NBS-GPIB

NuBus to IEEE-488 Interface

Software Reference Manual

PROM Rev 1.1

This page intentionally left blank

fishcamp engineering
4860 Ontario way
Santa Maria, CA 93455

TEL: (805) 345-2324
FAX: (805) 345-2325

Printed in U.S.A. May 1998

Limited Warranty

The information provided in this manual is believed to be correct, however fishcamp engineering assumes no responsibility for errors contained within. The software programs are provided "as is" without warranty of any kind, either expressed or implied.

No other warranty is expressed or implied. Fishcamp engineering shall not be liable or responsible for any kind of damages, including direct, indirect, special, incidental, or consequential damages, arising or resulting from its products, the use of its products, or the modification to its products. The warranty set forth above is exclusive and in lieu of all others, oral or written, express or implied.

The information covered in this manual is subject to change without notice.

This page intentionally left blank

The NBS-GPIB interface card was designed to have its driver code reside in ROM resident on the card. The driver code delivered with the card contains an extensive set of routines which compliment the hardware capabilities of the interface card. Full talk/listener/controller capability is provided with most of the low level details of programming for the GPIB interface handled by the routines of the driver.

The driver code conforms to the interface guide-lines set forth by Apple Computer in *Inside Macintosh* for device drivers. All driver routine calls can be made thru the Macintosh device manager thus assuring a high level of compatibility with future releases of the Mac operating system.

Along with the NBS-GPIB card is included a PASCAL interface file which makes the job of coding software for GPIB applications even easier.

This manual documents the software routines of the driver code as well as that of the provided interface files. For further information regarding the IEEE-488 interface please refer to:

ANSI/IEEE Std 488.1-1987
and
ANSI/IEEE Std 488.2-1987.

Published by
The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street
New York, NY 10017

For further information regarding the TMS9914A GPIB controller chip used on the NBS-GPIB card refer to the following documents:

TMS9914A General Purpose Interface Bus (GPIB) Controller Data Manual
and
TMS9914A GPIB Controller User's Guide

Published by
Texas Instruments
P.O. Box 1087
Richardson, Texas 75080

Application programs written to take full advantage of the NBS-GPIB interface card will be written in a hierarchical format. As can be seen in figure 2.1, most I/O calls are made from the application to the PASCAL interface routine (or other language) for the appropriate call. The PASCAL interface routine is the highest level interface provided for the user with the NBS-GPIB card.

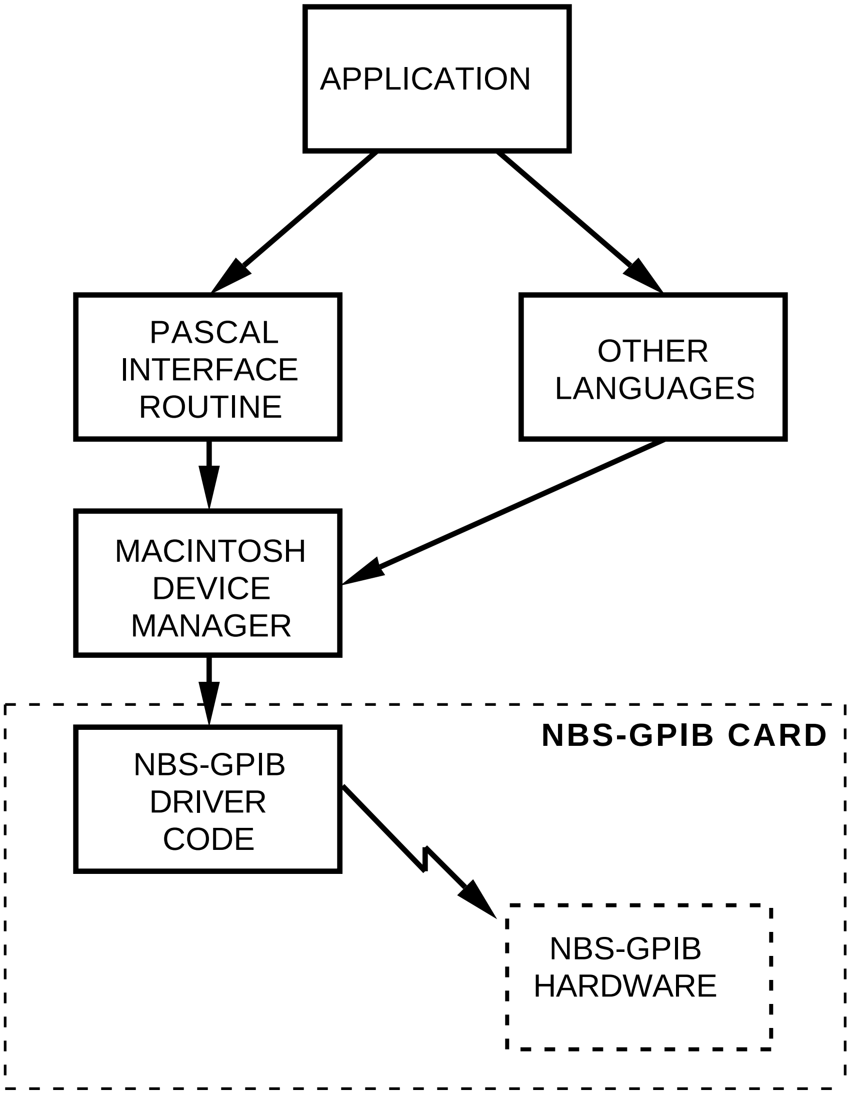


Figure 2.1 - Software Hierarchy

The interface routines take care of accepting/providing parameters from/to the calling program in the most concise and understandable manor. Only the values absolutely necessary for the proper functioning of the respective driver call are included in the pass parameter list of the interface routines. The interface routine further takes care of allocating temporary storage and setting up the parameters for calls to the driver code. These calls are all made thru the Macintosh device manager. With the exception of the 'GpibOpen' and 'GpibClose' routines, all driver calls are made with the device manager 'PBControl' call.

As stated above, the application writer wishing to use the NBS-GPIB interface card in the execution of his/her program will most likely want the utilize the PASCAL glue routines provided on disk with the board. This is the easiest way of developing software that uses the card because most of the work has been already done for the user by the fishcamp people in writing the code for these routines. Most likely the user would generate routines looking very similar to these if they had not been provided with the card.

On a lower interface level, the routines can be called from any language capable of calling the device manager routines of the Macintosh operating system. The NBS-GPIB driver code has been written to conform to the guide-lines set forth by Apple Computer for device drivers, and thus is compatible with many other programming languages the user may wish to use. As long as the pass-parameter conventions established by fishcamp engineering for the calls to the driver routines are adhered to, the programmer should have little problem in using the card with other languages. Please refer to the section on driver usage for information on calling the routines thru the Macintosh device manger.

And lastly, the programmer can always by-pass any of the supplied software routines and access the hardware directly. This may be desired when specialized routines peculiar to an application are required or maybe when the user wants to optimize the execution of a certain portion of code. This task will require a significant amount of work to implement, as well as requiring the user to have a thorough understanding of the architecture of the NBS-GPIB card. Every effort to provide the pertinent information on the design of the card has been done in order to assist the programmer in this task. Please reference the NBS-GPIB hardware reference manual for information specific to the architecture of the card.

The NBS-GPIB card is an 8-bit interface card with all hardware devices on the card memory mapped to distinct memory locations in the NuBus address space. All data accesses to/from the card are carried out over byte lane three of the NuBus interface. This translates to MC68020 cpu memory accesses from the Mac II with A0 and A1 bits set to 1's.

The card maps the NuBus slot address space into five distinct sections:

- PROM
- RAM
- Configuration Latch
- TMS9914A controller registers
- Interrupt enable/disable latch

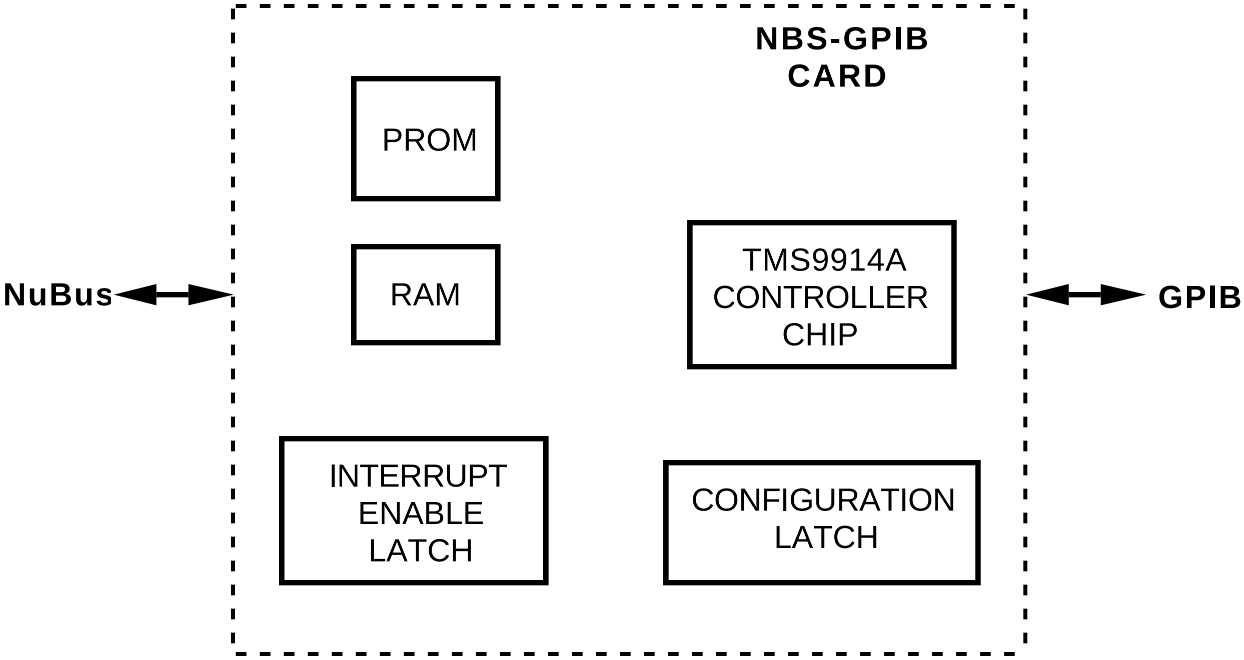


Figure 3.1 - NBS-GPIB Logical Devices.

The first section occupies the upper portion of the address space allocated to the card in the NuBus slot address space and is used to address the contents of the PROM containing the driver code for the card. This PROM has an 8K-byte total capacity. The Mac operating system reads the driver code from this PROM into system memory at reset time and then executes the code out of system memory from then on. The PROM is usually never accessed after this.

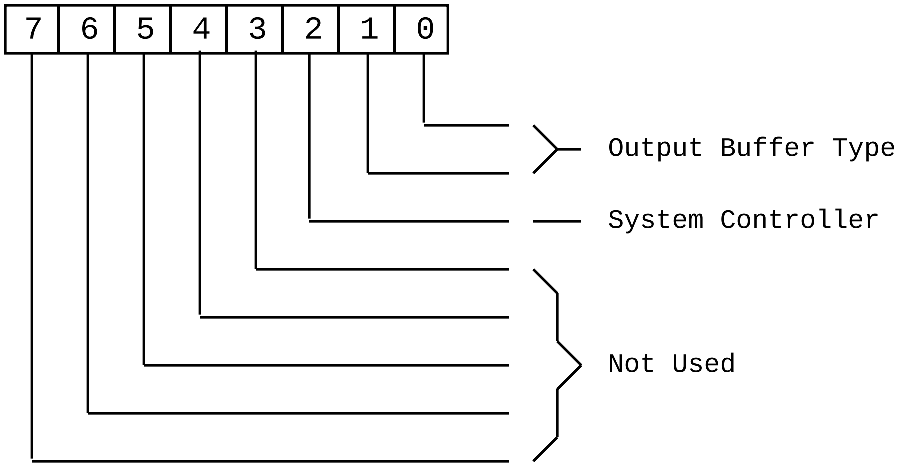
The second memory section of the card has a 2K-byte RAM buffer mapped to it. The driver routine in the PROM of the card uses some of the memory locations at the beginning of the RAM for local storage of variables used in execution of the driver routines of the card. The remaining locations are not currently used and may be accessed by the application programmer. Please refer to the driver source code listing for information as to which locations are currently used.

The third memory device on the card is a configuration output latch used to set some hardware lines necessary for operation of the NBS-GPIB card. The latch occupies a single byte in the memory map and is a write-only hardware device. The address of the configuration latch is defined as 'swaddr' in the include file for the driver. The supplied driver code maintains a RAM image of the configuration latch in order to be able to change the state of individual bits of the latch without affecting the state of the other bits. Applications writing to this latch should be aware of this because it can affect operation of the card at some time later when a driver routine attempts to alter a configuration bit. To be safe the application should also update the 'swimage' memory location after each memory write to the configuration latch.

Only the three least significant bits of the configuration latch are used in the hardware of the NBS-GPIB card. The two low order bits define the type of output buffers used for the data lines of the GPIB port. The NBS-GPIB card can be configured to have either three-state or open collector drivers on the GPIB data bus lines. Three-state type of buffers allow faster data transfers over the interface, but have the disadvantage of not being compatible with parallel-poll operations. During parallel-poll operations, each configured device on the bus must drive one bit of the eight bit data bus. Thus open collector drivers must be employed.

The design of the NBS-GPIB card also allows a hybrid mode of operation which gives the interface the best of both types of buffer outputs. This third mode sets the output buffers to their three-state output type except during parallel-poll operations, during which time the buffers automatically switch to the open-collector type of driver. After the parallel-poll operation completes, the buffers revert back to three-state operation. This is the mode which the driver defaults to upon a call to the 'GpibOpen' routine.

Bit 2 of the configuration latch defines whether or not the interface is configured as the 'system controller' on the GPIB bus. This needs to be set in order to configure the buffer which drives the control signals of the GPIB bus. In particular, it allows the card ultimate control of the 'ren' and 'ifc' lines of the bus.



Output Buffer Type:

Type	Value
open-collector	00
three-state	01
3-state with parallel poll	10

Figure 3.2 - Configuration Latch Bit Definition.

The fourth and most important block of memory addresses on the card map directly to the I/O registers of the TMS9914A GPIB controller chip used on the board. The chips data bus lines D7-D0 are mapped to the NuBus AD24-AD31 lines respectively. For definitions of the bits of the controller chip's registers consult the Texas Instrument's documentation on the device.

The last hardware device in the memory map is really two memory locations used in conjunction with each other to set the state of the interrupt enable latch on the card. The two locations are defined 'intenaddr' and 'intdisaddr' in the include file for the driver. The hardware design of the card uses the state of the interrupt enable latch to qualify any interrupts from the TMS9914A chip before passing them along to the NuBus 'NMRQ' interface line. Thus, to utilize interrupt operation on the NBS-GPIB card, the application must first set the appropriate bit in one of the two interrupt mask registers of the TMS9914A controller chip in order to enable the interrupt condition to be detected by the chip, and then secondly, set the interrupt enable latch in order to pass the interrupt on to the MAC. Any access to 'intenaddr' will enable interrupts from the card. Similarly, any access to 'intdisaddr' will disable interrupts from the interface card. The interrupt enable latch is always reset (interrupts disabled) after a power-up or system reset of the MAC.

	PROM - 8K BYTES	$FSFF 8003	$FSFF FFFF
	RAM - 2K BYTES	$FS00 0003	$FS00 1FFF
	INTENADDR	$FS04 0003	
	INTDISADDR	$FS06 0003	
	SWADDR	$FS08 0003	
	TMS9914A CHIP:		
READ ONLY	GPIBINT0	$FS02 0003	
	GPIBINT1	$FS02 0013	
	GPIBADST	$FS02 000B	
	GPIBBUS	$FS02 001B	
	GPIBCMD	$FS02 000F	
	GPIBDATAIN	$FS02 001F	
WRITE ONLY	GPIBINTM0	$FS02 0003	
	GPIBINTM1	$FS02 0013	
	GPIBAUXCMD	$FS02 001B	
	GPIBADDR	$FS02 0007	
	GPIBSERPOL	$FS02 0017	
	GPIBPARPOL	$FS02 000F	
	GPIBDATAOUT	$FS02 001F	

NOTE:
Only byte lane-3 addresses used by card.

Figure 3.1 - NBS-GPIB Memory Map

Because only byte lane three of the NuBus interface is used on the card, only every fourth memory location is valid in the NuBus address space. For instance, the 2K byte block of RAM is addressed starting at NuBus address $FS00 0003. The next byte of RAM is located at address $FS00 0007. And so on thru the remaining addresses. Application writers need to keep this in mind when writing the code for their program.

Included on the disk that comes with the interface card is an 'include file' the user may wish to use while writing programs which utilize the NBS-GPIB card. This file defines certain data structures and constants which are used by the driver routine for the card.

The 'GpibCtlBlk' structure is the single most important data type defined, in that all information passed to or from the driver routines are passed in various fields of this structure. This record is a 20 byte long data type with 7 distinct fields within it used. The format of 'GpibCtlBlk' is:

```
GpibCtlBlk =  RECORD
              csVar:        INTEGER;      { general purpose word has call specific
                                          data.  Refer to control call desired
                                          for variable definition. }
              csFlag:INTEGER;      { general purpose word has call specific
                                          data.  Refer to control call desired
                                          for variable definition. }
              csStatus:     INTEGER;      { call returned status information }
              csError:      INTEGER;      { call returned error information }
              csCount:      LONGINT;      { max characters to be inputted from the
                                          bus or the exact number of bytes to be sent
                                          out over the bus. For all operations,
                                          the actual number of bytes received or
                                          transmitted will be returned in this value}
              csDataBuf:    Ptr;          { used for actual data to/from the driver }
              csAddrList:   Ptr;          { pointer to a list of valid GPIB addresses
                                          of devices which will be partaking in the
                                          following transaction.  List will contain
                                          valid addresses terminated by the first
                                          non-valid address for Listeners.  For
                                          talkers there can only be one so only
                                          the byte pointed to is valid and no
                                          terminator is needed.  Not used for 'Send
                                          command'. }
              END;
```

Figure 4.1 - GpibCtlBlk Structure Definition.

Before calling the driver the application must first set the fields of the GpibCtlBlk correctly for the particular driver routine it is about to call. Each driver routine expects certain parameters in the various fields of the GpibCtlBlk. Not all of the fields are used at all times. Refer to the 'Driver Functions Interface' section of this manual for specifics about the field definitions for the driver function of interest.

Two fields within the GpibCtlBlk always have a consistent definition across the driver routines and are used to return error and status codes back to the calling program. These variables are the .csError and the .csStatus fields of the record.

The .csError field of the GbibCtlBlk structure is a 2 byte word used to return error code words about the operation of the driver routine during its execution. The following error codes have been defined for the current version of the driver:

```
*      Control call Error codes returned in 'csError'
ctlNoErr      EQU    $0000                ; default error code for control calls
ctlTimeEQU    $0001                ; timeout over GPIB buss
ctlBaddr      EQU    $0002                ; bad device address
ctlUnkErr     EQU    $0003                ; unknown error
ctlNinChg     EQU    $0004                ; interface not controller in charge
ctlInChg      EQU    $0005                ; interface not configured as device
```

Normal execution of a driver routine will return the ctlNoErr error code and the application should invoke its error recovery handler if the driver returns anything but this value. The ctlTime code usually signifies that there is some problem over the GPIB bus. This can happen if the application is trying to address a device which is not currently connected to the GPIB bus or is malfunctioning is some way. Refer to the 'SetTimot' driver function call for more information about timeouts.

Many routines of the driver are intended to be called when the interface is configured as a GPIB controller or a GPIB device, but not both. Exceptions do exist such as the routines which set the EOS character or which actually set-up the interface for operation as a GPIB controller. When routines expect to be configured a certain way, they check the current configuration of the interface card before execution of the driver routine. The two error codes ctlNinChg and ctlInChg signify that the interface was not properly configured for the type of driver routine called at the time of the error. When this happens, the routine will simply return with the error code set without completing execution of the routine.

Upon completion of driver routine calls, a status word is also returned along with the .csError word just described. The .csStatus field of the GbibCtlBlk structure is a 2 byte word used to return status bits about the operation of the driver routine during its execution. Each bit within the .csStatus word has been defined to signify a particular status condition. Figure 4.2 shows the status bits that have been defined for the current version of the driver.

