FOR-488 MANUAL

MBC MetraByte Corporation

3/85 60880A

MetraByte Corporation, 440 Myles Standish Boulevard, Taunton Mass. 02780 U.S.A. Phone: (617) - 880 - 3000 Telex: 503989

INDEX

INDEX

1.0 INTRODUCTION

The GPIBDVR.COM program is a DOS resident driver Extension for use with the MetraByte IE-488 interface card. The resident driver provides all the same functions and commands as the Basic ROM Interpreter plus the additional capability of linking to all high level and low level languages on the market today. The resident driver Extension is written in 8086/8088 assembly language and will run an any MS-DOS (1) compatible computer using the 8088/8086 instruction set.

The driver is accessed via a software interrupt in the range of F1 to FF (hex). This minimizes the effects of using a large portion of compiler code area when using the interpreter.

The following sections will describe how to load the driver extension and change the entrance characteristics for full multi-user / multi-task operation. The driver is capable of handling two IE-488 boards under a multi-user / multi-task environment. All COMMAND STRING formats are the same as explained in the IE-488 manual. This manual will explain the variations of the different string and variable pointer constraints of the various compilers on the market. This manual has a separate operation section for each of the languages specified (FORTRAN,TURBO PASCAL, and LARGE/SMALL models).

(1) MS-DOS is a trade mark of MicroSoft Corp.

2.0 GPIB (IEEE-488) RESIDENT DOS DRIVER FOR IE488 INTERFACE

2.1 GPIBDVR.COM OVERVIEW

The file GPIBDVR.COM is the main DOS resident driver extension. This file should be loaded once during power up or before the IE-488 board is used. The functions and commands the DOS resident driver extension will recognize are the same as shown in Chapter 4.0 (USERS COMMANDS) in this manual and the IE-488 manual. The DOS resident driver extension intercepts the ROM interpreter at the entry point and conditions the variable pointers on the STACK to conform to the ROM command line interpreter. This way only a small patch is required to link to any new compiler presented on the market today or in the future. This resident interpreter will in fact link to any language with very little programming. The libraries included are FORTRAN and BASIC libraries which setup the pointers to the variables in various ways and save special registers like SI and DI that may be used with different compilers such as the new MicroSoft C compiler version 3.0. TURBO PASCAL (trade name of Borland International) file is included (IE488TUR.COM) and may be changed to IE488.COM when compiling the .PAS program. There is a separate section on TURBO PASCAL programming in this manual.

2.2 LOADING THE GPIBDVR.COM FILE

Loading the file is straightforward. The default vector link is set at F1 hex. This means an INT 0F1H instruction will enter the extension. Once the driver extension is loaded only restarting the system (powerup) will remove the driver extension. The resident driver extension cannot be reloaded at the same vector, an error message will be displayed acknowledging this action. Entering the following will load and keep resident the driver interpreter.

```
A>GPIBDVR <enter>
METRABYTE GPIB DRIVER LOADED (c) 1985
A>
```

The driver is now part of the DOS system and is accessed via the **F1** hex vector software interrupt.

LOADING AT A DIFFERENT VECTOR

To load the GPIBDVR.COM interpreter at a different vector between **F1** and **FF** hex just enter the vector after the file name.

2.0 GPIB (IEEE-488) RESIDENT DOS DRIVER FOR IE488 INTERFACE

```
A>GPIBDVR F3 <enter>

METRABYTE GPIB DRIVER LOADED (c) 1985

A>
```

The driver is loaded and may be accessed via the **F3** hex software interrupt. This VECTOR is loaded in the IE488 ROM interpreter card when initialized in order to inform the ROM interpreter where the driver is located. Access to the driver is automatic when the interpreter is called.

If the driver is reloaded at the same vector such as:

```
A>GPIBDVR F3 <enter>

METRABYTE GPIB DRIVER ALREADY INSTALLED.

A>
```

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

The DOS resident driver Extension is setup for4 variations of linking. One of the main differences between different types of compilers is the way STRING variables are handled. The FORTRAN compiler (MicroSoft ver 3.2) does not pass the byte count for the character variable. This is a fixed length string assignment. The TURBO PASCAL (Borland Int.) and the MicrSoft PASCAL pass the byte count as the first byte in the string. The BASIC Compiler (MicroSoft) uses a four byte STRING DESCRIPTER where the first word is the byte count and the second word is the offset pointing to the first character. The BASIC Interpreter (MicroSoft) uses a three byte string descripter where the first byte is the byte count (255 max.) and the 2nd and 3rd byte are the offset pointer to the first character in the string. Some compilers pass the variable data on the stack while others pass either offsets (2 byte pointers) or both segments and offsets (4 byte pointers). The variations are endless and can become confusing to over come in a multi-language environment. The DOS resident driver extension software provides the basic tools to link to all types of compilers while maintaining full command string similarity. Thisallows easy upgrades in the event the compiler manufacturers change the assembly language link requirements to the compiler.

The ROM interpreter has five entry points and two coded return points for reseting the variables on the STACK before returning with a FAR RET instruction. The following is a description of the interpreter entrance requirements.

The entrance offsets are:

1: FULL STRING DESCRIPTER 2 BYTE POINTERS (DS = DATA SEGMENT)
DOS RESIDENT DRIVER EXTENSION NOT REQUIRED
ASSEMBLY LANGUAGE PREFERRED LINK ENTRY

ROMSEG:0000H	--- BASIC INTERPRETER (3 byte string desc.)
ROMSEG:0002H	--- BASIC COMPILER (4 byte string desc.)

2: NO STRING DESCRIPTER 4 BYTE VARIABLE POINTERS
(requires DOS resident driver extension)
LIBRARY FILE -- **IE488LRG.LIB**

ROMSEG:0006H --- LARGE MODEL (SEG:OFF variable pointers)

3: NO STRING DESCRIPTER 2 BYTE VARIABLE POINTERS
(requires DOS resident driver extension)
LIBRARY FILE -- **IE488SML.LIB**

ROMSEG:000AH --- SMALL MODEL (OFFSET variable pointers)
DS = data segment for all variables

4: SPECIAL CASE ENTRANCE FOR ADVANCED PROGRAMMERS

CMDLINE --- USER SOFTWARE LOADS COMMANDS STRING
AND DATA/FLAG SEGMENTS THEN EXECUTES.

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

ROM/RAM MEMORY MAP

The following is a memory layout of the 16 K byte IE488 interpreter. The GPIB Interpreter contains 12K byte of ROM and 4K bytes of static ram. **Entry points 0006 & 000A requirethe DOS resident driver extension. Entry points 0000,0002,0074 are fully implemented in the ROM interpreter and do not require the DOS driver extension.**

GPIB IEEE488 16 K BYTE INTERPRETER MAP

```
HEX ADDRESS
              |--------------------------------------------|
ROMSEG:0000   |  3 byte string descripter entrance point   |
              |                                            |
ROMSEG:0002   |  4 byte string descripter entrance point   |
              |                                            |
ROMSEG:0006   |  No string desc. (4 byte variable pointers)|
              |                                            |
ROMSEG:000A   |  No string desc. (2 byte variable pointers)|
              |                                            |
ROMSEG:0074   |  command line interpreter entry point      |
              |   vseg:voff,  fseg:foff variables set      |
              |     advanced programming entry point       |
              |                                            |
              |           {12 K ROM interpreter}           |
              |                                            |
ROMSEG:3000   |------------ RAM BUFFER BEGINS -------------|
              |    INTERNAL RAM BUFFERS FOR INTERPRETER    |
              |                2 Kbytes                    |
              |                                            |
              |                                            |
ROMSEG:3800   |--------------------------------------------|
              |                                            |
              |                                            |
              |                                            |
              |                                            |
              |                                            |
              |                                            |
              |       USER RAM AREA FOR SCRATCH PAD        |
              |              2048 bytes                    |
              |         NOT USED BY INTERPRETER            |
ROMSEG:3FFF   |---------- END OF RAM BUFFERS --------------|
```

ROMSEG = is the switch address the user selected at installation. This is on a 16K boundary anywhere in the 1 Megbyte address space.

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

3.1 ENTRANCE REQUIREMENTS FOR ROMSEG:0000H HIGH LEVEL LANGUAGES

Thisentry point requires the use of a full string descripter of the three byte type. The first byte will contain the byte count and the next two bytes contain the offset into the DS data segment of the first character in the string. The FLAG and BASE_ADDR variables are always 2 byte integers and are pointed to by the offset pushed on the stack. The last variable is the DATA VARIABLE which may be string or integer. If the VARIABLE is a string then the offset into the DS segment will point to the three byte string descripter. The first variable COMMAND is always a string and is decoded that way. The command string will contain a character which will define the VARIABLE type when interpreted, (see COMMANDS in the IE488 manual chapter 4.0). The BASIC Interpreter uses this entry point when a CALL IE488(var...) is executed.

