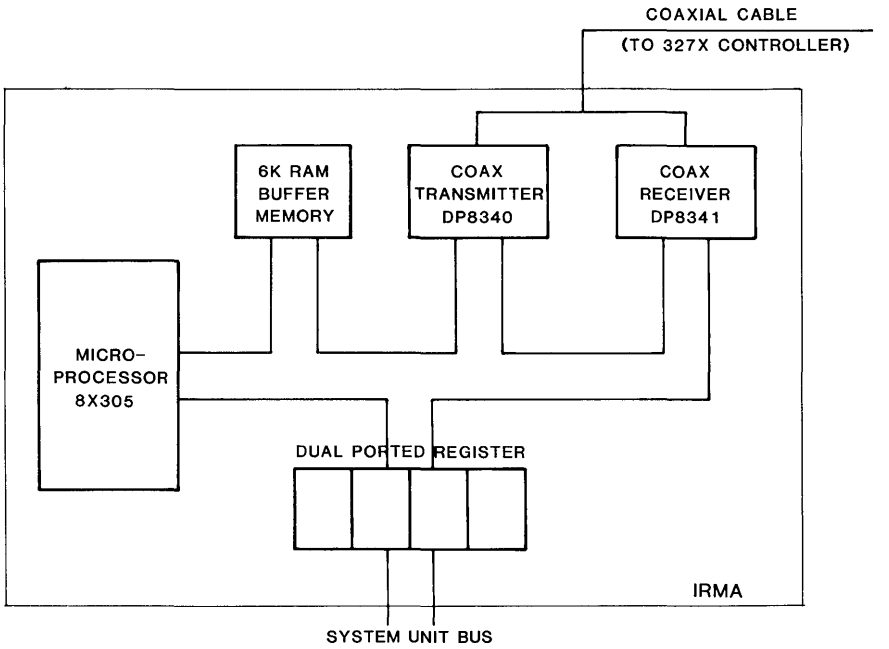
IRMA,™ the Decision Support Interface™, is a printed circuit board which plugs into the IBM Personal Computer System Unit. It can be installed in any slot in the System Unit and provides a back panel BNC connector for attachment by a coaxial cable to either a 3274 controller, 3276 controller, or an integral controller.

IRMA operates in a stand-alone mode, using an on-board microprocessor to handle the 327x protocol and CRT buffer. Whenever power is applied to IRMA, it responds to commands from the controller as if an IBM 3278-2 terminal were attached to the coaxial cable. The CRT buffer is accessed from the PC System Unit as an I/O device, the device codes being 220H to 227H. IRMA does not occupy any of the memory address space.

In order to meet the requirements of the 327x protocol, the Decision Support Interface (DSI) uses high speed microprocessor technology which is independent of the 8088 microprocessor of the System Unit. This allows the user to ignore the timing requirements of 327x, and operate with a buffer of data just as the 3278 CRT does.

When operating, the DSI takes commands from a four byte dual ported register array in addresses 220H to 223H. This array is accessed by I/O commands from the System Unit. The four single byte words are arranged as the command and up to three arguments. Command words allow the System Unit program to read or write bytes in the screen buffer, send keystrokes, and access the special features available on the DSI. This array is also handled on the DSI by the microprocessor which manages the 327x protocol. When the DSI is idle between messages from the controller, any commands left in the array by System Unit programs are processed as required. This processing occurs only when the higher priority 327x communication is idle. This idle state is indicated by a busy/done flag mechanism for both the DSI and System Unit microprocessors. This allows the System Unit to declare that a new command is available to the DSI, and for the DSI to signal the completion of this command.

IRMA BLOCK DIAGRAM



Major Component Definitions

8X305 Microprocessor

The 8X305 microprocessor provides the intelligence to handle the 327x protocol. Polling and answering, data transfer, handshaking, and screen buffer maintenance are performed by this processor.

DP8340 and DP8341 327x Coax Transmitter/Receiver Interface

These two ICs provide the interface from the microprocessor to the 327x coaxial cable. Serialization and deserialization of data take place in these two parts.

Screen Buffer

The DSI contains 6K bytes of fast RAM memory for screen buffers and temporary storage. This is divided into two 2K byte buffers for the CRT screen and the Extended Attribute buffers, leaving 2K bytes for local storage used by the 8X305 processor.

Dual Port Register Array

The four byte dual ported register array is shared by the 8X305 and the System Unit 8088 processors. These registers are used for all communication between the two microprocessors. Data to be transferred from one processor to the other is written into specific locations (addresses) in the array (220H-223H) and the 'Command Request' flag (226H) is set. When the receiving processor has read the register, this flag is cleared. Each processor can test the state of this flag to see if data transfer can begin and to determine when the transfer is complete.

Operation

The dual ported communication array is used to pass commands from the System Unit and its program to the DSI. The registers are organized as four I/O addresses (220H to 223H), using one 8 bit word at each address. The base address of the standard DSI command is 220H. Arguments for the command are placed into addresses 221H, 222H, and 223H. Standard command operation for the DSI requires the user program to set values into the dual port register as appropriate for each command. First, the user program sets the 'Command Request' flag. IRMA then reads the data, performs the operation, and leaves any resulting data in the dual port array. The Command Request flag is cleared at that time.

Ten commands are defined for the DSI program. These commands are given the following values:

Command Code	Command Definition
0	Read buffer data
1	Write buffer data
2	Read status/cursor position
3	Clear main status bits
4	Send keystroke
5	Light pen transmit
6	Execute Power-on-Reset
7	Load trigger data and mask
8	Load trigger address
9	Load attention mask

Programming Considerations

The DSI is accessed as an I/O device on the System Unit bus. It uses eight I/O device codes, 220H through 227H. Device codes 220H, 221H, 222H, and 223H are a dual access register array which is attached to both processors. The four bytes are used to pass commands from the 8088 to the DSI and to return any answers. Device code 224H and 225H are reserved for future use. Device codes 226H and 227H are used for Command Request and Attention Request flags.

All commands use word 0, device code 220H, as a command selection register. To begin a command, the program must set word 0 equal to the command number. The other 3 words are used to pass the arguments of the command. When the specified command is completed, word 0 contains the main status bits and the other three words contain output data.

A flag is provided to allow the user program to check for a command in progress. This flag is set by the 'Command Request' operation, and cleared when the command is finished. Programs must check this flag before modifying the register array. When the flag is clear, the array may be modified and a new command begun. If the program has not cleared the flag, an Invalid Response/Status may be returned to the user program.

Programmer's Notes on Status Bits

Main Status bits indicate specific conditions. The 'Aux Status Change' bit is set anytime the Aux Status changes. The 'Trigger Occurred' bit is set whenever the trigger data match occurs (see Load Trigger Data and Mask command (7)). The 'Key Buffer Empty' bit is set when the key scan code buffer is empty (see Send Keystroke command (4)). The 'Unit Reset' bit is set whenever the controller sends a 3270 reset command. The 'Buffer Modified' bit is set when any buffer write occurs. The 'Cursor Position Set' bit is set whenever the controller positions the terminal's cursor in the buffer.

The 'Aux Status' bits are defined as follows: the 'Unit Polled' bit is set by a poll command. This bit clears after reading. Since the controller polls about 1000 times per second, this status bit will be set often. The other 6 Aux status bits are defined exactly as 3270 protocol defines them.

The 'Command Interrupt Request' bit should not be used to tell when another command can be started. The hardware flag

'Command Request' must be used for this purpose. The 'Command Interrupt Request' bit will clear when a command is begun, and will be set at the end of the command. However, it is not cleared immediately upon 'Command Request'.

The 'Key Buffer Empty' bit, which is used to check the buffer before sending a 'Keystroke' command is only guaranteed valid if no command is in progress. This is because the empty bit does not clear immediately upon the keystroke command. Again, use the hardware flag to check for a command in progress before checking the empty flag bit.

The Main Status in word 0 is updated each time any of the conditions specified occurs. This word can be read and used with the above limitations at any time. The 'Read Status' command is not necessary except to read the 'Aux Status' or 'Cursor Position'.

The returned Main Status byte consists of 8 bit flags as follows:

Bit		Meaning
7	(MSB)	Aux Status change has occurred (*)
6		Trigger Occurred (*)
5		Key Buffer Empty
4		UNUSED
3		Unit Reset by controller (*)
2		Command interrupt request (+)
1		Buffer Modified (*)
0		Cursor Position Set, or search backward (*)

(*) = Bits which must be cleared by user program

(+) = This bit allows the attention request/interrupt request mechanism to be used with commands. Programmed I/O operation should use the hardware flag for all busy/done checking.

(MSB) = Most Significant Bit

The bit flags in the Aux Status are defined as follows:

Bit		Meaning
7	(MSB)	UNUSED
6		Unit Polled since last Status Read
5		Sound Alarm
4		Display Inhibited
3		Cursor Inhibited
2		Reverse Cursor Enabled
1		Cursor Blink Enabled
0		Keyboard Click Enabled

Command Descriptions

Read Buffer Data Command (0)

For a read data command, word 0 is set to zero. Word 1 is the low order 8 bits of the buffer address to be read. Word 2 is the high order 3 bits (right justified) of the address. Word 3 is unused. Upon completion of the command, Word 2 contains the associated Extended Attribute Data and Word 3 contains the data from the specified buffer location:

The internal IRMA screen buffer is 2000 characters long. This corresponds to the 25 lines by 80 characters per line. Even though the screen is displayed with the status line on the bottom, the status line is actually the first line in memory. The starting addresses of each line are listed later in this section.

The Read Data command returns the buffer data, EAB data, and the main status. Each command returns main status in word 0.

Word	Value	Input	Output
0	0	Command to DSI	Main Status
1	ADDR(L)	Address (low) to read	UNUSED
2	ADDR(H)	Address (high) to read	EAB* Data from DSI
3	DATA	UNUSED	Data from DSI

* EAB — Extended Attribute Buffer (Refer to Attribute Character explanations later in this document and in the *Terminal Emulator User's Guide.)*

Write Buffer Data Command (1)

The write buffer command is used to write (modify) the contents of the screen buffer. Like read data, Word 1 and Word 2 contain the address of the buffer location where data is to be written. Word 3 is the data to be written. At command completion, the buffer is updated and main status is returned.

Word	Value	Input	Output
0	1	Command to DSI	Main status
1	ADDR(L)	Address (low) for write	UNUSED
2	ADDR(H)	Address (high) for write	UNUSED
3	DATA	Data for write	UNUSED

Read Status/Cursor Position Command (2)

This command reads the current status and cursor position from the DSI. The status is returned as two bytes of bit flags. The cursor address is in the same format as the buffer address in read/write data commands.

Word	Value	Input	Output
0	2	Command to DSI	Main status
1	ADDR(L)	UNUSED	Cursor address (low)
2	ADDR(H)	UNUSED	Cursor address (high)
3	DATA	UNUSED	Aux Status

Clear Main Status Bits Command (3)

This command clears one or more of the main status bits.

Word	Value	Input	Output
0	3	Command to DSI	Main status
1	UNUSED	UNUSED	UNUSED
2	UNUSED	UNUSED	UNUSED
3	MASK	Bit clear mask	UNUSED

Five of the main status bits are set by specific conditions, but cleared only upon command. This allows each of these bits to be tested and cleared by different sections of the program. The clear mast controls which bits will be cleared. For each 1 bit in the clear mask, the corresponding bit in the status is cleared (i.e. set to 0). The clear mask can be used to reset multiple bits. Note that the returned status reflects the status AFTER the clear command has been executed.

For ease of implementation, word 0 is always maintained as the most current status. The normal program sequence would include the following: a read and test of word 0, a branch on condition to a service routine that is specific to the bits found to be set, and the Clear Main Status Bits command.

Send Keystroke Command (4)

The send keystroke command causes the DSI to send the controller a key scan code. This is the function used to simulate a key active condition. The key scan code is the exact code which a 3278-2 terminal would normally send, NOT an ASCII or EBCDIC character code.

Word	Value	Input	Output
0	4	Command to DSI	Main status
1	UNUSED	UNUSED	UNUSED
2	UNUSED	UNUSED	UNUSED
3	CODE	Key scan code to send	UNUSED

This command causes the Key Buffer Empty flag in the status byte to clear. It also checks the status of the Command Request flag. The Key Buffer Empty flag is guaranteed to be valid when a command is not in progress. The Key Buffer Empty flag must have a value of one before the key scan code can be sent.

Send Selector Pen Location (5)

This command causes the cursor position of the light pen to be sent to the controller. This code is NOT in cursor address format.

Word	Value	Input	Output
0	5	Command to DSI	Main status
1	ROW	Row on screen	UNUSED
2	FIELD ID	Field ID on screen	UNUSED
3	UNUSED	UNUSED	UNUSED

Execute Power-On-Reset Command (6)

This command causes the DSI to appear to the controller as if the terminal has just been reset. This is used to signal the controller that the DSI needs power-up service and initialization.

Word	Value	Input	Output
0	6	Command to DSI	Main status
1	UNUSED	UNUSED	UNUSED
2	UNUSED	UNUSED	UNUSED
3	UNUSED	UNUSED	UNUSED

Load Trigger Data AND Mask Command (7)

This command loads a trigger system with data and mask values. This system is used to put a "watch" on a specific buffer memory location. The watch can be set to check for an exact match on a new value, an inexact (masked) match on a new value, or on any change to the current value. The mask word determines which type of test occurs.