GpibCtlBlk.csStatus Word:

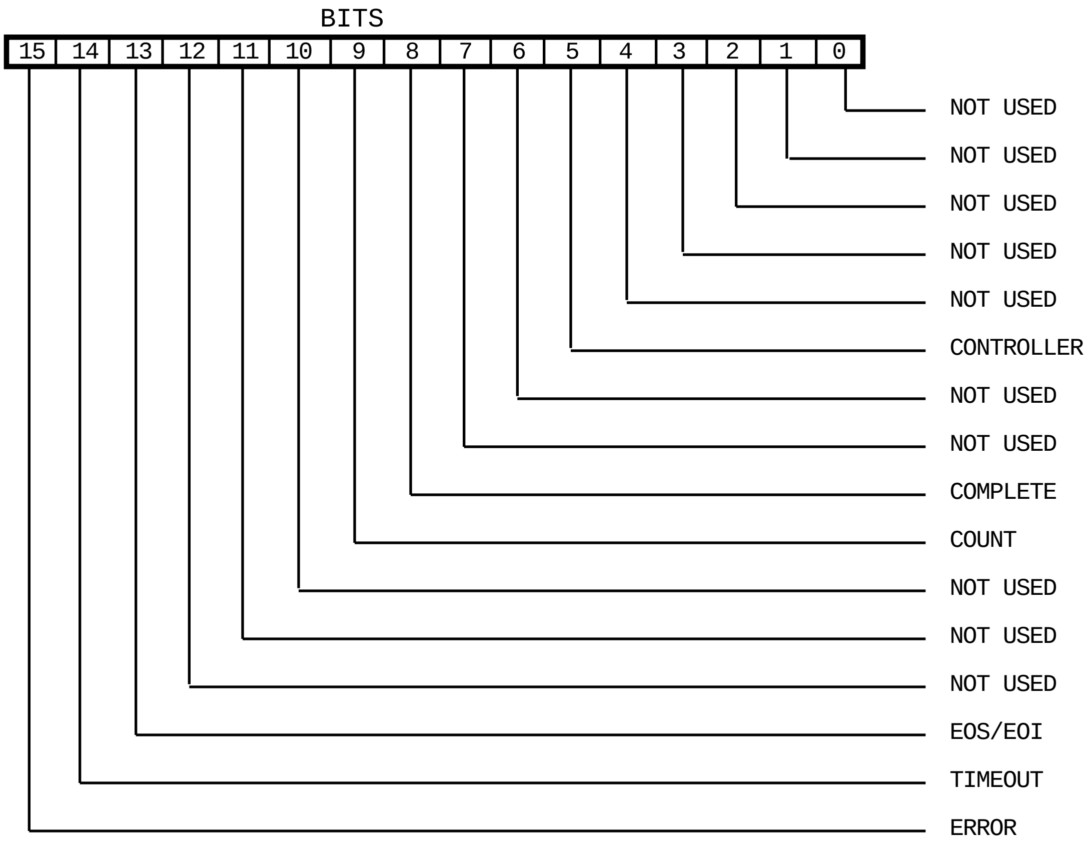


Figure 4.2 - GpibCtlBlk.csStatus Bit Definitions.

These bit definitions are defined in the include file as constants:

```
*      Status bit codes returned in 'csStatus'
stErr         EQU    $8000                ; error occurred during call
stTime        EQU    $4000                ; timeout occurred during call
stEnd         EQU    $2000                ; END or EOS occurred during operation
stCnt         EQU    $0200                ; I/O operation buffer size reached
stCmpltEQU    $0100                ; I/O operation completed during call
stCic         EQU    $0020                ; interface controller in charge
```

Bit 15 of the .csStatus field signifies that an error occurred during the execution of the driver call. It will always be accompanied by an error code of non-zero in the .csError field of the record. The other bits of the status word give the user more detailed information about the execution of the driver call and usually do not indicate error conditions.

All calls to the driver routines should be made thru the Device Manager of the Macintosh operating system. Consult the *Inside Macintosh* documentation for more specific information about device driver calls.

In order to make any calls to the driver of the NBS-GPIB card, or any driver for that matter, the driver must first be opened. The driver may be opened by calling the 'OpenSlot' function call of the Macintosh slot manager. Refer to the 'GpibOpen' function call documented in the 'Driver Functions Interface' section of this manual for more information. The call to open the driver will return a driver reference number which must be used for all subsequent calls to the driver.

All other calls to the NBS-GPIB driver, with the exception of the open and close calls, are made via device manager 'Control' calls. The driver does not support 'Prime' or 'Status' calls. The standard way of calling the Control call routine of a device driver is made with a call to the Device Manager 'Control' function or the lower level 'PBControl' function. The PASCAL interface routines supplied uses the 'PBControl' routine.

In either case the application must first set up the 'GpibCtlBlk' record as defined in the previous section of this manual. Then, a pointer to the GpibCtlBlk is passed in the first four bytes of a ParamBlockRec.csParam field. Finally, the PBControl call is made by using the driver reference number and a pointer to the ParamBlockRec as pass parameters. The NBS-GPIB driver only supports synchronous calls so the 'async' parameter should always be set to FALSE.

All driver control calls are made this way. The only difference is the ParamBlockRec.csCode parameter used and the way the GpibCtlBlk record is set up. The ParamBlockRec.csCode field should be set to the number of the particular control call being made.

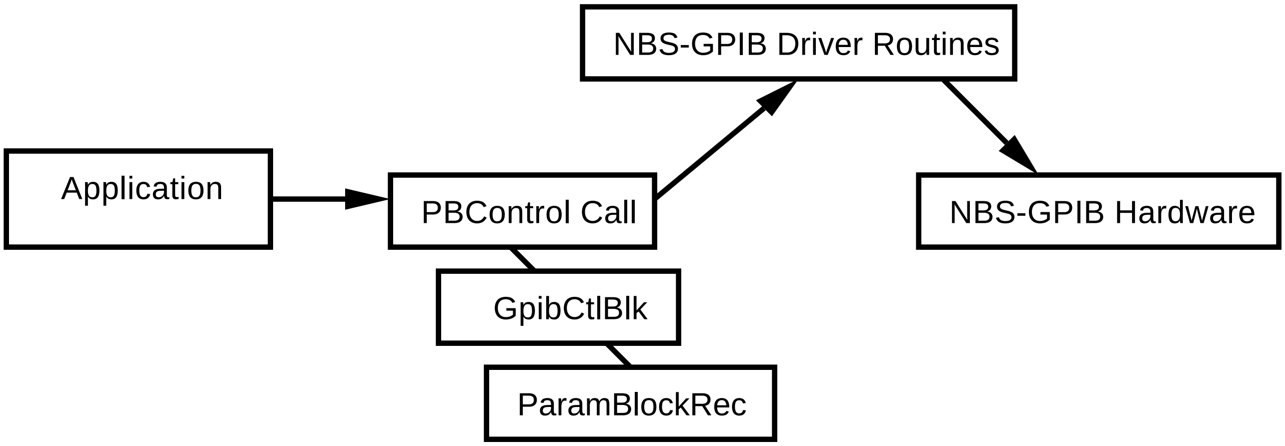
The NBS-GPIB driver only supports the following .csCode values:

Value	Driver routine
0	ContInit;
1	KillIo;
2	RemEnable;
3	Local;
4	Ifc;
5	SetEos;
6	SetMyAddr;
7	Trig;
8	DevClr;
9	PpEnable;
10	PpDisable;
11	PpUConfig;
12	CParPoll;
13	CSerPoll;
14	CRcv;
15	CSend;
16	SendCmd;
17	NContInit;
18	CXfer;
19	CPassCntrl;
20	CRcvCntrl;
21	Rcv;
22	Send;
23	EnInter;
24	SetOut;
25	Read;
26	Write;
27	NewTimout;

The way the GpibCtlBlk is set up is determined by the particular driver routine being called. Each driver routine documents these values in the 'Driver Functions Interface' section of this manual.

Finally, after all calls to the driver have been made, the driver must be closed. This is done with the 'CloseDriver' function call.

The reader should refer to the PASCAL interface file source code listing for examples of what was just described.



```
Open Driver

For Each Driver Call Made:
    Set up GpibCtlBlk
    Set up ParamBlockRec
    Put Pointer to GpibCtlBlk in ParamBlockRec.csParam
    Call PBControl

Close Driver
```

Figure 5.1 - Driver call method.

ContInit Initialize interface as GPIB Controller **ContInit**

Purpose: This call is used to set up the NBS-GPIB card as the controller in charge of the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '0' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs must call this routine to initialize the card to perform as the controller in charge of the GPIB bus before calling any of the routines designated for use by a GPIB controller.

```
ContInit:
     Set flag as 'controller' in local storage
     Issue software reset to TMS9914 chip
     Disable all interrupt mask bits
     Write address from local storage to TMS9914 chip
     Set fast T1 mode
     Set 3-state GPIB drivers
     Set 'system controller' bit
     Clear software reset to TMS9914 chip
     Send 'interface clear' for 1 ms
     Turn on the GPIB 'REN' line
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     refNum:              INTEGER;
     paramAddr:           LONGINT;
     myStatus:            INTEGER;
     myError:             INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                   { not used }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := NIL;             { not used }
     myGpibCtlBlk.csAddrList := NIL;            { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;            { not used }
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 0;                     { for 'ContInit' }
     paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;          { interface's status }
     myError := myGpibCtlBlk.csError;            { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

CParPoll Conduct a Parallel Poll **CParPoll**

Purpose: This call is used to conduct a parallel poll over the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '12' for this call
```

Output:

```
     gpibCtlBlk.csVar      - parallel poll response byte
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to conduct a parallel poll over the GPIB bus. The driver will wait approximately 125 uS for a response from all devices on the GPIB bus to be returned and then return the response in the low byte of gpibCtlBlk.csVar back to the calling program. This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CParPoll:
     Send 'RPP' command to TMS9914 chip
     Delay 125 uS
     Read poll response byte
     Send 'RPPCLR' command to TMS9914 chip
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;
     pollResponse:  signedByte;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := NIL;           { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 12;                  { for 'CParPoll' }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }
     pollResponse := signedByte(myGpibCtlBlk.csVar);   { result of poll }

{ The parallel poll response byte will be returned in the low byte of
'myGpibCtlBlk.csVar'.  The success of the device manager call is returned in
'err'.  The driver's status and result codes are returned in
myGpibCtlBlk.csStatus and myGpibCtlBlk.csError respectively.  The driver
reference number used is that which was returned by the call to 'GpibOpen'.}
END;
```

CPassCntrl Pass Control CPassCntrl

Purpose: This call is used to pass the function of active controller to another device on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
gpibCtlBlk.csAddrList  - pointer to new controller address
paramBlock.ioRefNum    - value returned from 'GpibOpen' call
paramBlock.csCode      - '19' for this call
```

Output:

```
gpibCtlBlk.csStatus    - call return status information
gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine when they are currently the active controller of the GPIB bus and they wish to relinquish control of the bus to another device. The specified device will then be the active controller, if the handoff goes well, after the return from this call. Normally some software protocol is set up to inform the new controller to expect the transfer of control. It is assumed also that the specified device has the capability to perform as a controller of the GPIB bus. The routine will return the 'ctlNinChg' error code if the board is not currently the active controller of the GPIB bus. See the 'CRcvCntrl' function for information on receiving control.

The calling program must pass a pointer to a talker address of the device it wishes to pass control to. This address should be a byte corresponding to a valid GPIB talker address in the range of 0x40 to 0x5e. The routine returns the 'ctlBaddr' error code if the value is not a valid talker address or the caller specified the address of the interface itself (MTA).

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CPassCntrl:
     Get talker address
     IF address is in range 0x40 to 0x5e and is not 'MTA' THEN
            Send 'TCT' command over the GPIB bus
            Wait for command to be accepted
            Send 'RLC' command to TMS9914 chip
            Clear 'am-controller' flag in local storage
     END IF valid talker address
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;
     talker:        Str255;              { the new controller }

BEGIN
     { first set up the talker address.  This is a pointer to an address
            of the device we wish to pass control to.  Remember that
            talker addresses are offset by + 0x40 in the IEEE-488 world.
            Later we will pass a pointer to the address by using the
     @talker[1] nomenclature.  Remember that in the PASCAL language
     the first byte of a string is the string length parameter and
     that it is the second byte which is the real first character of
     the string.  }

     talker:= 'U';                               { talker address '21' }

     { next, set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                    { not used }
     myGpibCtlBlk.csFlag := 0;                   { not used }
     myGpibCtlBlk.csStatus := 0;                 { a return value }
     myGpibCtlBlk.csError := 0;                  { a return value }
     myGpibCtlBlk.csCount := 0;                  { not used }
     myGpibCtlBlk.csDataBuf := NIL;              { not used }
     myGpibCtlBlk.csAddrList := @talker[1];      { pointer to talker }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                  { not used }
     paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
     paramBlock.csCode := 19;                    { for 'CPassCntrl' call }
     paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;          { interface's status }
     myError := myGpibCtlBlk.csError;            { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
```

```
END;
```

CRcv Controller Receive Data **CRcv**

Purpose: This call is used to allow the controller to receive data from a device on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csDataBuf    - pointer to receive data buffer
     gpibCtlBlk.csAddrList   - pointer to talker address
     gpibCtlBlk.csCount      - maximum number of bytes to receive
     gpibCtlBlk.csFlag       - 'look for EOS character' flag
     paramBlock.ioRefNum     - value returned from 'GpibOpen' call
     paramBlock.csCode       - '14' for this call
```

Output:

```
     gpibCtlBlk.csDataBuf    - receive data
     gpibCtlBlk.csCount      - actual number of bytes received
     gpibCtlBlk.csStatus     - call return status information
     gpibCtlBlk.csError      - call return error code
```

Details: Application programs call this routine to allow the controller to receive data from a device on the GPIB bus. The driver routine will take care of all of the GPIB bus addressing in order to initiate the actual data transfer.

The calling program must pass a pointer to a talker address of the device it wishes to receive data from in the `gpibCtlBlk.csAddrList` field. This address must be a valid GPIB talker address in the range of 0x40 to 0x5e. Only one GPIB talker can be specified.

It is the responsibility of the calling program to allocate the buffer space used to hold the receive characters. A pointer to the first byte of this buffer should be passed in the `gpibCtlBlk.csDataBuf` field. Enough space to hold `gpibCtlBlk.csCount` characters should be allocated for the buffer. The driver will terminate the data transfer when this maximum character count has been received if the transfer is not terminated earlier by some other condition. This is usually not the normal way a GPIB data transfer terminates however and will probably leave the talker in a strange state which may later have to be cleared by the controller. The `gpibCtlBlk.csCount` parameter is a longword variable which is used to return to the calling program the actual number of characters received during the current transaction.

During normal data transfers, the receive operation is terminated when the EOI line is driven by the talker during a data byte transfer. It is also possible to terminate on the occurrence of a particular byte in the data stream. This byte is called the EOS character and is specified to the driver by a call to the 'SetEos' control call. The calling program must set the `gpibCtlBlk.csFlag` parameter to a non-zero value in order to enable termination on the EOS character, otherwise no data checking will occur.

During the execution of this call, the driver will attempt to detect if there is no-response from the talker on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to receive a data byte over the bus it will enter a loop which verifies whether or not the data byte has been sent over the GPIB bus. If the byte is not sent after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CRcv:
    Set pointer to input buffer
    Set pointer to talker address
    IF valid talker address
          Output talker address over GPIB bus
          Send 'UNL' command over GPIB bus
          Make ourselves the listener on the GPIB bus
          Send 'HDFA' command to TMS9914 chip
          Send 'LON' command to TMS9914 chip
          Send 'GTS' command to TMS9914 chip
          While still receiving data
                Wait for data byte from GPIB bus
                If byte has EOI with it
                      Get data byte from GPIB bus
                      Store byte in buffer
                      Increment character count
                      Flag EOI received in .csStatus field
                      Send 'TCS' command to TMS9914 chip
                      Send 'RDHF' command to TMS9914 chip
                      Send 'HDACLR' command to TMS9914 chip
                      Put character count in .csCount field
                      Return to caller
                End if byte had EOI with it
                Else if byte did not have EOI with it
                      Get data byte from GPIB bus
                      Store data byte in buffer
                      Increment data buffer pointer
                      Increment character count
                      If we should be checking for EOS character
                            If this byte was the EOS character
                                  Flag EOI received in .csStatus
                                  Send 'TCS' command to TMS9914
                                  Send 'RDHF' command to TMS9914
                                  Send 'HDACLR' cmd to TMS9914
                                  Put char count in .csCount
```

```
                              Return to caller
                        End if this was the EOS character
                  End if we are checking for EOS character
                  If max buffer size reached
                        Flag buffer size hit in .csStatus
                        Send 'TCS' command to TMS9914 chip
                        Send 'RDHF' command to TMS9914 chip
                        Send 'HDACLR' command to TMS9914
                        Put character count in .csCount
                        Return to caller
                  End if max buffer size reached
                  Send 'RDHF' command to TMS9914 chip
            End if byte did not have EOI with it
      End While still receiving data
End If valid talker address
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     aStr:         Str255;
     byteCnt:      LONGINT;              { data count }
     dataBuffer:   Handle;       { Rx data goes here }

BEGIN
     { first set up the talker address.  This is the address of the device
            we wish to receive data from.  Remember that talker addresses
            are offset by + 0x40 in the IEEE-488 world.  Only one talker
            address can be specified.  Later we will pass a pointer to the
            talker by using the @aStr[1] nomenclature. }

     aStr:= 'H';                         { talker at address '8' }

     { next allocate space for the receive data buffer. }
     byteCnt := 1000;                           { max chars to receive }
     dataBuffer := NewHandle(byteCnt);          { reserve memory for data }

     IF (dataBuffer <> NIL) THEN                { if we have enough memory }
            BEGIN
            HLock(dataBuffer);                  { lock the memory block
                                                  during I/O operation }

            { next set up the driver's control call parameters }
            myGpibCtlBlk.csVar := 0;            { not used }
            myGpibCtlBlk.csFlag := 1;           { check for EOS character }
            myGpibCtlBlk.csStatus := 0;         { a return value }
            myGpibCtlBlk.csError := 0;          { a return value }
            myGpibCtlBlk.csCount := byteCnt ;   { max buffer size }
            myGpibCtlBlk.csDataBuf := dataBuffer^; { the input buffer }
            myGpibCtlBlk.csAddrList := @aStr[1];   { the device address }

            { now set up the device manager's control call parameters }
            paramBlock.ioCompletion := NIL;
            paramBlock.ioVRefNum := 0;          { not used }
            paramBlock.ioRefNum := refNum;      { from 'GpibOpen' call }
            paramBlock.csCode := 14;            { for 'CRcv' call }
            paramAddr := LONGINT(@myGpibCtlBlk);{ address of GPIB params }
            paramBlock.csParam[1] := LoWord(paramAddr);
            paramBlock.csParam[0] := HiWord(paramAddr);

            err := PBControl(@paramBlock, FALSE);
            myStatus := myGpibCtlBlk.csStatus;  { interface's status }
            myError := myGpibCtlBlk.csError;    { driver's result code }

            HUnlock(dataBuffer);                { no more need to lock }

            END    { if databuffer allocated }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.  The data will be placed in
dataBuffer and 'myGpibCtlBlk.csCount' will contain the actual number of
characters transferred into the dataBuffer. }
END;
```

CRcvCntrl Receive Control **CRcvCntrl**

Purpose: This call is used to receive control from the currently active controller on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
paramBlock.ioRefNum     - value returned from 'GpibOpen' call
paramBlock.csCode       - '20' for this call
```

Output:

```
gpibCtlBlk.csStatus     - call return status information
gpibCtlBlk.csError      - call return error code
```

Details: An application program calls this routine when it is expecting control to be relinquished to it from the currently active controller on the GPIB bus. The local interface will then be the active controller, if the handoff goes well, after the return from this call. Normally some software protocol is set up to inform the local device to expect the transfer of control. The routine will return the 'ctlInChg' error code if the board is currently the active controller of the GPIB bus. See the 'CPassCntrl' function for more information on transferring control.

The calling program must have enabled 'DAC' holdoffs on 'unrecognized commands' in the TMS9914 chip before calling this routine. No assumptions are made in the driver about which interrupt mask bits are set by the application so it is the responsibility of the calling program to set up the 'DAC' holdoff. This can be accomplished by calling the 'GpibWrAddr' function prior to calling the 'CRcvCntrl' routine. See the example code below for more information on enabling the 'DAC' holdoff. More information can also be found in the documentation from Texas Instruments on the TMS9914A chip.