ASSEMBLY LANGUAGE LINK

TheIE488 interface card also allows the user to use all the user commands as described in section 4.0 (USER COMMANDS) using the same parameter passing conventions as the BASIC Interpreter. The user should be familiar with the 8086/8088 assembly language format before attempting to utilize this function. The user will initiate a FAR CALL to the ROM (the address of the switch settings selected for the ROM segment on the IE488). The Stack is used to transfer all variable pointers and data. The DS register is the data segment pointer for the variable. The segment will be the same value as the switch setting on the IE488 interface board. The user should save any register contents which he does not want destroyed. The interface to the ROM Interpreter has two entry points, the first starting at the ROM_SEG:0000 and the second at ROM_SEG:0002. The main difference between the two is the way in which the string variables are interpreted. The first entry at ROM_SEG:0000 expects the string descripter to be three bytes. This limits the string length to 255 bytes. The second entry point at ROM_SEG:0002 expects the string descripter to be four bytes long, thus allowing a maximum of 32767 bytes for a string length (15 bits). The example following uses the first entry and sets up the string descripter accordingly. The second entry point is primarily used for the IBM Basic Compiler link which uses a four byte string descripter. The interpreter assumes that DS is the only data segment for the variables passed to it and a correct DS should be insured before entry to the Interpreter.

There is also available an assembly language macro library for the IEEE488 interface board which allows the use of MACRO's similar to the Basic CALL statement for all the commands. This IEEE488 MACRO library allows the user to link to assembly language with the same format as the command string.

EXAMPLE:

```
DATA   SEGMENT   DATA
;------------------------- BOARD CONTROL DATA -------------------

     IE488_ROM_SEG   DD   0C0000000H   ;C000:0000 pointer to ROMS

     RTN_FLAG        DW   0000        ;return flag code variable
     BASE_ADDRESS    DW   0000        ;board number 0 (first board)

;-------------------------  COMMAND STRING ---------------------
;        string descripter same as the Basic Interpreter

     CMD_STRING      DB   'OUTPUT 11,12,14[$,2,15]'

     CMD_DESCRP      DB   $ - CMD_STRING               ;Byte count
                     DW   CMD_STRING

;--------------- STRING DESCRIPTER / DATA ARRAY ----------------
;        string descripter same as the Basic Interpreter


     DATA_ARRAY     DB  'THIS  IS THE DATA TO TRANSFER',10,12

     DATA_DESCRP    DB  $ - DATA_ARRAY             ;byte count
                    DW  DATA_ARRAY                 ;pointer to data

;------------------- VARIABLE POINTERS FOR COMMAND -------------

     VARIABLE_1      DW   OFFSET CMD_DESCRPT        'command
     VARIABLE_2      DW   OFFSET STRING_DATA        'data string
     VARIABLE_3      DW   OFFSET RTN_FLAG           'return flags
     VARIABLE_4      DW   OFFSET BASE_ADDRESS       'board number

DATA ENDS

;------------------- SETUP STACK AND EXECUTE COMMAND ------------

IE488     PROC NEAR

          MOV SI,OFFSET VARIABLE_1      ;get pointers
          PUSH [SI]                     ;stack command
          PUSH [SI+2]                   ;stack variable
          PUSH [SI+4]                   ;stack flag
          PUSH [SI+6]                   ;stack base address
          CALL DWORD PTR IE488_ROM_SEG  ;call device driver

RETURN:   CMP RTN_FLAG,00               ;any errors on return ?
          JNE ERROR_HANDLER             ;exit to error handler
     .... continue users program ......

IE488     ENDP
```

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

The stack FRAME or STRUCTURE at the ROM entry point is shown. This structure must not be changed in any way. The data segment for all pointers is assumed to be the DS segment register. This register must be set before the CALL FAR to the rom segment.

```
STACK_FRAME    STRUC

           BASE_ADDR ESS DD ;base address pointer
           RTN_FLAG        DW   ;flag return pointer
           STRING_DATA     DW   ;variable pointer
           CMD_DESCRPT     DW   ;command variable pointer

STACK_FRAME    ENDS
```

The resident DOS interpreter assumes a Segment and offset for each variable passed in the CALL statement.

3.2 ENTRANCE REQUIREMENTS FOR ROMSEG:0006H HIGH LEVEL LANGUAGES

This entrance is used with the DOS resident driver expansion and must be installed to function. If the DOS expansion is not installed an error message will be displayed on the screen and the system will halt.

This entrance assumes the pointers on the stack are four byte type SEGMENT:OFFSET pointers. If strings are used the variable pointers point to the first character in the string. All strings are enclosed in quotation marks, ("this is a string"). Fortran uses apostrophe marks ('...') to define a string, therefore to define a string the user would enclose the quotation marks in apostrophe marks, ('"this is a string"'). This allows variable string lengths of up to 64k bytes in size. The DMA vectors allow string transfer sizes up to 64k bytes also. Since Fortran only allows 127 byte string length, the user may use an integer array and convert it at a later date, this technique is allowed by the interpreter. All output commands using string variables must use the enclosed quotation technique else an error message will be generated. No string descripter is used for this entrance. This entrance is considered a LARGE model library and has the file name IE488LRG.LIB. This is the link the FORTRAN compiler uses to connect to FORTRAN (MicroSoft ver 3.2). The Large C compiler ver 3.0 may be configured to handle this type of string and stack format easily.

THE INTERPRETER IS EXPECTING THE STACK TO BE SET AS FOLLOWS WITH NO DEVIATIONS.

```
 GPIB_FRAME     STRUC

                BASE_ADDRESS DD ;base address pointer
                RTN_FLAG        DD  ;flag return pointer
                STRING_DATA     DD  ;variable pointer
                CMD_DESCRPT     DD  ;command variable pointer

 GPIB_FRAME     ENDS
```

All string variables are handled in the same manor. The VARIABLE string pointer will point to the first character of the string, and the string will be enclosed in quotes ("..."). The maximum length for output is 127 bytes maximum.

This method was designed to accommodate the different compilers which do not pass string descripter pointers to assembly language subroutines such as the Fortran Compiler.

3.3 ENTRANCE REQUIREMENTS FOR ROMSEG:000AH HIGH LEVEL LANGUAGES

This entry point is similar to the previous ROMSEG:0006H except the variable pointers on the stack are two byte offsets which pointto the variables passed in the call statement. The segment is assumed to be the DS segment register. This is considered a small model linker which allows only one data segment and one code segment. All string variables are handled in the same manner as the LARGE model entry point.

THE INTERPRETER IS EXPECTING THE STACK TO BE SET AS FOLLOWS WITH NO DEVIATIONS.

```
 GPIB_FRAME      STRUC

                 BASE_ADDRESS DD ;base address pointer
                 RTN_FLAG      DW   ;flag return pointer
                 STRING_DATA   DW   ;variable pointer
                 CMD_DESCRPT   DW   ;command variable pointer

 GPIB_FRAME      ENDS
```

All string variables are handled in the same manner. The VARIABLE string pointer will point to the first character of the string, and the string will be enclosed in quotes ("..."). The maximum length for output is 127 bytes maximum.

The DOS resident driver expansion must be installed for this link to function properly. An error message will be displayed and the system will halt if entry is attempted with no DOS driver.

3.4 ENTRANCE REQUIREMENTS FOR ROMSEG:0074H HIGH LEVEL LANGUAGES

This entry point requires some advanced programming experience to use. It is the command line interpreter parse section. The ROM interpreter consists of three partitions, the Command Line Interpreter, The Command Line device builder, and the command execution module. These three modules are very independent functioning modules which share a common variable array of RAM memory starting at offset 3000h and ending at 37FFh. The RAM stores the initial command string transmitted by the compiler and the four byte variable pointers to the data VARIABLE and the four byte pointer to the return FLAG variable. The board BASE ADDRESS is decoded and stored as a word in the RAM buffer. If the above links are not suitable for the current compiler being used, users may write their own. The user is required to complete three section of code:-

1. store the command string in the interpreter's command string buffer and add the "*" character at the end of the string.

2. set the data VARIABLE's segment and offset pointers in the RAM buffer.

3. set a group of interpreter flags to identify the return type from the interpreter and insure the return address (segment:offset) is on the stack.

Once the above has been completed the user may enter the interpreter and execute the specified command. A list of the pointer offsets of the RAM are specified for custom applications.