For each 1 bit in the mask, the data and the buffer MUST match for a trigger to occur. To make an exact comapre, the mask is set to OFFH (all ones) and the data is set to the desired value. When a match occurs, the Trigger Occurred bit in the main status will be set. If the mask does not contain all ones, only those bits which are one will be checked for a match for the trigger to occur. For example:

Buffer Value	0	1	0	1	0	1	0	1
Search Value	0	1	0	1	0	1	1	0
Bit by Bit Compare (1's = Difference) (Exclusive OR)	0	0	0	0	0	0	1	1
Mask	1	1	1	1	1	1	1	0
Logical AND of Mask and Compare	0	0	0	0	0	0	1	0

The result is non-zero; the trigger did not occur

The special case of a mask of all zeros is used to handle a test for change of state. At the time the mask is set to 0, the current value of the location in memory is saved by IRMA. This is then compared and any change is reported as a trigger.

Word	Value	Input	Output
0	7	Command to DSI	Main status
1	DATA	Data value for the compare	UNUSED
2	MASK	Mask value for the compare	UNUSED
3	UNUSED	UNUSED	UNUSED

Load Trigger Address Command (8)

The load trigger address command sets the buffer position for the data/mask compare. This address is in the same format as a cursor position and data read/write. After executing the Load Data and Address Trigger commands, the Trigger occurred bit should be cleared. Clear Main Status Bits Command (3) The Load Address command should follow the Load Data command

Word	Value	Input	Output
0	8	Command to DSI	Main status
1	ADDR(L)	Address (low) for checking	UNUSED
2	ADDR(H)	Address (high) for checking	UNUSED
3	UNUSED	UNUSED	UNUSED

Load Attention Mask Command (9)

The attention mask is applied to the regular status word. Any time a bit is SET in the status word, the corresponding bit in the Attention Mask is checked. If this bit is one, the Attention request flag is set. This only applies to bits changing from zero to one.

Word	Value	Input	Output
0	9	Command to DSI	Main status
1	UNUSED	UNUSED	UNUSED
2	UNUSED	UNUSED	UNUSED
3	MASK	Mask for status change	UNUSED

Command Request / Attention Request Flags

There are two flags which allow the 8X305 microprocessor and the System Unit 8088 CPU to handshake over commands. The two flags are Command Request and Attention Request. The 8088can set Command Request, indicating that a new command has been placed in the dual ported memory. The Attention Request flag is set by the 8X305 processor to indicate a status change with Attention Mask bit set in the corresponding bit. The Atttention flag, set by the DSI 8X305 processor can be cleared only by the 8088. The current status of both flags can be read by the 8088 (and the 8X305).

The status flag read command is an I/O read on device code 227H. The resulting 8 bit number contains both flags as follows:

Bit	Meaning
7	Attention Request flag is set by DSI and cleared by user program in the System Unit.
6	Command Request flag is set by the user program in the System Unit to indicate a new command and is cleared by the DSI after command is accepted
5-0	Unused

To set the Command Request flag, the user program should execute an I/O write to device code 226H. To clear the Attention Request flag, the program should execute an I/O write to device code 227H. In either case, the data written is unimportant.

Device Code	Input	Use
226H	I/O Write	Set Command Request Flag
227H	I/O Write	Clear Attention Request Flag
227H	I/O Read	Read Current Flags

Key Scan Codes

In normal 327x terminal operation, each keystroke is sent as a special key scan code to the controller. The controller responds by updating the screen (or screen buffer) to show the echo of this new keystroke. In the DSI, keystrokes are sent to the controller via a keystroke command using a key scan code. The controller generated screen buffer update occurs just like a terminal, and the program can read this buffer as desired. Note that some characters are generated by multiple scan codes, such as shift up, character, shift down.

Key scan codes are specific to 327x systems, and are NOT ASCII or EBCDIC. The following table lists the keys and the proper scan codes:

KEY SCAN CODES (327x)

Key	Scan Code (HEX)	Key	Scan Code (HEX)
A	4D, 60, CD	a	60
B	4D, 61, CD	b	61
C	4D, 62, CD	c	62
D	4D, 63, CD	d	63
E	4D, 64, CD	e	64
F	4D, 65, CD	f	65
G	4D, 66, CD	g	66
H	4D, 67, CD	h	67
I	4D, 66, CD	i	66
J	4D, 69, CD	j	69
K	4D, 6A CD	k	6A
L	4D, 6B, CD	l	6B
M	4D, 6C, CD	m	6C
N	4D, 6D, CD	n	6D
O	4D, 6E, CD	o	6E
P	4D, 6F, CD	p	6F
Q	4D, 70, CD	q	70
R	4D, 71, CD	r	71
S	4D, 72, CD	s	72
T	4D, 73, CD	t	73
U	4D, 74, CD	u	74
V	4D, 75, CD	v	75
W	4D, 76, CD	w	76
X	4D, 77, CD	x	77
Y	4D, 78, CD	y	78
Z	4D, 77, CD	z	77

Key	Scan Code (HEX)	Key	Scan Code (HEX)
)	4D, 20, CD	0	20
!	4D, 21, CD	1	21
@	4D, 22, CD	2	22
#	4D, 23, CD	3	23
$	4D, 24, CD	4	24
%	4D, 25, CD	5	25
^	4D, 26, CD	6	26
&	4D, 27, CD	7	27
*	4D, 28, CD	8	28
(	4D, 29, CD	9	29
PF 1	4F, 21, CF	=	11
PF 2	4F, 22, CF	+	4D, 11, CD
PF 3	4F, 23, CF	,	33
PF 4	4F, 24, CF	.	32
PF 5	4F, 25, CF	\	15
FF 6	4F, 26, CF	|	4D, 15, CD
PF 7	4F, 27, CF	Attn	50
PF 8	4F, 28, CF	Sys Req	4F, 50, CF
PF 9	4F, 29, CF	Cursor	51
PF 10	4F, 20, CF	Clear	4F, 51, CF
PF 11	4F, 30, CF	Erase	53
PF 12	4F, 11, CF	Blink	54
PF 13	40	Erase EOF	55
PF 14	41	Print	56
PF 15	42	Click	57
PF 16	43	Return	08
PF 17	44	Up	0E
PF 18	45	Down	13
PF 19	46	Left	16
PF 20	47	Right	1A
PF 21	48	Dup	5F
PF 22	49	Mark	5E
PF 23	4A	Del	0D
PF 24	4B	Reset	34
PA1	4F, 5F, CF	Enter	18
PA2	4F, 5E, CF	Space	10
PA3	4F, 0C, CF	Shift down	4D
'	12	Shift up	CD
"	4D, 12, CD	Alt down	4F
;	7E	Alt up	CF
:	4D, 7E, CD	Tab fwd	36
/	14	Tab bkwd	35
?	4D, 14, CD	Home	4F, 35, CF
-	30	Backspace	31
Underscore	4D, 30, CD	<	09
		>	40, 09, CD

The DSI Screen Buffer

The screen buffer maintained in the DSI can be read by using the Read Data command. The user program must supply the address in the screen buffer as part of the Read Data command, and receive the data at that location upon completion.

The screen buffer contains the 2000 characters which are normally displayed on the screen of a terminal. The first line of the screen begins in buffer location 50H. Each line then consists of 50H (80 decimal) characters in consecutive order. The starting address of each line is listed below:

Line #	Starting Address Hex	Decimal
1	50	80
2	AO	160
3	F0	240
4	140	320
5	190	400
6	1E0	480
7	230	560
8	280	640
9	2D0	720
10	320	800
11	370	880
12	3C0	960
13	410	1040
14	460	1120
15	4B0	1200
16	500	1280
17	550	1360
18	5A0	1440
19	5F0	1520
20	640	1600
21	690	1680
22	6E0	1760
23	730	1840
24	780	1920
Status	0	0

Bytes removed from the buffer are translated into characters using the following table:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul	sp	0	&	à	ä	À	Ä	a	q	A	Q		P		
1	em	=	1	-	è	ë	È	Ë	b	r	B	R		S		
2	ff	'	2	.	ì	ï	Ì	Ï	c	s	C	S				
3	nl	"	3	,	ò	ö	Ò	Ö	d	t	D	T				
4	stp	/	4	:	ù	ü	Ù	Ü	e	u	E	U				
5	cr	\	5	+	ã	â	Ã	Â	f	v	F	V		6		
6		ı	6	¬	õ	ê	Õ	Ê	g	w	G	W				X
7		¦	7	-	ÿ	î	Y	Î	h	x	H	X				
8	>	?	8	º	à	ô	A	Ô	i	y	I	Y				
9	<	!	9		è	û	E	Û	j	z	J	Z				
A	[	$	ß	^	é	á	E	Á	k	ae	K	AE				
B	]	¢	£	~	ì	é	I	É	l	ø	L	Ø				
C	)	¿	#	••	ò	í	O	Í	m	å	M	Å		B		4
D	(	¥	@	`	ù	ó	U	Ó	n	ç	N	Ç				A
E	}	Pts	%	´	ü	ú	Y	Ú	o	¨	O	;		?		
F	{	¤	_	¸	ç	ñ	C	Ñ	p	¯	P	*				

For example, 2C is '#'; B8 is 'Y'; B3 is 'T'.

Attribute Characters

Proper interpretation of data from the buffer requires attention to the attribute bytes. Normally, an attribute byte (byte from the last 4 columns of the preceding table) will precede and a second will follow any field on the screen. The byte preceding the field defines how that field will be handled. The data word bits for attribute characters are defined as follows:

7	6	5	4	3	2	1	0	Bit Number
1	1	a	b	c	c	d	e	Attribute Character

1,1 = Attribute identifier
a = 0 Unprotected
1 Protected
b = 0 Alphameric
1 Numeric
c,c = 00 Normal display, non-detectable
01 Normal display, detectable
10 Bright display, detectable
11 Non display, non-detectable
d = Reserved (Must always be zero)
e = Modified Data Tag
0 — Field has not been modified.
1 — Field has been modified by the operator.

The Extended Attribute Buffer (EAB) is subdivided into two types of attributes, the Extended Field Attribute (EFA) and the Extended Character Attribute (ECA). The EFA defines the field attributes while the ECA defines each character. The ECA is dependent upon the most recent EFA. When the ECA is 0, the attributes defined by the last EFA remain in effect. The chart shown here indicates this relationship.

Extended Attribute Buffer	EFA GREEN	ECA 0	ECA 0	ECA BLUE	ECA 0	EFA RED	ECA 0
Attribute Buffer	UNPROT	A	B	C	D	PROT	E

Characters 'A' and 'B' are defined to be 'green' and in an unprotected field. Character 'C' is still part of the same unprotected field but has been redefined to 'blue'. Character 'E' is in a protected field and is defined as 'red'. Characters 'A' and 'B' have an ECA value of 0 and their Extended Attribute characteristics revert to the most recent EFA. Character 'D' also has a value of 0. Its characteristic reverts to the most recent EFA of unprotected and green. An ECA is a temporary redefinition of a character and does not affect ECA characters with a value of 0.

For the Extended Field Attributes (EFA) the data word bits are:

7 6 5 4 3 2 1 0	Bit Number
a a b b b c c c	Extended Field Attributes

Bits a,a = 00 Normal Mode
01 Blink Character
10 Reverse Video Characters
11 Underline Character

Bits b,b,b = 000 Default to Base Color
001 Blue
010 Red
011 Magenta
100 Green
101 Cyan
110 Yellow
111 White

Bits c,c,c = 000 Base character set
001 APL
010 PS 2 (191 character)
011 PS 3 " "
100 PS 4 " "
101 PS 5 " "
110 PS 6 " "
111 PS 7 " "

For the Extended Character Attributes (ECA) the data word bits are:

7 6 5 4 3 2 1 0	Bit Number
a a b b b c c c	Extended Field Attributes

Bits a,a = 00 Reverts to most recent EFA
01 Blink Character
10 Reverse Video Characters
11 Underline Character

Bits b,b,b = 000 Reverts to EFA
001 Blue
010 Red
011 Magenta
100 Green
101 Cyan
110 Yellow
111 White

Bits c,c,c = 000 Reverts to EFA
001 APL
010 PS 2 (191 character)
011 PS 3 " "
100 PS 4 " "
101 PS 5 " "
110 PS 6 " "
111 PS 7 " "

When processing a screen buffer, it is necessary for the programmer to remember the most recent attribute byte encountered. Note that the screen is 1920 x 1 characters for attribute purposes. Ends of lines take no part in attribute interpretation. Also, attributes are displayed as blanks on the screen.

Subroutines in BASICA are provided to handle line and field reads, keystroke sends, and status checks. These routines implement all necessary tabular character translations and proper handshaking. Using these subroutines is by far the easiest approach to this programming since all of the complicated functions are correctly handled. See the IRMA(TM) insert, *BASICA Subroutines* in the PC *BASIC* manual for these subroutines.

Installation

IRMA requires NO pre-installation configuration. The circuit board may be installed into any available slot in the System Unit. See general notes in The IBM manual, *Guide to Operations,* Section 5, for installation assistance.

INSTALLATION OF ANY BOARD INTO A PC MUST BE DONE ONLY WHEN UNPLUGGED. TO ENSURE SAFETY, UNPLUG THE SYSTEM UNIT LINE CORD BEFORE REMOVING THE COVER. AFTER INSTALLATION, COMPLETELY RE-ASSEMBLE THE CABINET BEFORE APPLYING POWER.

After the IRMA board is installed, the PC exterior backpanel includes a BNC female connector. The BNC connector is standard for 3270 systems, and for many other types of coaxial connections.

The BNC connector is a 1/8th turn bayonet type device. Attachment requires only a gentle insertion push and a 1/8th turn clockwise to lock. Reverse the procedure to disconnect. The BNC is normally attached to coaxial cable of the type RG-62AU. IBM specifications allow up to 1500 meters of cable to the controller, and the DSI conforms to the specification.

Data on the cable is transferred at a bit rate of 2.3587 MHz. The data is encoded in a Manchester-like code, and transmitted base band. The IBM protocol is designed as a single drop (one terminal per coaxial cable) system.