This call should only be made if the NBS-GPIB card is not already the active controller of the GPIB bus. The routine will return the 'ctlInChg' error code if the board is currently the active controller of the GPIB bus.

```
CRcvCntrl:
    Get INT1 register status byte from TMS9914 chip
    IF 'UCGM' bit set THEN
          IF it is the 'TCT' command THEN
                IF we were addressed to talk THEN
                      Send 'RQC' command to TMS9914 chip
                      Send 'DACR' command to TMS9914 chip
                      Clear 'am-controller' flag
                END IF we were addressed to talk
          END IF it was the 'take control' command
    END IF 'unrecognized command' detected
```

Example:

```
         VAR
              err:          OSErr;
              paramBlock:   ParamBlockRec;
              myGpibCtlBlk: GpibCtlBlk;
              paramAddr:    LONGINT;
              refNum:       INTEGER;
              myStatus:     INTEGER;
              myError:      INTEGER;

         BEGIN
              { First we must enable 'data accepted' holdoffs on the GPIB bus when
         'unrecognized commands' are detected by the TMS9914A chip.  This is
         accomplished by setting bit 5 of the interrupt mask register 1 on the GPIB
         controller chip of the NBS-GPIB board.  This register is located at board
         address $FS20010 on the interface card (ref. board memory map), where 'S'
         refers to the NuBus slot address where the board is plugged into.  The driver
         function call 'GpibWrAddr' is used to accomplish the setting of the mask bit
         by writing a 0x20 (bit 5) to the register.    }

                     { enable DAC holdoff on 'unrecognized command' }
              err := GpibWrAddr(refNum, $20010, $20, myStatus, myError);

              { next, set up the driver's control call parameters }
              myGpibCtlBlk.csVar := 0;                  { not used }
              myGpibCtlBlk.csFlag := 0;                 { not used }
              myGpibCtlBlk.csStatus := 0;               { a return value }
              myGpibCtlBlk.csError := 0;                { a return value }
              myGpibCtlBlk.csCount := 0;                { not used }
              myGpibCtlBlk.csDataBuf := NIL;            { not used }
              myGpibCtlBlk.csAddrList := NIL;           { not used }

              { now set up the device manager's control call parameters }
              paramBlock.ioCompletion := NIL;
              paramBlock.ioVRefNum := 0;                { not used }
              paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
              paramBlock.csCode := 20;                  { for 'CRcvCntrl' call }
              paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
              paramBlock.csParam[1] := LoWord(paramAddr);
              paramBlock.csParam[0] := HiWord(paramAddr);

              REPEAT              { keep trying until control is passed to me }
                     BEGIN
                     err := PBControl(@paramBlock, FALSE);
                     END; UNTIL (myGpibCtlBlk.csError = 0);

              myStatus := myGpibCtlBlk.csStatus;        { interface's status }
              myError := myGpibCtlBlk.csError;          { driver's result code }

         { The success of the device manager call is returned in 'err'.  The driver's
         status and result codes are returned in myGpibCtlBlk.csStatus and
         myGpibCtlBlk.csError respectively.  The driver reference number used is that
         which was returned by the call to 'GpibOpen'.}
END;
```

CSend Controller Send Data **CSend**

Purpose: This call is used to allow the controller to send data to one or more devices on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
gpibCtlBlk.csDataBuf    - pointer to data buffer
gpibCtlBlk.csAddrList   - pointer to listener addresses
gpibCtlBlk.csCount      - number of bytes to send
gpibCtlBlk.csVar        - 'look for EOS character' flag
gpibCtlBlk.csFlag       - 'send EOI with last character' flag
paramBlock.ioRefNum     - value returned from 'GpibOpen' call
paramBlock.csCode       - '15' for this call
```

Output:

```
gpibCtlBlk.csCount      - actual number of bytes sent
gpibCtlBlk.csStatus     - call return status information
gpibCtlBlk.csError      - call return error code
```

Details: Application programs call this routine to allow the controller to send data to one or more devices on the GPIB bus. The driver routine will take care of all of the GPIB bus addressing in order to initiate the actual data transfer.

The calling program must pass a pointer to a list of listener addresses, of the devices it wishes to send data to, in the `gpibCtlBlk.csAddrList` field. This list should be composed of a string of bytes, each one corresponding to a valid GPIB listener address in the range of 0x20 to 0x3e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

A pointer to the first byte of the data buffer should be passed in the `gpibCtlBlk.csDataBuf` field. The calling program shall use the `gpibCtlBlk.csCount` field to specify the number of characters to transmit. The driver will terminate the data transfer when this character count has been sent if the transfer is not terminated earlier by some other condition. The `gpibCtlBlk.csCount` parameter is a longword variable which is used to return to the calling program the actual number of characters transmitted during the current transaction.

During normal data transfers, the transmit operation is terminated when the specified number of characters have been sent. It is also possible to terminate on the occurrence of a particular byte in the data stream. This byte is called the EOS character and is specified to the driver by a call to the 'SetEos' control call. The calling program must set the `gpibCtlBlk.csVar` parameter to a non-zero value in order to enable termination on the EOS character, otherwise no data checking will occur.

The last character sent will be sent with the EOI line on the GPIB bus pulled low if the `gpibCtlBlk.csFlag` variable is set to a non-zero value.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CSend:
     Set pointer to data buffer
     Send our address as the talker address on the GPIB bus
     Send 'UNL' over GPIB bus
     Set pointer to listener address list
     WHILE valid listener address
           Output listener address over GPIB bus
     End WHILE valid listener address
     Send 'TON' command to TMS9914 chip
     Send 'GTS' command to TMS9914 chip
     Clear character counter
     While still sending data
           Get data byte
           Increment data byte pointer
           If last byte to send
                 Signal count hit in .csStatus
                 IF .csFlag set then
                       Send 'FEOI' command to TMS9914 chip
                       signal 'EOI' sent in .csStatus
                 End if .csFlag set
                 Send data byte over GPIB bus
                 Increment character counter
                 Send 'TCA' command to TMS9914 chip
                 Put character count in .csCount field
                 Return to caller
           End if last byte to send
           If we should be checking for EOS character
                 If this byte was the EOS character
                       signal 'EOI' sent in .csStatus
                       IF .csFlag set then
                             Send 'FEOI' command to TMS9914 chip
```

```
                        signal 'EOI' sent in .csStatus
                  End if .csFlag set
                  Send data byte over GPIB bus
                  Increment character counter
                  Send 'TCA' command to TMS9914 chip
                  Put character count in .csCount field
                  Return to caller
            End if this was the EOS character
      End if we are checking for EOS character
      Send data byte over GPIB bus
      Increment character counter
      Wait for data byte to be accepted over GPIB bus
End While still sending data
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     listeners:    Str255;        { listener address list }
     byteCnt:      LONGINT;              { data count }
     sendData:     Str255;        { Tx data goes here }

BEGIN
     { first set up the listener address list.  This is a list of addresses
            of devices we wish to receive the data.  Remember that
            listener addresses are offset by + 0x20 in the IEEE-488 world.
            The list should be terminated by a non-valid listener address.
            In this case we use the ASCII <z> which meets the requirement
            by having a value of 0x7a.  Later we will pass a pointer to the
            first listener by using the @listeners[1] nomenclature.
            Remember that in the PASCAL language the first byte of a string
            is the string length parameter and that it is the second byte
            which is the real first character of the string.  }

     listeners := '(+7z';         { 3 listeners at addresses 8, 11, and 23 }

     sendData := 'hello world';          { actual data to send }
     byteCnt := LONGINT(Length(sendData));

     { next set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;            { don't check for EOS character }
     myGpibCtlBlk.csFlag := 1;           { send EOI with last character }
     myGpibCtlBlk.csStatus := 0;         { a return value }
     myGpibCtlBlk.csError := 0;          { a return value }
     myGpibCtlBlk.csCount := byteCnt ;   { max buffer size }
     myGpibCtlBlk.csDataBuf := @sendData[1]; { the first data byte }
     myGpibCtlBlk.csAddrList := @listeners[1];  { the device addresses }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;          { not used }
     paramBlock.ioRefNum := refNum;      { from 'GpibOpen' call }
     paramBlock.csCode := 15;            { for 'CSend' call }
     paramAddr := LONGINT(@myGpibCtlBlk);{ address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);
     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;  { interface's status }
     myError := myGpibCtlBlk.csError;    { driver's result code }


{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.  The 'myGpibCtlBlk.csCount'
field will contain the actual number of characters transferred over the GPIB
bus. }
END;
```

CSerPoll Serial Poll **CSerPoll**

Purpose: This call is used to serial poll one or more devices on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

gpibCtlBlk.csAddrList	- pointer to talker list
gpibCtlBlk.csDataBuf	- pointer to response buffer
paramBlock.ioRefNum	- value returned from 'GpibOpen' call
paramBlock.csCode	- '13' for this call

Output:

gpibCtlBlk.csDataBuf	- response bytes in buffer
gpibCtlBlk.csStatus	- call return status information
gpibCtlBlk.csError	- call return error code

Details: Application programs call this routine to perform a serial poll operation on one or more devices on the GPIB bus. Serial polls are usually performed in response to one of the instruments on the bus pulling the SRQ line low. Since when more than two devices are interconnected on the bus there is an uncertainty over which device is requesting service, the controller can poll each device sequentially to determine the address of the requesting instrument.

This routine allows the calling application to specify a series of addresses upon which to perform the serial poll. The program must pass a pointer to a list of talker addresses of devices it wishes to poll. This list should be composed of a string of bytes, each one corresponding to a valid GPIB talker address in the range of 0x40 to 0x5e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

The calling application must also pass a pointer to a buffer which will be used to hold the response bytes from all of the poll operations. The poll response bytes will be stored in sequential byte locations within the buffer in the same order as the device addresses were stored in the talker list. The memory space for the buffer should be allocated prior to calling the routine and contain enough space to hold all of the requested response bytes.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CSerPoll:
    Set pointer to first talker
    Set pointer to first location in response buffer
    Send 'SPE' over GPIB bus
    Send 'HDFA' command to TMS9914 chip
    While current address is in range 0x40 to 0x5e
          Send address of current talker over GPIB bus
          Increment address list pointer
          Send 'LON' command to TMS9914 chip
          Send 'GTS' command to TMS9914 chip
          Wait for 'byte in' flag from TMS9914 chip
          Send 'TCS' command to TMS9914 chip
          Get serial poll response byte from GPIB bus
          Store response byte in response buffer
          Increment pointer to next location in response buffer
          Send 'RHDF' command to TMS9914 chip
    End While
    Send 'SPD' command over GPIB bus
    Send 'HDACLR' command to TMS9914 chip
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;
     talkers:       Str255;              { talker address list }
     responses:     Str255;              { response bytes }
     serPoll:       ARRAY[1..3] OF INTEGER;     { individual responses }

BEGIN
     { first set up the talker address list.  This is a list of addresses
            of devices we wish to perform a serial poll on.  Remember that
            talker addresses are offset by + 0x40 in the IEEE-488 world.
            The list should be terminated by a non-valid talker address.
            In this case we use the ASCII <z> which meets the requirement
            by having a value of 0x7a.  Later we will pass a pointer to the
            first talker by using the @talkers[1] nomenclature.
            Remember that in the PASCAL language the first byte of a string
            is the string length parameter and that it is the second byte
            which is the real first character of the string.  }

     talkers := 'HKWz';          { 3 talkers at addresses 8, 11, and 23 }

     responses := '123';         { arbitrary 3 char string for responses }
     responses[1] := char(0);
     responses[2] := char(0);
     responses[3] := char(0);

     { next set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                   { not used }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := @responses[1];   { pointer to response buf }
     myGpibCtlBlk.csAddrList := @talkers[1];    { pointer to talker list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 13;                   { for 'CSerPoll' call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }
     serPoll[1] := INTEGER(responses[1]);
     serPoll[2] := INTEGER(responses[2]);
     serPoll[3] := INTEGER(responses[3]);

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The serial poll response bytes for the
three devices end up in the three locations of the 'serPoll' array.  The
driver reference number is that which was returned by the call to
'GpibOpen'.}
END;
```

CXfer Initiate the transfer of data CXfer

Purpose: This call is used by the controller in charge of the GPIB bus to initiate a data transfer between two devices where the controller does not participate in the transfer.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csAddrList   - pointer to device list
     paramBlock.ioRefNum     - value returned from 'GpibOpen' call
     paramBlock.csCode       - '18' for this call
```

Output:

```
     gpibCtlBlk.csStatus     - call return status information
     gpibCtlBlk.csError      - call return error code
```

Details: The program must pass a pointer to a list of device addresses which will participate in the transfer. This list should be composed of a string of bytes, the first one corresponding to a valid GPIB talker address in the range of 0x40 to 0x5e, followed by one or more valid GPIB listener addresses in the range of 0x20 to 0x3e.. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
CXfer:
     Set pointer to first device
     If address is in range 0x40 to 0x5e then
           Send talker address over GPIB bus
           Wait for GPIB bus free
     Else
           Return 'bad address' to caller
     End If
     Send 'UNL' over GPIB bus
```

```
While current address is in range 0x20 to 0x3e
      Send address of current listener over GPIB bus
      Increment address list pointer
      Wait for GPIB bus free
End While
Send 'SHDW' command to TMS9914 chip
Send 'HDFE' command to TMS9914 chip
Send 'LON' command to TMS9914 chip
Send 'GTS' command to TMS9914 chip
Wait for 'EOI' received             ; data being transferred
Send 'TCS' command to TMS9914 chip
Wait for GPIB bus free
Send 'RHDF' command to TMS9914 chip
Send 'HDECLR' command to TMS9914 chip
Send 'SHDCLR' command to TMS9914 chip
Return to caller
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     devices:      Str255;               { device address list }

BEGIN
     { First set up the device address list.  This is a list of addresses
            of devices we wish to participate in the data transfer.
            The first byte in the list specifies the talker and is followed
     by the desired listeners in the transaction.  Multiple listener
     addresses can be specified.  Remember that talker addresses are
     offset by + 0x40 and listener address are offset by + 0x20 in
     the IEEE-488 world.  The list should be terminated by a non-
     valid GPIB address.  In this case we use the ASCII <z> which
     meets the requirement by having a value of 0x7a.  Later we will
     pass a pointer to the talker address by using the @devices[1]
     nomenclature.  Remember that in the PASCAL language the first
     byte of a string is the string length parameter and that it is
     the second byte which is the real first character of the string.
     }

     devices:= 'U!&z';            { talker '21', listeners '1' and '6' }

     { next set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                    { not used }
     myGpibCtlBlk.csFlag := 0;                   { not used }
     myGpibCtlBlk.csStatus := 0;                 { a return value }
     myGpibCtlBlk.csError := 0;                  { a return value }
     myGpibCtlBlk.csCount := 0;                  { not used }
     myGpibCtlBlk.csDataBuf := NIL;              { not used }
     myGpibCtlBlk.csAddrList := @devices[1];     { pointer to device list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                  { not used }
     paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
     paramBlock.csCode := 18;                    { for 'CXfer' call }
     paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
```

```
              paramBlock.csParam[1] := LoWord(paramAddr);
              paramBlock.csParam[0] := HiWord(paramAddr);

              err := PBControl(@paramBlock, FALSE);
              myStatus := myGpibCtlBlk.csStatus;         { interface's status }
              myError := myGpibCtlBlk.csError;           { driver's result code }

         { The success of the device manager call is returned in 'err'.  The driver's
         status and result codes are returned in myGpibCtlBlk.csStatus and
         myGpibCtlBlk.csError respectively.  The driver reference number is that which
         was returned by the call to 'GpibOpen'.}
END;
```

DevClr Device Clear **DevClr**

Purpose: This call is used to cause a device clear (SDC) command to be sent to designated devices on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
gpibCtlBlk.csAddrList  - pointer to listener address list
paramBlock.ioRefNum    - value returned from 'GpibOpen' call
paramBlock.csCode      - '8' for this call
```

Output:

```
gpibCtlBlk.csStatus    - call return status information
gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine to cause a device clear (SDC) command to be sent to selected devices on the GPIB bus. This is usually done to clear the interface logic of a particular device on the GPIB bus. This is distinct from the IFC command which affects every device connected on the bus. The interface manual for the particular instrument should be consulted for specifics about it's response to this command.

The calling program must pass a pointer to a list of listener addresses of devices it wishes to send the device clear command to. This list should be composed of a string of bytes, each one corresponding to a valid GPIB listener address in the range of 0x20 to 0x3e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
DevClr:
```

```
Send 'universal unlisten' over the GPIB bus
Point to first listener address
While current address is in range 0x20 to 0x3e
      Send address of current listener over GPIB bus
      Increment address list pointer
End While
Send 'SDC' command over the GPIB bus
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     listeners:    Str255;              { the list of listeners }

BEGIN
     { first set up the listener address list.  This is a list of addresses
           of devices we wish to receive the 'SDC' command.  Remember that
           listener addresses are offset by + 0x20 in the IEEE-488 world.
           The list should be terminated by a non-valid listener address.
           In this case we use the ASCII <z> which meets the requirement
           by having a value of 0x7a.  Later we will pass a pointer to the
           first listener by using the @listeners[1] nomenclature.
           Remember that in the PASCAL language the first byte of a string
           is the string length parameter and that it is the second byte
           which is the real first character of the string.  }

     listeners := '(+7z';        { 3 listeners at addresses 8, 11, and 23 }

     { next, set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := @listeners[1]; { pointer to listener list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 8;                   { for 'DevClr' call }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;        { interface's status }
     myError := myGpibCtlBlk.csError;          { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

EnInter Enable/Disable Board Interrupts **EnInter**

Purpose: This call is used to enable or disable the ability of the NBS-GPIB board to interrupt the MAC.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csFlag     - Enable/Disable BOOLEAN.
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '23' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine in order to enable or disable the ability of the NBS-GPIB card to interrupt the MAC. The interface card has been designed such that any interrupts generated by the TMS9914A GPIB controller chip used on the card can be allowed to interrupt the MAC.

In order for an interrupt to be generated however, two conditions must be met. First, the interrupt mask registers in the TMS9914A chip must be programmed to allow the controller chip to generate an interrupt. Refer to the documentation from Texas Instruments on the TMS9914A chip for more detailed information regarding the possible interrupt conditions available for the controller chip. The second condition which must be satisfied before the NBS-GPIB card can interrupt the MAC is that the interrupt enable latch for the board must be set in order to pass the interrupt from the controller chip thru to the MAC. This driver function call has been provided to allow the application to set the state of the board's interrupt enable latch.

The interrupt enable latch defaults to the state which inhibits all interrupts from the board upon a reset of the MAC computer. Revision 1.1 of the NBS-GPIB driver does not utilize the interrupt capability of the card. This routine is provided for the use of those application developers wishing to write their own interrupt driven routines for the card. The application should disable interrupts before closing the driver if they had been previously enabled during the execution of the program. Refer to 'The Device Manager' chapter of *Inside Macintosh Volume V* for more information regarding interrupts and slot devices.

```
EnInter:
     IF .csFlag non-zero THEN
            Enable board interrupts
     ELSE
            Disable Board interrupts
```

Example:

```
   VAR
        err:                 OSErr;
        paramBlock:          ParamBlockRec;
        myGpibCtlBlk:        GpibCtlBlk;
        paramAddr:           LONGINT;
        refNum:              INTEGER;
        myStatus:            INTEGER;
        myError:             INTEGER;

   BEGIN
        { first set up the driver's control call parameters }
        myGpibCtlBlk.csVar := 0;                  { not used }
        myGpibCtlBlk.csFlag := $ffff;             { enable interrupts }
        myGpibCtlBlk.csStatus := 0;               { a return value }
        myGpibCtlBlk.csError := 0;                { a return value }
        myGpibCtlBlk.csCount := 0;                { not used }
        myGpibCtlBlk.csDataBuf := NIL;            { not used }
        myGpibCtlBlk.csAddrList := NIL;           { not used }

        { now set up the device manager's control call parameters }
        paramBlock.ioCompletion := NIL;           { not used }
        paramBlock.ioVRefNum := 0;                { not used }
        paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
        paramBlock.csCode := 23;                  { for 'EnInter' call }
        paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        myStatus := myGpibCtlBlk.csStatus;        { interface's status }
        myError := myGpibCtlBlk.csError;          { driver's result code }

   { The success of the device manager call is returned in 'err'.  The driver's
   status and result codes are returned in myGpibCtlBlk.csStatus and
   myGpibCtlBlk.csError respectively.  The driver reference number used is that
   which was returned by the call to 'GpibOpen'.}
 END;
```

GpibClose Close Driver **GpibClose**

Purpose: This call is used to close the previously opened NBS-GPIB driver.

Format:

```
FUNCTION CloseDriver(refNum: INTEGER): OSErr;
```

Parameters: Input:

```
     refNum - the driver reference number returned from the 'GpibOpen' call.
```

Output:

```
     none
```

Details: Application programs should call this routine after all I/O is done. It is customary to do this at the end of the application program just before terminating. The driver should have been previously opened by a call to 'GpibOpen'.

Upon return from this function, the card will be left configured as a GPIB bus 'device'. The interrupt mask bits in the TMS9914A chip will be reset and the GPIB bus buffers will be set to 3-state outputs and 'non-system controller' operation.

```
GpibClose:
     Init card as a GPIB device          ; call NCInit
     Flag driver closed in local storage ;
```

Example:

```
VAR
     err:          OSErr;
     refNum:INTEGER;

BEGIN
     err := CloseDriver(refNum);

{ The success of the call is returned in 'err'.  The driver reference number
is that which was returned by the call to 'GpibOpen'.}
END;
```

GpibOpen Open Driver GpibOpen

Purpose: This call is used to open and initialize the NBS-GPIB driver.

Format: `FUNCTION OpenSlot(@paramBlock, FALSE): OSErr;`

Parameters: Parameters required by 'OpenSlot' function.

Details: Application programs must call this routine before making any calls to the other routines in the driver package. This is usually done once at the beginning of the application. The complimentary 'GpibClose' routine should be called after all I/O is done. Again, it is customary to do this at the end of the application program just before terminating.

It should be noted that upon return from this function, the board is configured as a GPIB bus 'device'. The application will need to call the 'ContInit' routine before making calls to any routines expecting the card to be configured as a 'controller'.

The call to the slot manager routine 'OpenSlot', documented in Inside Macintosh Volume V, is used to indirectly open the NBS-GPIB driver. See the example below for sample code.