IE488 INTERPRETER RAM BUFFER POINTERS

VARIABLE NAME	OFFSET HEX	DESCRIPTION
RAMPTR	3000	;First buffer location
BASADR	300C	;base address or board #
CMPLR	300A	;compiler type flag
FLGSEG	300E	;FLAG variable segment pointer
FLGOFF	3010	;FLAG variable offset pointer
VAROFF	3012	;data VARIABLE offset pointer
VARSEG	3014	;data VARIABLE segment pointer
TPASC	30A2	;type of PASCAL compiler
IEEBSY	30AE	;interpreter busy flag
CMDSTR	30D8	;interpreter command string pointer

```
ROMENTRY   DD  0XXXX0074H      ;entry point for ROM
                               ;xxxx = THE SELECTED ROM SEGMENT
```

The following example uses this technique to intercept the ROM interpreter and initialize the ROM interpreter for string variables where the first character in the string is the byte count. This is typical of programs like PASCAL and TURBO

PASCAL(1). Due to the unique external call procedure that Turbo Pascal uses this would be a good example of the advanced level interface.

There are two constraints the external procedure in Turbo Pascal has to over come. The first is the relative position of the IP register for the procedure which leads to the second. All references to variables in the external procedure must be with the DS register or the Stack. This is due to the SEGMENT technique used by the 8086/88/286 type processor series. The problem is to get the IP register contents at the entry point of the external procedure. Since the IP register cannot be pushed on the stack and any near call is absolute IP this can create a problem. The GPIBDVR.COM resident DOS driver has a program to return the CS:IP registers to the user upon request. This is *accomplished by loading the AX register with 81F1 hex and issuing* an interrupt to where the GPIBDVR driver was loaded. In this example it was assumed loaded at 0F1 hex. This interrupt function will return the CS:IP register in DX:DI of the next instruction to be executed. This utility may be used by any program to obtain the CS:IP registers. All variables internal to the procedure will be referenced as [DI]+variable.

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

```
PAGE 58,132
TITLE   ** IEEE-488 TURBO PASCAL MODEL LIBRARY (STRNG LENGTH) **
SUBTTL  -     --------- GENERAL DESCRIPTION ------------    -

COMMENT \
```

This driver library is designed for any compiler which uses a FOUR byte pointer on the stack. The library will use string descripter to get the byte count and point to the first character of the string. The variable pointer will point to the string descripter. The first byte of the string descripter will contain the string byte count and bytes 2 and 3 will point to the first character of the string to be processed.

The library links to the ROM's via a FAR JUMP to ROMSEG:0002 hex. The ROM segment may be changed by the user, however it is set at 0C000:0000 hex by default. The program CHGVEC.EXE will allow the user to change the segment if required.

The library link sets up the stack to look like a basic interpreter and runs accordingly. This means making one pass through the command string to see if the variable is a string, and if yes then set up a string descripter.

CALL SEQUENCE:

```
type
        CMD = string [127];
        DAX = string [50];
        FLG =Integer;
        BAD = Integer;

var
        C:COMMAND = 'OUTPUT 12[$]'
        F:FLG = 0
        B:BAD = 0
        V:VAX ='

begin

IE488(var C:CMD,var V:VAX,var F:FLG,var B:BAD);external IE488.COM

end
```

If the DOS resident driver is not installed then an error message will be displayed and the program will halt.

The compilers this model will work with are:

TURBO PASCAL   Ver 2.0 and higher

e.c.\

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

```
PAGE
SUBTTL  * LINK SEGMENT INITIALIZATION FOR SMALL MODEL *
SUBTTL



;------------ STACK VARIABLE DATA STRUCTURE -------------


FRAME   STRUC

        SAVEBP          DW      ?  ;BP register
        RETURN_ADDRESS  DW      ?  ;return address pointer
        BRDADDR         DD      ?  ;board address pointer
        FLAG            DD      ?  ;flag variable pointer
        VARIABLE        DD      ?  ;data variable pointer
        COMMAND         DD      ?  ;command string pointer

FRAME   ENDS


;--------------- SEGMENT IDENTIFIERS ---------------------



DATA    SEGMENT PUBLIC  'DATA'
DATA    ENDS


IE488TUR  SEGMENT   BYTE  'CODE'
DGROUP    GROUP DATA
          ASSUME CS:IE488TUR, DS:DGROUP, ES:DGROUP, SS:DGROUP

;--------------------------------------------------------------
PAGE
SUBTTL  **** MAIN TURBO PASCAL LINK CODE ****
SUBTTL

IE488   PROC    NEAR

MOV     AX,81F1H                ;get IP in DI
INT     OF1H                    ;get this IP group
JMP     CONECT                  ;execute group

;----- VARIABLE DATA POINTERS AND IDENTIFIERS -------------

RAMPTR  EQU     3000H           ;Interpreter ram pointer
TPASC   EQU     30A2H           ;pascal compiler code pointer
CMPLR   EQU     300AH           ;reset compiler flag
IEEBSY  EQU     30AEH           ;ROM busy flag
CMDSTR  EQU     30D8H           ;command string pointer
ERFLG   EQU     3010H           ;error flag offset pointer
ERSEG   EQU     300EH           ;error flag segment pointer
VAROFF  EQU     3012H           ;variable offset pointer
VARSEG  EQU     3014H           ;data segment pointer
```

3.0 ENTRANCE REQUIREMENTS FOR ALL HIGH LEVEL LANGUAGES

```
BASADR  EQU     300CH           ;base address of board

;---------- ROM INTERPRETER SEGMENT:OFFSET --------------

PTRCOD  DB      'ROMSEG$'       ;rom segment ID

ROMOFF  DW      0074H           ;rom segment pointer
ROMSEG  DW      0C000H

VARPTR  DB      00H             ;variable string count
        DW      STGBUF          ;variable pointer offset
STGBUF  DB      100H DUP(0)     ;string buffer for pascal

TPSTRG  DW      0000H           ;pascal string flag

;-------------------------------------------------------

CONECT: PUSH    BP                      ;save BP register
        MOV     BP,SP                   ;point to the variables
        PUSH    DS
        SUB     DI,0005H                ;adjustment for IP
        PUSH    DI                      ;save for later
        MOV     BX,CS:ROMSEG[DI]        ;get pointer
        MOV     ES,BX                   ;point to ROM's
        MOV     DI,IEEBSY               ;busy flag
        CMP     ES:WORD PTR[DI],0000H   ;are we busy ?
        JE      GPIBOK                  ;all ok
        LES     BX,[BP]+FLAG            ;set flag to busy code
        MOV     AX,0060H                ;busy code
        MOV     ES:WORD PTR[BX],AX      ;set flag
        POP     DI
        POP     DS                      ;restore ds
        POP     BP                      ;restore bp
        RET     16                      ;return to caller

GPIBOK: LDS     SI,[BP]+COMMAND         ;point to command string
        SUB     CX,CX
        MOV     CL,DS:BYTE PTR[BX]      ;string count
        PUSH    CX                      ;save it
        INC     SI                      ;point to first byte
        MOV     BX,CS:ROMSEG[DI]        ;set seg
        MOV     ES,BX                   ;get es:di rom segment
        MOV     DI,CMDSTR[DI]           ;command string pointer
        MOV     ES:BYTE PTR[DI],CL      ;put in byte count
        INC     DI                      ;point to first byte
        CLD                             ;set count direction

REP     MOVS    BYTE PTR ES:[DI],DS:[SI];move command string
        MOV     DS:BYTE PTR[DI],'*'     ;term. of command string
        POP     CX                      ;restore byte count
        LES     BX,[BP]+COMMAND         ;point to command string

LPX:    CMP     ES:BYTE PTR[BX],'$'     ;the variable a string ?
        JE      EXTSTR                  ;exit if match
        INC     BX                      ;bump pointer +1
        LOOP    LPX                         ;more to come
        POP     SI                          ;restore IP
        PUSH    SI                          ;save it for later
        JMP     SHORT VARINT                ;integer variable ok

EXTSTR: LES     BX,[BP]+VARIABLE            ;get data pointer
        MOV     AX,ES                       ;can't move direct
        MOV     DI,VARSEG                   ;point to data seg ROM
        MOV     DS:[DI],AX                  ;save data segment ptr
        MOV     DI,OFFSET CS:VARPTR[SI]     ;getstring descript ptr
        MOV     AL,ES:BYTE PTR[BX]          ;get byte count
        MOV     CS:BYTE PTR[DI],AL          ;set string byte count
        INC     DI                          ;point  first character
        INC     BX                          ;point  first character
        MOV     CL,AL                       ;get byte count
        AND     CX,00FFH                    ;<255 BYTES
        MOV     DI,OFFSET CS:STGBUF[SI]     ;initialize buffers
        MOV    CS:WORD PTR TPSTRG[SI],01 ;init. strg flg

STGLP:  MOV     AL,ES:BYTE PTR[BX]          ;get byte
        MOV     CS:BYTE PTR [DI],AL         ;put in buffer
        INC     BX
        INC     DI                          ;bump pointers
        DEC     CX                          ;byte count -1
        JNZ     STGLP                       ;more to come
        JMP     SHORT SETFLG                ;set flag pointers

VARINT: LES     BX,[BP]+VARIABLE            ;get data pointer
        MOV     AX,ES                       ;can't move direct
        MOV     DI,VARSEG                   ;point data seg in ROM
        MOV     DS:[DI],AX                  ;save data seg ptr
        MOV     DI,VAROFF                   ;pointer offset
        MOV     DS:[DI],BX                  ;point to data

SETFLG: LES     BX,[BP]+FLAG                ;get flag pointer
        MOV     AX,ES                       ;not allowed direct
        MOV     DI,ERFLG                    ;offset
        MOV     DS:WORD PTR[DI],BX          ;point to data
        MOV     DI,ERSEG                    ;segment pointer
        MOV     DS:WORD PTR[DI],AX          ;segment set
        LES     BX,[BP]+BRDADDR             ;get board address
        MOV     AX,ES:[BX]                  ;get address
        MOV     DI,BASADR                   ;RAM ptr to BASE ADDRESS
        MOV     DS:WORD PTR[DI],AX          ;set base address
        MOV     DI,CMPLR                    ;reset flags
        MOV     DS:WORD PTR[DI],0000H       ;compiler flag clear
        MOV     DI,TPASC                    ;pascal flag
        MOV     DS:WORD PTR[DI],0000H       ;flag clear

;setup dummy stack for ROM interpreter return procedure

        PUSH    DX                          ;variable 1
        PUSH    DX                          ;variable 2
        PUSH    DX                          ;variable 3
        PUSH    DX                          ;variable 4
        PUSH    CS
        MOV     BX,OFFSET RXTN[SI]      ;return pointer
        PUSH    BX                      ;CS:IP return on stack
        PUSH    BP                      ;simulates ROM stack
        PUSH    ES                      ;pointers
        PUSH    SS
        PUSH    DS
        JMP     DWORD PTR ROMOFF[SI]    ;execute

RXTN:   POP     SI                      ;restore IP offset
        CMP     CS:TPSTRG[SI],00        ;is it a string =1
        JE      TBRTN                   ;return to user
        LES     BX,[BP]+VARIABLE        ;get variable
        MOV     DI,OFFSET CS:VARPTR[SI] ;get pointer
        MOV     CL,CS:BYTE PTR[DI]      ;byte count
        MOV     DI,OFFSET CS:STGBUF[SI] ;buffer
        INC     BX                      ;initialize pointer

RTNLP:  MOV     AL,CS:BYTE PTR[DI]      ;get byte
        MOV     ES:BYTE PTR[BX],AL      ;put it in PASCAL BUFFER
        INC     BX
        INC     DI                      ;bump pointers
        DEC     CX                      ;more to come ?
        JNZ     RTNLP                   ;yes

TBRTN:  POP     DS                      ;return to caller
        POP     BP
        RET     16                      ;return to caller



TBRTN:  POP     DS                      ;pop variables
        POP     BP                      ;restore BP
        RET     16                      ;return to caller

IE488           ENDP
IE488TUR        ENDS
                END
```