The 327x controller can be configured via system generation in several ways. IRMA emulates the 3278-2 command structure, and thus requires 3278-2 with a typewriter keyboard system configuration. In particular, terminals (like the 3278) and printers (like the 3287) are very different and coaxial cables configured for one will NOT work on the other.

APPENDIX A

```
DEMO.BAS       - Technical Analysis Corporation - 12-21-1982  09:38:18

10 DEFINT A-Z                                       ' All integers for speed

100 DIM I.TAB%(1279),I.BUF%(1920)                   '*Dimension IRMA's tables

102 GOSUB 50000                                     '*Initialize IRMA's variables
104 P%=0                                            ' Init current display row
200 CLS                                             ' Start with blank screen

202 PRINT "         IRMA BASIC Subroutines Demonstration " '*Frame user area
204 PRINT "                Tiny Terminal Emulator"
206 PRINT
208 PRINT "                  123456789.123456789.
210 PRINT
212 PRINT "              1                          1
214 PRINT "              2                          2
216 PRINT "              3                          3
218 PRINT "              4                          4
220 PRINT "              5                          5
222 PRINT
224 PRINT "                  123456789.123456789.

300 REM Main program loop                           '*Main loop, set cursor
302 GOSUB 51400                                     ' Get 3278 cursor postion
304 IF I.VRO%<1 THEN 314                            ' Test for out of range cond.
306 IF I.VRO%>5 THEN 314
308 IF I.VCL%>20 THEN 314
310 LOCATE 5+I.VRO%,16+I.VCL%,1                     ' Position visible cursor
312 GOTO 400
314 LOCATE 4,14,1                                   ' Off screen, but visible!

400 A$=INKEY$                                       '*Get a user keystroke
402 IF A$=CHR$(27) THEN SYSTEM                      ' EXIT request is ESC key
404 IF LEN(A$)=0 THEN 500                           ' No keystroke here...
406 I.VST$=A$
408 GOSUB 50600                                     ' Send keystrokes

500 REM Refresh screen                              '*Refresh screen
502   I.VCL%=X : I.VRO%=P%+1 : I.VRR%=20
504   GOSUB 51300                                   ' RDABS
506   LOCATE P%+6,16,0                              ' Move invisible cursor
508   PRINT I.VST$                                  ' Print buffer contents
510   P%=P%+1 : IF P%>4 THEN P%=0                   ' Inc row number
512 A#=FRE(I$)                                      ' Force string garbage collect
514 GOTO 300                                        ' Goto main loop
```

```
SAMPLES.BAS   - Technical Analysis Corporation - 12-21-1982  20:02:15

50 DEFINT A-Z
52 PRINT "IRMA BASIC SUBROUTINES DEMONSTRATION - IRMASUBS DEMO 1.01
54 PRINT : PRINT "Initializing IRMASUBS variables & tables

100 DIM I.TAB%(1279),I.BUF%(1920)                    '*Dimension IRMA's tables

102 GOSUB 50000                                      '*Initialize IRMA's variables
110 PRINT

200 REM Field Report                                 '*Field Report
202 I.VFL%=1                                         ' Start with the first field
204 PRINT "Field Row Column Length Contents"         ' Print field info header
206   FH$=" ####  ##   ##    ####  &"                ' Print using definition
210 GOSUB 50700                                      ' Find the first field
212 WHILE I.VER%=0                                   ' Until no fields found
214   GOSUB 50900                                    ' Read field contents
216   I.VOO%=1                                       ' Get string from buffer start
218   GOSUB 51100                                    ' Read string I.VST$
220   I.VST$=LEFT$(I.VST$,40)                        ' Limit the length to screen
222   PRINT USING FH$;I.VFL%,I.VRO%,I.VCL%,I.VFS%,I.VST$
230   GOSUB 50800                                    ' Find next field
240 WEND

300 REM Modify a few fields                          '*Modify field
302 I.VFL%=2                                         ' Second screen field
310 GOSUB 50700                                      ' Find the first field
320 I.VST$="Iwo"                                     ' A simple string
322 I.VOO%=1                                         ' Beginning of data area
324 GOSUB 51200                                      ' Put string in buffer
326 GOSUB 51000                                      ' Write the field
328 IF I.VER%<>0 THEN PRINT "Error: ";I.VER%         ' Report possible error
330 GOSUB 50800                                      ' Point to next field
332 I.VST$="Three"                                   ' More nice data
334 I.VOO%=1                                         ' Buffer start
336 GOSUB 51200                                      ' Put new string in buffer
338 IF I.VFS%>LEN(I.VST$) THEN I.VFS%=LEN(I.VST$)    ' Shorten write length
340 GOSUB 51000                                      ' Write the field
345 STOP

400 REM Display part of the screen                   '*Display
402 I.VCL%=1 : I.VRR%=40                             ' Column one
404 FOR I.VRO%=1 TO 5                                ' Top five lines
406   GOSUB 51300                                    ' Read a short line
```

```
410   PRINT I.VST$                                  ' Print the line
412 NEXT I.VRO%                                     ' Again, with feeling

500 REM General status info                         '*Status info
502 GOSUB 51400                                     ' Read cursor postion
504 PRINT "Buffer pointer ROW:";I.VRO%;"  COLUMN:";I.VCL%
510 GOSUB 50200                                     ' Get slave status
520 GOSUB 900                                       ' Display status

600 REM Send some keystrokes                        '*Keystrokes
602 I.VST$=CHR$(0)+CHR$(15)+"AAA"                   ' Back tab character w/ string
604 GOSUB 50600                                     ' Send keystrokes
612 I.VST$=CHR$(9)+"bbb"                            ' Forward tab
614 GOSUB 50600                                     ' Send keystrokes
620 STOP

700 REM Clear any status                            '*Clear status
702 GOSUB 50200                                     ' Read current status
706 GOSUB 900                                       ' Display current status
710 GOSUB 50300                                     ' Clear all set bits
712 GOSUB 50200                                     ' Get the status again
714 GOSUB 900                                       ' Display the new status

800 REM Trigger stuffs                              '*Trigger tests
802 I.VRO%=1 : I.VCL%=1                             ' Upper left corner
804 I.VMS%=0 : I.VVL%=0                             ' Any change is a trigger
806 GOSUB 50400                                     ' Set trigger data & addr
808 PRINT "Waiting for any change in row 1, column 1"
820 WHILE LEN(INKEY$)=0                             ' Until a key is pressed
822   I.VTO%=2                                      ' 2 second timeout
824   GOSUB 50500                                   ' Wait for trigger event
826   IF I.VTO%<0 THEN PRINT "."; : GOTO 840
830   PRINT "Trigger! "
836   GOTO 842
840 WEND
842 I.VST%=64                                       ' Clear trigger bit
844 GOSUB 50300                                     '
848 PRINT "Waiting for an upper case A in row 1, column 1"
850 I.VMS%=255 : I.VVL%=160                         ' Specific Upper case A
852 GOSUB 50400                                     ' Set the trigger again
860 WHILE LEN(INKEY$)=0                             ' Until a key is pressed
862   I.VTO%=2                                      ' 2 second timeout
864   GOSUB 50500                                   ' Wait for trigger event
```

```
SAMPLES.BAS   - Technical Analysis Corporation - 12-21-1982  20:02:15

866   IF I.VTO%<0 THEN PRINT "."; : GOTO 870
867   PRINT "Trigger! "
868   GOTO 872
870 WEND
872 I.VST%=64                                   ' Clear trigger bit
874 GOSUB 50300                                 '
899 END


900 REM Display status words                    '!Status display
912 PRINT
918 PRINT "Main status word:"
920 IF (I.VST% AND I.MAX%) THEN PRINT "  Aux status change"
922 IF (I.VST% AND I.MTG%) THEN PRINT "  Trigger occurred"
924 IF (I.VST% AND I.MKY%) THEN PRINT "  Key buffer empty"
926 IF (I.VST% AND I.MPR%) THEN PRINT "  Controller issued reset"
928 IF (I.VST% AND I.MCC%) THEN PRINT "  Last command complete"
930 IF (I.VST% AND I.MDI%) THEN PRINT "  Buffer dirty (modified)"
932 IF (I.VST% AND I.MCM%) THEN PRINT "  Buffer pointer moved"
938 PRINT "Aux status word:"
940 IF (I.VAX% AND I.MPO%) THEN PRINT "  Poll occurred
942 IF (I.VAX% AND I.MAL%) THEN PRINT "  Alarm requested"
944 IF (I.VAX% AND I.MDD%) THEN PRINT "  Display disabled (Inhibited)"
946 IF (I.VAX% AND I.MCI%) THEN PRINT "  Cursor inhibited"
948 IF (I.VAX% AND I.MRC%) THEN PRINT "  Reverse video cursor"
950 IF (I.VAX% AND I.MBC%) THEN PRINT "  Blinking cursor"
952 IF (I.VAX% AND I.MCK%) THEN PRINT "  Keyboard clicker enabled"
980 RETURN
```

```
50000 REM Initialize IRMA interface variables   '!INIT - IRMASUBS Rev 1.01
50002 RESTORE 50036                             ' Point to initial values
50004 READ I.CRD%,I.CWR%,I.CAC%,I.CCL%,I.CKY%   ' Load command numbers
50006 READ I.CSP%,I.CXP%,I.CMD%,I.CTA%,I.CIM%
50008 READ I.MAX%,I.MTG%,I.MKY%,I.MXX%,I.MPR%   ' Main status masks
50010 READ I.MCC%,I.MDI%,I.MCM%
50012 READ I.MXX%,I.MPO%,I.MAL%,I.MDD%,I.MCI%   ' Aux status masks
50014 READ I.MRC%,I.MBC%,I.MCK%
50016 READ I.RG0%,I.RG1%,I.RG2%,I.RG3%          ' Communication registers
50018 READ I.RST%,I.RAK%,I.RAF%
50020 READ I.MAT%,I.MBS%                        ' Handshake masks
50022 READ I.VER%,I.VST%,I.VRO%,I.VCL%,I.VMS%   ' General variables
50024 READ I.VVL%,I.VFG%,I.VST$,I.VFL%,I.VT0%
50026 READ I.VT1%,I.VT2%,I.VT3%,I.VS0$,I.VS1$
50028 READ I.VAX%,I.VS2$,I.VS3$,I.VS4$,I.VS5$
50030 READ I.VS6$,I.VS7$,I.VS8$,I.VS9$,I.VT4%
50032 READ I.VT5%,I.VCB%,I.VCE%,I.VT8%,I.VT9%
50034 READ I.VOO%,I.VRR%,I.VFS%,I.VTO%
50036 DATA 0,1,2,3,4,5,6,7,8,9
50038 DATA 128,64,32,16,8,4,2,1
50040 DATA 128,64,32,16,8,4,2,1
50042 DATA &H220,&H221,&H222,&H223
50044 DATA &H226,&H227,&H227
50046 DATA 128,64
50048 DATA 0,0,0,0,0,0,0,"",0,0,0,0,0,"","",
50050 DATA 0,"","","","","","","","","",0
50052 DATA 0,0,0,0,0,0,0,0,5
50058 DEF SEG
50060 BLOAD "IRMATABS.OVR",VARPTR(I.TAB%(0))
50062 RETURN

50100 REM Power on reset simulation             '*XPOR ( I.VER% )
50102 I.VER%=0                                  ' Reset error flag
50104 OUT I.RG0%,I.CXP%                         ' Set commmand in place
50106 GOSUB 58000                               ' Start & wait for slave
50108 RETURN

50200 REM Get slave status                      '*GSTAT ( I.VST%, I.VER% )
50202 I.VER%=0                                  ' Reset error flag
50204 OUT I.RG0%,I.CAC%                         ' Get aux status & cursor
50206 GOSUB 58000                               ' Start & wait for slave
50208 I.VAX%=INP(I.RG3%)                        ' Get aux
50210 I.VST%=INP(I.RG0%)                        ' Get main
```

```
50212 RETURN                                          ' Exit!

50300 REM Reset slave status bits                     '*RSTAT ( I.VST%, I.VER% )
50302 I.VER%=0                                        ' Reset error flag
50304 OUT I.RG0%,I.CCL%                               ' Clear status command
50306 OUT I.RG3%,I.VST%                               ' Status bits to clear
50308 GOSUB 58000                                     ' Start & wait for slave
50310 RETURN                                          ' Exit!

50400 REM Set trigger event & address                 '*STDNM ( I.VRO%, I.VCL% )
50401 I.VER%=0                                        '         I.VMS%, I.VVL%
50402 GOSUB 58200                                     ' Check row and column values
50403 IF I.VER%<>0 THEN RETURN                        ' Error if bad input
50404 OUT I.RG0%,I.CTA%                               '         I.VER%          )
50406 I.VT0%=(I.VCL%-1)+(I.VRO%*80)                   ' Compute address
50408 OUT I.RG2%,I.VT0%\&H100                         ' High part of address
50410 OUT I.RG1%,I.VT0% AND &HFF                      ' Low part of address
50411 GOSUB 58000                                     ' Start slave
50412 IF ((I.VVL% OR I.VMS% ) AND &HFF00)<>0 THEN I.VER%=4 : RETURN ' Bad byte
50417 IF I.VER%<>0 THEN RETURN                        ' Give up if dead slave
50418 OUT I.RG0%,I.CMD%                               ' Setup mask & data
50419 OUT I.RG1%,I.VVL%                               ' Data
50420 OUT I.RG2%,I.VMS%                               ' Mask
50422 GOSUB 58000                                     ' Set trigger.
50424 RETURN                                          ' Exit!

50500 REM Wait for trigger event                      '*WTRIG ( I.VT0%, I.VER% )
50502 I.VER%=0                                        ' Reset error flag
50504 I.VS0$=TIME$                                    ' Set the stopwatch
50506 IF (INP(I.RG0%) AND I.MTG%)<>0 THEN RETURN      ' All done!
50508 I.VER%=10                                       ' Potential timeout.
50510 IF I.VTO%<0 THEN RETURN                         ' Time has run out.
50512 IF I.VS0$=TIME$ THEN 50502                      ' Still time, try again
50514 I.VTO%=I.VTO%-1                                 ' Drop a grain of sand.
50516 GOTO 50502                                      ' Do it some more.