```
GpibOpen:
     Save DCE pointer in local RAM        ;
     Set default 'EOS' byte               ; <LF> character
     Set default local GPIB address       ; '0'
     Flag driver 'open' in local storage  ;
     Set default timeout constant         ; $00002000
     Init card as a GPIB device           ; call NCInit
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     nameStr:       Str255;
     mySlot:        SignedByte;
     refNum:        INTEGER;

BEGIN
     mySlot := $B;                { slot number board is plugged into }
     paramBlock.ioCompletion := NIL;
     nameStr := .Fc_gpib;                { taken from driver header }
     paramBlock.ioNamePtr := @nameStr;
     paramBlock.ioPermssn := fsCurPerm;
     paramBlock.ioMix := NIL;
     paramBlock.ioFlags := 0;
     paramBlock.ioSlot := mySlot;
     paramBlock.ioId := -128;            { the GPIB driver ID }

     err := OpenSlot(@paramBlock, FALSE);
     refnum := paramBlock.ioRefNum;
```

```
{ The success of the call is returned in 'err'.  The driver reference number
is returned in 'paramBlock.ioRefNum and is used to reference the open driver
in all subsequent calls.}

END;
```

Ifc Interface Clear/Abort Ifc

Purpose: This call is used to pulse the interface clear line (IFC) on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '4' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to pulse the interface clear line (IFC) on the GPIB bus. This causes devices on the bus to go to a known state. The interface manual for the particular instrument should be consulted for specifics about it's response to this command. The driver routine will assert the IFC line for a minimum of 1 ms. This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
Ifc:
     Send 'sic' command to TMS9914 chip
     Delay 1ms
     Send 'siclr' command to TMS9914 chip
```

Example:

```
VAR
     err:                  OSErr;
     paramBlock:           ParamBlockRec;
     myGpibCtlBlk:         GpibCtlBlk;
     paramAddr:            LONGINT;
     refNum:               INTEGER;
     myStatus:             INTEGER;
     myError:              INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := NIL;           { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;           { not used }
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 4;                   { for 'Ifc' control call }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

KillIO Halt any I/O in process KillIO

Purpose: This call is used to terminate any I/O operation on the device driver.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '1' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: This call has limited use with the NBS-GPIB driver because it currently supports only synchronous calls from the device manager. It is included to provide conformance with the Mac's device manager control calls.

```
KillIO:
     Return noErr to caller
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     paramAddr:           LONGINT;
     refNum:              INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                    { not used }
     myGpibCtlBlk.csFlag := 0;                   { not used }
     myGpibCtlBlk.csStatus := 0;                 { not used }
     myGpibCtlBlk.csError := 0;                  { not used }
     myGpibCtlBlk.csCount := 0;                  { not used }
     myGpibCtlBlk.csDataBuf := NIL;              { not used }
     myGpibCtlBlk.csAddrList := NIL;             { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;             { not used }
     paramBlock.ioVRefNum := 0;                  { not used }
     paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
     paramBlock.csCode := 1;                     { for KillIO }
     paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);

{ The success of the device manager call is returned in 'err'.  The driver
reference number used is that which was returned by the call to 'GpibOpen'.}
```

```
END;
```

Local Local **Local**

Purpose: This call is used to de-assert the remote enable line (REN) on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '3' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to de-assert the remote enable line (REN) on the GPIB bus. Devices on the bus will go local immediately. This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
Local:
     Send 'sreclr' command to TMS9914 chip
```

Example:

```
VAR
     err:                  OSErr;
     paramBlock:           ParamBlockRec;
     myGpibCtlBlk:         GpibCtlBlk;
     paramAddr:            LONGINT;
     refNum:               INTEGER;
     myStatus:             INTEGER;
     myError:              INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                   { not used }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := NIL;             { not used }
     myGpibCtlBlk.csAddrList := NIL;            { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;            { not used }
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 3;                    { for 'Local' control call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }
```

```
{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

NContInit Initialize interface as GPIB Device NContInit

Purpose: This call is used to set up the NBS-GPIB card as a device on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '17' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs can call this function to re-initialize the card as a simple GPIB bus device. Normally this is not necessary, as the call to the 'GpibOpen' routine also calls this routine.

```
NContInit:
     Set flag as 'non-controller' in local storage
     Issue software reset to TMS9914 chip
     Disable all interrupt mask bits
     Write address from local storage to TMS9914 chip
     Set 3-state GPIB drivers
     Reset 'system controller' bit
     Clear software reset to TMS9914 chip
```

Example:

```
VAR
     err:                  OSErr;
     paramBlock:           ParamBlockRec;
     myGpibCtlBlk:         GpibCtlBlk;
     refNum:               INTEGER;
     paramAddr:            LONGINT;
     myStatus:             INTEGER;
     myError:              INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := NIL;           { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;           { not used }
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 17;                  { for 'NContInit' }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
```

```
          paramBlock.csParam[0] := HiWord(paramAddr);

          err := PBControl(@paramBlock, FALSE);
          myStatus := myGpibCtlBlk.csStatus;         { interface's status }
          myError := myGpibCtlBlk.csError;           { driver's result code }

     { The success of the device manager call is returned in 'err'.  The driver's
     status and result codes are returned in myGpibCtlBlk.csStatus and
     myGpibCtlBlk.csError respectively.  The driver reference number used is that
     which was returned by the call to 'GpibOpen'.}
END;
```

NewTimot Set new timeout value NewTimot

Purpose: This call is used to change the timeout constant used in the driver routines of the NBS-GPIB card.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
gpibCtlBlk.csCount    - new timeout value
paramBlock.ioRefNum   - value returned from 'GpibOpen' call
paramBlock.csCode     - '27' for this call
```

Output:

```
gpibCtlBlk.csStatus   - call return status information
gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to define a new timeout constant used by certain routines of the NBS-GPIB driver. The driver uses this value in order to determine how long to wait for operations over the GPIB bus to complete before assuming that the bus has 'hung' and some sort of error must be reported. Some of the functions that are timed are data byte transfers to/from other devices on the bus, or addressing operations performed by the interface when it is the controller in charge of the GPIB bus.

The following code excerpt is typical of the way the driver times a GPIB bus operation. The routine starts out with register D6 containing the timeout constant currently defined.

```
CRcv11      SUBI.L        #1,D6                 ; decrement pass count
            BEQ           CRcvTime              ; if bus not responding
            MOVE.B        (A4),D0        ; get interrupt 0 status
            ANDI.B        #eoimk+bim,D0         ; check for EOI or BI
            BEQ.S         CRcv11                ; wait until set
```

The default timeout constant defined when the driver opens is the value $2000.

The following routines use the timeout constant:

- Trig
- PpEnable
- PpUConfig
- CSerPoll
- CSend
- CXfer
- CRcvCntrl
- Send
- DevClr
- PpDisable
- CParPoll
- CRcv
- SendCmd
- CPassCntrl
- Rcv

```
         NewTimot:
              Save new timeout value in local storage
```

Example:

```
         VAR
              err:                  OSErr;
              paramBlock:           ParamBlockRec;
              myGpibCtlBlk:         GpibCtlBlk;
              paramAddr:            LONGINT;
              refNum:               INTEGER;
              myStatus:             INTEGER;
              myError:              INTEGER;
              newValue:             LONGINT;

         BEGIN
              newValue:= $00050000;               { new timeout constant }

              { first set up the driver's control call parameters }
              myGpibCtlBlk.csVar := 0;                   { not used }
              myGpibCtlBlk.csFlag := 0;                  { not used }
              myGpibCtlBlk.csStatus := 0;                { a return value }
              myGpibCtlBlk.csError := 0;                 { a return value }
              myGpibCtlBlk.csCount := newValue;          { new timeout constant }
              myGpibCtlBlk.csDataBuf := NIL;             { not used }
              myGpibCtlBlk.csAddrList := NIL;            { not used }

              { now set up the device manager's control call parameters }
              paramBlock.ioCompletion := NIL;            { not used }
              paramBlock.ioVRefNum := 0;                 { not used }
              paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
              paramBlock.csCode := 27;                   { for 'NewTimot' call }
              paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
              paramBlock.csParam[1] := LoWord(paramAddr);
              paramBlock.csParam[0] := HiWord(paramAddr);

              err := PBControl(@paramBlock, FALSE);
              myStatus := myGpibCtlBlk.csStatus;         { interface's status }
              myError := myGpibCtlBlk.csError;           { driver's result code }

         { The success of the device manager call is returned in 'err'.  The driver's
         status and result codes are returned in myGpibCtlBlk.csStatus and
         myGpibCtlBlk.csError respectively.  The driver reference number used is that
         which was returned by the call to 'GpibOpen'.}
END;
```

PpDisable Parallel Poll Disable PpDisable

Purpose: This call disables one or more devices on the GPIB bus from responding to a parallel poll operation.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csAddrList  - pointer to listener address list
     paramBlock.ioRefNum    - value returned from 'GpibOpen' call
     paramBlock.csCode      - '10' for this call
```

Output:

```
     gpibCtlBlk.csStatus    - call return status information
     gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine to disable one or more devices on the GPIB bus from responding to a parallel poll operation.

The calling program must pass a pointer to a list of listener addresses of devices it wishes to send the 'PPD' command to. This list should be composed of a string of bytes, each one corresponding to a valid GPIB listener address in the range of 0x20 to 0x3e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
PpDisable:
     Point to first listener address
     Send 'universal unlisten' over the GPIB bus
     While current address is in range 0x20 to 0x3e
          Send address of current listener over GPIB bus
          Increment address list pointer
     End While
```

```
Send 'PPC' over the GPIB bus
Send 'PPD' command over the GPIB bus
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;
     listeners:     Str255;              { the list of listeners }

BEGIN
     { first set up the listener address list.  This is a list of addresses
            of devices we wish to receive the 'PPD' command.  Remember that
            listener addresses are offset by + 0x20 in the IEEE-488 world.
            The list should be terminated by a non-valid listener address.
            In this case we use the ASCII <z> which meets the requirement
            by having a value of 0x7a.  Later we will pass a pointer to the
            first listener by using the @listeners[1] nomenclature.
            Remember that in the PASCAL language the first byte of a string
            is the string length parameter and that it is the second byte
            which is the real first character of the string.  }

     listeners := '(+7z';        { 3 listeners at addresses 8, 11, and 23 }

     { next, set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                { not used }
     myGpibCtlBlk.csFlag := 0;               { not used }
     myGpibCtlBlk.csStatus := 0;             { a return value }
     myGpibCtlBlk.csError := 0;              { a return value }
     myGpibCtlBlk.csCount := 0;              { not used }
     myGpibCtlBlk.csDataBuf := NIL;          { not used }
     myGpibCtlBlk.csAddrList := @listeners[1];  { pointer to listener list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;              { not used }
     paramBlock.ioRefNum := refNum;          { from 'GpibOpen' call }
     paramBlock.csCode := 10;                { for 'PpDisable' call }
     paramAddr := LONGINT(@myGpibCtlBlk);    { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;      { interface's status }
     myError := myGpibCtlBlk.csError;        { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

PpEnable Parallel Poll Enable PpEnable

Purpose: This call is used to configure one or more devices on the GPIB bus to respond to a parallel poll operation.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csAddrList  - pointer to listener address list
     gpibCtlBlk.csDataBuf   - pointer to configuration bytes
     paramBlock.ioRefNum    - value returned from 'GpibOpen' call
     paramBlock.csCode      - '9' for this call
```

Output:

```
     gpibCtlBlk.csStatus    - call return status information
     gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine to configure one or more devices on the GPIB bus to respond to a parallel poll operation. The device must implement the PP1 subset in order to respond to parallel polling.

Because parallel polling is usually used in a configuration where a response from more than one device is desired, this call allows the designation of multiple devices and a corresponding configuration byte for each to be specified. Pointers to two lists are passed as parameters to this call.

The first list is a list of listener addresses of devices the calling program expects to configure. This list should be composed of a string of bytes, each one corresponding to a valid GPIB listener address in the range of 0x20 to 0x3e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

The second list is a string of configuration bytes, one per device, in the same order as the devices appear in the listener list. The configuration bytes should have the format: `X X X X E B3 B2 B1`. The three least significant bits tell the device which bit of the parallel poll response byte it owns when responding to a parallel poll. The 'E' bit tells the device which state to put the owned bit in when it wants to signal that it needs attention. The other bits are unused.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
PpEnable:
    Point to first listener address
    Point to first configuration byte
    Loop:
          Send 'universal unlisten' over the GPIB bus
          If current address is in range 0x20 to 0x3e
                Send address of current listener over GPIB bus
                Increment address list pointer
                Send 'PPC' over the GPIB bus
                Get current listener's configuration byte
                Increment configuration byte list pointer
                OR the configuration byte with the 'PPE' command
                Send the 'PPE' byte over the GPIB bus
          End If
    End Loop (if no more valid listeners )
```

Example:

```
VAR
    err:          OSErr;
    paramBlock:   ParamBlockRec;
    myGpibCtlBlk: GpibCtlBlk;
    paramAddr:    LONGINT;
    refNum:       INTEGER;
    myStatus:     INTEGER;
    myError:      INTEGER;
    listeners:    Str255;             { the list of listeners }
    configs:      Str255;             { list of config bytes }

BEGIN
    { first set up the listener address list.  This is a list of addresses
           of devices we wish to receive the 'PPE' command.  Remember that
           listener addresses are offset by + 0x20 in the IEEE-488 world.
           The list should be terminated by a non-valid listener address.
           In this case we use the ASCII <z> which meets the requirement
           by having a value of 0x7a.  Later we will pass a pointer to the
           first listener by using the @listeners[1] nomenclature.
           Remember that in the PASCAL language the first byte of a string
           is the string length parameter and that it is the second byte
           which is the real first character of the string.  }

    listeners := '(+7z';         { 3 listeners at addresses 8, 11, and 23 }

    { next we need to set up the configuration bytes for the above three
           devices.  We will use the 'configs' string variable for this
           and send a pointer to the first valid byte in the string similar
           to the way we passed a pointer to the listener list. }

    configs := '123';                   { arbitrary 3 char long string }
    configs[1] := char(0);              { bit 0 for address 8 }
    configs[2] := char(1);              { bit 1 for address 11 }
    configs[3] := char(2);              { bit 2 for address 23 }
```

```
     { next, set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                   { not used }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := @configs[1];     { pointer to config bytes }
     myGpibCtlBlk.csAddrList := @listeners[1];  { pointer to listener list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 9;                    { for 'PpEnable' call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

PpUConfig Parallel Poll Unconfigure **PpUConfig**

Purpose: This call deconfigures the parallel poll response of all devices on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '11' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to issue the 'parallel poll unconfigure' command over the GPIB bus. This command instructs all devices to not respond to parallel poll operations. This is an unaddressed command and thus affects all devices on the bus. This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
PpUConfig:
     Send 'PPU' command over the GPIB bus
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                   { not used }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := NIL;             { not used }
     myGpibCtlBlk.csAddrList := NIL;            { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 11;                   { for 'PpUConfig' call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;          { interface's status }
     myError := myGpibCtlBlk.csError;            { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

Rcv Device Receive Data **Rcv**

Purpose: This call is used to allow the application to receive data, as a device on the GPIB bus, when it is addressed to listen.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csDataBuf   - pointer to receive data buffer
     gpibCtlBlk.csCount     - maximum number of bytes to receive
     gpibCtlBlk.csFlag      - 'look for EOS character' flag
     paramBlock.ioRefNum    - value returned from 'GpibOpen' call
     paramBlock.csCode      - '21' for this call
```

Output:

```
     gpibCtlBlk.csDataBuf   - receive data
     gpibCtlBlk.csCount     - actual number of bytes received
     gpibCtlBlk.csStatus    - call return status information
     gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine when they are configured as a device and have been addressed to listen in order to receive data over the GPIB bus.

It is the responsibility of the calling program to allocate the buffer space used to hold the receive characters. A pointer to the first byte of this buffer should be passed in the `gpibCtlBlk.csDataBuf` field. Enough space to hold `gpibCtlBlk.csCount` characters should be allocated for the buffer. The driver will terminate the data transfer when this maximum character count has been received if the transfer is not terminated earlier by some other condition. This is usually not the normal way a GPIB data transfer terminates however and will probably leave the talker in a strange state which may later have to be cleared by the controller. The `gpibCtlBlk.csCount` parameter is a longword variable which is used to return to the calling program the actual number of characters received during the current transaction.

During normal data transfers, the receive operation is terminated when the EOI line is driven by the talker during a data byte transfer. It is also possible to terminate on the occurrence of a particular byte in the data stream. This byte is called the EOS character and is specified to the driver by a call to the 'SetEos' control call. The calling program must set the `gpibCtlBlk.csFlag` parameter to a non-zero value in order to enable termination on the EOS character, otherwise no data checking will occur.

During the execution of this call, the driver will attempt to detect if there is no-response from the talker on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to receive a data byte over the bus it will enter a loop which verifies whether or not the data byte has been sent over the GPIB bus. If the byte is not sent after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is currently configured as a 'device' on the GPIB bus.

```
Rcv:
     Set pointer to input buffer
     Send 'HDFA' command to TMS9914 chip
     While still receiving data
           Wait for data byte from GPIB bus
           If byte has EOI with it
                 Get data byte from GPIB bus
                 Store byte in buffer
                 Increment character count
                 Flag EOI received in .csStatus field
                 Send 'RDHF' command to TMS9914 chip
                 Send 'HDACLR' command to TMS9914 chip
                 Put character count in .csCount field
                 Return to caller
           End if byte had EOI with it
           Else if byte did not have EOI with it
                 Get data byte from GPIB bus
                 Store data byte in buffer
                 Increment data buffer pointer
                 Increment character count
                 If we should be checking for EOS character
                       If this byte was the EOS character
                             Flag EOS received in .csStatus
                             Send 'RDHF' command to TMS9914
                             Send 'HDACLR' cmd to TMS9914
                             Put char count in .csCount
                             Return to caller
                       End if this was the EOS character
                 End if we are checking for EOS character
                 If max buffer size reached
                       Flag buffer size hit in .csStatus
                       Send 'RDHF' command to TMS9914 chip
                       Send 'HDACLR' command to TMS9914
                       Put character count in .csCount
                       Return to caller
                 End if max buffer size reached
                 Send 'RDHF' command to TMS9914 chip
           End if byte did not have EOI with it
     End While still receiving data
```

Example:

VAR

```
       err:          OSErr;
       paramBlock:   ParamBlockRec;
       myGpibCtlBlk: GpibCtlBlk;
       paramAddr:    LONGINT;
       refNum:       INTEGER;
       myStatus:     INTEGER;
       myError:      INTEGER;
       byteCnt:      LONGINT;             { data count }
       dataBuffer:   Handle;       { Rx data goes here }
       myByte:       SignedByte;
       isThere:      BOOLEAN;

  BEGIN
       { read the address status register from the TMS9914A chip.  The address
       of the register is at $FS020008.  The address passed to the read
routine specifies the offset from the base address of the board. }

       err := GpibRdAddr(gRefNum, $020008, myByte, myStatus, myError);

       { check if the 'LADS' bit is set.  This will be TRUE when we have been
addressed to listen.  We assume we can only listen. }

       IF (BitAnd(LONGINT(myByte), 4) <> 0) THEN
              isThere := TRUE             { we were addressed to listen }
       ELSE
              isThere := FALSE;

       IF isThere THEN
              BEGIN
              { allocate space for the receive data buffer. }
              byteCnt := 1000;                   { max chars to receive }
              dataBuffer := NewHandle(byteCnt);   { reserve memory for data }

              IF (dataBuffer <> NIL) THEN        { if we have enough memory }
                     BEGIN
                     HLock(dataBuffer);          { lock the memory block
                                                   during I/O operation }

                     { next set up the driver's control call parameters }
                     myGpibCtlBlk.csVar := 0;    { not used }
                     myGpibCtlBlk.csFlag := 1;   { check for EOS character }
                     myGpibCtlBlk.csStatus := 0; { a return value }
                     myGpibCtlBlk.csError := 0;  { a return value }
                     myGpibCtlBlk.csCount := byteCnt ;   { max buffer size }
                     myGpibCtlBlk.csDataBuf := dataBuffer^; { input buffer }
                     myGpibCtlBlk.csAddrList := NIL;     { not used }

                     { set up the device manager's control call parameters }
                     paramBlock.ioCompletion := NIL;
                     paramBlock.ioVRefNum := 0;  { not used }
                     paramBlock.ioRefNum := refNum; { from 'GpibOpen' call }
                     paramBlock.csCode := 21;    { for 'Rcv' call }
                     paramAddr := LONGINT(@myGpibCtlBlk);{ GPIB params }
                     paramBlock.csParam[1] := LoWord(paramAddr);
                     paramBlock.csParam[0] := HiWord(paramAddr);

                     err := PBControl(@paramBlock, FALSE);
                     myStatus := myGpibCtlBlk.csStatus;  { status }
                     myError := myGpibCtlBlk.csError;    { result code }

                     HUnlock(dataBuffer);        { no more need to lock }

                     END;   { if databuffer allocated }
              END;   { if we were addressed to listen }
  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in myGpibCtlBlk.csStatus and
  myGpibCtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'GpibOpen'.  The data will be placed in
  dataBuffer and 'myGpibCtlBlk.csCount' will contain the actual number of
  characters transferred into the dataBuffer. }

END;
```

Read Read Memory Location **Read**

Purpose: This call is used to allow the application to read a memory address from the NBS-GPIB card.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csAddrList - desired memory address.
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '25' for this call
```

Output:

```
     gpibCtlBlk.csVar      - Byte at specified memory address.
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Applications call this routine in order to read a byte from the memory space of the NBS-GPIB card.

This routine has been included in the driver to allow the application programmer complete access to all of the hardware functions with which the interface card is capable of. With this call an application can, for instance, read any of the status registers on the TMS9914A chip.

Addresses should be specified by sending the lower 24 bits of the address desired on the card, with the two LSB's zero. The driver will complete the address used for the access by adding `$FS000003` to the value passed to the routine ('S' being the slot address where the card is installed). Remember that the NBS-GPIB card only supports data transfers over byte lane 3 of the NuBus interface.

The user should consult the NBS-GPIB memory map given in another part of this manual for a list of addresses used on the card.