4.0 LIBRARY OVERVIEW

The disk is shipped with several files on it. The main file GPIBDVR.COM is the DOS resident driver. The file CHGVEC.EXE is for changing the vector to call the library. The remaining files have the extension .LIB which are the individual library files for the specified compiler. All the .LIB files begin with IE488xxx.LIB where xxx is the three letter code for the particular compiler.

LIBRARY FILE NAME: **IE488xxx.LIB**

where: xxx is the library identifier

FOR	--	FORTRAN MicroSoft version 3.2
TBP	--	TURBO PASCAL version 2.0/3.0 (IE488TUR.COM)
SML	--	SMALL MODEL COMPILER ENTRANCE
LRG	--	LARGE MODEL COMPILER ENTRANCE
BAS	--	BASICA COMPILER MicroSoft version 1.xx/2.xx

The commands included in the library are shown in the table below and are explained in the following sections.

ABORT	
CLEAR	dev1, dev2, ... devN
CONFIG	TALK=dev1/MTA, LISTEN=dev2, dev3,...,(MLA)
ENTER	dev.secad [image]
EOI	dev[image]
LOCAL	dev1, dev2, ... devN
LOCKOUT	dev1, dev2, ... devN
OUTPUT	dev1, dev2, ... devN [image]
PARPOL	
PASCTL	dev
PPCONF	dev
PPUNCF	dev
REMOTE	dev1, dev2, ... devN
REQUEST	(1)
RXCTL	
STATUS	dev.secad
SYSCON	MAD=dev,CIC=0/1/2/3,NOB=1/2,BA0=&Hddd,BA1=&Hddd
TIMEOUT	
TRIGGER	dev1, dev2, ... devN
ITEST	(var, #NUM)
PRINT/h	(var, Nob)

4.0 LIBRARY OVERVIEW

All modes of operation are determined by an **ASCII STRING** in a command (COMMAND or CMD) referenced within a CALL statement. The variable is declared a CHARACTER COMMAND*127 at the beginning of the program. All non character variables are INTEGER*2 type and must be declared as such. The CALL statement format is:

CALL IE488 (COMMAND, var, FLAG, BASADR)

or if used as a FUNCTION:

if IE488 (COMMAND, var, FLAG, BASADR) then goto ERRORS

where:

COMMAND - is the COMMAND including device addresses or secondary commands and [image terminators]. This is always a **STRING** and is decoded by the Command Line Interpreter in the IE-488 library. The COMMAND is separated from the operands (devices etc.) by one or more SPACES, any other delimiters will cause a SYNTAX error in command line. The separator for devices is alwaysthe comma "," and secondary address is always a period "." . The IMAGE string isidentified by brackets "[]". The Command Line Interpreter is relatively tolerant of syntax error identification and will send back the appropriate error code to isolate the error. The format is:-

CMD = '"**COMMAND dev1, dev2, ...,devn [image]**"'

The **[image]** specifier allows the user to specify the variable field operations for the beginning and end of the data transfer variable. The variable may be a variable name, array identifier, numeric data value or a string. The user must match the image to the data type or an error will be generated in the data transfer. No check is made in the match of the image to the variable type, this is the responsibility of the user. The [image] codings are explained in section 3.1 (IMAGE SPECIFIERS).

var ------ isthe data variable OUTPUT/INPUT to be transferred from/to. Datais transferred as specified by the image terminator/specifier. If the image specifier isnot used the data is treated as anINTEGER*2. The data may be of String or Integer type.

FLAG ---- is the transfer status of the CALL statement. If an error occurs FLAG will contain a **HEX** number representing the error condition. A set of error and transfer message codes are generated at the completion of each CALL. Type is INTEGER*2 only.

BASADR -- is the address of the interface board being used. BASADR% may be 0 or 1, or actual base address e.g. 768. The type is INTEGER*2 only.

4.1 IMAGE SPECIFIER

The main reason for the IMAGE specifier is to allow the ROM interpreter to identify the DATA type of VARIABLE in the CALL statement. It is the users responsibility to insure that strings and integer data types are declared in the image specifier. The IMAGE specifier will also condition the data with odd/even or no parity if the variable is a string, or allow transfer to the high or low byte of an integer variable, or sequential bytes if the variable is a 16 bit word. Also the user may transfer a portion of the variable by selecting the starting and ending limits. The Interpreter will check the starting and ending limits of all VARIABLE strings and return the appropriate error code if the limits are exceeded. The Interpreter will use the IMAGE specifier to identify the VARIABLE data type and transfer the data. It is the programmers responsibility to insure the data types match the IMAGE specifier. The IMAGE specifier also determines the data transfer type either program control or DMA transfer. The following are the codes for the IMAGE specifier.

[$(p)(x),m,z] -	Input/Output the number of Bytes to/from the variable string starting at position m and ending at position z, with parity p (E=even, O=odd, none). If m, z and p are omitted the entire string will be output as in the string variable$ as specified by the image terminator (x) without parity. If no terminator is used then the string will end with EOI.
[B(H/L)(x),m,z]-	Input/Output the specified H/L number of Bytes to or from the the specified integer variable array starting at (m) array location and ending at the (z) position. The data transferred will not change the the other half of the 16 bit integer, only the byte specified is changed on an ENTER command. There is no change to the data with the OUTPUT command. [BL#,2,10] will transfer the low byte of position 2 thru position 10 of the integer variable array. Note, the number of bytes transferred is nine, position two and ten are included. Transfer termination is specified by the image terminator. It is the user's responsibility to insure that the array size and the type of array are correct. No check is made on data types. The values of m and z may be reversed which will transfer data in the reverse order. If m and z arethe same then only one word is transferred. If m and z are omitted then the integer variable is not considered an array and the variable is transferred with or without an EOI depending on the image terminator (x).