50600 REM Send keystrokes from I.VST$                 '*KEYS ( I.VST$, I.VER% ))
50602 I.VER%=0                                        ' Reset error flag
50604 I.VT0%=LEN(I.VST$)                              ' Count of remaining chars
50606 I.VT1%=1                                        ' Current pointer
50608 WHILE I.VT0%>0                                  ' Only until none remain...
50610   I.VT2%=ASC(MID$(I.VST$,I.VT1%,1))             ' Get character value
50612   IF I.VT2%>0 THEN 50620                        ' Not an extended code...
```

```
50614      I.VT0%=I.VT0%-1 : I.VT1%=I.VT1%+1       ' Get set to eat next char
50616      IF I.VT0%<1 THEN 50658                  ' EXIT if partial char
50618      I.VT2%=ASC(MID$(I.VST$,I.VT1%,1))+256   ' Offset into extended table
50620    I.VT2%=I.TAB%(I.VT2%+&H200)               ' Look up key codes
50622    IF (I.VT2% AND &HFF)=0 THEN 50634         ' Skip shift key
50624      I.VT3%=I.TAB%((I.VT2% AND &HFF)+&H400)  ' Get scan code of shift
50626      I.VT3%=I.VT3% AND &H7F                  ' Strip up/dn control bit
50628      GOSUB 58100                             ' Transmit scan code
50630      IF I.VER%=0 THEN 50634                  ' Skip error exit
50632        I.VT0%=-1 : GOTO 50658                ' Error on key attempt ABORT!
50634    IF (I.VT2% AND &HFF00)=0 THEN 50644       ' Handle possible lone shift
50636      I.VT3%=I.TAB%((I.VT2%\&H100)+&H400)     ' Get scan code
50638      GOSUB 58100                             ' Transmit it.
50640      IF I.VER%=0 THEN 50644                  ' Skip error exit
50642        I.VT0%=-1 : GOTO 50658                ' PUNT.
50644    IF (I.VT2% AND &HFF)=0 THEN 50654         ' Skip shift key
50646      I.VT3%=I.TAB%((I.VT2% AND &HFF)+&H400)  ' Get scan code of shift
50648      GOSUB 58100                             ' Transmit scan code
50650      IF I.VER%=0 THEN 50654                  ' Skip error exit
50652        I.VT0%=-1 : GOTO 50658                ' Error on key attempt ABORT!
50654    I.VT0%=I.VT0%-1                           ' One less character to send
50656    I.VT1%=I.VT1%+1                           ' Point to next character
50658 WEND                                         ' Do until done or error
50660 RETURN

50700 REM Find field I.VFL%                        '*FIND ( I.VFL%, I.VCL% ))
50702 I.VER%=0                                     '        I.VRO%, I.VER% )
50703 IF (I.VFL%<1) OR (I.VFL%>1920) THEN I.VER%=7 : RETURN
50704 I.VCE%=80 : I.VCB%=80 : I.BUF%(0)=&HC0       ' Start at upper left
50706 OUT I.RG0%,I.CRD%                            ' Preset the read command
50708 IF I.VFL%=1 THEN 50738                       ' Default screen condition
50710 I.VT1%=1                                     ' Current field number
50712 OUT I.RG1%,I.VCE% AND &HFF                   ' Low order address
50714 OUT I.RG2%,I.VCE%\&H100                      ' High order address
50716 GOSUB 58000                                  ' Start slave & wait
50718 IF I.VER%<>0 THEN RETURN                     ' PUNT if read error.
50720 I.VT2%=INP(I.RG3%)                           ' Get data
50722 IF I.VT2%<&HC0 THEN 50730                    ' Skip if not attribute
50724   IF (I.VT2% AND &H20)>0 THEN 50730          ' Skip if protected
50725     I.VCB%=I.VCE%                            ' Save start position
50726     I.VT1%=I.VT1%+1                          ' We've found the next field!
50727     I.BUF%(0)=I.VT2%                         ' Init buffer attribute
50728     IF I.VT1%=I.VFL% THEN 50736              ' Exit the search loop
```

```
50730 I.VCE%=I.VCE%+1                                  ' Try next location
50732 IF I.VCE%>=2000 THEN I.VER%=7 : RETURN            ' Field not found anywhere!
50734 GOTO 50712                                       ' Read the next character
50736 I.VCE%=I.VCE%+1                                  ' Point to first char of field
50738 IF I.VCE%>=2000 THEN I.VER%=7 : RETURN            ' Field at last col/row
50740 I.VRO%=I.VCE%\80                                 ' Make the row number
50742 I.VCL%=(I.VCE% MOD 80)+1                         ' And column number
50744 I.VFS%=0                                         ' Length of field
50746 OUT I.RG1%,I.VCE% AND &HFF                       ' Low order address
50748 OUT I.RG2%,I.VCE%\&H100                          ' High order address
50750 GOSUB 58000                                      ' Start read operation
50752 IF I.VER%<>0 THEN RETURN                         ' Quit if slave is dead
50754 I.VT2%=INP(I.RG3%)                               ' Get data
50756 IF I.VT2%>=&HC0 THEN RETURN                      ' Next attribute found
50758 I.VFS%=I.VFS%+1                                  ' Count the chars in field
50760 I.VCE%=I.VCE%+1                                  ' Point to next char
50762 IF I.VCE%>=2000 THEN RETURN                      ' Ran off end of screen!
50764 GOTO 50746                                       ' Continue eating chars

50800 REM Find NEXT field I.VFL%+1                     '*FNEXT ( I.VFL%, I.VCL%,)
50801 I.VER%=0                                         '       ( I.VRO%, I.VER% )
50802 I.VT1%=I.VFL% : I.VFL%=I.VFL%+1                  ' Next field
50804 IF I.VCE%<80 THEN I.VER%=11 : RETURN             ' Illegal cursor value
50806 IF I.VCE%>=2000 THEN I.VER%=7 : RETURN           ' No next field.
50808 OUT I.RG0%,I.CRD%                                ' Set read command in place
50810 OUT I.RG1%,I.VCE% AND &HFF                       ' Low order address
50812 OUT I.RG2%,I.VCE%\&H100                          ' High order address
50814 GOSUB 58000                                      ' Get data at current position
50816 IF I.VER%>0 THEN RETURN                          ' Give up if slave dead
50820 I.VT9%=INP(I.RG3%)                               ' This should be attribute
50822 IF I.VT9%<&HC0 THEN I.VER%=11 : RETURN           ' Invalid cursor position
50830 GOTO 50712                                       ' Enter general FIND code

50900 REM Read field                                   '*RDFLD ( I.VER% )I.VCL%,)
50902 I.VER%=0 : I.VT2%=0                              ' Reset error flag & buffer ptr
50904 IF I.VCB%=80 THEN 50924                          ' Special case of upper corner
50906 IF I.VCB%<80 THEN I.VER%=11                      ' Invalid start address
50908 IF I.VCB%>=2000 THEN I.VER%=11                   ' Beyond end of screen
50910 OUT I.RG0%,I.CRD%                                ' Set read command
50912 OUT I.RG1%,I.VCB% AND &HFF                       ' Low order address
50914 OUT I.RG2%,I.VCB%\&H100                          ' High order address
50916 GOSUB 58000                                      ' Read the leading attribute
50918 I.VT3%=INP(I.RG3%)                               ' Data word
```

```
50920 IF (I.VT3% AND &HE0)<>&HC0 THEN I.VER%=5  : RETURN ' Bad field type
50922 GOTO 50930                                ' Scan and eat field
50924 I.VT2%=1                                  ' Increment buffer pointer
50926 I.BUF%(0)=&HC0                            ' Fake attribute
50930 OUT I.RG0%,I.CRD%                         ' Setup the read command
50934 WHILE I.VT2%<=I.VFS%                      ' Until out of characters
50936   I.VT9%=I.VT2%+I.VCB%                    ' Offset to character
50938   IF I.VT9%>=2000 THEN I.VER%=11 : GOTO 50950 ' bad field specs
50940   OUT I.RG1%,I.VT9% AND &HFF              ' Set low address
50942   OUT I.RG2%,I.VT9%\&H100                 ' High address
50946   GOSUB 58000                             ' Xecute read operation
50948   IF I.VER%=0 THEN 50960                  ' Abort if slave dead
50950     I.VT2%=9999                           ' Exit loop
50952     GOTO 50970                            '
50960   I.BUF%(I.VT2%)=CVI(CHR$(INP(I.RG3%))+CHR$(INP(I.RG2%)))
50962   I.VT2%=I.VT2%+1                         ' Point to next location
50970 WEND
50972 RETURN                                    ' All done

51000 REM Write field                           '*WRFLD ( I.VER% )I.VCL%,)
51002 I.VER%=0                                  ' Reset error flag
51004 IF I.VCB%<80 THEN I.VER%=11               ' Illegal address
51006 IF I.VCB%>=2000 THEN I.VER%=11            ' Too large
51008 IF I.VCB%+I.VT1%>2000 THEN I.VER%=12      ' Field too long for screen
51010 IF I.VER%<>0 THEN RETURN                  ' Quit if bad parameters
51012 OUT I.RG0%,I.CRD%                         ' Read attribute from screen
51014 OUT I.RG1%,I.VCB% AND &HFF                ' Low address
51016 OUT I.RG2%,I.VCB%\&H100                   ' High address
51018 GOSUB 58000                               ' Xecute!
51020 IF I.VER%<>0 THEN RETURN                  ' Abandon if dead slave
51022 IF (INP(I.RG3%) AND &HFE)<>(I.BUF%(0) AND &HFE) THEN I.VER%=13 : RETURN
51024 I.BUF%(0)=I.BUF%(0) OR 1                  ' Set the MDT flag
51026 I.VT2%=0                                  ' First byte of buffer to write
51028 IF (I.BUF%(0) AND &H10)=0 THEN 51052      ' Check for numeric only
51030   I.VT2%=1                                ' First buffer location to chk
51032   WHILE I.VT2%<=I.VT1%                    ' Do until all chars done
51034     I.VT0%=I.BUF%(I.VT2%) AND &HFF        ' Mask off any previous EAB
51036     IF I.VT0%>=&H20 THEN IF I.VT0%<=&H29 THEN 51048 ' 0-9 is Ok
51038       IF I.VT0%=&H31 THEN 51048           ' Minus Ok
51039       IF I.VT0%=&H32 THEN 51048           ' Period Ok
51040       IF I.VT0%=&H35 THEN 51048           ' Plus Ok
51042     I.VER%=6 : I.VT2%=9999                ' "Non-numeric" char
51048     I.VT2%=I.VT2%+1                       ' Next character
```

```
51050   WEND                                    ' Loop
51052 I.VT2%=0                                  ' Start from the beginning
51054 OUT I.RG0%,I.CWR%                         ' Set write command
51056 WHILE I.VT2%<=I.VFS%                      ' Until all characters written
51058   I.VT0%=I.VCB%+I.VT2%                    ' Compute address
51060   OUT I.RG1%,I.VT0% AND &HFF              ' Low address
51062   OUT I.RG2%,I.VT0%\&H100                 ' High address
51064   OUT I.RG3%,I.BUF%(I.VT2%) AND &HFF      ' Data w/o EAB
51066   GOSUB 58000                             ' Make slave do it.
51068   IF I.VER%<>0 THEN I.VT2%=9999           ' Force exit if slave DOA
51070   I.VT2%=I.VT2%+1                         ' Next location to write
51072 WEND                                      ' Do it again
51074 RETURN                                    ' All done!

51100 REM Get string from buffer                '*GSTR( I.VST$, I.VT1%
51105 I.VER%=0 : I.VT2%=0                       '       I.VT0%          )
51108 IF I.VOO%>1920 THEN I.VER%=12 : RETURN    ' Offset too large
51110 IF I.VOO%<0 THEN I.VER%=12 : RETURN       ' Offset negative
51112 I.VST$=""                                 ' Clear string
51114 WHILE (I.VOO%<=I.VFS%) AND (I.VT2%=0)     '
51116   I.VST$=I.VST$+CHR$(I.TAB%((I.BUF%(I.VOO%) AND &HFF)+&H100))
51118   I.VOO%=I.VOO%+1                         ' Point to next char
51120   IF LEN(I.VST$)=254 THEN I.VT2%=1        ' Overrun
51122 WEND
51126 RETURN

51200 REM Put string in buffer                  '*PSTR( I.VST$, I.VT1%
51202 I.VER%=0                                  '       I.VT0%          )
51204 IF I.VOO%+LEN(I.VST$)-1>I.VFS% THEN I.VER%=12 : RETURN ' Too long
51206 I.VT3%=LEN(I.VST$) : IF I.VT3%=0 THEN RETURN ' Zero strings easy!
51208 I.VT2%=0                                  ' Offset into buffer
51210 WHILE I.VT2%<I.VT3%                       ' While still characters
51212   I.BUF%(I.VOO%+I.VT2%)=I.TAB%(ASC(MID$(I.VST$,I.VT2%+1,1))+&H0)
51214   I.VT2%=I.VT2%+1                         ' Move pointer
51216 WEND
51218 I.VOO%=I.VOO%+I.VT2%                      ' Pointer for next
51220 RETURN                                    ' Transfer complete

51300 REM Read screen absolute                  '*ABSRD ( I.VER%, I.VST$,
51302 I.VER%=0                                  '         I.VS0$, I.VRO%,
51304 I.VT0%=I.VRO%*80+I.VCL%-1                 '         I.VCL%, I.VRR% )
51306 IF I.VT0%<0 THEN I.VER%=11 : RETURN       ' Invalid cursor address
51308 IF I.VT0%>=2000 THEN I.VER%=11 : RETURN   '
```

```
51310 IF I.VT0%+I.VRR%>2000 THEN I.VER%=12 : RETURN ' Can't read that much
51312 I.VT1%=0 : I.VST$="" : I.VS0$=""              ' Offset from start
51314 OUT I.RG0%,I.CRD% : I.VT2%=I.VT0%+I.VRR%     ' Set a read commnad
51316 WHILE I.VT0%<I.VT2%                          ' Until all characters read
51318   OUT I.RG1%,I.VT0% AND &HFF                 ' Low order address
51320   OUT I.RG2%,I.VT0%\&H100                    ' High order address
51322   GOSUB 58000                                ' Execute read operation
51324   IF I.VER%<>0 THEN I.VT0%=9999              ' Force loop exit
51326   I.VST$=I.VST$+CHR$(I.TAB%(INP(I.RG3%)+&H100)) ' Get char & convert
51328   I.VS0$=I.VS0$+CHR$(INP(I.RG2%))            ' Unmodified EAB
51340   I.VT0%=I.VT0%+1                            ' Next character
51342 WEND
51344 RETURN

51400 REM Read 3278 cursor position                '*CPOS( I.VRO%, I.VCL%,
51402 I.VER%=0                                     '       I.VER%          )
51406 OUT I.RG0%,I.CAC%                            ' Status & cursor read
51408 GOSUB 58000                                  ' Start the slave
51410 I.VT0%=(INP(I.RG2%)*&H100)+INP(I.RG1%)       ' Get absolute address
51411 IF I.VER%<>0 THEN RETURN                     ' Dead slave, quit action
51412 I.VRO%=I.VT0%\80                             ' Compute row
51414 I.VCL%=(I.VT0% MOD 80)+1                     ' And column
51416 RETURN
```