```
Read:
     Get specified address.
     AND address with $00FFFFFF.
     ADD address with the board's base address ($FS000003).
     Read the byte at the calculated address.
     AND byte with $000000FF.
     Put requested byte in the low byte of the .csVar field.
     Return.
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     paramAddr:           LONGINT;
```

```
       refNum:               INTEGER;
       myStatus:             INTEGER;
       myError:              INTEGER;
       theByte:              SignedByte;

  BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                  { a return value }
       myGpibCtlBlk.csFlag := 0;                 { not used }
       myGpibCtlBlk.csStatus := 0;               { a return value }
       myGpibCtlBlk.csError := 0;                { a return value }
       myGpibCtlBlk.csCount := 0;                { not used }
       myGpibCtlBlk.csDataBuf := NIL;            { not used }
       myGpibCtlBlk.csAddrList := $020008;       { address of TMS9914A's
                                                   address status register }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;           { not used }
       paramBlock.ioVRefNum := 0;                { not used }
       paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
       paramBlock.csCode := 25;                  { for 'Read' call }
       paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       myStatus := myGpibCtlBlk.csStatus;        { interface's status }
       myError := myGpibCtlBlk.csError;          { driver's result code }
       theByte := SignedByte(myGpibCtlBlk.csVar); { the returned byte }

  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in myGpibCtlBlk.csStatus and
  myGpibCtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'GpibOpen'.}
END;
```

RemEnable Remote enable **RemEnable**

Purpose: This call is used to assert the remote enable line (REN) on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '2' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to assert the remote enable line (REN) on the GPIB bus. Devices on the bus will not go into remote until they are later addressed to listen. This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
RemEnable:
     Send 'sre' command to TMS9914 chip
```

Example:

```
VAR
     err:                  OSErr;
     paramBlock:           ParamBlockRec;
     myGpibCtlBlk:         GpibCtlBlk;
     paramAddr:            LONGINT;
     refNum:               INTEGER;
     myStatus:             INTEGER;
     myError:              INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := NIL;           { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;           { not used }
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 2;                   { for RemEnable }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;        { interface's status }
     myError := myGpibCtlBlk.csError;          { driver's result code }
```

```
{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

Send Device Send Data Send

Purpose: This call is used to allow the application to send data, as a device on the GPIB bus, when it is addressed to talk.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csDataBuf    - pointer to data buffer
     gpibCtlBlk.csCount      - number of bytes to send
     gpibCtlBlk.csVar        - 'look for EOS character' flag
     gpibCtlBlk.csFlag       - 'send EOI with last character' flag
     paramBlock.ioRefNum     - value returned from 'GpibOpen' call
     paramBlock.csCode       - '22' for this call
```

Output:

```
     gpibCtlBlk.csCount      - actual number of bytes sent
     gpibCtlBlk.csStatus     - call return status information
     gpibCtlBlk.csError      - call return error code
```

Details: Application programs call this routine when they are configured as a device and have been addressed to talk in order to send data over the GPIB bus.

A pointer to the first byte of the data buffer should be passed in the `gpibCtlBlk.csDataBuf` field. The calling program shall use the `gpibCtlBlk.csCount` field to specify the number of characters to transmit. The driver will terminate the data transfer when this character count has been sent if the transfer is not terminated earlier by some other condition. The `gpibCtlBlk.csCount` parameter is a longword variable which is used to return to the calling program the actual number of characters transmitted during the current transaction.

During normal data transfers, the transmit operation is terminated when the specified number of characters have been sent. It is also possible to terminate on the occurrence of a particular byte in the data stream. This byte is called the EOS character and is specified to the driver by a call to the 'SetEos' control call. The calling program must set the `gpibCtlBlk.csVar` parameter to a non-zero value in order to enable termination on the EOS character, otherwise no data checking will occur.

The last character sent will be sent with the EOI line on the GPIB bus pulled low if the `gpibCtlBlk.csFlag` variable is set to a non-zero value.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is currently configured as a 'device' on the GPIB bus.

```
Send:
    Set pointer to data buffer
    Clear character counter
    While still sending data
          Get data byte
          Increment data byte pointer
          If last byte to send
                Signal count hit in .csStatus
                IF .csFlag set then
                      Send 'FEOI' command to TMS9914 chip
                      signal 'EOI' sent in .csStatus
                End if .csFlag set
                Send data byte over GPIB bus
                Wait for data byte to be accepted over GPIB bus
                Increment character counter
                Put character count in .csCount field
                Return to caller
          End if last byte to send
          If we should be checking for EOS character
                If this byte was the EOS character
                      signal 'EOI' sent in .csStatus
                      IF .csFlag set then
                            Send 'FEOI' command to TMS9914 chip
                            signal 'EOI' sent in .csStatus
                      End if .csFlag set
                      Send data byte over GPIB bus
                      Wait for byte to be accepted over GPIB bus
                      Increment character counter
                      Put character count in .csCount field
                      Return to caller
                End if this was the EOS character
          End if we are checking for EOS character
          Send data byte over GPIB bus
          Increment character counter
          Wait for data byte to be accepted over GPIB bus
    End While still sending data
```

Example:

```
VAR
    err:          OSErr;
    paramBlock:   ParamBlockRec;
    myGpibCtlBlk: GpibCtlBlk;
```

```
       paramAddr:     LONGINT;
       refNum:        INTEGER;
       myStatus:      INTEGER;
       myError:       INTEGER;
       listeners:     Str255;        { listener address list }
       byteCnt:       LONGINT;       { data count }
       sendData:      Str255;        { Tx data goes here }
       myByte:        SignedByte;
       isThere:       BOOLEAN;

  BEGIN
       { read the address status register from the TMS9914A chip.  The address
       of the register is at $FS020008.  The address passed to the read
routine specifies the offset from the base address of the board. }

       err := GpibRdAddr(gRefNum, $020008, myByte, myStatus, myError);

       { check if the 'TADS' bit is set.  This will be TRUE when we have been
addressed to talk.  We assume we can only talk. }

       IF (BitAnd(LONGINT(myByte), 2) <> 0) THEN
              isThere := TRUE             { we were addressed to listen }
       ELSE
              isThere := FALSE;

       IF isThere THEN
              BEGIN
              sendData := 'hello worldxx'; { actual data to send }
              sendData[12] := CHAR(13);           { <CR> }
              sendData[13] := CHAR(10);           { <LF> }
              byteCnt := LONGINT(Length(sendData));

              { next set up the driver's control call parameters }
              myGpibCtlBlk.csVar := 0;     { don't check for EOS character }
              myGpibCtlBlk.csFlag := 1;    { send EOI with last character }
              myGpibCtlBlk.csStatus := 0;  { a return value }
              myGpibCtlBlk.csError := 0;   { a return value }
              myGpibCtlBlk.csCount := byteCnt ;   { max buffer size }
              myGpibCtlBlk.csDataBuf := @sendData[1]; { the first data byte }
              myGpibCtlBlk.csAddrList := NIL;     { not used }

              { now set up the device manager's control call parameters }
              paramBlock.ioCompletion := NIL;
              paramBlock.ioVRefNum := 0;          { not used }
              paramBlock.ioRefNum := refNum;      { from 'GpibOpen' call }
              paramBlock.csCode := 22;            { for 'Send' call }
              paramAddr := LONGINT(@myGpibCtlBlk);{ address of GPIB params }
              paramBlock.csParam[1] := LoWord(paramAddr);
              paramBlock.csParam[0] := HiWord(paramAddr);

              err := PBControl(@paramBlock, FALSE);
              myStatus := myGpibCtlBlk.csStatus;  { interface's status }
              myError := myGpibCtlBlk.csError;    { driver's result code }
              END;          { if we were addressed to talk }


  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in myGpibCtlBlk.csStatus and
  myGpibCtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'GpibOpen'.  The 'myGpibCtlBlk.csCount'
  field will contain the actual number of characters transferred over the GPIB
  bus. }

END;
```

SendCmd Controller Send Command String **SendCmd**

Purpose: This call is used to allow the controller to send a command string directly over the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
gpibCtlBlk.csDataBuf    - pointer to data buffer
gpibCtlBlk.csCount      - number of bytes to send
paramBlock.ioRefNum     - value returned from 'GpibOpen' call
paramBlock.csCode       - '16' for this call
```

Output:

```
gpibCtlBlk.csCount      - actual number of bytes sent
gpibCtlBlk.csStatus     - call return status information
gpibCtlBlk.csError      - call return error code
```

Details: Application programs call this routine to instruct the controller to send a command string directly over the GPIB bus. The data bytes in the specified buffer will be sent over the GPIB bus with the ATN line asserted. No other addressing or set-up bytes are sent over the GPIB bus prior to the data transfer. The EOI line is *not* asserted with the last data byte sent.

A pointer to the first byte of the data buffer should be passed in the `gpibCtlBlk.csDataBuf` field. The calling program shall use the `gpibCtlBlk.csCount` field to specify the number of characters to transmit. The driver will terminate the data transfer when this character count has been sent. The `gpibCtlBlk.csCount` parameter is a longword variable which is used to return to the calling program the actual number of characters transmitted during the current transaction.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus. Use this call with caution for invalid command bytes can leave the bus in an undefined state.

```
SendCmd:
    Set pointer to data buffer
    Clear character counter
    While still sending data
          Get data byte
          Increment data byte pointer
          If last byte to send
                Send data byte over GPIB bus
                Increment character counter
                Put character count in .csCount field
                Signal count hit in .csStatus
                Return to caller
          End if last byte to send
          Send data byte over GPIB bus
          Increment character counter
          Wait for data byte to be accepted over GPIB bus
    End While still sending data
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     byteCnt:      LONGINT;             { data count }
     sendData:     Str255;       { Tx data goes here }

BEGIN
     { first set up the command byte buffer.  In the following example we
     send the 'unlisten' command followed by the listener address 23
            and the talker address 8. }

     sendData := '?7H';                 { actual command bytes to send }

     byteCnt := LONGINT(Length(sendData));

     { next set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;           { not used }
     myGpibCtlBlk.csFlag := 0;          { not used }
     myGpibCtlBlk.csStatus := 0;        { a return value }
     myGpibCtlBlk.csError := 0;         { a return value }
     myGpibCtlBlk.csCount := byteCnt ;  { max buffer size }
     myGpibCtlBlk.csDataBuf := @sendData[1]; { the first data byte }
     myGpibCtlBlk.csAddrList := NIL;    { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;         { not used }
     paramBlock.ioRefNum := refNum;     { from 'GpibOpen' call }
     paramBlock.csCode := 16;           { for 'SendCmd' call }
     paramAddr := LONGINT(@myGpibCtlBlk);{ address of GPIB params }
```

```
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        myStatus := myGpibCtlBlk.csStatus;  { interface's status }
        myError := myGpibCtlBlk.csError;    { driver's result code }


  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in myGpibCtlBlk.csStatus and
  myGpibCtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'GpibOpen'.  The 'myGpibCtlBlk.csCount'
  field will contain the actual number of characters transferred over the GPIB
  bus. }
END;
```

SetEos Define new EOS character SetEos

Purpose: This call is used to define a new 'End-of-String' character (EOS) for the interface.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csVar      - new EOS character in low byte of word
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '5' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to define a new EOS character for the interface. Upon driver initialization, by a call to GpibOpen, the firmware sets a default EOS character of an ASCII line-feed. This call can subsequently be used to change the character recognized as the EOS character.

The EOS character is sometimes used to terminate I/O operations across the GPIB bus. The driver's send and receive data calls allow the calling program to specify whether or not the EOS character will be used as a message terminator for the selected operation. If so then the data transfer operation will terminate whenever the currently defined EOS character is detected in the data stream. This will happen regardless of the value of the csCount parameter or the state of the GPIB EOI line at the time of the occurrence. See the particular data transfer routine for more information regarding data transfer termination.

```
SetEos:
     Save new EOS character in local storage
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     paramAddr:           LONGINT;
     refNum:              INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;
     newChar:             Char;

BEGIN
     newChar := Char(4);                           { new EOS of ASCII <EOT>}

     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := INTEGER(newChar);    { pass new character value }
     myGpibCtlBlk.csFlag := 0;                  { not used }
     myGpibCtlBlk.csStatus := 0;                { a return value }
     myGpibCtlBlk.csError := 0;                 { a return value }
     myGpibCtlBlk.csCount := 0;                 { not used }
     myGpibCtlBlk.csDataBuf := NIL;             { not used }
     myGpibCtlBlk.csAddrList := NIL;            { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;            { not used }
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 5;                    { for 'SetEos' call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

SetMyAddr Designate new local GPIB address SetMyAddr

Purpose: This call is used to designate a new GPIB address for the local device.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csVar      - new local address in low byte of word
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '6' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Application programs call this routine to specify a new local GPIB address for the interface. This call can be made for both controller and non-controller configured interfaces. The address of each device on the GPIB bus must be unique and usually does not change during a sequence of commands. The address is used to determine which devices participate in any subsequent data transfer operations.

Valid address values to be used in this control call will fall in the range of 0 to 31. The appropriate driver routines will take care of translating this value to its corresponding talker and listener addresses which are actually sent over the GPIB bus during an addressing sequence of the operation. When the NBS-GPIB driver GpibOpen routine is called, the driver defaults to a local address of '0'.

```
SetMyAddr:
     Save new address value in local storage
     Write new address to TMS9914 chip
```

Example:

```
VAR
     err:           OSErr;
     paramBlock:    ParamBlockRec;
     myGpibCtlBlk:  GpibCtlBlk;
     paramAddr:     LONGINT;
     refNum:        INTEGER;
     myStatus:      INTEGER;
     myError:       INTEGER;
     myAddress:     INTEGER;

BEGIN
     myAddress := 5;                             { set new address }

     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := myAddress ;           { pass new address value }
     myGpibCtlBlk.csFlag := 0;                   { not used }
     myGpibCtlBlk.csStatus := 0;                 { a return value }
     myGpibCtlBlk.csError := 0;                  { a return value }
     myGpibCtlBlk.csCount := 0;                  { not used }
     myGpibCtlBlk.csDataBuf := NIL;             { not used }
     myGpibCtlBlk.csAddrList := NIL;            { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
     paramBlock.csCode := 6;                    { for 'SetMyAddr' call }
     paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;         { interface's status }
     myError := myGpibCtlBlk.csError;           { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

SetOut Set GPIB bus' Output Buffer Configuration **SetOut**

Purpose: This call is used to set the type of output buffers used on the GPIB bus data lines.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
    gpibCtlBlk.csVar      - Buffer output type.
    paramBlock.ioRefNum   - value returned from 'GpibOpen' call
    paramBlock.csCode     - '24' for this call
```

Output:

```
    gpibCtlBlk.csStatus   - call return status information
    gpibCtlBlk.csError    - call return error code
```

Details: Applications call this routine in order to set the type of output buffers used on the GPIB bus data lines.

The NBS-GPIB card can be configured to have either three-state or open collector drivers on the GPIB data bus lines. Three-state type of buffers allow faster data transfers over the interface, but have the disadvantage of not being compatible with parallel-poll operations. During parallel-poll operations, each configured device on the bus must drive one bit of the eight bit data bus. Thus open collector drivers must be employed.

The design of the NBS-GPIB card also allows a hybrid mode of operation which gives the interface the best of both types of buffer outputs. This third mode sets the output buffers to their three-state output type except during parallel-poll operations, during which time the buffers automatically switch to the open-collector type of driver. After the parallel-poll operation completes, the buffers revert back to three-state operation. This is the mode which the driver defaults to upon a call to the 'GpibOpen' routine.

The application specifies the configuration type to this function by passing one of the following values in the .csVar parameter.

.csVar value	Output Type
0	Always open-collector outputs
1	Always three-state outputs
2	Three-state except during parallel-poll

```
SetOut:
    Get current configuration from local memory image
    Mask off 'system bit'
    OR new configuration into value
```

```
Store in configuration register
Store a memory image in local storage
Return
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     paramAddr:           LONGINT;
     refNum:              INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 1;                  { Set 3-state outputs }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := NIL;           { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;           { not used }
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 24;                  { for 'SetOut' call }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;        { interface's status }
     myError := myGpibCtlBlk.csError;          { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

Trig Group Execute Trigger **Trig**

Purpose: This call is used to cause a group execute trigger (GET) command to be sent to designated devices on the GPIB bus.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
     gpibCtlBlk.csAddrList  - pointer to listener address list
     paramBlock.ioRefNum    - value returned from 'GpibOpen' call
     paramBlock.csCode      - '7' for this call
```

Output:

```
     gpibCtlBlk.csStatus    - call return status information
     gpibCtlBlk.csError     - call return error code
```

Details: Application programs call this routine to cause a group execute trigger (GET) command to be sent to designated devices on the GPIB bus. This is usually done to synchronize a number of instruments.

The calling program must pass a pointer to a list of listener addresses of devices it wishes to send the trigger command to. This list should be composed of a string of bytes, each one corresponding to a valid GPIB listener address in the range of 0x20 to 0x3e. The list end will be presumed by the driver to be the first byte not in the above mentioned range.

During the execution of this call, the driver will attempt to detect if there is no-response from the addressed device on the bus. This can happen if the device's address is not properly set or the instrument is malfunctioning in some way. It does this by using the value of the 'timot' parameter described in another portion of this manual. Whenever the driver is attempting to send a data byte over the bus it will enter a loop which verifies whether or not the data byte has been accepted over the GPIB bus. If the byte is not accepted after 'timot' number of passes thru the check loop, the operation will terminate and the driver will return the 'ctlTime' error result to the calling program. This should prevent most programs from 'hanging' if there is some failure on the GPIB bus.

This call should only be made if the NBS-GPIB card is the controller in charge of the GPIB bus.

```
Trig:
     Send 'universal unlisten' over the GPIB bus
     Point to first listener address
     While current address is in range 0x20 to 0x3e
            Send address of current listener over GPIB bus
            Increment address list pointer
     End While
     Send 'GET' command over the GPIB bus
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     myGpibCtlBlk: GpibCtlBlk;
     paramAddr:    LONGINT;
     refNum:       INTEGER;
     myStatus:     INTEGER;
     myError:      INTEGER;
     listeners:    Str255;              { the list of listeners }

BEGIN
     { first set up the listener address list.  This is a list of addresses
            of devices we wish to receive the 'GET' command.  Remember that
            listener addresses are offset by + 0x20 in the IEEE-488 world.
            The list should be terminated by a non-valid listener address.
            In this case we use the ASCII <z> which meets the requirement
            by having a value of 0x7a.  Later we will pass a pointer to the
            first listener by using the @listeners[1] nomenclature.
            Remember that in the PASCAL language the first byte of a string
            is the string length parameter and that it is the second byte
            which is the real first character of the string.  }

     listeners := '(+7z';        { 3 listeners at addresses 8, 11, and 23 }

     { next, set up the driver's control call parameters }
     myGpibCtlBlk.csVar := 0;                  { not used }
     myGpibCtlBlk.csFlag := 0;                 { not used }
     myGpibCtlBlk.csStatus := 0;               { a return value }
     myGpibCtlBlk.csError := 0;                { a return value }
     myGpibCtlBlk.csCount := 0;                { not used }
     myGpibCtlBlk.csDataBuf := NIL;            { not used }
     myGpibCtlBlk.csAddrList := @listeners[1];  { pointer to listener list }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
     paramBlock.csCode := 7;                   { for 'Trig' control call }
     paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := myGpibCtlBlk.csStatus;        { interface's status }
     myError := myGpibCtlBlk.csError;          { driver's result code }

{ The success of the device manager call is returned in 'err'.  The driver's
status and result codes are returned in myGpibCtlBlk.csStatus and
myGpibCtlBlk.csError respectively.  The driver reference number used is that
which was returned by the call to 'GpibOpen'.}
END;
```

Write Write Memory Location **Write**

Purpose: This call is used to allow the application to write a byte to a memory address on the NBS-GPIB card.