[W(x),m,z] - Input / Output the specified number of 16 bit words to / from the specified integer variable (array) starting with position (m) and ending with position (z). The number of words transferred is defined as {z - m + 1}. Termination is specified by the image terminator. It is the user's responsibility to insure that the array size and type of array are correct. No check is made on data types. The values of m and z may be reversed which will transfer data in the reverse order. If m and z are the same then only one word istransferred. Ifm and z are omitted then the integer variable is not considered an array and the variable is transferredwith or without an EOI depending on the image terminator (x).

4.2 IMAGE TERMINATORS

%(t) The % image terminator cancels both the carriage return, line feed and EOI terminators during an OUTPUT command execution. During an INPUT command the entry will terminate when the array size or the input count is reached (m + count = z) or EOI.

#(t) The # image terminator ends the data output with an EOI only. No carriage return or line feed is inserted at the end of the data output transfer. The data is terminated by an EOI during the INPUT or OUTPUT command. The ENTER also terminates if the last item in data list is entered which sets the FLAG% variable with an error code of &H0020.

+(t) The + image terminator adds a carriage return, line feed and EOI during an OUTPUT command only. The INPUT command is in the default mode (INPUT terminates with EOI or last entry). If carriage return and line feed are part of the data being transferred they will be sent as normal data.

No image terminator code defaults to an EOI only at the last byte to be transferred. The operation is the same as the # terminator.

(t) The transfer terminator t determines the type of transfer the GPIB is to perform. The following transfer codes are available. If this specifier is not used the data transfer is under program control.

D = Direct Memory Access (DMA) to the specified array. The m and z specifiers must be used with this type of transfer. Structure programming must be used when this mode is active. All variables must be assigned before the CALL is executed and no new variables are allowed to be introduced after the execution of the CALL statement. See APPENDIX A for details on DMA transfers.

4.3 FLAG RETURN CODES

The following codes are returned in the FLAG% variable upon completion of the CALL statement. The flag return codes are grouped into 3 categories.

****************** DATA TRANSFER ******************

DECIMAL	HEX		DESCRIPTION
00000	#0000	=	TRANSFERRED OK
00032	#0020	=	NO INPUT EOI or LINE FEED
00048	#0030	=	DEVICE TIME OUT
00064	#0040	=	RESERVED
00080	#0050	=	DMA CHANNEL BUSY
00096	#0060	=	GPIB BUSY

******************** HARDWARE *********************

DECIMAL	HEX		DESCRIPTION
00256	#0100	=	HARDWARE FAILURE
00512	#0200	=	TIME OUT ON DATA TRANSFER
00768	#0300	=	DEVICE NOT ACTIVE CONTROLLER
01024	#0400	=	IBM-PC ACTIVE CONTROLLER
01280	#0500	=	SYSTEM NOT INITIALIZED
01536	#0600	=	CONFIGURATION ERROR

******************* FORMAT **********************

DECIMAL	HEX		DESCRIPTION
04096	#1000	=	UNDEFINED COMMAND
04352	#1100	=	SYNTAX ERROR IN COMMAND LINE
08192	#2000	=	UNDEFINED IMAGE
12288	#3000	=	DEVICE RANGE ERROR
12544	#3100	=	TOO MANY DEVICES
12880	#3200	=	TALKER/LISTENER CONFLICT
16384	#4000	=	COMMAND/DATA OUT OF RANGE
20480	#5000	=	COMMAND REQUIRES DEVICE
24576	#6000	=	UNDEFINED DEVICE CODE
28672	#7000	=	INPUT ARRAY NOT INITIALIZED
-28672	#9000	=	IBM MUST BE TALKER or LISTENER

4.4 USER COMMANDS

The following set of commands explain the use of the FORTRAN driver only in the applications. The use of other compilers will only change the CALL sequence to a specified PROCEDURE and type/var pseudo operators to call the device.

The following user commands are available. The string variable **COMMAND** is the same string format as described in the Fortran manual. All command strings must end enclosed in quotes ("....") typical command string would be as follows. Please note that all commands must be assigned in string form before using the CALL statement. The single quotes define a string in Fortran while the double quotes mark the begining and end of the string for the resident interpreter. Since Fortran does not pass the byte count of the string to the subroutine this method was incorporated.

```
COMMAND= '"OUTPUT 03.13.20,05[WD#,2,20]"'
```

This command string would output integer words (16 bit) two thru and including word 20 to device primary address 03 with secondary addresses 13 and 20 and also to device primary address 05. The data transfer uses the DMA mode for fast access. The device codes must be in decimal within the range of 00 to 30. This allows the user a maximum of 31 device addresses to choose from. However the maximum number of devices which may physically be connected to the bus is 15.

The transfer of String data is limited to single element arrays and must be initialized. The Maximum string size is 127 bytes as defined in this Users manual.

ABORT - Terminate the current command issued by the IBM. The command executes an IFC and resets the IBM board addressed. DMA and Interrupts are disabled. The IBM-PC is assumed to be the main system controller and unconditionally takes control of the bus and remains the controller in charge until PASCTL command is executed.No device is necessary.

COMMAND$ FORMAT:

"ABORT"

EXAMPLE:

```
      CALL IE488 ('"ABORT"', VAR, FLG, BRD)   'execute command
      IF(FLG .EQ. 0) GOTO 100                'test for errors ?
      WRITE(*,(\( 'ERROR IN ABORT'))) FLG
      STOP
      END

  100 ...... user program continues .........
```

CLEAR - Clear or Reset the selected devices or all devices. If no device is given the GPIB is cleared. The IBM PC must be the active controller or an error message will be generated.

COMMAND$ FORMAT:

"CLEAR dev1,dev2,......devN"

EXAMPLE:

```
      CALL IE488 ('"CLEAR 10,12,14.22"', VAR, FLG, BRD)
```

CONFIG - Configure the GPIB to the devices specified in the command string. The GPIB will remain in this state until reconfigured by issuing an ENTER or OUTPUT command. The VAR variable is not changed in this command. If the TALK = dev1 is omitted the IBM-PC is assumed to be the controller only. The user may enter MTA to make the IBM-PC the talker or enter the actual device number using the TALK variable name. The IBM-PC may be addressed as a listener by using the name MLA as the last device in the COMMAND string. The FLAG variable will contain the error code if any conflicts occur.

COMMAND FORMAT:

"CONFIG (TALK=dev1 /MTA,)LISTEN=dev2,dev3,..,(MLA)"

ENTER - Input GPIB data from selected talker to specified string array. The string array must have been previously dimensioned. The FLAG will contain error codes if an error occurs. The IBM - PC must have been previously programmed as a listener.If the IBM - PC is not the controller then the ENTER command will return error code 9000H to inform the caller that the IBM is not in the listen mode. The command may be re-entered until the controller in charge programs the IBM to listen. Only one device is allowed with this command.

COMMAND FORMAT:

"ENTER dev.secad [image]"

```
      CALL IE488 ('"ENTER 12.05[$]"', DVM, FLG, BRD)
```

EOI - Sends a data byte on the selected device with EOI asserted. The bus must have been programmed to talk before the command is executed. The variable contains the data to be transferred. It is the users responsibility to insure the data and type match. If a string variable is used the entire string is transferred ending with an EOI. If Integer mode is used only one transfer (byte) or two (word) will be executed. The limit parameters are ignored. Only one device is allowed. No device is generally required if the Talker (IBM-PC) has been previously programmed to talk by the controller in charge. If the IBM-PC is not the controller in charge and not programmed as a talker then an error code &h9000 will be returned until the controller in charge programs the IBM-PC as a talker before data is transferred.

COMMAND FORMAT:

"EOI dev [image]"

EXAMPLE:

```
      VAR = '"THIS IS A ""STRING"" WITH QUOTES"'
C     'define last byte to transfer

      CMD = '"EOI 12[$]"'          'define command
      CALL IE488 (CMD, VAR, FLG, BRD)
      IF (FLG .EQ. 0) GOTO 200
      WRITE (*(a\('ERROR IN LINE 120'))) FLG
      STOP
      END

  200 ......... continue users program ...........
```

This routine will transfer the STRING in the VAR variable and issue an EOI command with the last byte of the STRING to signal the receiver on the bus that the data transfer will end.

The image specifiers for the removal of the line feeds and carriage returns are ignored during the command, no parity is used.

LOCAL - Set selected device(s) to the local state. If no device is specified then all devices on the bus are set to local. The IBM-PC must be the active controller or an error message will be generated.