SAMPLES.BAS - Technical Analysis Corporation - 12-21-1982 20:02:15

```
58000 REM Start & wait for slave                  '!Start & wait for slave
58002 I.VS9$=TIME$ : I.VT9%=0                     ' A simulated stopwatch
58004 OUT I.RS1%,0                                ' Start the slave
58006 I.VT8% = INP( I.RAF% ) AND I.MBS%           ' Get slave busy bit
58008 IF I.VT8% = 0 THEN RETURN                   ' Return with command complete
58010 IF I.VS9$=TIME$ THEN 58006                  ' Loop until clock ticks
58012 I.VS9$=TIME$ : I.VT9%=I.VT9%+1              ' Three times
58014 IF I.VT9%<3 THEN 58006                      '
58016 I.VER%=1                                    ' Slave timeout
58018 RETURN                                      '

58100 REM Send key scan code from I.VT3%          '*Send key scan code
58102 I.VS9$=TIME$ : I.VT9%=0                     ' Make stopwatch
58104 IF (INP(I.RG0%) AND I.MKY%)>0 THEN 58114    ' Key buffer is ready
58106 IF I.VS9$=TIME$ THEN 58104                  ' Loop until clock tick
58108 I.VT9%=I.VT9%+1 : IF I.VT9%<4 THEN 58104    ' Time has not run out
58110 I.VER%=9                                    ' Keystroke timeout
58112 RETURN
58114 IF I.VT3%=0 THEN I.VER%=15 : RETURN         ' Invalid scan code
58115 OUT I.RG0%,I.CKY%                           ' Keystroke command
58116 OUT I.RG3%,I.VT3%                           ' Scan code
58118 GOSUB 58000                                 ' Fire up the slave
58120 RETURN                                      ' And we're done!

58200 REM - Verify Row and Column                 '*Verify ROW/COL
58206 IF (I.VRO%<0) OR (I.VRO%>24) THEN I.VER%=2 : RETURN
58208 IF (I.VCL%<1) OR (I.VCL%>80) THEN I.VER%=3 : RETURN
58210 RETURN
```

```
IRMATABS.BAS  - Technical Analysis Corporation - 12-21-1982  17:33:44

100 PRINT "Building IRMATABS.OVR for IRMASUBS package"
110 DEFINT A-Z
115 DEF SEG
120 DIM I.TAB%(1279)                                  ' 256+256+512+256
130 RESTORE
135 FOR I.VT0%=0 TO 1279
140   READ I.TAB%(I.VT0%)
145 NEXT
200 BSAVE "IRMATABS.OVR",VARPTR(I.TAB%(0)),2560 ' Save the whole array
210 PRINT "Build complete. "
220 NEW
300 REM Offsets into I.TAB% are as follows:
302 REM
304 REM      000 - ASCII to buffer code table     (256 entrys)
306 REM      256 - Buffer code to ASCII table     (256 entrys)
308 REM      512 - Normal keycodes to key no.     (256 entrys)
310 REM      768 - Extended keycodes to key no.   (256 entrys)
312 REM     1024 - Key number to scan code        (256 entrys)
320 REM
59600 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59601 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59602 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59603 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59604 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59605 DATA &h0000, &h0000, &h0010, &h0019, &h003C, &h002C
59606 DATA &h001A, &h002E, &h0030, &h0012, &h000D, &h000C
59607 DATA &h00BF, &h0035, &h0033, &h0031, &h0032, &h0014
59608 DATA &h0020, &h0021, &h0022, &h0023, &h0024, &h0025
59609 DATA &h0026, &h0027, &h0028, &h0029, &h0034, &h00BE
59610 DATA &h0009, &h0011, &h0008, &h0018, &h002D, &h00A0
59611 DATA &h00A1, &h00A2, &h00A3, &h00A4, &h00A5, &h00A6
59612 DATA &h00A7, &h00A8, &h00A9, &h00AA, &h00AB, &h00AC
59613 DATA &h00AD, &h00AE, &h00AF, &h00B0, &h00B1, &h00B2
59614 DATA &h00B3, &h00B4, &h00B5, &h00B6, &h00B7, &h00B8
59615 DATA &h00B9, &h000B, &h0015, &h000A, &h0036, &h002F
59616 DATA &h003D, &h0080, &h0081, &h0082, &h0083, &h0084
59617 DATA &h0085, &h0086, &h0087, &h0088, &h0089, &h008A
59618 DATA &h008B, &h008C, &h008D, &h008E, &h008F, &h0090
59619 DATA &h0091, &h0092, &h0093, &h0094, &h0095, &h0096
59620 DATA &h0097, &h0098, &h0099, &h000F, &h0017, &h000E
59621 DATA &h003B, &h0000, &h0000, &h0000, &h0000, &h0000
59622 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59623 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59624 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
```

```
59625 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59626 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59627 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59628 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59629 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59630 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59631 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59632 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59633 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59634 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59635 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59636 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59637 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59638 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59639 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59640 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59641 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59642 DATA &h0000, &h0000, &h0000, &h0000
59700 DATA &h0020, &h0020, &h0020, &h0020, &h0020, &h0020 : ' EBCDIC
59701 DATA &h0020, &h0020, &h003E, &h003C, &h005B, &h005D
59702 DATA &h0029, &h0028, &h007D, &h007B, &h0020, &h003D
59703 DATA &h0027, &h0022, &h002F, &h005C, &h007C, &h007C
59704 DATA &h003F, &h0021, &h0024, &h0063, &h006C, &h0079
59705 DATA &h0070, &h006F, &h0030, &h0031, &h0032, &h0033
59706 DATA &h0034, &h0035, &h0036, &h0037, &h0038, &h0039
59707 DATA &h0062, &h0073, &h0023, &h0040, &h0025, &h005F
59708 DATA &h0026, &h002D, &h002E, &h002C, &h003A, &h002B
59709 DATA &h002D, &h005F, &h002E, &h0020, &h005E, &h007E
59710 DATA &h0022, &h0060, &h0027, &h0035, &h0061, &h0065
59711 DATA &h0069, &h006F, &h0075, &h0061, &h006F, &h0079
59712 DATA &h0061, &h0065, &h0065, &h0069, &h006F, &h0075
59713 DATA &h0075, &h0063, &h0061, &h0065, &h0069, &h006F
59714 DATA &h0075, &h0061, &h006F, &h0079, &h0061, &h0065
59715 DATA &h0065, &h0069, &h006F, &h0075, &h0075, &h0063
59716 DATA &h0041, &h0045, &h0049, &h004F, &h0055, &h0041
59717 DATA &h004F, &h0059, &h0041, &h0045, &h0045, &h0049
59718 DATA &h004F, &h0055, &h0059, &h0043, &h0041, &h0045
59719 DATA &h0049, &h004F, &h0055, &h0041, &h0045, &h0049
59720 DATA &h004F, &h0055, &h0041, &h0045, &h0049, &h004F
59721 DATA &h0055, &h004E, &h0061, &h0062, &h0063, &h0064
59722 DATA &h0065, &h0066, &h0067, &h0068, &h0069, &h006A
59723 DATA &h006B, &h006C, &h006D, &h006E, &h006F, &h0070
59724 DATA &h0071, &h0072, &h0073, &h0074, &h0075, &h0076
59725 DATA &h0077, &h0078, &h0079, &h007A, &h0061, &h006F
```

```
59726 DATA &h0061, &h0063, &h003B, &h002A, &h0041, &h0042
59727 DATA &h0043, &h0044, &h0045, &h0046, &h0047, &h0048
59728 DATA &h0049, &h004A, &h004B, &h004C, &h004D, &h004E
59729 DATA &h004F, &h0050, &h0051, &h0052, &h0053, &h0054
59730 DATA &h0055, &h0056, &h0057, &h0058, &h0059, &h005A
59731 DATA &h0041, &h004F, &h0041, &h0043, &h003B, &h002A
59732 DATA &h0020, &h0020, &h0020, &h0020, &h0020, &h0020
59733 DATA &h0020, &h0020, &h0020, &h0020, &h0020, &h0020
59734 DATA &h0020, &h0020, &h0020, &h0020, &h0050, &h0053
59735 DATA &h0041, &h001E, &h0042, &h0036, &h0010, &h0016
59736 DATA &h001A, &h00E9, &h0006, &h0001, &h0042, &h0003
59737 DATA &h00A8, &h00DB, &h0020, &h0020, &h0020, &h0020
59738 DATA &h0020, &h0020, &h0020, &h0020, &h0020, &h0020
59739 DATA &h0020, &h0020, &h0020, &h0020, &h0020, &h0020
59740 DATA &h0015, &h0017, &h005A, &h005F, &h0009, &h000A
59741 DATA &h0058, &h0016, &h001B, &h0025, &h00FB, &h00B7
59742 DATA &h0034, &h0041, &h00E9, &h0002
59800 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59801 DATA &h0000, &h0000, &h1000, &h1500, &h3400, &h0000
59802 DATA &h0000, &h4C00, &h0000, &h0000, &h0000, &h0000
59803 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59804 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59805 DATA &h0000, &h0000, &h4A00, &h2039, &h3239, &h0639
59806 DATA &h0739, &h0839, &h0A39, &h3200, &h0C39, &h0D39
59807 DATA &h0B39, &h0F39, &h4200, &h0E00, &h4300, &h4400
59808 DATA &h0D00, &h0400, &h0500, &h0600, &h0700, &h0800
59809 DATA &h0900, &h0A00, &h0B00, &h0C00, &h3139, &h3100
59810 DATA &h3A00, &h0F00, &h3A39, &h4439, &h0539, &h2839
59811 DATA &h3F39, &h3D39, &h2A39, &h1839, &h2B39, &h2C39
59812 DATA &h2D39, &h1D39, &h2E39, &h2F39, &h3039, &h4139
59813 DATA &h4039, &h1E39, &h1F39, &h1639, &h1939, &h2939
59814 DATA &h1A39, &h1C39, &h3E39, &h1739, &h3C39, &h1B39
59815 DATA &h3B39, &h0439, &h2100, &h2000, &h0939, &h0E39
59816 DATA &h0300, &h2800, &h3F00, &h3D00, &h2A00, &h1800
59817 DATA &h2B00, &h2C00, &h2D00, &h1D00, &h2E00, &h2F00
59818 DATA &h3000, &h4100, &h4000, &h1E00, &h1F00, &h1600
59819 DATA &h1900, &h2900, &h1A00, &h1C00, &h3E00, &h1700
59820 DATA &h3C00, &h1B00, &h3B00, &h3300, &h2139, &h3339
59821 DATA &h0339, &h1000, &h0000, &h0000, &h0000, &h0000
59822 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59823 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59824 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59825 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59826 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
```

```
59827 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59828 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59829 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59830 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59831 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59832 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59833 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59834 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59835 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59836 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59837 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59838 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59839 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59840 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59841 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59842 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59843 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59844 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59845 DATA &h0000, &h2200, &h4D00, &h4E00, &h4F00, &h5000
59846 DATA &h5100, &h5200, &h5300, &h5400, &h5500, &h5600
59847 DATA &h0000, &h0000, &h0000, &h0000, &h5700, &h5800
59848 DATA &h464B, &h474B, &h1100, &h1200, &h114B, &h124B
59849 DATA &h234B, &h0000, &h0000, &h0000, &h0000, &h0000
59850 DATA &h3B4B, &h3C4B, &h3D4B, &h3E4B, &h3F4B, &h404B
59851 DATA &h414B, &h0000, &h0000, &h0000, &h0000, &h0000
59852 DATA &h0000, &h0000, &h0000, &h0100, &h024B, &h1300
59853 DATA &h144B, &h2500, &h2600, &h3700, &h3800, &h3400
59854 DATA &h4800, &h0000, &h0000, &h224B, &h3500, &h2700
59855 DATA &h0000, &h4600, &h0000, &h4700, &h0000, &h0000
59856 DATA &h3600, &h3900, &h2300, &h2400, &h0000, &h0000
59857 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59858 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59859 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59860 DATA &h014B, &h0200, &h134B, &h1400, &h254B, &h264B
59861 DATA &h374B, &h384B, &h0000, &h484B, &h0000, &h464B
59862 DATA &h474B, &h0000, &h0000, &h5900, &h044B, &h054B
59863 DATA &h064B, &h074B, &h084B, &h094B, &h0A4B, &h0B4B
59864 DATA &h0C4B, &h0D4B, &h0E4B, &h0F4B, &h0000, &h0000
59865 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59866 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59867 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59868 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59869 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59870 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
```

```
59871 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59872 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59873 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59874 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59875 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59876 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59877 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59878 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59879 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59880 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59881 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59882 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59883 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59884 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59885 DATA &h0000, &h0000
59890 DATA &H0000                                          : ' SCODE
59900 DATA &h0050, &h0051, &h003D, &h0021, &h0022, &h0023
59901 DATA &h0024, &h0025, &h0026, &h0027, &h0028, &h0029
59902 DATA &h0020, &h0030, &h0011, &h0031, &h005F, &h005E
59903 DATA &h0052, &h0053, &h0036, &h0070, &h0076, &h0064
59904 DATA &h0071, &h0073, &h0078, &h0074, &h0068, &h006E
59905 DATA &h006F, &h001B, &h0015, &h0035, &h000C, &h000D
59906 DATA &h0054, &h0055, &h00CC, &h0060, &h0072, &h0063
59907 DATA &h0065, &h0066, &h0067, &h0069, &h006A, &h006B
59908 DATA &h007E, &h0012, &h000F, &h0008, &h000E, &h0013
59909 DATA &h0056, &h0057, &h00CD, &h0009, &h0079, &h0077
59910 DATA &h0062, &h0075, &h0061, &h006D, &h006C, &h0033
59911 DATA &h0032, &h0014, &h00CE, &h0016, &h001A, &h0034
59912 DATA &h0034, &h0010, &h00CF, &h0018, &h0040, &h0041
59913 DATA &h0042, &h0043, &h0044, &h0045, &h0046, &h0047
59914 DATA &h0048, &h0049, &h004A, &h004B, &h0000, &h0000
59915 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59916 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59917 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59918 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59919 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59920 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59921 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59922 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59923 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59924 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59925 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59926 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59927 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
```

```
59928 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59929 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59930 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59931 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59932 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59933 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59934 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59935 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59936 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59937 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59938 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59939 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59940 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59941 DATA &h0000, &h0000, &h0000, &h0000, &h0000, &h0000
59942 DATA &h0000, &h0000, &h0000
```