Format:

```
FUNCTION PBControl(@paramBlock, FALSE): OSErr;
```

Parameters: Input:

```
     gpibCtlBlk.csVar      - byte to write in lower 8 bits of .csVar
     gpibCtlBlk.csAddrList - desired memory address.
     paramBlock.ioRefNum   - value returned from 'GpibOpen' call
     paramBlock.csCode     - '26' for this call
```

Output:

```
     gpibCtlBlk.csStatus   - call return status information
     gpibCtlBlk.csError    - call return error code
```

Details: Applications call this routine in order to write a byte to the memory space of the NBS-GPIB card.

This routine has been included in the driver to allow the application programmer complete access to all of the hardware functions with which the interface card is capable of. With this call an application can, for instance, write to any of the control registers on the TMS9914A chip.

Addresses should be specified by sending the lower 24 bits of the address desired on the card, with the two LSB's zero. The driver will complete the address used for the access by adding $FS000003 to the value passed to the routine ('S' being the slot address where the card is installed). Remember that the NBS-GPIB card only supports data transfers over byte lane 3 of the NuBus interface.

The user should consult the NBS-GPIB memory map given in another part of this manual for a list of addresses used on the card.

```
Write:
     Get specified address.
     AND address with $00FFFFFF.
     ADD address with the board's base address ($FS000003).
     Write the byte at the calculated address.
     Return.
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     myGpibCtlBlk:        GpibCtlBlk;
     paramAddr:           LONGINT;
     refNum:              INTEGER;
     myStatus:            INTEGER;
```

```
      myError:              INTEGER;
      theByte:              SignedByte;

  BEGIN
      { in this example we will enable interrupts on 'BI' from the gpib bus
      from the TMS9914A chip.  We do this by setting bit 5 of the interrupt
      mask '0' register on the chip. }

      theByte := SignedByte($20);              { set bit 5 }

      { next set up the driver's control call parameters }
      myGpibCtlBlk.csVar := theByte;           { value to be written }
      myGpibCtlBlk.csFlag := 0;                { not used }
      myGpibCtlBlk.csStatus := 0;              { a return value }
      myGpibCtlBlk.csError := 0;               { a return value }
      myGpibCtlBlk.csCount := 0;               { not used }
      myGpibCtlBlk.csDataBuf := NIL;           { not used }
      myGpibCtlBlk.csAddrList := $020000;      { address of TMS9914A's
                                                 'int mask 0' register }

      { now set up the device manager's control call parameters }
      paramBlock.ioCompletion := NIL;          { not used }
      paramBlock.ioVRefNum := 0;               { not used }
      paramBlock.ioRefNum := refNum;           { from 'GpibOpen' call }
      paramBlock.csCode := 26;                 { for 'Write' call }
      paramAddr := LONGINT(@myGpibCtlBlk);     { address of GPIB params }
      paramBlock.csParam[1] := LoWord(paramAddr);
      paramBlock.csParam[0] := HiWord(paramAddr);

      err := PBControl(@paramBlock, FALSE);
      myStatus := myGpibCtlBlk.csStatus;       { interface's status }
      myError := myGpibCtlBlk.csError;         { driver's result code }

  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in myGpibCtlBlk.csStatus and
  myGpibCtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'GpibOpen'.}
END;
```

```
{
  File: GpibGlu.p

       Version 1.0   15 March, 1989

Copyright © 1988-1989 by fishcamp engineering.  All rights reserved.
}


UNIT GpibGlu;


INTERFACE

USES
       {$LOAD MacIntf.LOAD}
              MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf;
       {$LOAD}


{
CONST
}


TYPE



{
       the following structure will be used for all driver 'Control calls'
       for passing information into/from the driver.
}




       GpibCtlBlk =  RECORD
                     csVar:        INTEGER;      { general purpose word has call specific
                                                 data.  Refer to control call desired
                                                 for variable definition. }
                     csFlag:       INTEGER;      { general purpose word has call specific
                                                 data.  Refer to control call desired
                                                 for variable definition. }
                     csStatus:     INTEGER;      { call returned status information }
                     csError:      INTEGER;      { call returned error information }
                     csCount:      LONGINT;      { max characters to be inputted from the
                                                 bus or the exact number of bytes to be sent
                                                 out over the bus. For all operations,
                                                 the actual number of bytes received or
                                                 transmitted will be returned in this value}
                     csDataBuf:    Ptr;          { used for actual data to/from the driver }
                     csAddrList:   Ptr;          { pointer to a list of valid GPIB addresses
                                                 of devices which will be partaking in the
                                                 following transaction.  List will contain
                                                 valid addresses terminated by the first
                                                 non-valid address for Listeners.  For
                                                 talkers there can only be one so only
                                                 the byte pointed to is valid and no
                                                 terminator is needed.  Not used for 'Send
                                                 command'. }
```

```
                                    END;

       GpibCtlBlkPtr = ^GpibCtlBlk;






FUNCTION GpibOpen(gpibSlot: SignedByte; VAR refNum: INTEGER): OSErr;


       {      initialize the card as system controller }
FUNCTION GpibController(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       {      initialize the card as a Device }
FUNCTION GpibDevice(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       {      send remote enable  }
FUNCTION GpibRemote(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       {      send 'local'  }
FUNCTION GpibLocal(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       {      send 'interface clear'  }
FUNCTION GpibIfc(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       {      mark a new EOS character  }
FUNCTION GpibNewEos(refNum: INTEGER; theChar: CHAR; VAR status, error: INTEGER): OSErr;


       {      Set new GPIB address  }
FUNCTION GpibNewAddr(refNum, theAddr: INTEGER; VAR status, error: INTEGER): OSErr;


       { send 'group execute trigger' to listener list }
FUNCTION GpibGet(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;


       { send 'selected device clear' to listener list }
FUNCTION GpibSdc(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;


       { receive data (as controller) from a selected talker }
FUNCTION GpibCRcv(refNum: INTEGER; VAR count: LONGINT; talker: CHAR; bufferPtr: Ptr;
                                    eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;


       { send data (as controller) to selected listener(s) }
FUNCTION GpibCSend(refNum: INTEGER; VAR count: LONGINT; listeners: Str255; bufferPtr: Ptr;
                             sendEoi, eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
       { send command string (as controller) }
FUNCTION GpibSendCmd(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                                                          VAR status, error: INTEGER): OSErr;


       {
              Configure the listener list with the specified configuration bytes
              in order that the devices may respond to a parallel poll operation.
       }
FUNCTION GpibPpEn(refNum: INTEGER; listeners: Str255; configStr: Str255;
                                                          VAR status, error: INTEGER): OSErr;


       { Disable one or more specified devices from responding to a parallel poll. }
FUNCTION GpibPpDis(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;


       { Disables parallel poll operation on all devices on the GPIB bus. }
FUNCTION GpibPpUConfig(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       { Perform a parallel poll operation on the GPIB bus. }
FUNCTION GpibCParPoll(refNum: INTEGER; VAR response: SignedByte;
                                                          VAR status, error: INTEGER): OSErr;


       { serial poll the devices specified }
FUNCTION GpibCSerPoll(refNum: INTEGER; talkers: Str255; bufferPtr: Ptr;
                                                          VAR status, error: INTEGER): OSErr;


       { enable/disable board interrupts }
FUNCTION GpibIntEn(refNum: INTEGER; operation: BOOLEAN; VAR status, error: INTEGER): OSErr;


       { write a byte to an address on the card }
FUNCTION GpibWrAddr(refNum: INTEGER; address: UNIV Ptr; theByte: SignedByte;
                                                          VAR status, error: INTEGER): OSErr;


       { read a byte from an address on the card }
FUNCTION GpibRdAddr(refNum: INTEGER; address: UNIV Ptr; VAR theByte: SignedByte;
                                                          VAR status, error: INTEGER): OSErr;


       { set the output buffer type }
FUNCTION GpibSetOut(refNum: INTEGER; theConfig: INTEGER; VAR status, error: INTEGER): OSErr;


       { Transfer data from a talker to a listener on the bus where the controller does not
              participate in the transaction }
FUNCTION GpibXfr(refNum: INTEGER; addresses: Str255; VAR status, error: INTEGER): OSErr;


       { receive data as a device }
FUNCTION GpibRcv(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                                    eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
       { send data as a device }
FUNCTION GpibSend(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                             sendEoi, eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;


       { receive controll from the currently active controller }
FUNCTION GpibRcvCntrl(refNum: INTEGER; VAR status, error: INTEGER): OSErr;


       { pass controll to a device on the bus }
FUNCTION GpibPassCntrl(refNum: INTEGER; device: CHAR; VAR status, error: INTEGER): OSErr;


       { set new timeout constant }
FUNCTION GpibNewTimout(refNum: INTEGER; value: LONGINT; VAR status, error: INTEGER): OSErr;


FUNCTION GpibClose(refNum: INTEGER): OSErr;




IMPLEMENTATION




FUNCTION GpibOpen(gpibSlot: SignedByte; VAR refNum: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       nameStr:             Str255;

BEGIN
       paramBlock.ioCompletion := NIL;
       nameStr := '.Fc_gpib';             { taken from driver header (not needed ???) }
       paramBlock.ioNamePtr := @nameStr;
       paramBlock.ioPermssn := fsCurPerm;  { any available permission }
       paramBlock.ioMix := NIL;
       paramBlock.ioFlags := 0;
       paramBlock.ioSlot := gpibSlot;      { the slot the user plugged into }
       paramBlock.ioId := -128;            { the GPIB driver id }

       err := OpenSlot(@paramBlock, FALSE);
       refNum := paramBlock.ioRefNum;      { return the driver reference number }
       GpibOpen := err;                    { success code }
END;


       {      initialize the card as system controller }

FUNCTION GpibController(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
       VAR
              err:                 OSErr;
              paramBlock:          ParamBlockRec;
              myGpibCtlBlk:        GpibCtlBlk;
              paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                  { not used }
       myGpibCtlBlk.csFlag := 0;                 { not used }
       myGpibCtlBlk.csStatus := 0;                 { a return value }
       myGpibCtlBlk.csError := 0;                  { a return value }
       myGpibCtlBlk.csCount := 0;                  { not used }
       myGpibCtlBlk.csDataBuf := NIL;              { not used }
       myGpibCtlBlk.csAddrList := NIL;             { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;             { not used }
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 0;                     { for 'ContInit' }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }

       GpibController := err;
END;




       {      initialize the card as a device }

FUNCTION GpibDevice(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
       VAR
              err:                  OSErr;
              paramBlock:           ParamBlockRec;
              myGpibCtlBlk:         GpibCtlBlk;
              paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                    { not used }
       myGpibCtlBlk.csFlag := 0;                   { not used }
       myGpibCtlBlk.csStatus := 0;                 { a return value }
       myGpibCtlBlk.csError := 0;                  { a return value }
       myGpibCtlBlk.csCount := 0;                  { not used }
       myGpibCtlBlk.csDataBuf := NIL;              { not used }
       myGpibCtlBlk.csAddrList := NIL;             { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;             { not used }
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 17;                    { for 'NContInit' }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }

       GpibDevice := err;
END;
```

```
       {      send remote enable  }

FUNCTION GpibRemote(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                  { not used }
       myGpibCtlBlk.csFlag := 0;                 { not used }
       myGpibCtlBlk.csStatus := 0;               { a return value }
       myGpibCtlBlk.csError := 0;                { a return value }
       myGpibCtlBlk.csCount := 0;                { not used }
       myGpibCtlBlk.csDataBuf := NIL;            { not used }
       myGpibCtlBlk.csAddrList := NIL;           { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;           { not used }
       paramBlock.ioVRefNum := 0;                { not used }
       paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
       paramBlock.csCode := 2;                   { for RemEnable }
       paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;          { interface's status }
       error := myGpibCtlBlk.csError;            { driver's result code }

       GpibRemote := err;

END;



       {      send 'local'  }

FUNCTION GpibLocal(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                  { not used }
       myGpibCtlBlk.csFlag := 0;                 { not used }
       myGpibCtlBlk.csStatus := 0;               { a return value }
       myGpibCtlBlk.csError := 0;                { a return value }
       myGpibCtlBlk.csCount := 0;                { not used }
       myGpibCtlBlk.csDataBuf := NIL;            { not used }
       myGpibCtlBlk.csAddrList := NIL;           { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;           { not used }
       paramBlock.ioVRefNum := 0;                { not used }
```

```
        paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
        paramBlock.csCode := 3;                     { for 'Local' control call }
        paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        status := myGpibCtlBlk.csStatus;            { interface's status }
        error := myGpibCtlBlk.csError;              { driver's result code }

        GpibLocal := err;

END;




        {      send 'interface clear'  }

FUNCTION GpibIfc(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
        err:                  OSErr;
        paramBlock:           ParamBlockRec;
        myGpibCtlBlk:         GpibCtlBlk;
        paramAddr:            LONGINT;

BEGIN
        { first set up the driver's control call parameters }
        myGpibCtlBlk.csVar := 0;                    { not used }
        myGpibCtlBlk.csFlag := 0;                   { not used }
        myGpibCtlBlk.csStatus := 0;                 { a return value }
        myGpibCtlBlk.csError := 0;                  { a return value }
        myGpibCtlBlk.csCount := 0;                  { not used }
        myGpibCtlBlk.csDataBuf := NIL;              { not used }
        myGpibCtlBlk.csAddrList := NIL;             { not used }

        { now set up the device manager's control call parameters }
        paramBlock.ioCompletion := NIL;             { not used }
        paramBlock.ioVRefNum := 0;                  { not used }
        paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
        paramBlock.csCode := 4;                     { for 'Ifc' control call }
        paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        status := myGpibCtlBlk.csStatus;            { interface's status }
        error := myGpibCtlBlk.csError;              { driver's result code }

        GpibIfc := err;

END;
```

```
       {      mark a new EOS character  }

FUNCTION GpibNewEos(refNum: INTEGER; theChar: CHAR; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := INTEGER(theChar);    { pass new character value }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := NIL;            { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;            { not used }
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 5;                    { for 'SetEos' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibNewEos := err;
END;



       {      Set new GPIB address  }

FUNCTION GpibNewAddr(refNum, theAddr: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := theAddr ;            { pass new address value }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := NIL;            { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 6;                    { for 'SetMyAddr' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibNewAddr := err;
END;





       { send 'group execute trigger' to listener list }

FUNCTION GpibGet(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                   { not used }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := @listeners[1];  { pointer to listener list }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 7;                    { for 'Trig' control call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibGet := err;
END;
```

```
       { send 'selected device clear' to listener list }

FUNCTION GpibSdc(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                 { not used }
       myGpibCtlBlk.csFlag := 0;                { not used }
       myGpibCtlBlk.csStatus := 0;              { a return value }
       myGpibCtlBlk.csError := 0;               { a return value }
       myGpibCtlBlk.csCount := 0;               { not used }
       myGpibCtlBlk.csDataBuf := NIL;           { not used }
       myGpibCtlBlk.csAddrList := @listeners[1];  { pointer to listener list }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;               { not used }
       paramBlock.ioRefNum := refNum;           { from 'GpibOpen' call }
       paramBlock.csCode := 8;                  { for 'DevClr' call }
       paramAddr := LONGINT(@myGpibCtlBlk);     { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;         { interface's status }
       error := myGpibCtlBlk.csError;           { driver's result code }

       GpibSdc := err;
END;




       { receive data (as controller) from a selected talker }

FUNCTION GpibCRcv(refNum: INTEGER; VAR count: LONGINT; talker: CHAR; bufferPtr: Ptr;
                                    eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;
       myStr:               Str255;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                 { not used }
       IF eosCheck THEN                         { check for EOS character in data stream? }
              myGpibCtlBlk.csFlag := 1
       ELSE
              myGpibCtlBlk.csFlag := 0;
       myGpibCtlBlk.csStatus := 0;              { a return value }
       myGpibCtlBlk.csError := 0;               { a return value }
       myGpibCtlBlk.csCount := count;           { max buffer size }
       myGpibCtlBlk.csDataBuf := bufferPtr;     { the input buffer }
       myStr := '1';                            { placeholder}
       myStr[1] := talker;
       myGpibCtlBlk.csAddrList := @myStr[1];      { the device address }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 14;                   { for 'CRcv' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }
       count := myGpibCtlBlk.csCount;             { return the actual # chars received }

       GpibCRcv := err;
END;


       { send data (as controller) to selected listener(s) }

FUNCTION GpibCSend(refNum: INTEGER; VAR count: LONGINT; listeners: Str255; bufferPtr: Ptr;
                            sendEoi, eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       IF eosCheck THEN                           { check for EOS character ? }
              myGpibCtlBlk.csVar := 1
       ELSE
              myGpibCtlBlk.csVar := 0;
       IF sendEoi THEN                            { send last byte with EOI ? }
              myGpibCtlBlk.csFlag := 1
       ELSE
              myGpibCtlBlk.csFlag := 0;
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := count;             { max buffer size }
       myGpibCtlBlk.csDataBuf := bufferPtr;       { the first data byte }
       myGpibCtlBlk.csAddrList := @listeners[1];  { the device addresses }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 15;                   { for 'CSend' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }
       count := myGpibCtlBlk.csCount;             { return actual number of characters sent }

       GpibCSend := err;
END;
```

```
       { send command string (as controller) }

FUNCTION GpibSendCmd(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                                                           VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                     { not used }
       myGpibCtlBlk.csFlag := 0;                    { not used }
       myGpibCtlBlk.csStatus := 0;                  { a return value }
       myGpibCtlBlk.csError := 0;                   { a return value }
       myGpibCtlBlk.csCount := count;               { max buffer size }
       myGpibCtlBlk.csDataBuf := bufferPtr;         { the first data byte }
       myGpibCtlBlk.csAddrList := NIL;              { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                   { not used }
       paramBlock.ioRefNum := refNum;               { from 'GpibOpen' call }
       paramBlock.csCode := 16;                     { for 'SendCmd' call }
       paramAddr := LONGINT(@myGpibCtlBlk);         { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;             { interface's status }
       error := myGpibCtlBlk.csError;               { driver's result code }
       count := myGpibCtlBlk.csCount;               { return actual number of characters sent }

       GpibSendCmd := err;
END;




{
       Configure the listener list with the specified configuration bytes
       in order that the devices may respond to a parallel poll operation.
}

FUNCTION GpibPpEn(refNum: INTEGER; listeners: Str255; configStr: Str255;
                                                           VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                     { not used }
       myGpibCtlBlk.csFlag := 0;                    { not used }
       myGpibCtlBlk.csStatus := 0;                  { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := @configStr[1];   { the first configuration data byte }
       myGpibCtlBlk.csAddrList := @listeners[1];  { the device addresses }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 9;                    { for 'GpibPpEn' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibPpEn := err;
END;





{
       Disable one or more specified devices from responding to a parallel poll.
}

FUNCTION GpibPpDis(refNum: INTEGER; listeners: Str255; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                   { not used }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := @listeners[1];  { the device addresses }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 10;                   { for 'GpibPpDis' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibPpDis := err;
END;
```

```
{
       Disables parallel poll operation on all devices on the GPIB bus.
}

FUNCTION GpibPpUConfig(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                   { not used }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := NIL;            { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 11;                   { for 'GpibPpUConfig' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibPpUConfig := err;
END;
```

```
{
       Perform a parallel poll operation on the GPIB bus.
}

FUNCTION GpibCParPoll(refNum: INTEGER; VAR response: SignedByte;
                                                         VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                   { a return value }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := NIL;             { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 12;                    { for 'GpibCParPoll' call }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }
       response := SignedByte(myGpibCtlBlk.csVar);

       GpibCParPoll := err;
END;




FUNCTION GpibCSerPoll(refNum: INTEGER; talkers: Str255; bufferPtr: Ptr;
                                                          VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                    { not used }
       myGpibCtlBlk.csFlag := 0;                   { not used }
       myGpibCtlBlk.csStatus := 0;                 { a return value }
       myGpibCtlBlk.csError := 0;                  { a return value }
       myGpibCtlBlk.csCount := 0;                  { not used }
       myGpibCtlBlk.csDataBuf := bufferPtr;        { pointer to response buf }
       myGpibCtlBlk.csAddrList := @talkers[1];     { pointer to talker list }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 13;                    { for 'CSerPoll' call }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }

       GpibCSerPoll := err;
END;
```

```
              { enable/disable board interrupts }

FUNCTION GpibIntEn(refNum: INTEGER; operation: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                  { not used }
       IF operation = TRUE THEN
              myGpibCtlBlk.csFlag := $ffff { flage interupt enabled }
       ELSE
              myGpibCtlBlk.csFlag := 0;          { flage inteerrupt disabled }
       myGpibCtlBlk.csStatus := 0;               { a return value }
       myGpibCtlBlk.csError := 0;                { a return value }
       myGpibCtlBlk.csCount := 0;                { not used }
       myGpibCtlBlk.csDataBuf := NIL;            { not used }
       myGpibCtlBlk.csAddrList := NIL;           { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                { not used }
       paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
       paramBlock.csCode := 23;                  { for 'EnInter' call }
       paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;          { interface's status }
       error := myGpibCtlBlk.csError;            { driver's result code }

       GpibIntEn := err;
END;





              { write to a board address }

FUNCTION GpibWrAddr(refNum: INTEGER; address: UNIV Ptr; theByte: SignedByte;
                                                         VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := theByte;            { the byte we are writing }
       myGpibCtlBlk.csFlag := 0;                 { not used }