COMMAND FORMAT:

"LOCAL dev1,dev2,......devN"

EXAMPLE:

```
     CMD = '"LOCAL 10,11,12,14"'           'define command
 120 CALL IE488 (CMD, VAR, FLG, BRD)      'execute command
     IF (FLG .EQ. 0 ) GOTO 200            'test for errors
     WRITE (*,\('ERROR IN LINE 120')) FLG
     STOP
     END

xx200  ....... continue users program ........
```

The above program sets devices 10,11,12,14 to the local state and returns to the user's program. The LOCKOUT command is very similar in structure to the LOCAL command except the LOCKOUT does not allow the user to manually select the device to local.

LOCKOUT - Local Lockout the specified device. If no device is given all devices on the bus will be set to local lockout. The IBM-PC must be the active controller or an error message will be generated. The devices cannot be set to local except by the GPIB controller. The FLG% variable contain the error code.

COMMAND FORMAT:

"LOCKOUT dev1,dev2,.....devN"

This command is the same as the LOCAL command except the user is NOT allowed to manually select the device to local.

OUTPUT - Output selected string to selected listener(s) on GPIB. The variable will contain the data to be transferred. The image specifier will contain the data type and terminators. The FLAG will contain the error codes if an error occurs. Up to 14 devices may be accessed in the list. If the IBM-PC is not the controller in charge, the IBM-PC must be programmed by the controller in charge before data is transferred.

COMMAND FORMAT:

"OUTPUT dev1.secad,dev2...[image]"

EXAMPLE:

```
     VAR  = '"THIS IS A TEST"'          'define bytes to transfer
     CMD  = '"OUTPUT 12,11[$E]"'        'define command
 120 CALL IE488 (CMD, VAR, FLG, BRD)
     IF (FLG .EQ. 0 ) GOTO 200
     WRITE (*(\('ERROR IN LINE 120'))) FLG
     STOP
     END

'This command line will output the entire string "THIS IS A TEST"
'with out the quotes using even parity and ending with a EOI code
'to show the end of 'the string. The FLG  variable will have any
'error transfer codes 'if an error was detected during transfer.
'All string transfers must be enclosed in ,quotes.

 200  ......... continue users program ...........

     DIM MYDATA (2,400)                 'my integer data array
     CMD  = '"OUTPUT 12,11[BL,0,100]"'  'setup image

C output data in 2,0 from element 0 to 100 low byte only
 420 CALL IE488 (CMD, MYDATA(2,0), FLG, BRD)
     IF(FLG .EQ. 0) GOTO 500
     WRITE (*(\('ERROR IN LINE 420'))) FLG
     STOP
     END
         ......continue users program.......
 500 'setup for DMA transfer
     CMD  = '"OUTPUT 12,10,15[WD,0,8192]"' 'DECIMAL ONLY

' transfer data in DMA mode

 550 CALL IE488 (CMD, MYDATA(2,0), FLG, BRD )
     IF (FLG .EQ. 0 ) GOTO 600
     WRITE (*(\('ERROR IN LINE 550'))) FLG
     STOP
     END
'If error code in DMA is not &H50 or 0 then issue an ABORT
command to clear interface device.

600     ........ user program continues ......
```

PARPOL - Reads the 8 Status Bit messages for the devices on the GPIB which have been set for parallel poll configuration. The VAR will contain the 8 bit message. The IBM-PC must be the active controller or an error will occur.

COMMAND FORMAT:

"PARPOL"

PROGRAMMING EXAMPLE:

```
    VAR = 0     'Parallel Poll return byte initialized
    CMD = '"PARPOL"'
120 CALL IE488 (CMD, VAR, FLG, BRD)
    IF (FLG .EQ. 0 ) GOTO 200

'if error, the flag is printed out.

    WRITE (*(\('ERROR IN LINE 120'))) FLG
    STOP
    END

200 'process parallel poll return byte code in character VAR
                ..........
                ..........
```

This command responds as programmed in the parallel configuration command. The VAR will contain the eight bit poll response. See the Parallel Poll Configure command (PPCONF) for the details of the bit pattern.

PASCTL - The Active control of the GPIB is transferred to the specified device address and the IBM-PC becomes the standard listener/talker but not controller. The IBM - PC must be the active controller or an error will occur. The IBM-PC is not allowed to Talk until programmed by the controller in charge.

COMMAND FORMAT:

"PASCTL dev"

EXAMPLE:

```
    CMD = '"PASCTL 6"'
110 CALL IEE488 (CMD, X, FLG, BRD)
    IF (FLG .EQ. 0) GOTO 200
    WRITE (*(\('ERROR IN LINE 110'))) FLG
    STOP
    END

200  ............... continue users program ..................
```

The IBM-PC is inactive at this point and no controller commands are allowed. To receive control back the command RXCTL must be used as follows.

```
      CMD = '"RXCTL"'                                 'define command
      VAR = 0                                        'set VAR$ to false

  330 CALL IE488 (CMD, VAR, FLG, BRD)     'test for control
      IF (FLG .EQ. 0 ) GOTO 360
      WRITE (*(\('ERROR IN LINE 330'))) FLG
      STOP
      END

  360  ..... user continues program ......
       IF VAR = -1 THEN THE CONTROL IS BACK ELSE NOT IN CONTROL
                   ......
```

Note: It is the responsibility of the controller in charge to program the IBM-PC to the talk mode before the transfer of control is executed.

PPCONF - Sets up the desired parallel poll bus configuration for the user. The VAR integer contains the poll sequence (00-FF). IBM-PC must be the active controller or an error will occur.
COMMAND FORMAT:
"PPCONF dev"

The PARALLEL POLL function provides a means of sending one bit of status information if the controller is requesting the response. Unlike SERIAL POLL, which is initiated by the device, the parallel poll is initiated by the controller in charge. There are two methods to configure a device for parallel poll, remote and local configurations. In remote configuration (PPI), he controller uses the following bit codes to configure the device addressed.

msb	B6	B5	B4	B3	B2	B1	lsb
X	1	1	0	S	P3	P2	P1

Were Pn = the device bit code 0 to 7 for PPR1 to PPR8 and S is the Send of the Parallel Poll Response, S = response. Adevice may be configured so that it never responds to a parallel poll. PPD (&H70) is the parallel poll disable command, which places the device in the parallel poll idle state (PPIS). The value of the individual status (IST) can be set by bit B4 in the VAR byte.

B4 = 0	IST = Parallel Poll Flag
B4 = 1	IST = SRQS

EXAMPLE:

```
    BRD = 0
    A = 19                  'parallel configure bit code for dev 14
    CMD = '"PPCONF 14"'
140 CALL IE488 (CMD, A, FLG, BRD)
    IF (FLG .EQ. 0 ) GOTO 170
    WRITE (*(\('ERROR IN LINE 140'))) FLG
    STOP
    END
170 ...... continue program ..........
```

In the local configuration (PP2), the specifications are made from the device. Writing 0 11 U S P3 P2 P1 to the VAR configures the controller for a Parallel Poll Response. When U = 0, this command is the LPE (local poll enable) local message. When U = 1, the TLC does not respond to the poll. The TLC is configured in the S bit. The PPRn will be sent true only if the Parallel Poll Flag (IST individual status local message) matches this bit. During normal operation, The value of VAR on entry will set or clear PPF (IST if B4 = 0) according to the device's need for service.

PPUNCF - Resets the parallel poll type configuration of the selected device. The IBM-PC must be the active controller or an error will occur. The specified device will not respond to a parallel poll command.

COMMAND FORMAT:

"PPUNCF dev"

PROGRAMMING EXAMPLE:

```
    BRD  = 0
    A   =  #0A
    CMD = '"PPUNCF 14"'
130 CALL IE488 ( CMD, A, FLG, BRD )
    IF (FLG .EQ. 0 ) GOTO 300

' error is processed here
    WRITE (*(\('ERROR IN LINE 130'))) FLG
    STOP
    END

300 'program continues here if ok
        ....................
         ..................
```

This routine will only disable device 14 to respond to a Parallel Poll command. If no device code is used the entire bus is disabled.

REMOTE - Sets the selected devices or device on the GPIB to go into the remote position. The IBM must be theactive controller or an error will occur. If an erroroccurs the FLAG% will contain the error code.

COMMAND FORMAT:

"REMOTE dev1,dev2,.......devN"

EXAMPLE:

```
     VAR  = 0                              'dummy variable not used
     BRD  = 0                              'define board number
     CMD  = '"REMOTE 10,12,14"'             'define command
 140 CALL IE488 (CMD, VAR, FLG, BRD )
     IF (FLG .EQ. 0 ) GOTO 200
     WRITE (*(\('ERROR IN LINE 140'))) FLG
     STOP
     END


 200  ......... continue users program ...........
```

This command is the counterpart to the LOCAL command. Devices 10,12,14 are set in the remote state and ready for a command sequence. The error flag FLG% will contain any error codes if an error was detected.