APPENDIX B

Limited Product Warranty

Technical Analysis Corporation (TAC) warrants the Decision Support Interface™, IRMA™, product hardware, with the exception of the supplied diskette, to be free from defects in material and workmanship under normal, proper and intended use in its unmodified condition for one year from the date of purchase by the first End User. TAC's sole obligation under this hardware warranty shall be to furnish parts and labor for the repair or replacement of the product found by TAC to be defective in material or workmanship during the warranty period. This obligation applies only to the first End User Purchaser of the product and does not apply to subsequent purchasers through resale by the first End User.

Warranty repairs will be performed at the point of manufacture. Equipment authorized by TAC for return for warranty service shall be accompanied by a written description of the defect, returned postpaid to the TAC factory and upon repair or replacement will be redelivered by TAC freight prepaid to the End User. The warranty of TAC does not cover normal wear and tear, or damage caused by accident, negligence, vandalism, alteration, abuse, misuse, improper installation, environmental stress, or acts of God.

The diskette supplied with the product is covered by the above provisions for a period of thirty days. The End User is responsible for making adequate copies of the diskette for back-up and recovery should a diskette be damaged by the diskette drive mechanism or by improper handling of the diskette.

TAC warrants that the product firmware will conform to TAC's product specifications prevailing at the time of product delivery to the first Purchaser of the product. This firmware warranty makes no claim of compatibility with equipment or software supplied or to be supplied in the future by others.

(Continued on Reverse Side)

Software contained on the diskette supplied with the product is provided to the End User as a convenience; it has been placed in the Public Domain by TAC and therefore, comes with no warranty of any kind.

Generally distributed firmware updates will also be supplied during the warranty period to those End Users who return their FIRMWARE/SOFTWARE UPDATE REGISTRATION card to TAC.

THIS EXPRESS LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR USE. TAC SHALL NOT BE LIABLE FOR ANY DAMAGES SUSTAINED BY PURCHASER OR ANY OTHER PARTY ARISING FROM OR RELATING TO THE USE OR PERFORMANCE OF THE PRODUCT, OR TO ANY EQUIPMENT FAILURE, INCLUDING, BUT NOT LIMITED TO CONSEQUENTIAL DAMAGES, NOR SHALL TAC HAVE ANY LIABILITY FOR DELAYS IN REPLACEMENT OR REPAIR OF RELATED EQUIPMENT OR THE TAC PRODUCT.

Information Request

In order to provide complete and accurate documentation, your comments would be greatly appreciated. You are encouraged to report any discrepancies found in this or any other TAC manual. TAC will expend its best effort to investigate and take corrective action on any verified errors.

1. Was the documentation easy to read? If it was not, please indicate confusing sections.

2. Were you able to find information easily? If not, indicate the information that was difficult to find.

3. Were the technical terms defined adequately? List any which were not.

4. Was the information accurate? List any discrepancies.

5. Was the documentation complete in its information? List any area where further discussion was needed.

6. General comments:

TECHNICAL ANALYSIS CORPORATION
120 W. WIEUCA RD., N.E.
ATLANTA, GA. 30042 U.S.A.

SOFTWARE UPDATE #1

Revision 1.08, released February 7, 1983, offers two significant enhancements. The first enhancement supports the PC-DOS MODE command screen centering commands. E78 now recognizes the information made available by the MODE command and uses it appropriately. This will be helpful to users with some types of non-IBM monitors which were reported to have centering problems.

The second enhancement allows the user to save as many as nine screen images for later review or recall. This feature requires that E78 be called with a new command sequence. If the old sequence is used, the emulator program will work as before with no screen images available to the user. The new calling sequence is:

E78/n <filename>

where n is the number of screen images the user wishes to have available. If memory is insufficient for the number of screen images requested, E78 allocates as many as will fit. This number is particularly important when E78 is to be made resident. Each screen image allocated to the resident emulator requires 4K bytes (4096 bytes) of system memory in addition to the normal memory consumed by the resident emulator.

In the case where both a resident and non-resident emulator are both active, E78 will share previously allocated resident buffers. For example, the resident E78 has allocated two screen images and the non-resident E78 has allocated five. In this case screen images one and two would be resident and three through five would be non-resident.

The new sequences for accessing the screen image feature are described below:

CONTROL & END Provides for the selection of the screen image to be used. The prompt: Select screen memory for STORE will appear on the status line. Press a number key (1-9) to select the screen image. This command does not affect the screen currently displayed.

END	Provides the means to RECALL and display a screen image. The prompt: Select screen memory for RECALL will appear on the status line. Press a number (0-9) to select. Zero is used here to select the current 3278 screen.

For the functions press the key or keys listed on the left to use these features.

While a screen image is being displayed, the message 'Screen memory n' will be displayed at the right most end of the status line indicating that screen image 'n' (1-9) is being displayed. Error messages will appear briefly on the status line for the following conditions:

— Memory not allocated

— Memory empty

— Invalid number (not 1-9)

To go back to displaying the current 3278 screen after a screen image has been recalled, press the 'RECALL' key (IBM's END key) followed by the digit 0. Pressing any key that causes a character to be sent to the host computer/controller will also cause the display to revert back to the current 3278 screen. These keys include all keys except those which change screen modes or keys which have no current function (dead keys).

Display mode function keys may be used to view the recalled screens in all the modes available to the normal screen. Be aware that saving a screen with non-displayed fields present, such as passwords, will allow someone to later recall that screen with the password displayed. Users should be cautioned against STOREing screens containing privileged or secure information in non-displaying fields.

November 2, 1983
IRMA UPDATE - PRELIMINARY

REVISION NOTICE

Revision 1.25 of E78 supercedes all previous revisions of the E78 program. Revision 1.25 is an enhancement of the earlier revision of 1.10 which was released in February, 1983.

Revision 1.25 fixes the two known bugs left in 1.10, screens with no attributes and screens being displayed backwards. The first bug occasionally caused screens which contained no attribute characters to be treated as a non-displayed field. E78 now properly handles this situtation. The second bug, in rare instances, caused E78's screen buffer to be displayed backwards when entered as resident from some programs. Running full screen programs, such as spread sheets and full screen text editors aggravated the problem.

Revision 1.25 contains many new features and a new program "GENX" has also been included in the release to provide users a simple approach to modifying the E78 program which does not require modifying the assembly language E78 program. In addition to the GENX program, file transfer utilities for VM/CMS and VMS/TSO operating systems are also included on the same diskette as the executable E78 program. Features added in this revision include the following:

- MOD 4 screen support (80X43). (IBM PC OR PC/XT MUST BE EQUIPED WITH A REVISION C IRMA BOARD.)

- File transfer utilities for CMS and TSO

are now included on the executable E78 diskette.

- Revision 1.25 also remedies the problems with the screen RECALL function.

- Light pen support for applications requiring the IBM 'Selector Pen'. This feature requires that the E78 be used with the IBM Color Display Adapter and a light pen which connects to the display adapter such as the one sold by FTG Data Systems.

- Support of all 16 possible IBM keyboard types, including IBM 'Reserved' types used by IBM for custom keyboards.

- Two keyboard layouts are also provided. The standard default keyboard was designed to accommodate the combination of APL and 3278 functions. The selectable optional keyboard is the same layout as was initially released with IRMA. (Pre-1.20 keyboard) Both of these keyboards are compatible with software previously developed for IRMA.

- ASYNC character input support. This feature allows data entry to IBM mainframes using character serial devices attached to the PC COM1: RS-232 ASYNC card. The interface supports such options as barcode or OCR readers and touch input screens, such as the screen manufactured by Touch Technoloy.

- Complete keyboard reconfiguration ability. All E78 keyboard sequences are now controlled by simple tables which may be user modified through the GENX utility. (Instructions for this procedure are available upon request from the IRMA Technical Support Group.)

- Support of IBM APL-I character set and keyboard when using a display adapter equipped with TAC or STSC's APL*PLUS/PC (Trademark of STSC, Inc.) character generator ROM. In order to support the 3278 and APL functions, the user should NOT select the pre-1.20 keyboard.

- Support of 32 line screens. E78 may be configured to appear to the mainframe as a model 2 (24x80) or model 3 (32x80) 3278 or 3279 with full seven color support. Screens longer than 32 lines are handled by scrolling key functions and an automatic cursor tracking system.

- Support of PC 'look-similars' using 8086 processors, such as the Eagle 1600. Also included is support for hybrid COLOR/MONOCHROME screens, such as used on the COMPAQ portable computer. An option to accomodate the PCXT monochrome cursor is also included.

- A program function is provided to completely disable the display of NON-DISPLAY type fields even when the 'Display Attributes and Non-Display Fields' mode is selected.

- GENX, a generic menu driven program, provides the user with the ablity to customize the .EXE type programs. The GENX program can be used to create customized versions of E78 with specialized keyboard, color, and communications defaults.

- E78 can now be made resident upon execution. With this feature an 'auto-resident' E78 could be placed in the AUTOEXEC.BAT file to make the E78 programs instantly available after machine startup.

GENX

The GENX utility allows the user to configure IRMA to accommodate features and function modes that best suit his/her system requirements. Calibrating the LIGHT PEN, enabling COM1: inputs, setting screen and keyboard types, and selecting PC 'look similar' modifications are all handled by the GENX program.

The standard GENX menu provided with this E78 release includes menu items to select all possible keyboards. It should be noted, however, that selecting a specific keyboard does not necessarily mean that the resulting keyboard configuration is useful. For example, TEXT and APL keyboards should not be selected unless an extended character set is installed in the display adapter board. If one of these keyboards is selected without having the character set installed, the TEXT and APL special characters will be displayed as musical notes and game symbols. Selecting either of the DATA ENTRY keyboards moves the numeric and PF keys into bizarre locations. Making this particular keyboard funtionable requires several decisions to be made. These decisions are based on the specific applications involved and the layout of the PC keyboard versus the 3278 keyboard. If DATA ENTRY keyboards are essential to the application, instructions for altering the keyboard and adding menu options are available upon request from the IRMA Technical Support Group.

The GENX program is executable in two forms One format is for the general user who wants to make use of the normal defaults without altering the keyboard mappings or altering the E78 program to accommodate unique requirements. The second format is for those users who require special configurations that are not included in the default selections. The following discussion describes the first format. The second format is discussed in detail in "Application Note - GENX". This second format is for use primarily in creating specialized keyboard and menu items.

GENX presents the user with a menu of options. The user selects options from this menu that describe the parameters necessary fo operation. Selecting the appropriate options causes the GENX program to 'patch' these options into the executable E78 program. Upon subsequent power-ups, these selected options are the default parameters. If different or additional options are required for other users, a second version of the GENX program should be generated. Therefore, it is feasible that the user will have several customizations of the GENX program to meet each specific application required. Be sure to label each of the custom versions appropriately.