       myGpibCtlBlk.csStatus := 0;               { a return value }
       myGpibCtlBlk.csError := 0;                { a return value }
       myGpibCtlBlk.csCount := 0;                { not used }
       myGpibCtlBlk.csDataBuf := NIL;            { not used }
       myGpibCtlBlk.csAddrList := address;       { the address we wish to write }
```

```
        { now set up the device manager's control call parameters }
        paramBlock.ioCompletion := NIL;
        paramBlock.ioVRefNum := 0;                { not used }
        paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
        paramBlock.csCode := 26;                  { for 'Write' call }
        paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        status := myGpibCtlBlk.csStatus;          { interface's status }
        error := myGpibCtlBlk.csError;            { driver's result code }

        GpibWrAddr := err;
END;




                { read from a board address }

FUNCTION GpibRdAddr(refNum: INTEGER; address: UNIV Ptr; VAR theByte: SignedByte;
                                                          VAR status, error: INTEGER): OSErr;
VAR
        err:                 OSErr;
        paramBlock:          ParamBlockRec;
        myGpibCtlBlk:        GpibCtlBlk;
        paramAddr:           LONGINT;

BEGIN
        { first set up the driver's control call parameters }
        myGpibCtlBlk.csVar := 0;                  { a return value }
        myGpibCtlBlk.csFlag := 0;                 { not used }
        myGpibCtlBlk.csStatus := 0;               { a return value }
        myGpibCtlBlk.csError := 0;                { a return value }
        myGpibCtlBlk.csCount := 0;                { not used }
        myGpibCtlBlk.csDataBuf := NIL;            { not used }
        myGpibCtlBlk.csAddrList := address;       { the address we wish to write }

        { now set up the device manager's control call parameters }
        paramBlock.ioCompletion := NIL;
        paramBlock.ioVRefNum := 0;                { not used }
        paramBlock.ioRefNum := refNum;            { from 'GpibOpen' call }
        paramBlock.csCode := 25;                  { for 'Read' call }
        paramAddr := LONGINT(@myGpibCtlBlk);      { address of GPIB params }
        paramBlock.csParam[1] := LoWord(paramAddr);
        paramBlock.csParam[0] := HiWord(paramAddr);

        err := PBControl(@paramBlock, FALSE);
        status := myGpibCtlBlk.csStatus;          { interface's status }
        error := myGpibCtlBlk.csError;            { driver's result code }
        theByte := SignedByte(myGpibCtlBlk.csVar);

        GpibRdAddr := err;
END;
```

```
              { set output buffer configuration }

FUNCTION GpibSetOut(refNum: INTEGER; theConfig: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := theConfig;           { the buffer type }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := NIL;            { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
       paramBlock.csCode := 24;                   { for 'SetOut' call }
       paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;           { interface's status }
       error := myGpibCtlBlk.csError;             { driver's result code }

       GpibSetOut := err;
END;





       { Transfer data from a talker to a listener on the bus where the controller does not
              participate in the transaction }
FUNCTION GpibXfr(refNum: INTEGER; addresses: Str255; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                   { not used }
       myGpibCtlBlk.csFlag := 0;                  { not used }
       myGpibCtlBlk.csStatus := 0;                { a return value }
       myGpibCtlBlk.csError := 0;                 { a return value }
       myGpibCtlBlk.csCount := 0;                 { not used }
       myGpibCtlBlk.csDataBuf := NIL;             { not used }
       myGpibCtlBlk.csAddrList := @addresses[1];  { pointer to device list }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                 { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 18;                    { for 'CXfer' call }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }

       GpibXfr := err;
END;




       { receive data as a device }
FUNCTION GpibRcv(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                                    eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                    { not used }
       IF eosCheck THEN                            { check for EOS character in data stream? }
              myGpibCtlBlk.csFlag := 1
       ELSE
              myGpibCtlBlk.csFlag := 0;
       myGpibCtlBlk.csStatus := 0;                 { a return value }
       myGpibCtlBlk.csError := 0;                  { a return value }
       myGpibCtlBlk.csCount := count;              { max buffer size }
       myGpibCtlBlk.csDataBuf := bufferPtr;        { the input buffer }
       myGpibCtlBlk.csAddrList := NIL;             { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 21;                    { for 'Rcv' call }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }
       count := myGpibCtlBlk.csCount;              { return the actual # chars received }

       GpibRcv := err;
END;
```

```
       { send data as a device }
FUNCTION GpibSend(refNum: INTEGER; VAR count: LONGINT; bufferPtr: Ptr;
                             sendEoi, eosCheck: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       IF eosCheck THEN                            { check for EOS character ? }
              myGpibCtlBlk.csVar := 1
       ELSE
              myGpibCtlBlk.csVar := 0;
       IF sendEoi THEN                             { send last byte with EOI ? }
              myGpibCtlBlk.csFlag := 1
       ELSE
              myGpibCtlBlk.csFlag := 0;
       myGpibCtlBlk.csStatus := 0;                 { a return value }
       myGpibCtlBlk.csError := 0;                  { a return value }
       myGpibCtlBlk.csCount := count;              { max buffer size }
       myGpibCtlBlk.csDataBuf := bufferPtr;        { the first data byte }
       myGpibCtlBlk.csAddrList := NIL;             { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                  { not used }
       paramBlock.ioRefNum := refNum;              { from 'GpibOpen' call }
       paramBlock.csCode := 22;                    { for 'Send' call }
       paramAddr := LONGINT(@myGpibCtlBlk);        { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;            { interface's status }
       error := myGpibCtlBlk.csError;              { driver's result code }
       count := myGpibCtlBlk.csCount;              { return actual number of characters sent }

       GpibSend := err;
END;




       { receive controll from the currently active controller }
FUNCTION GpibRcvCntrl(refNum: INTEGER; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       err := GpibWrAddr(refNum, $20010, $20, status, error);    { enable DAC holdoff on
                                                          'unrecognized command' }

       IF (err = 0) THEN
              BEGIN
              { first set up the driver's control call parameters }
              myGpibCtlBlk.csVar := 0;                    { not used }
              myGpibCtlBlk.csFlag := 0;                  { not used }
              myGpibCtlBlk.csStatus := 0;                { a return value }
              myGpibCtlBlk.csError := 0;                 { a return value }
              myGpibCtlBlk.csCount := 0;                 { not used }
              myGpibCtlBlk.csDataBuf := NIL;             { not used }
              myGpibCtlBlk.csAddrList := NIL;            { not used }

              { now set up the device manager's control call parameters }
              paramBlock.ioCompletion := NIL;
              paramBlock.ioVRefNum := 0;                 { not used }
              paramBlock.ioRefNum := refNum;             { from 'GpibOpen' call }
              paramBlock.csCode := 20;                   { for 'CRcvCntrl' call }
              paramAddr := LONGINT(@myGpibCtlBlk);       { address of GPIB params }
              paramBlock.csParam[1] := LoWord(paramAddr);
              paramBlock.csParam[0] := HiWord(paramAddr);

              err := PBControl(@paramBlock, FALSE);
              status := myGpibCtlBlk.csStatus;           { interface's status }
              error := myGpibCtlBlk.csError;             { driver's result code }
              END;

       GpibRcvCntrl := err;
END;


       { pass controll to a device on the bus }
FUNCTION GpibPassCntrl(refNum: INTEGER; device: CHAR; VAR status, error: INTEGER): OSErr;
VAR
       err:                 OSErr;
       paramBlock:          ParamBlockRec;
       myGpibCtlBlk:        GpibCtlBlk;
       paramAddr:           LONGINT;
       myStr:               Str255;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                          { not used }
       myGpibCtlBlk.csFlag := 0;                         { not used }
       myGpibCtlBlk.csStatus := 0;                       { a return value }
       myGpibCtlBlk.csError := 0;                        { a return value }
       myGpibCtlBlk.csCount := 0;                        { not used }
       myGpibCtlBlk.csDataBuf := NIL;                    { not used }
       myStr := '1';                                     { placeholder}
       myStr[1] := device;
       myGpibCtlBlk.csAddrList := @myStr[1];             { the device address }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                        { not used }
       paramBlock.ioRefNum := refNum;                    { from 'GpibOpen' call }
       paramBlock.csCode := 19;                          { for 'CPassCntrl' call }
       paramAddr := LONGINT(@myGpibCtlBlk);              { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;                  { interface's status }
       error := myGpibCtlBlk.csError;                    { driver's result code }

       GpibPassCntrl := err;
END;
```

```
FUNCTION GpibNewTimout(refNum: INTEGER; value: LONGINT; VAR status, error: INTEGER): OSErr;
VAR
       err:                  OSErr;
       paramBlock:           ParamBlockRec;
       myGpibCtlBlk:         GpibCtlBlk;
       paramAddr:            LONGINT;

BEGIN
       { first set up the driver's control call parameters }
       myGpibCtlBlk.csVar := 0;                           { not used }
       myGpibCtlBlk.csFlag := 0;                          { not used }
       myGpibCtlBlk.csStatus := 0;                        { a return value }
       myGpibCtlBlk.csError := 0;                         { a return value }
       myGpibCtlBlk.csCount := value;                     { new timeout constant }
       myGpibCtlBlk.csDataBuf := NIL;                     { not used }
       myGpibCtlBlk.csAddrList := NIL;                    { not used }

       { now set up the device manager's control call parameters }
       paramBlock.ioCompletion := NIL;
       paramBlock.ioVRefNum := 0;                         { not used }
       paramBlock.ioRefNum := refNum;                     { from 'GpibOpen' call }
       paramBlock.csCode := 27;                           { for 'NewTimout' call }
       paramAddr := LONGINT(@myGpibCtlBlk);               { address of GPIB params }
       paramBlock.csParam[1] := LoWord(paramAddr);
       paramBlock.csParam[0] := HiWord(paramAddr);

       err := PBControl(@paramBlock, FALSE);
       status := myGpibCtlBlk.csStatus;                   { interface's status }
       error := myGpibCtlBlk.csError;                     { driver's result code }

       GpibNewTimout := err;
END;




FUNCTION GpibClose(refNum: INTEGER): OSErr;
VAR
       err:                  OSErr;

BEGIN
       err := CloseDriver(refNum);

       GpibClose := err;
END;




END.
```

```
*   Version 1.0   15 March, 1989
*   Version 1.1   15 May,   1991
*       • changed 'Rev2' level constant



* File gpibincl.a
*{Copyright © 1988-1991 by fishcamp engineering.  All rights reserved.}




*************
* Constants *
*************

ramaddr        EQU        $000000           ; start of ram from base address of board

gpibint0       EQU        $020000           ; 9914 int status 0 read register
gpibint1       EQU        $020010           ; 9914 int status 1 read register
gpibadst       EQU        $020008           ; 9914 address status read register
gpibbus        EQU        $020018           ; 9914 bus status read register
gpibcmd        EQU        $02000c           ; 9914 Cmd pass thru read register
gpibdatain     EQU        $02001c           ; 9914 data in register

gpibintm0      EQU        $020000           ; 9914 int mask 0 write register
gpibintm1      EQU        $020010           ; 9914 int mask 1 write register
gpibauxcmd     EQU        $020018           ; 9914 auxiliary cmd write register
gpibaddr       EQU        $020004           ; 9914 address write register
gpibserpol     EQU        $020014           ; 9914 serial poll write register
gpibparpol     EQU        $02000c           ; 9914 parallel poll write register
gpibdataout    EQU        $02001c           ; 9914 data out register

intenaddr      EQU        $040000           ; interupt enable address
intdisaddr     EQU        $060000           ; interupt disable address
swaddr         EQU        $080000           ; address of on board dip-switch
romaddr        EQU        $ff8000           ; start of rom from base address of board



*****************************************************************************************
*   The following structure is used to pass data to and from the driver during all
*   Control calls to the driver.
*
*   GpibCtlBlk =  RECORD
*                    csVar:     INTEGER;      { general purpose word has call specific
*                                                 data.  Refer to control call desired
*                                                 for variable definition. }
*                    csFlag:    INTEGER;      { general purpose word has call specific
*                                                 data.  Refer to control call desired
*                                                 for variable definition. }
*                    csStatus:  INTEGER;      { call returned status information }
*                    csError:   INTEGER;      { call returned error information }
*                    csCount:   LONGINT;      { max characters to be inputed from the bus
*                                                 or the exact number of bytes to be sent
*                                                 out over the bus. For all operations,
*                                          the actual number of bytes received/transmitted
*                                                 will be returned in this value.  }
*                    csDataBuf: Ptr;          { used for actual data to/from the driver }
*
```

```
*                                                 of devices which will be partaking in the
*                                                 following transaction.  List will contain
*                                                 valid addresses terminated by the first
*                                                 non-valid address for Listeners.  For
*                                                 talkers there can only be one so only
*                                                 the byte pointed to is valid and no
terminator
*                                                 is needed.  Not used for 'Send command'. }
*
*                 END;
*
*   GpibCtlBlkPtr = ^GpibCtlBlk;
*
*
*
csVar         EQU        0                  ; (word)     - call specific data
csFlag        EQU        csVar+2            ; (word)    - call specific data
csStatus      EQU        csFlag+2           ; (word)     - returned driver status
csError       EQU        csStatus+2         ; (word)     - returned error code
csCount       EQU        csError+2          ; (longword) -  # data bytes in buffer
csDataBuf     EQU        csCount+4          ; Pointer to output/input data string.
csAddrList    EQU        csDataBuf+4        ; Pointer to device list (talker/listener)
*
*****************************************************************************************



*   Control call operating system Error codes
gpibErr       EQU        -127               ; returned to the O.S.


*   Control call Error codes returned in 'csError'
ctlNoErr      EQU        $0000              ; default error code for control calls
ctlTime       EQU        $0001              ; timeout over GPIB buss
ctlBaddr      EQU        $0002              ; bad device address
ctlUnkErr     EQU        $0003              ; unknown error
ctlNinChg     EQU        $0004              ; interface not controller in charge
ctlInChg      EQU        $0005              ; interface not configured as device


*   Status bit codes returned in 'csStatus'
stGood        EQU        $0000              ; Default status returned
stErr         EQU        $8000              ; error occured during call
stTime        EQU        $4000              ; timeout occurred during call
stEnd         EQU        $2000              ; END or EOS occurred during operation
stCnt         EQU        $0200              ; I/O operation buffer size reached
stCmplt       EQU        $0100              ; I/O operation completed during call
stCic         EQU        $0020              ; interface controller in charge
stNCic        EQU        $FFDF              ; not mask 4 interface controller in charge


*   The following need to be supplied by Apple
*       sRsrc_Type values
*


GpibBoardId       EQU        $020B              ; As assigned by Apple DTS
CatCommunication  EQU        $0006              ;
TypIEEE488        EQU        $0004              ;
DrSwNBSGPIB       EQU        $0103              ;
DrHwNBSGPIB       EQU        $0100              ;
```

```
DrSwBoard          EQU        $0000             ; always 0 for board sResource
DrHwBoard          EQU        $0000             ; always 0 for board sResource

ROMSIZE            EQU        8192              ; size of on-board ROM
fhBlockSize        EQU        20                ; format/header is 20 bytes long
Rev2               EQU        2                 ; current revision level of this ROM
sRsrc_Board        EQU        1                 ; board sResource list ID
sRsrc_gpib         EQU        128               ; gpib sResource list ID

*   Apple defined sResource list ID numbers
sRsrc_Type         EQU        1                 ; type of resource
sRsrc_Name         EQU        2                 ; name of sResource
sRsrc_Icon         EQU        3                 ; Icon for the sResource
sRsrc_DrvrDir      EQU        4                 ; Driver directory for the sResource
sRsrc_LoadRec      EQU        5                 ; Load record for the sResource
sRsrc_BootRec      EQU        6                 ; Boot record
sRsrc_Flags        EQU        7                 ; sResource flags
sRsrc_HWDevId      EQU        8                 ; Hardware device Id

*   Apple defined Board sResource entry ID numbers
STimeOut           EQU        35                ; TimeOut constant



*   9914 equates

RLCM               EQU        $02               ; remote/local change
SPASM              EQU        $04               ; serial poll mask
eoim0              EQU        $08               ; EOI mask
bom                EQU        $10               ; byte out mask
bim                EQU        $20               ; byte in mask
INTR1              EQU        $40               ; reg 1 interrupt mask
INTR0              EQU        $80               ; reg 0 interrupt mask

IFCM               EQU        $01               ; interface clear mask
SRQM               EQU        $02               ; service request
MAM                EQU        $04               ; (MLA or MTA) and not SPMS
DCASM              EQU        $08               ; device clear state
APTM               EQU        $10               ; address pass thru
ucgm               EQU        $20               ; unidentified command
ERRM               EQU        $40               ; incomplete handshake
GETM               EQU        $80               ; group execute trigger

ULPAM              EQU        $01               ; LSM of last address rec.
tadsm              EQU        $02               ; talk addressed
LADSM              EQU        $04               ; listen addressed
TPASM              EQU        $08               ; pri. talk addressed
LPASM              EQU        $10               ; pri. listen addressed
ATNM               EQU        $20               ; attention status
LLOM               EQU        $40               ; local lockout mask
REMM               EQU        $80               ; remote enable mask

RENM               EQU        $01               ; remote enabled status
IFCMB              EQU        $02               ; interface clear status
SRQNM              EQU        $04               ; service requested
eoimk              EQU        $08               ; end or identify
NRFDM              EQU        $10               ; not ready for data
NDACM              EQU        $20               ; not data accepted status
DAVM               EQU        $40               ; data valid status
ATNMB              EQU        $80               ; attention status
```

```
CLRM              EQU        $7f                ; clear/set operation (clear)
SETM              EQU        $80                ; clear/set operation (set)

DAT               EQU        $20                ; disable talk mode
DAL               EQU        $40                ; disable listen mode
DALT              EQU        $60                ; disable both
EDPA              EQU        $80                ; enable dual pri. address mode

RSV               EQU        $04                ; request service


*   set/reset commands for 9914

swrst             EQU        $80                ; chip reset
swrstclr          EQU        $00                ; end reset
dacr              EQU        $01                ; release acds holdoff
IVASR             EQU        $01                ; invalid secondary address
VSADR             EQU        $81                ; valid secondary address
hdfa              EQU        $83                ; holdoff on all data
hdaclr            EQU        $03                ; release holdoff on all
hdfe              EQU        $84                ; holdoff on EOI only
hdeclr            EQU        $04                ; release holdoff on EOI
FGET              EQU        $86                ; force group execute trigger
FGTCLR            EQU        $06                ; end group execute trigger
RTL               EQU        $87                ; return to local
RTLCLR            EQU        $07                ; end return to local
lon               EQU        $89                ; listen only
LONCLR            EQU        $09                ; end listen only
ton               EQU        $8a                ; talk only
tonclr            EQU        $0a                ; end talk only
rpp               EQU        $8e                ; request parallel poll
rppclr            EQU        $0e                ; end parallel poll
sic               EQU        $8f                ; interface clear
siclr             EQU        $0f                ; end interface clear
sre               EQU        $90                ; send REM
sreclr            EQU        $10                ; reset REM
DAI               EQU        $93                ; disable all interrupts
DAICLR            EQU        $13                ; enable all interrupts
STDL              EQU        $95                ; set T1 delay
STDCLR            EQU        $15                ; reset T1 timer
shdw              EQU        $96                ; shadow handshake
shdclr            EQU        $16                ; reset shadow handshake
vstdl             EQU        $17                ; very fast T1
vstdlclr          EQU        $97                ; reset very fast T1
RSV2S             EQU        $18                ; service request #2
RSV2C             EQU        $98                ; reset RSV #2

*   pulse type commmands

rhdf              EQU        $02                ; release RFD holdoff
NBAF              EQU        $05                ; suppress byte sent
feoi              EQU        $08                ; send eoi with next byte
gts               EQU        $0b                ; goto standby
tca               EQU        $0c                ; take control asych.
tcs               EQU        $0d                ; take control sync.
rqc               EQU        $11                ; request control
rlc               EQU        $12                ; release control
PTS               EQU        $14                ; pass thru next secondary
```

```
*   GPIB commands

mla                 EQU        $20                 ; my listen address
mta                 EQU        $40                 ; my talk address
unl                 EQU        $3f                 ; universal unlisten
unt                 EQU        $5f                 ; universal untalk
dcl                 EQU        $14                 ; device clear
get                 EQU        $08                 ; group execute trigger
llo                 EQU        $11                 ; local lock out
ppc                 EQU        $05                 ; parallel poll configure
ppd                 EQU        $70                 ; parallel poll disable
ppe                 EQU        $60                 ; parallel poll enable
ppu                 EQU        $15                 ; parallel poll unconfigure
sdc                 EQU        $04                 ; selected device clear
spd                 EQU        $19                 ; serial poll disable
spe                 EQU        $18                 ; serial poll enable
tct                 EQU        $09                 ; take controll
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page   1
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

                                    * File gpibrom.a
                                    *
                                    *     change history:
                                    *           5/15/91     -     changed '_RevLevel' level string
                                    *
                                    *{Copyright © 1988-1991 by fishcamp engineering.  All rights reserved.}


                                                      MACHINE           MC68020
                                                      STRING            C
                                                      PRINT       ON




                                    **********************************************************************************************
                                    *     Begin declaration ROM
                                    **********************************************************************************************

00000                               gpibDeclRom MAIN
00000
00000
00000                               **********************************************************************************************
00000                               *                 Directory
00000                               **********************************************************************************************
00000                               _sRsrcDir   OSLstEntry  sRsrc_Board,_sRsrc_Board                  ; References the Board sResource
00000   0100 000C             1                       DC.L        (sRsrc_Board<<24)++((_sRsrc_Board-*)**$00FFFFFF)
00004                                                 OSLstEntry  sRsrc_gpib,_sRsrc_gpib              ; References the gpib sResource
00004   8000 0084             1                       DC.L        (sRsrc_gpib<<24)++((_sRsrc_gpib-*)**$00FFFFFF)
00008                                                 DatLstEntry EndOfList,0             ; end of the list
00008   FF00 0000             1                       DC.L        (EndOfList<<24)+0
0000C