REQUEST - The GPIB may request service from the active controller on the bus by executing the "REQUEST n" command. This command has two modes. the first when "n" is omitted which may be executed any time to monitor the status of the IBM interface board. The VAR (INTEGER) contains the status bits for the GPIB board addressed, [Hi Byte] = on board hardware registers, [Lo Byte] contains the IBM GPIB serial poll register status byte. The second mode when n is any number (0-31). This allows the user to set a serial poll status word to the controller in charge. The Low byte of the variable will contain the STATUS byte to be transferred to the controller.

```
 msb      GPIB  ON BOARD                   SERIAL POLL REGISTER   lsb
|--------------------------------|--------------------------------|
|15  14  13  12  11  10  09  08  |07  06  05  04  03  02  01  00  |
|--------------------------------|--------------------------------|
```

```
BIT 08 = INTERRUPT ENABLED 1 = on           BIT 00 = S0 BIT
BIT 09 = DMA ENABLED 1 = on                 BIT 01 = S1 BIT
BIT 10 = DMA CHANNEL                        BIT 02 = S2 BIT
         1=chan #1, 0=chan #3               BIT 03 = S3 BIT
BIT 11 = INTERRUPT vector level (1)         BIT 04 = S4 BIT
BIT 12 = INTERRUPT vector level (2)         BIT 05 = S5 BIT
BIT 13 = INTERRUPT vector level (4)         BIT 06 = rsv on send
BIT 14 = srq (CIC=1) PEND bit (CIC=0)             PEND on receive
BIT 15 = Controller In Charge (CIC)         BIT 07 = S7 BIT
         1 = yes,  0 = not in charge
```

EXAMPLE 1: ------- IBM NOT IN CONTROL --------

```
    BRD =0
    X  = (SERIAL POLL BIT PATTERN)
    CMD  = '"REQUEST 1"'
130 CALL IE488 (CMD, X, FLG, BRD)
    IF (FLG .EQ. 0 ) GOTO 200
    WRITE (*(\('ERROR IN LINE 130'))) FLG
    STOP
    END
 ......... process status flag code ............
200 WRITE (*(\('REQUEST 1 FLAG CODE =  '))) X
```

EXAMPLE 2: -------- IBM IS CONTROLLER IN CHARGE ---------

```
    BRD =0
    CMD  = '"REQUEST"'
130 CALL IE488 (CMD, X, FLG, BRD)
    IF (FLG .EQ. 0 ) GOTO 300
    WRITE (*(\('ERROR IN LINE 130'))) FLG
    STOP
    END
300 WRITE (*(\('REQUEST STATUS CODE = '))) X
```

Bit 09 would be used to determine if the DMA data transfer is complete (0 = off, 1 = on). The user may use the instruction at any time to monitor the state of the IBM-PC GPIB.

```
140 IF ITEST(X, #0200) THEN 130  ' this will loop until the DMA is
                                ' done.
```

RXCTL - Receive control of the bus.The VAR (integer) is set true if the IBM regains control of the bus else VAR is false.

COMMAND FORMAT:

"RXCTL"

EXAMPLE:

```
      BRD  = 0                           'define board number
      CMD  = '"RXCTL"'                    'define command

  150 CALL IE488 (CMD, VAR, FLG, BRD)

      IF (FLG .EQ. 0 ) GOTO 200
      WRITE (*(\('ERROR IN LINE 150'))) FLG
      STOP
      END

  200   ... continue users program until IBM is controller ...
     IF (VAR .EQ. 0 ) GOTO 150
                          ' the last instruction before control
  300    ..... THIS IS WHERE THE PROGRAM WILL CONTINUE .....
         .....    WHEN THE IBM RECEIVES CONTROL        .....
               ........ user program continues .......
                              ......
```

When control is received the IBM may issue all commands as outlined. The RXCTL command may be issued at any time to determine the state of the IEEE488 BUS.

STATUS - A serial polled devices status byte is read into the selectedvariable. The variable will contain the Statusbyte of the device specified as a serial poll. TheIBM-PC must be the active controller or an error will occur. Only one device is allowed with one secondary address. If no device is specified an error will occur.

COMMAND FORMAT:

"STATUS dev.secad"

EXAMPLE:

```
' three command sequence for Keithly Model 175 DVM
    BRD  = 0
    A   = '"M33X"'
    CMD = '"REMOTE 12"'
    DVMEOI = '"EOI 12[$]"'
    DVMSTATUS = '"STATUS 12"'

160 CALL IE488 (CMD, A, FLG, BRD )
    IF (FLG .EQ. 0 ) GOTO 190
    WRITE (*(\('ERROR IN LINE 160'))) FLG
    STOP
    END
C
190 CALL IE488 (DVMEOI, A, FLG, BRD )
    IF (FLG .EQ. 0) GOTO 220
    WRITE (*(\('ERROR IN LINE 190))) FLG
    STOP
    END
C
C  Status command issued here
220 CALL IE488 (DVMSTATUS, X, FLG, BRD )
    IF (FLG .EQ. 0) GOTO 250
    WRITE (*(\('ERROR IN LINE 220'))) FLG
    STOP
    END
C
250 WRITE (*(\('STATUS BYTE CODE RETURNED IS   =  '))) X
    STOP
    END
```

The above routine selects the DVM, sends out Set status info "M33X" then the status (serial poll) is executed on the device.

SYSCON - SYStem CONfiguration and initialization of the GPIB. The user must run this command once before using the GPIB. IF this is not run first an error will be generated. Base address data BAx is in HEX(&H) or DECIMAL. The SYSCON command checks for the conflict of all parameters if two boards are used. These are the BAS0, BAS1, interrupt vector and DMA channel settings which must be different. The BRD% and data variable are not used in this CALL since they have been defined in the COMMAND string.

COMMAND FORMAT:

"SYSCON MAD=dev,CIC=(0/1/2/3),NOB=(1/2),BA0=&Hddd(,BA1=&Hddd)"

where:

dev	=	the address of the IBM 00 to 30 decimal
MAD	=	My (IBM) device address
NOB	=	number of IE488 boards (1 or 2)
BA0	=	base address for board 1
BA1	=	base address for board 2
CIC	=	controller in charge, 0=none, 1=brd#1, 2=brd#2, 3=(brd#1 and brd#2) (separate GPIB busses).

EXAMPLE:

```
    CMD = '"SYSCON  MAD=3, CIC=1, NOB=1, BA0=&H300"'
130 CALL IE488 (CMD, A, FLG, BRD )
    IF (FLG .EQ. 0) GOTO 200
    WRITE (*(\('ERROR IN LINE 130'))) FLG
    STOP
    END

'The above lines of initialization code should always be placed
'at the beginning of your programs and precede any use of the IE-
'488.

200 .......... continue users program ..............
                     .............
                     .............
```

TIMEOUT - Sets the time out duration when transferring data to/from the devices. The Variable integer VAR% is set to a number from 0000 to 65000. The approximate time is the VAR% * 1.5 seconds for the IBM-PC/AT and VAR% * 3.5 for the IBM-PC/XT. No error flag is returned.

COMMAND$ FORMAT:
"TIMEOUT"

EXAMPLE:

```
100  TIMESET$ = "TIMEOUT"
110 DURATION% = 10        'approximately 30 seconds for PC
120  CALL IE488% ( TIMESET$, DURATION%, FLAG%, BASADR% )
130  ' continue user program  time out is set until changed.
```

TRIGGER - Sends a trigger message to the selected device or a group of devices. The IBM-PC must be the active controller or an error will occur.

COMMAND$ FORMAT:

"TRIGGER dev1, dev2,......... devN"

EXAMPLE:

```
xx120 BRD% = 0                          'define board number
xx140 CMD$ = "TRIGGER 11,12,15"         'define command
xx150 CALL IE488 (CMD$, VAR%, FLG%, BRD%)

    ..... devices 11,12,13 are triggered at the same time .....

xx160 IF NOT FLG% THEN 200
xx170 PRINT "ERROR ";HEX$(FLG%);"  IN LINE 160" : END

xx200  ......... continue users program ...........
```

DATA SUPPORT FUNCTIONS

THESE FUNCTIONS ARE ONLY AVAILABLE FOR THE FORTRAN LIBRARY

The following functions were added to the IE488FOR.LIB library file to allow the user to handle string functions and string/integer logical functions with out declaring the variable as a LOGICAL. The addition of a PRINT(h) (VAR, NOB) function for outputing integer arrays which are used as string arrays for GPIB data transfer.