For GENX to operate properly, several files must be available in the current directory. These files are noted below:

GENX.EXE - This file is the actual GENX program which displays the menus and patches the program file.

E78.MAP - This is the symbol table for the program to be modified. This file is produced by the DOS LINK program and contains entrys which tell GENX how to find the various tables and switches which it uses to apply patches to E78. This file is in a simple format and may be TYPEd.

E78.GEN - This is a text file initially prepared by TAC and contains the information used by GENX to display the menus and make the patches required by the user.

Several files are created in the current directory as a result of running GENX. These files are listed below:

CE78.EXE - This is the users customized version of E78. The user should use this name to execute a customized E78.

CE78.LOG - This is a text file in a format similar to E78.GEN mentioned above. This file contains a listing of all patches installed in CE78.EXE.

For this general use of the GENX program, a file called E78GEN.BAT contains th instructions to start the GENX program which produces the customized E78 program, CE78.EXE. Though the above mentioned files are used by the GENX utility, their use and modification is transparent to the user. These same files are also used by the second format for GENX. For the second format, some of the modification of these files must be done by the user. For example, to create a special keyboard modification that is not listed among the default settings, the genfile, E78.GEN, must be modified by the user before the actual CE78.EXE can be generated. However, for most applications, the first format is adequate for creating the customized version of E78.

USING THE GENX PROGRAM WITH MENU FORMAT

1. If you have not already done so, make several duplicate copies of the E78 diskette.
   DO NOT MODIFY THE ORIGINAL DISKETTE!!

2. To set the E78 program defaults, you must be in DOS. After receiving the DOS prompt, A> or B>, or C>, enter E78GEN followed by the 'Enter' key.

3. Following the copyright, trademark, and revision information, a menu is displayed on the CRT. This menu includes the options available. Those items preceeded by a dash are complete as listed; those items preceeded by an equal sign have submenus. Select the appropriate items which describe your system requirements. The menus are listed in the following section, Menu Options.

4. Once all of the desired options have been selected, select item # 99. Exiting the GENX program causes the selected options to become permanant patches to the E78.EXE program. From this point on, to use the customized version of E78.EXE, the user enters CE78 to activate the emulator program. After typing CE78 <ENTER>, the copyright and related information is again displayed on the CRT, followed by USER CUSTOMIZED VERSION.

After completing the GENX routine if the auto-resident patch has not been installed, the DOS prompt will be displayed. Typing CE78 <ENTER> will place you directly into the 3270 mode. Pressing both SHIFT keys simultaneously will return operational control to the PC. To return to 3270 mode CE78<ENTER> must be re-entered.

If the option to install auto-residency has been selected, after exiting the GENX routine, the DOS prompt will be displayed. Type CE78<ENTER> to activate the emulator. The copyright, trademark, and revision information is displayed followed by USER CUSTOMIZED VERSION. The DOS prompt appears again; press both SHIFT keys simultaneously to enter 3270 mode. Pressing the SHIFT keys again returns the operaitonal control to the PC mode. This process must be repeated each time the PC is "booted". (It is not necessary to re-run the GENX program, only the subsequent steps.) If desired the CE78 command may be place in the AUTOEXEC.BAT file, causing it to automatically execute each time the system is booted.

The ability to alternate between modes is available with both non-auto-resident and auto-resident; however, installing the auto-resident patch simplifies this process for the general user.

MENU OPTIONS

After entering E78GEN, the following information will be displayed on the CRT:

```
     E78 Terminal Emulator Customization Menu

 1 - Disable 24th line status display

 3 - Make emulator auto-resident

 5 - Make 2 color mode default
 6 - Make 7 color mode default

 8 - Make SHOW COLUMNS default

10 - Select pre 1.20 keyboard

12 = Select KEYBOARD & SCREEN type
13 = Set LIGHT PEN correction
14 = PC look-alike patches
15 = Set up COM1: input parameters


                              99 = Exit GENX program

Your selection:
```

Selecting item #2 disables the display o the status line.

Selecting item # 3 causes the customized E78 program to become resident upon power up.

Selecting item # 5 makes 2 color the default.

Selecting item # 6 makes 7 color the default.

Selecting item # 8 causes unprotected null fields to be filled with dots, showing the size of the field.

Entering item # 10 selects the pre-1.20 keyboard. This option should not be selecte if the user expects to later use non-typewriter keyboards (eg. APL, TEXT or DATA ENTRY).

Selecting item # 12 causes the following menu to be displayed on the CRT.

```
          Select KEYBOARD & SCREEN type

 1 - Typewriter (default)
 2 - Typewriter w/ Numeric lock
 3 - Typewriter, PSHICO

 5 - APL
 6 - APL w/ Numeric lock
 7 - APL, PSHICO

 9 - Text
10 - Text w/ Numeric lock

12 - Data Entry I
13 - Data Entry I w/ Numeric lock
14 - Data Entry II
15 - Data Entry II w/ Numeric lock

17 - No attached keyboard
18 - Reserved 0000
19 - Reserved 0011
20 - Reserved 1011

22 - Mod 2 Screen (24x80)
23 - Mod 3 Screen (32x80)

98 - Return to previous menu
99 - Exit GENX program

Your Selection:
```

Menu item # 13 should be used only when pressing the LIGHT PEN against the edge of screen field causes the adjacent field to be selected. If this occurs on the left edge of the field, use the MINUS (-) selections. If it occurs on the right edge of the field, use the PLUS (+) selections to calibrate the LIGHT PEN. If none of the available selections results in correct LIGHT PEN operation, the pen is probably broken or internally mis-adjusted. To enable the LIGHT PEN, attach the LIGHT PEN to the display adapter board.

Set LIGHT PEN correction

```
1 - -3 X. . o
2 - -2 . X. o
3 - -1 . . Xo
4 - >0     o
5 - +1     oX. .
6 - +2     o. X.
7 - +3     o. . X
```

98 = Return to previous menu
99 = Exit GENX program

Your selection:

Selecting item # 14 causes the following information to be displayed:

PC look-alike patches

1 - Eagle 1600 screen cleanup

3 - Force Horizontal update sync

5 - Disallow 26 line color display

7 - Always set cursor shape on exit

98 = Return to previous menu
99 = Exit GENX program

Your selection:

Item # 1 eliminates the spurious dots that sometimes appear on the CRT. This selection also enables '8086' type screen accesses and should be selected when IRMA/E78 is used in any 8086 based machine. Item # 3 is necessary to remove or reduce screen flicker on some non-IBM machines, notably the EAGLE PC. Item # 5 should be selected if using the COMPAQ color/monochrome display. Item # 7 solves the PCXT monochrome cursor problem.

Selecting #15 causes the following menu t be displayed:

Setup COM1: input parameters

1 - Enable COM1: character input
2 - Disable COM1: key clicks

4 - Select 4800 baud
5 - Select 2400 baud
6 - Select 1200 baud
7 - Select 300 baud
8 - Select 110 baud

98 = Return to the previous menu
99 = Exit GENX program

Item # 1 enables asychronous character input, such as barcode readers. Item # 2 Disables the key clicks. Items 4 - 8 establishes the baud rate for the device attached to COM1:.

Remember that once the E78 program has been customized into CE78, it is permanant. To change the customization, it is recommended to begin with an unmodified copy of the original diskette as some of the patches can be remodified and some cannot. It is therefore essential to make adequate copies of the original diskette. Once the E78 program has been modified to CE78, to activate the emulator enter CE78 followed by the <ENTER> key.

If the custom emulator was made resident by either the auto-resident patch or the CONTROL HOME function, alternating between the 3270 mode and the PC mode is done by pressing both shift keys simultaneously. To return to the 3270 mode, enter both shift keys again. If the custom emulator program, CE78.EXE, was not made resident, pressing both shift keys will alternate to the PC mode. To return to the 3270 mode, re-enter "CE78".

FILE TRANSFER

The IRMA file transfer programs are compatible with both Rev. 1.1 and 2.0 of DOS and all Revisions of IRMA software and firmware. There are five files necessary to execute the file transfer programs. They are included on the executable E78 diskette:

1. FT78X.EXE - the executable file transfer program for CMS\XEDIT

2. FT78T.EXE - the executable file transfer program for TSO

3. IRMA.XED - IRMA\XEDIT profile for use with the CMS file transfer utility. This file must exist on the host system with the filename IRMA and the filetype XEDIT in order for the FT78X program to work on the IBM-PC. See page 11 for a listing of this file.

4. IRMATABS.OVR - must be on file transfer diskette

5. FTSAMPLE.TXT - a sample text file for testing file transfer.

ENVIRONMENT

The host environment is limited to:

1. VM/CMS - SP, using XEDIT, or
2. MVS/TSO - Using the EDIT function of TSO.

The PC environment must include:

1. 128K of memory.

IRMA FILE TRANSFER UNDER CMS

The file transfer program was designed to be simple to use. A HELP function, plus a question and answer format provides the user with all the information required to transfer files. To begin the file transfer the user must first perform the following:

1. Enter the IRMA.XED profile at mainframe. This file must exist in each USERID that will be using the file transfer program.

2. Be logged on to the mainframe.

It would be useful at this time to run the help function for a listing of the two formats and the switches that can be used with the single line format. A printed copy of this help function is provided on page 28.

ENTER: FT78X/H

Under this file transfer program, it is possible to transfer binary data. The binary data is translated into an intermediate text file format for sending the file to the host. The file is stored on the host in a special format. When the file is transmitted back to a PC, it is received as binary data.

This binary mode was implemented for such operations as the transfer of BAS type files from PC to PC. In order to sucessfully transfer binary data, the IRMA.XEDIT Profile must include the option to enable lower case characters or the host filetype must allow lower case by default.

FORMAT 1 - Question and Answer

PROMPT: Confirm selections prior to transfer? (Y,N)

ENTER: Y for Yes, N for No

This selection causes a line similar to the following text line to appear on the CRT after all the PROMPTS have been answered listing the received file and the save file:

"Receive from host <filename>, save as local <filename>."

This confirmation is followed by

"OK to continue? (Y/N)

If the filenames are listed correctly, enter Y for Yes. If they are not correct, enter N for No and the FT78X program will be terminated. Restart the FT78X program from the beginning to re-specify the operation.

PROMPT: Transfer direction. (R/S)

R = Receive a file on the PC from the host

S = Send a file from the PC to the host

ENTER: R or S

PROMPT: Transfer binary file. (Y/N)

If it is necessary to transfer a file containing binary data, Y for Yes must be entered.

PROMPT: Display copy to CON: (Y/N)

If it is desirable to display the transferring data to the CRT, enter Y for Yes. This will slow the transfer down. It is also not advisable to alternate between PC mode and 3278 mode while the transfer is taking place. Doing so may cause interuptions in the transfer.

For the transfer to take place the program must know the source and destination file. The prompts for supplying information are in the same order no matter whether sending or receiving a file. You must always supply the local filename and host filename and filetype. The FT78X program will interpret which one is the source and which one is the destination.

PROMPT: What is the local filename? (Must be one word)

ENTER: <filename>

PROMPT: Host filename

ENTER: <filename>

PROMPT: What is the file type on the host?

(This is specific to CMS) It is suggested to specify SCRIPT as the file type because it allows a variable length record up to 132 characters. DATA type files allow for a fixed 80 character record length.

In order to transmit a file from the host to the PC, the file must already exist at the host. The destination filename should not exist prior to the transfer. However, if it should exist and be determined to be non-empty, an option to clear the data is offered after all the PROMPTS have been answered. The following question is displayed if the PC is receiving the file and if the filename already exists:

PROMPT: Do you want to delete and write over that file? (Y/N)

ENTER: Y or N

If Y is entered, the file will be replaced with the file from the mainframe.

If N is entered, FT78X will prompt the user to respecify the filename.

If the host filename is the destination (on send) and it is determined to be non-empty, am option to clear the data is offered. Unlike the local file which can be deleted, only the contents of the host file are deleted, not the actual file.

PROMPT: Host filename already exists, clear it? (Y/N)

ENTER: Y or N

Y will clear the file and allow user to continue
N will terminate the file transfer program.

If the user has specified the confirm option, the text line specifing the file to receive and to send will be displayed on the CRT. At this point the user has the option of continuing or terminating the file transfer program.

FORMAT 2 - Single command line

ENTER: FT78X{switches} <local_filename>
<host_filename> <host_filetype>

This second format places all the necessary information in one command line. It also allows you to add "switches" to the command line for verification and confirmation of information used in the transfer. This second format is also useful with batch files. The switches are listed on the following page and in the FT78X/H listing.

FT78X/H - HELP FILE

There are two formats for using the file transfer utility.

```
    FT78X
or
    FT78X {switches} <local_filename>
         <host_filename> <host_filetype>
```

The first format causes a question and answer dialogue to be provided to specify the information. The second format contains all of the information in the command line and is useful with batch files. If fewer than the required number of items are included with the second format, the user will be prompted for the remaining items.

Global Switches

/B - Binary transfer mode

/C - Confirm the pending operation prior to execution

/F - Display host file size on receive

/O - Override the delete-file query and automatically delete the local file if it exists on the receive

/R - Receive a file from the host

/S - Send a file to the host

/V - Display data on CON: during the transfer

IRMA.XED PROFILE

This is the IRMA\XEDIT profile for use with the file transfer utility. You must enter this profile at the host with the filename IRMA and the file type of XEDIT in order for FT78X to work on the IBM-PC. It must be entered exactly as shown here, as this profile sets up the functions that the transfer program uses. A copy of this profile must exist in each USERID using the file transfer program.

```
SET SCALE OFF
SET NUMBER ON
SET CURLINE ON 3
SET CMDLINE BOTTOM
SET NULLS ON
SET PF6 QQUIT
SET PF18 QQUIT
SET PF7 FORWARD
SET PF19 FORWARD
SET PF8 TOP
SET PF20 TOP
SET PF9 FILE
SET PF21 FILE
```

The following are OPTIONAL items which may be included if desired.

```
SET CASED MIXED    to allow lower
                   case as the default
                   (Must be set for
                   binary transfer.)
```

NOTES ON ASCII TO EBCDIC CONVERSION

GENERAL CONVERSION LOSSES

In all cases, the following ASCII characters are translated by IRMATABS.OVR. into the equivalent EBCDIC characters. The conversion to EBCDIC occurs when a file is sent to the host from the PC.