ITEST (var, num)

This function allows the user to TEST any bit(s) in the specified "var" byte. The variables are all INTEGER*2 however the function only works on the INTEGER and returns the integer to the specified variable. The function performs a logical AND on the specifiedvariable integer without changing the contents of the variable. The returned integer is the logical AND of the variable integer. This allows the user to single out any bit in the integer for a set/not set condition without the declaration of a LOGICAL type command.

EXAMPLE:

```
C     **** SETUP VARIABLES *****
      INTEGER*2 VAR1, LNUM, ITEST, RTNUM
C
      VAR1 = #13AB
      LNUM = #0081
      RTNUM =0
C
      RTNUM = ITEST ( VAR1, LNUM )
C
C     **** FUNCTIONS RETURNS HEX 81 SINCE THESE BITS WERE SET
C          AND THE VAR1 VALUE REMAINS UNCHANGED.
C
      IF ( ITEST( VAR1, #80 ) .NE. 0 ) GOTO 100
C
C     ... WILL EXECUTE THIS LINE IF BIT 8 IS NOT
C          SET (MSB OF THE BYTE)
C
 100 .... PROGRAM WILL BEGIN HERE IF VAR1 BIT 8 IS SET
```

PRINT(H) (var, Nob)

This subroutine allows the user to print a variable or array in hex or ASCII form. The user must define the type of print desired. PRINT outputs the data in ASCII form as characters in the range 00 to 255. PRINTH outputs the characters in two byte HEX format (00 to FF). The Nob variable is the number of bytes transferred to the console. The range is (0000 to 65535).

EXAMPLE:

```
C     ****** SET THE VARIABLES AND DIMENSIONS *****
C
      INTEGER*2 ARRAY, NUM, I
      CHARACTER ASCIIB*120
C
      DIMENSION  ARRAY(2,100)
C
      DO 01 I,1,100
      ARRAY(1,I)
   01 CONTINUE
C
C     **** ARRAY HAS ASCII CHARACTERS IN IT ***
C
      DO 02 I = 1, 100
      ARRAY (2,I) = 65+I
   02 CONTINUE
C
      PRINT (ARRAY(2,1), 100)
C
C     *** THIS WILL PRINT THE ARRAY (2,1) FIRST 100 BYTES IN ASCII
C                    ON THE SCREEN
C
      PRINTH (ARRAY(1,1), 100)
C
C     *** THIS WILL PRINT THE ARRAY (1,1) FIRST 100 BYTES IN HEX
C                    2 BYTE FORMAT NO SPACES. (FFADFGHH .... )
C
      STOP
      END
```

5.0 HIGH LEVEL LANGUAGES

This section explains how to use the different libraries for different languages. Not all the languages are covered for this release, however as the languages evolve the updates will be available for them. For the latest release and delivery in the future contact MetraByte Corporation.

Due to the various language differences a library has been generated for each language. A program has been generated to change the starting ROM segment of the IE-488 board at which the library accesses the resident driver. This program is called CHGVEC.EXE and examples are shown in Chapter 6 of this manual. All the libraries supplied have a default ROM segment of 0C000 hex and need not be changed unless a conflict with other third party software exists.

5.1 FORTRAN LIBRARY

This library is for the MicroSoft Fortran compiler version **3.xx** and has a library name of IE488FOR.LIB. The Fortran library includes the functions ITEST(var, #num) as a means of performing a logical AND function on any type variable and return the logical AND of the comparison as the return variable. This function would be used when using the REQUEST function for the controllers hardware status register (16 bit integer).

To use the Fortran (IE488FOR.LIB) library, just compile the Fortran source as described in the Fortran users manual. The compiled Fortran will produce .OBJ files which must be linked together to generate an executable file (.EXE).

```
C>link <enter>
MicroSoft Linker version x.xx etc.

Object modules   [.OBJ]: filespec
Run File [filespec.EXE]: <return>
List Map [ NUL.MAP ]: <return>
Libraries [ .LIB ]: IE488FOR <enter>
```

At this point the file IE488FOR.LIB will automatically be linked with the Fortran library and include all the necessary subroutines to generate an executable file (filespec.EXE).

```
C>filespec
      this will execute the compiled program.
```

5.3 TURBO PASCAL (2)

The file named IE488TUR.COM is the Turbo Pascal file. At compile time rename this file IE488.COM so the Pascal compiler will know which external file to link in the code. All commands are available to the user. The user should use the **var** variable parameter identifier to pass a four byte pointer on the stack.

CALL SEQUENCE:

```
type
        CMD = string [127];
        DAX = string [50];
        FLG = Integer;
        BAD = Integer;

var
        C:COMMAND = 'OUTPUT 12[$]'
        F:FLG = 0
        B:BAD = 0
        V:VAX = '                                                  '

begin

IE488(var C:CMD,var V:VAX,var F:FLG,var B:BAD);external IE488.COM

end
```

The DOS driver extension is not necessary for the TURBO PASCAL link since the (IE488.COM) program simulates the basic interpreter link.

The compilers this model will work with are:

TURBO PASCAL   Ver 2.0 and higher

6.0 CHANGING THE ENTRY SEGMENT [CHGVEC.EXE]

This section explains the use of the program file CHGVEC.EXE which allows the user to set the library file segment pointers to where the user set the memory address switches. The IE488 board is shipped with the SEGMENT switches set a 0C000:0000 hex address. If the user desires a different address then the program CHGVEC.EXE must be run to change the starting locating. If no address change is anticipated then this section may be ignored.

The library file must reside in the current directory for proper operation. The CHGVEC program does not support the PATH directory function. The user may change the library file name before starting to eliminate confusion. The procedure following will suggest a method for easy identification of the library.

EXAMPLE:
Change the starting ROM address from 0C000 hex to 0CE00 hex for the FORTRAN library IE488FOR.LIB.

```
C>COPY IE488FOR.LIB IE488FCC.LIB
```

The library duplication for FCC is Fortran Start CC.

```
C>\utility\CHGVEC.EXE
```

load the change vector from the utility directory
follow the directions the program requests.

```
            IE488.LIB SEGMENT ADDRESS UPDATE PROGRAM

ENTER FILE TO UPDATE (d:filespec.ext) [DIR for DIRECTORY] ? DIR
ENTER DRIVE LETTER AS (A,B,C,D,E,F) ? A

IE488FOR.LIB   IE488SML.LIB   IE488LRG.LIB   IE488TUR.COM
CHGVEC  .EXE     IE488FCC.LIB

ENTERFILETOUPDATE (d:filespec.ext) [DIRforDIRECTORY]?
                                                   A:IE488FCC.LIB

THE CURRENT SEGMENT IS HEX  C000
DO YOU WISH TO CHANGE THE SEGMENT (Y/N) ? Y
 SELECT ONE OF THE FOLLOWING
 1 = C000          6 = D800          11 = EC00
 2 = C400          7 = DC00
 3 = CC00          8 = E000
 4 = D000          9 = E400
 5 = D400         10 = E800

PLEASE ENTER THE SELECTION OR 0 TO END ? 3
SEGMENT START ADDRESS NOW CHANGED.
C>
```

The file is now changed and ready to be linked to your program.

7.0 PROGRAMMING EXAMPLES

The following example uses a Kiethly 175 DVM to collect data into a string and display the string on the console. The DVM address is set to device number 12 decimal.

```
      PROGRAM   DVM175

C     Declare variable integer / string types
C
      INTEGER*2 FLG, BRD, IVAR
C
C    The character  strings for the command string are variable
C     length and are preassigned  at  127  bytes in length. The
C     user may assign any length as long as the entire string will
C    fit  into the length. All command strings begin and end in
C     QUOTES (".....")
C
      CHARACTER*127  SYSCON, REMOTE, ENTER, SVAR
C
C     Initialize variables
C
      BRD = 0
      FLG = 0
      IVAR = 0
      SYSCON = '"SYSCON MAD=3, CIC=1, NOB=1, BAO=&H300"'
      REMOTE = '"REMOTE 12"'
      ENTER  = '"ENTER 12[$]"'
C
C     Initialize the IE-488 interface board the firat time only
C
      CALL IE488 (SYSCON, IVAR, FLG, BRD)
C
C     Set the DVM to remote for data collection
C
      CALL IE488 (REMOTE, IVAR, FLG, BRD)
C
C    Initialize string  and Collect the DVM data. The variable
C     SVAR is initialized each time to insure data integrity.
C
   01 SVAR = '                                                    '
      CALL IE488 (ENTER, SVAR, FLG, BRD)
C
C     Display the data on the screen with the PRINT ( VAR, #bytes)
C
      CALL PRINT ( SVAR, 19 )
C
C     request to repeat the function
C
      WRITE (*,'(A\)') '+Type 0 to End, 1 to Repeat data ?  '
      READ  (*,I1) IVAR
      IF (IVAR .NE. 0) GOTO 01
      STOP
      END
```