ASCII	EBCDIC
[	¢
]	!
^	¬

CMS/XEDIT OPERATION LOSSES

Due to the use of POWERINPUT in XEDIT to improve performance on a SEND, two of the ASCII characters are not available. The "^" and "[" are reserved as special characters. The ^ and [characters are effectively changed to spaces and are NOT recoverable on subsequent receives. A warning to this effect will be displayed on the CRT if either of these characters are encountered during a send operation.

IRMA FILE TRANSFER UNDER TSO

The file transfer program for TSO was also designed to be simple to use. A HELP function, plus a question and answer format provides the user with all the information required to transfer files. To begin the file transfer the user must first perform the following:

1. Be logged on to the mainframe.

2. Must be in TSO with READY prompt.

It would be useful at this time to run the help function for a listing of the two formats and the switches that can be used with the single line format. A printed copy of the help function is provided on page 40.

ENTER: FT78T/H

Under this file transfer program, it is possible to transfer binary data. The binary data is translated into an intermediate text file format for sending the file to the host. The file is stored on the host in a special format. When the file is transmitted back to a PC, it is received as binary data. This binary mode was implemented for such operations as the transfer of .BAS type files from PC to PC. When using the binary transfer mode, the user must specify the ASIS operand or select a dataset type which allows lower case by default.

The TSO version of file transfer allows the user to fully specify the dataset name and associated operands to be used with EDIT. The data set name must be a single word with no imbedded spaces. However, the word can be as detailed as necessary. This allows easy specification of partitioned datasets. The operands can also be as detailed or as simple as necessary and they are not limited to a single word. This provides the users with the means to add items such as, ASIS (lower case) and/or NONUM (un-numbered dataset). Several examples are included later in this discussion.

Format 1 - Question and Answer

PROMPT: Confirm selections prior to transfer? (Y,N)

ENTER: Y for Yes, N for No

This selection causes the following text line to appear on the CRT after all the PROMPTS have been answered listing the received file and the save file:

"Receive from host <filename>, save as local <filename>."

This confirmation is followed by:

"OK to continue? (Y/N)

If the filenames are listed correctly, enter Y for Yes. If they are not correct, enter N for No and the FT78T program will be terminated. Restart the FT78T program from the beginning.

PROMPT: Transfer direction. (R/S)

R = Receive a file on the PC from the host

S = Send a file from the PC to the host

ENTER: R or S

PROMPT: Transfer binary file. (Y/N)

If it is necessary to transfer a file containing binary data, Y for Yes must be entered.

PROMPT: Display copy to CON: (Y/N)

If it is desirable to display th transferring data to the CRT, enter Y for Yes. This will slow the transfer down. It is also not advisable to alternate between PC mode and 3278 mode while the transfer is taking place. Doing so may cause interuptions in the transfer.

For the transfer to take place the program must know the source and destination file. The prompts for supplying information are in the same order no matter whether sending or receiving a file. You must always supply the local filename and host filename and filetype. The FT78T program will interpret which one is the source and which one is the destination.

PROMPT: What is the local filename? (Must be one word)

ENTER: <filename>

PROMPT: What is the Data-set-name?

ENTER: <data-set-name>

PROMPT: What are the operands for host?

ENTER: <operands> [none]

The "[none]" is the default. If no operands are required, simply enter a <NL or ENTER>.

In order to transmit a file from the host to the PC, the file must already exist at the host. The destination filename should not exist prior to the transfer. However, if it should exist and be determined to be non-empty, an option to clear the data is offered after all the PROMPTS have been answered. The following question is displayed if the PC is receiving the file and if the filename already exists:

PROMPT: Do you want to delete and write over that file? (Y/N)

ENTER: Y or N

If Y is entered, the file will be replaced with the file from the mainframe.

If N is entered, FT78T will prompt the user to respecify the filename.

If the host filename is the destination (on send) and it is determined to be non-empty, an option to clear the data is offered. Unlike the local file which can be deleted, only the contents of the host file are deleted, not the actual file.

PROMPT: Host file already exists, clear it? (Y/N)

ENTER: Y or N

Y will clear the file and allow the user to continue.
N terminates the file transfer program.

If the user has specified the confirm option, the text line specifing the file to receive and to send will be displayed on the CRT. At this point the user has the option of continuing or restarting the file transfer program.

Under TSO files usually contain line numbers; however, files on the PC do not have line numbers. Files sent to the host from the PC will be automatically numbered, beginning at 10 and incrementing by 10 for a maximum number of 9,999 lines of text. Files received by the PC will be temporarily renumbered, beginning at 1 and incrementing by 1 for a maximum of 99,999 lines. These line numbers are automatically removed before writing the file to the diskette on the PC.

Additionally, un-numbered data sets can be sent and received by including the NONUM operand. This causes FT78T to ignore line numbers entirely and not attempt to renumbe the data set on a receive.

FORMAT 2 - Single command line

ENTER: FT78T{switches} <local_filename>
<data_set_name> <operands>

This second format places all the necessary information in one command line. It also allows you to add "switches" to the command line for verification and confirmation of information used in the transfer. This second format is also useful with batch files.

For easy reference, a copy of FT78T/H is included with the update.

FT78T/H - HELP FILE

There are two formats for using the file transfer utility.

```
    FT78T
or
    FT78T {switches} <local_filename>
         <host_filename> <operands>
```

The first format causes a question and answer dialogue to be provided to specify the information. The second format contains all of the information in the command line and is useful with batch files. If fewer than the required number of items are included with the second format, the user will be prompted for the remaining items.

Global Switches

/B - Binary tranfer mode

/C - Confirm the pending operation prior to execution.

/D - Override the delete-file querry and automatically delete the local file if it exists on the receive. At the host on send, deletes the only the contents of file, not actual file.

/R - Receive a file from the host.

/S - Send a file to the host (See note 1)

/V - Display data on CON: during the transfer.

SAMPLE TSO TRANSFERS

SAMPLE ONE FTSAMPLE.TXT

A copy of the sample test file, FTSAMPLE.TXT, is to be sent to the host as the numbered, sequential data set, TEST.TXT. The following dialogue would achieve the transfer:

PROMPT	RESPONSE
A>	FT78T <NL or ENTER>
Confirm selections prior to transfer (Y/N)	N <NL or ENTER>
Transfer direction (R/S)	S <NL or ENTER>
Transfer binary file (Y/N)	N <NL or ENTER>
Display copy to CON:	N <NL or ENTER>
Local filename:	FTSAMPLE.TXT <NL or ENTER>
Data set name:	TEST.TEXT <NL or ENTER>
Operands: [none]:	<NL or ENTER>

Transfer takes place at this point.

SAMPLE TWO PARTITION TRANSFER

A copy of the member, MEM1 in th partitioned, numbered data set PPS.DATA, is to be stored in the IBM PC as FILE1.DAT.

PROMPT	RESPONSE
A>	FT78T <NL or ENTER>
Confirm selections prior to transfer (Y/N)	N <NL or ENTER>
Transfer direction (R/S)	R <NL or ENTER>
Transfer binary file (Y/N)	N <NL or ENTER>
Display copy to CON:	N <NL or ENTEF
Local filename:	FILE1.DAT <NL or ENTER>
Data-set-name:	1. PPS.DATA(MEM1)<NL or ENTER> or 2. PPS(MEM1) <NL or ENTER>
Operands [none]:	1. <NL or ENTER> or 2. DATA <NL or ENTER>

When entering the data-set-name, the firs entry includes the filetype and therefore does not require the file type to specified as an operand.

If using the second entry, it is necessary to enter the file type as an operand. For this example when entering the data-set-name and the operands, match the numbered responses for each entry.

Transfer takes place at this point.

SAMPLE THREE BINARY TRANSFER

The BASICA program, PROG.BAS, is to be sent to the host as TEMPPROG.DATA. This requires lower cas on the host as the binary transfer mode will be used.

PROMPT	RESPONSE
A>	FT78T <NL or ENTER>
Confirm selections prior to transfer (Y/N)	N <NL or ENTER>
Transfer direction (R/S)	S <NL or ENTER>
Transfer binary file (Y/N)	Y <NL or ENTER>
Display copy to CON:	N <NL or ENTER>
Local filename:	PROG.BAS <NL or ENTER>
Data set name:	TEMPPROG.DATA <NL or ENTER>
Operands: [none]:	ASIS <NL or ENTER>

Transfer takes place at this point.

SAMPLE FOUR SINGLE COMMAND LINE

To perform the same transfer as listed in the previous example using thesingle command line format, enter the following data.

```
FT78T/S/B PROG.BAS TEMPPROG.DATA ASIS
```

or

```
FT78T/S/B PROG.BAS TEMPPROG DATA ASIS
```

Note that in the second case, DATA is specified as an operand and in the first case, it is specified as part of the data-set-name.

SAMPLE FIVE PC TO PC TRANSFER

Using the binary transfer mode, first uploa the specified file to the mainframe. Once it is located on the mainframe in its storage state, other PC's can access this file and have it downloaded to a PC using the same file transfer utility. The file will be sent to the PC in its original state. (.BAS or Wordstar(TM) format.)

SAMPLE SIX TO SPECIFIED DISK DRIVE

To specifiy an alternate disk drive for the file coming from the mainframe, simply preceed the destination filename with the device name, such as :Bfilename or :C filename.

THE ASCII TO EBCDIC CONVERSION

GENERAL CONVERSION LOSSES

In all cases, the following ASCII characters are translated by IRMATABS.OVR. into the equivalent EBCDIC characters. The conversion to EBCDIC occurs when a file is sent to the host from the PC.

ASCII	EBCDIC
[	¢
]	!
^	¬

OPERATIONAL LOSSES

The EDIT mode of TSO changes the following characters to a ":" during transfer:

\
`
{
}
~

This causes a loss of information which is non-recoverable even though E78 echos the ASCII symbol.

For example:

This is a table on the IBM/PC as sent to the mainframe using TSO/EDIT:

```
 0 1 2 3 4 5 6 7 8 9 A B C D E F
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   ! " # $ % & ' ( ) * + , - . /
 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
 @ A B C D E F G H I J K L M N O
 P Q R S T U V W X Y Z [ \ ] ^ _
 ` a b c d e f g h i j k l m n o
 p q r s t u v w x y z { | } ~
```

This is the table as it was received back from the mainframe on the PC:

```
 0 1 2 3 4 5 6 7 8 9 A B C D E F
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   ! " # $ % & ' ( ) * + , - . /
 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
 @ A B C D E F G H I J K L M N O
 P Q R S T U V W X Y Z [ : ] ^ _
 : a b c d e f g h i j k l m n o
 p q r s t u v w x y z : | : :
```

In the event that the transfer is not accomplished correctly, refer to th following profile for the proper settings of various options. Set your profile this way and try the transfer again. This profile may or may not be complete for your system; it is offered here as a suggestion.

```
CHAR(0)
LINE(0)
PROMPT
INTERCOM
NOPAUSE
NOMSGID
NOMODE
NOWTPMSG
NORECOVER
DEFAULT LINE/CHARACTER DELETE CHARACTERS IN
 EFFECT FOR THIS TERMINAL
```

PROM INSTALLATION

IT IS RECOMMENDED THAT THIS INSTALLATION GUIDE BE READ IN ITS ENTIRITY PRIOR TO BEGINNGING THE ACTUAL INSTALLATION.

1. Never attempt to remove the IRMA board without first turning off all power to the IBM PC. Always use care when removing the board from the card cage.

2. Refer to Figure 1 for the location of the IRMA firmware proms.

3. To remove an existing PROM, pry gently and evenly between the PROM and its socket with a narrow blades screw driver or small knife to raise the PROM from its socket. After the PROM is partially raised from its socket, take hold of the PROM with your fingers and carefully withdraw it from the socket.

4. To insert the replacement PROM(s), first determine the PIN 1 location on the PROM. PIN 1 is indicated by an embossed dot on the PROM or it is the pin to the left of the semi-circle notch on the TOP of the PROM. See Figure II.

5. The PROMs must be installed with PIN 1 in the upper left hand coner of the PROM socket. Be sure the IRMA board is oriented as indicated in Figure 1.

6. Align the pins of the PROM with the pin holes in the socket. Be sure that the pins are aligned properly.

7. Press the PROM gently into its socket.

8. Check the pins on the inserted PROM to insure that none of the pins were bent or failed to mate with the holes in the socket. If any pins are bent or not mated, gently remove the PROM from the socket. Staighten any pins and re-align the pins. Re-insert the PROM.

Figure I - Location of IRMA firmware PROMS

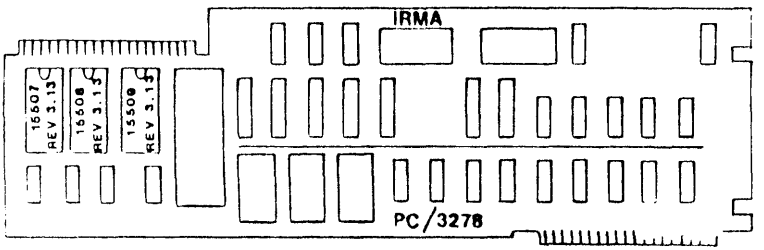


Figure II - Chip Orientation