0000C                               **********************************************************************************************
0000C                               *                 sRsrc_Board List
0000C                               **********************************************************************************************
0000C                               _sRsrc_Board      OSLstEntry  sRsrc_Type,_BoardType               ; References the sResource type
0000C   0100 0014             1                       DC.L        (sRsrc_Type<<24)++((_BoardType-*)**$00FFFFFF)
00010                                                 OSLstEntry  sRsrc_Name,_BoardName               ; References the sResource name
00010   0200 0018             1                       DC.L        (sRsrc_Name<<24)++((_BoardName-*)**$00FFFFFF)
00014                                                 DatLstEntry BoardId,GpibBoardId           ; the board Id
00014   2000 020B             1                       DC.L        (BoardId<<24)+GpibBoardId
00018                                                 OSLstEntry  VendorInfo,_VendorInfo              ; references the vendor information list
00018   2400 0034             1                       DC.L        (VendorInfo<<24)++((_VendorInfo-*)**$00FFFFFF)
0001C                                                 DatLstEntry EndOfList,0             ; end of the list
0001C   FF00 0000             1                       DC.L        (EndOfList<<24)+0
00020
00020   0001                        _BoardType        DC.W  CatBoard                                  ; the Board sResource:  <Category>
00022   0000                                          DC.W  TypBoard                                  ;                                   <Type>
00024   0000                                          DC.W  DrSwBoard                                 ;                                   <DrvrSw>
00026   0000                                          DC.W  DrHwBoard                                 ;                                   <DrvrHw>
00028   6669736863616D              _BoardName        DC.L  'fishcamp engineering NBS-GPIB card'; board's official product name
0004C
0004C
0004C
0004C                               **********************************************************************************************
0004C                               *                 Vendor info record
0004C                               **********************************************************************************************
0004C                               _VendorInfo OSLstEntry  VendorId,_VendorId                  ; references the vendor Id
0004C   0100 0010             1                       DC.L        (VendorId<<24)++((_VendorId-*)**$00FFFFFF)
00050                                                 OSLstEntry  RevLevel,_RevLevel                  ; references the revision level
00050   0300 0024             1                       DC.L        (RevLevel<<24)++((_RevLevel-*)**$00FFFFFF)
00054                                                 OSLstEntry  PartNum,_PartNum                    ; references the part number
00054   0400 0028             1                       DC.L        (PartNum<<24)++((_PartNum-*)**$00FFFFFF)
00058                                                 DatLstEntry EndOfList,0             ; end of the list
00058   FF00 0000             1                       DC.L        (EndOfList<<24)+0
0005C
0005C   6669736863616D              _VendorId         DC.L        'fishcamp engineering'              ; the vendor id
00074   52657620312E31              _RevLevel         DC.L        'Rev 1.1'                           ; the revision level
0007C   4E42532D475049              _PartNum          DC.L        'NBS-GPIB'                          ; the part number
00088
00088
00088                               **********************************************************************************************
00088                               *                 sRsrc_gpib
00088                               **********************************************************************************************
00088                               _sRsrc_gpib OSLstEntry  sRsrc_Type,_GpibType                ; references the sResource type
00088   0100 0014             1                       DC.L        (sRsrc_Type<<24)++((_GpibType-*)**$00FFFFFF)
0008C                                                 OSLstEntry  sRsrc_Name,_GpibName                ; references the sResource name
0008C   0200 0018             1                       DC.L        (sRsrc_Name<<24)++((_GpibName-*)**$00FFFFFF)
00090                                                 OSLstEntry  sRsrc_DrvrDir,_GpibDrvrDir          ; references the driver directory
00090   0400 0038             1                       DC.L        (sRsrc_DrvrDir<<24)++((_GpibDrvrDir-*)**$00FFFFFF)
00094                                                 DatLstEntry sRsrc_HWDevId,1               ; the hardware device Id
00094   0800 0001             1                       DC.L        (sRsrc_HWDevId<<24)+1
00098                                                 DatLstEntry EndOfList,0             ; end of the list
00098   FF00 0000             1                       DC.L        (EndOfList<<24)+0
0009C
0009C   0006                        _GpibType         DC.W        CatCommunication                    ; Gpib sResource: <Category>
0009E   0004                                          DC.W        TypIEEE488                          ;                             <Type>
000A0   0103                                          DC.W        DrSwNBSGPIB                   ;                             <DrvrSw>
000A2   0100                                          DC.W        DrHwNBSGPIB                   ;                             <DrvrHw>
000A4
000A4   496E74656C4275              _GpibName         DC.L        'IntelBus_IEEE488_fishcamp_NBS-GPIB'
000C8
000C8                               **********************************************************************************************
000C8                               *                 driver directory
000C8                               **********************************************************************************************
000C8                               _GpibDrvrDir      OSLstEntry  sMacOS68020,_sMacOS68020              ; references the Macintosh-OS 68020
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   2
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

000C8   0200 0008             1                       DC.L        (sMacOS68020<<24)++((_sMacOS68020-*)**$00FFFFFF)
000CC                                                 DatLstEntry EndOfList,0             ; end of the list
000CC   FF00 0000             1                       DC.L        (EndOfList<<24)+0
000D0
000D0                               ; Driver-1 (68020)
000D0   0000 1D9E                   _sMacOS68020      DC.L        End020Drvr-_sMacOS68020       ; the physical block size
000D4                                                 INCLUDE           'gpibdrvr.a'                  ; the header/code
000D4                               *     Version 1.0   15 March, 1989
000D4                               *     Version 1.1   15 May,   1991
000D4                               *           - changed 'NCInit' routine to properly update the 'swImage' memory location
000D4
000D4
000D4
000D4                               * File gpibdrvr.a
000D4                               *{Copyright © 1988-1991 by fishcamp engineering.  All rights reserved.}
000D4
000D4
000D4
000D4
000D4                                                 BLANKS            ON
000D4                                                 STRING            ASIS
000D4
000D4
000D4
000D4
000D4
000D4                               ********************************************************************************************
000D4                               *                 local vars, definitions etc.
000D4                               ********************************************************************************************
000D4
000D4                               ; This is local storage, starting at 'ramaddr' in the local RAM on the card.
000D4                               ;           We multiply offsets by four because we only use byte lane '3' on the
000D4                               ;           NuBus card.
000D4
000D4   0000 0000                   DCEPtr            EQU         0                       ; Pointer to our DCE
000D4   0000 0010                   eos               EQU         DCEPtr+(4*4)            ; End-Of-String char for GPIB transfers
000D4   0000 0014                   gpibAddrSt        EQU         eos+(1*4)               ; my GPIB address
000D4   0000 0018                   gpibDrvrOpen      EQU         gpibAddrSt+(1*4)        ; flag indicating driver is currently open
000D4   0000 001C                   amController      EQU         gpibDrvrOpen+(1*4)      ; flag indicating configuration as controller
000D4   0000 0020                   timot             EQU         amController+(1*4)      ; longword count thru buss loop operations for
000D4                                                                                     ; 'noresponse' result.
000D4   0000 0030                   swImage           EQU         timot+(4*4)             ; memory image of config latch
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4                               ;---------------------------------------------
000D4                               ; Write the specified byte to the specified NuBus address.
000D4                               ; The address actually used will be the address specified
000D4                               ; added to the board's base address which should be contained in A1.
000D4                               ; The board's base address for slot 9 with byte lane 3 used
000D4                               ; would be $f9000003.
000D4                               ;
000D4                               ;     Call:                         A1 - board base address
000D4                               ;
000D4                               ;     Registers affected:           None
000D4                               ;
000D4                                                 MACRO
000D4                                                 MWrite            &Address,&Data
000D4                                                 MOVEM.L           D0/A0,-(SP)             ; save work registers
000D4
000D4                                                 MOVE.L            A1,D0                   ; from board base address
000D4                                                 ADD.L             #&Address,D0            ; add to where byte will go
000D4                                                 MOVEA.L           D0,A0                   ; A0 has address
000D4                                                 MOVE.B            #&Data,D0               ; set data
000D4                                                 BSR               NbWrite
000D4
000D4                                                 MOVEM.L           (SP)+,D0/A0             ; restore registers
000D4                                                 ENDM
000D4                               ;---------------------------------------------
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4                               ;---------------------------------------------
000D4                               ;Set up the flag in the control call return status word
000D4                               ;     which flags the controller status of the interface
000D4                               ;
000D4                               ;     Call:             A1 - board base address
000D4                               ;                       A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
000D4                               ;
000D4                               ;     Registers affected:           None
000D4                               ;
000D4                                                 MACRO
000D4                                                 MSetCIC
000D4
000D4                                                 BSR         AmIncharge                    ; are we the controller in charge?
000D4                                                 BNE.S       @StCIC1                       ; no ...
000D4                                                 ORI.W       #stCic,csStatus(A2)           ; flag CIC
000D4                                                 BRA.S       @StCIC2
000D4
```

```
MC680xx Assembler - Ver 3.2b6                                                 18-May-91  Page   3
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

000D4                               @StCIC1           ANDI.W            #stNCic,csStatus(A2)    ; flag /CIC
000D4
000D4                               @StCIC2           NOP
000D4
000D4                                                 ENDM
000D4                               ;---------------------------------------------
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4
000D4                               ********************************************************************************************
000D4                               *                       gpib driver header
000D4                               ********************************************************************************************
000D4
000D4   4400                        GpibDrvr          DC.W        $4400             ; ctl,needslock
000D6   0000 0000 0000                                DC.W        0,0,0             ; not used but required values
000DC
000DC                               ; Entry point offset table
000DC
000DC   001E                                          DC.W        GpibOpen-GpibDrvr ; open
000DE   0000                                          DC.W        GpibDrvr-GpibDrvr ; no prime
000E0   013A                                          DC.W        GpibCtl-GpibDrvr  ; control
000E2   0000                                          DC.W        GpibDrvr-GpibDrvr ; no status
000E4   0104                                          DC.W        GpibClose-GpibDrvr; close
000E6
000E6                                                 STRING      Pascal
000E6   082E46635F6770              GpibTitle         DC.B        '.Fc_gpib'        ; driver name
000EF                                                 STRING      ASIS
000EF   0000 00F0                                     ALIGN 2                       ; force alignment
000F0   0000                                          DC.W        0                 ; version - 0
000F2
000F2
000F2
000F2
000F2                               ********************************************************************************************
000F2                               *     GpibOpen initializes local storage.
000F2                               *
000F2                               *     Entry:      A0 - param blk pointer
000F2                               *                 A1 - DCE pointer
000F2                               *
000F2                               *     Locals:     A0 - temporary
000F2                               *                 A1 - board base address
000F2                               *                 A2 - Saved param block pointer
000F2                               *                 A3 - Saved DCE pointer
000F2                               *                 D0 - temporary
000F2                               *                 D1 - temporary
000F2                               ********************************************************************************************
000F2                               ; save registers
000F2                               GpibOpen
000F2 G 2448                                          MOVE.L      A0,A2             ; A2 <- param block pointer
000F4 G 2649                                          MOVE.L      A1,A3             ; A3 <- DCE pointer
000F6
000F6                               ; get base address of board
000F6   102B 0028                                     MOVE.B      dCtlSlot(A3),D0   ; get the slot address
000FA   E188                                          LSL.L       #8,D0             ; shift the 4 slot bits into proper position
000FC   E188                                          LSL.L       #8,D0
000FE   E188                                          LSL.L       #8,D0
00100   0080 F000 0003                                ORI.L       #$f0000003,D0     ; Slot space
00106   2240                                          MOVEA.L     D0,A1             ; A1 = board base address
00108
00108                               ; save DCE pointer in local storage
00108   2009                                          MOVE.L      A1,D0             ; from board base address
0010A G 0680 0000 000C                                ADD.L       #(DCEPtr+12),D0   ; add to where LSB byte will go
00110   2040                                          MOVEA.L     D0,A0             ; A0 has address
00112   200B                                          MOVE.L      A3,D0             ; get DCE pointer
00114   6100 1D3C       01E52                         BSR         NbWrite           ; write low byte
00118   E088                                          LSR.L       #8,D0             ; position next byte
0011A   2208                                          MOVE.L      A0,D1
0011C G 5981                                          SUB.L       #4,D1
0011E   2041                                          MOVEA.L     D1,A0             ; point to address of next byte
00120   6100 1D30       01E52                         BSR         NbWrite           ; write 2nd byte
00124   E088                                          LSR.L       #8,D0             ; position next byte
00126   2208                                          MOVE.L      A0,D1
00128 G 5981                                          SUB.L       #4,D1
0012A   2041                                          MOVEA.L     D1,A0             ; point to address of next byte
0012C   6100 1D24       01E52                         BSR         NbWrite           ; write 3rd byte
00130   E088                                          LSR.L       #8,D0             ; position next byte
00132   2208                                          MOVE.L      A0,D1
00134 G 5981                                          SUB.L       #4,D1
00136   2041                                          MOVEA.L     D1,A0             ; point to address of next byte
00138   6100 1D18       01E52                         BSR         NbWrite           ; write high byte
0013C
0013C                               ; set default 'eos' byte of <LF>
0013C                                                 MWrite      eos,$0a
0013C   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)       ; save work registers
00140                         1
00140   2009                  1                       MOVE.L      A1,D0             ; from board base address
00142 G 0680 0000 0010        1                       ADD.L       #eos,D0           ; add to where byte will go
00148   2040                  1                       MOVEA.L     D0,A0             ; A0 has address
0014A   103C 000A             1                       MOVE.B      #$0a,D0           ; set data
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   4
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0014E   6100 1D02       01E52 1                       BSR               NbWrite
00152                         1
00152   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0       ; restore registers
00156
00156                               ; set default 'gpibAddrSt' byte of 0
00156                                                 MWrite            gpibAddrSt,0
00156   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)       ; save work registers
0015A                         1
0015A   2009                  1                       MOVE.L            A1,D0             ; from board base address
0015C G 0680 0000 0014        1                       ADD.L             #gpibAddrSt,D0    ; add to where byte will go
00162   2040                  1                       MOVEA.L           D0,A0             ; A0 has address
00164 G 4200                  1                       MOVE.B            #0,D0             ; set data
00166   6100 1CEA       01E52 1                       BSR               NbWrite
0016A                         1
0016A   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0       ; restore registers
0016E
0016E                               ; flag driver open
0016E                                                 MWrite            gpibDrvrOpen,$ff
0016E   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)       ; save work registers
00172                         1
00172   2009                  1                       MOVE.L            A1,D0             ; from board base address
00174 G 0680 0000 0018        1                       ADD.L             #gpibDrvrOpen,D0  ; add to where byte will go
0017A   2040                  1                       MOVEA.L           D0,A0             ; A0 has address
0017C   103C 00FF             1                       MOVE.B            #$ff,D0           ; set data
00180   6100 1CD0       01E52 1                       BSR               NbWrite
00184                         1
00184   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0       ; restore registers
00188
00188                               ; set default timout constant
00188   2009                                          MOVE.L            A1,D0             ; from board base address
0018A G 0680 0000 002C                                ADD.L             #(timot+12),D0    ; add to where LSB byte will go
00190   2040                                          MOVEA.L           D0,A0             ; A0 has address
00192   203C 0000 2000                                MOVE.L            #$00002000,D0     ; this many times thru loop
00198   6100 1CB8       01E52                         BSR               NbWrite           ; write low byte
0019C   E088                                          LSR.L             #8,D0             ; position next byte
0019E   2208                                          MOVE.L            A0,D1
001A0 G 5981                                          SUB.L             #4,D1
001A2   2041                                          MOVEA.L           D1,A0             ; point to address of next byte
001A4   6100 1CAC       01E52                         BSR               NbWrite           ; write 2nd byte
001A8   E088                                          LSR.L             #8,D0             ; position next byte
001AA   2208                                          MOVE.L            A0,D1
001AC G 5981                                          SUB.L             #4,D1
001AE   2041                                          MOVEA.L           D1,A0             ; point to address of next byte
001B0   6100 1CA0       01E52                         BSR               NbWrite           ; write 3rd byte
001B4   E088                                          LSR.L             #8,D0             ; position next byte
001B6   2208                                          MOVE.L            A0,D1
001B8 G 5981                                          SUB.L             #4,D1
001BA   2041                                          MOVEA.L           D1,A0             ; point to address of next byte
001BC   6100 1C94       01E52                         BSR               NbWrite           ; write high byte
001C0
001C0                               ; default to init'd as a device
001C0 G 204A                                          MOVE.L            A2,A0             ; param blk pointer restored for next routine
001C2 G 224B                                          MOVE.L            A3,A1             ; and the DCE pointer
001C4   6100 10C0       01286                         BSR               NCInit            ; init as GPIB device
001C8
001C8 G 426A 0010                                     MOVE.W            #noErr,ioResult(A2) ; flag no error
001CC   7000                                          MOVEQ             #noErr,D0;
001CE   6006            001D6                         BRA.S             EndOpen
001D0
001D0                               ; error exit
001D0   357C FFE9 0010              OpError           MOVE.W            #openErr,ioResult(A2) ; couldn't open driver
001D6
001D6   4E75                        EndOpen           RTS                                 ; return
001D8
001D8
001D8
001D8
001D8
001D8
001D8
001D8
001D8
001D8
001D8
001D8                               ***********************************************************************************************
001D8                               *     GpibClose releases device's private storage
001D8                               *
001D8                               *     Entry:      A0 - param blk pointer
001D8                               *                 A1 - DCE pointer
001D8                               *
001D8                               *     Locals:     A0 - temporary
001D8                               *                 A1 - board base address
001D8                               *                 A2 - Saved param block pointer
001D8                               *                 A3 - Saved DCE pointer
001D8                               *                 D0 - temporary
001D8                               *
001D8                               *     Return:     D0 - error
001D8                               ***********************************************************************************************
001D8                               GpibClose
001D8 G 2448                                          MOVE.L            A0,A2             ; A2 <- param block pointer
001DA G 2649                                          MOVE.L            A1,A3             ; A3 <- DCE pointer
001DC
001DC   6100 10A8       01286                         BSR               NCInit            ; init as GPIB device
001E0
001E0                               ; get base address of board
001E0   102B 0028                                     MOVE.B            dCtlSlot(A3),D0   ; get the slot address
001E4   E188                                          LSL.L             #8,D0             ; shift the 4 slot bits into proper position
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   5
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

001E6   E188                                          LSL.L       #8,D0
001E8   E188                                          LSL.L       #8,D0
001EA   0080 F000 0003                                ORI.L       #$f0000003,D0     ; Slot space
001F0   2240                                          MOVEA.L     D0,A1             ; A1 = board base address
001F2
001F2                               ; flag driver closed
001F2                                                 MWrite      gpibDrvrOpen,0
001F2   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)       ; save work registers
001F6                         1
001F6   2009                  1                       MOVE.L      A1,D0             ; from board base address
001F8 G 0680 0000 0018        1                       ADD.L       #gpibDrvrOpen,D0  ; add to where byte will go
001FE   2040                  1                       MOVEA.L     D0,A0             ; A0 has address
00200 G 4200                  1                       MOVE.B      #0,D0             ; set data
00202   6100 1C4E       01E52 1                       BSR         NbWrite
00206                         1
00206   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0       ; restore registers
0020A
0020A   7000                                          MOVEQ       #noErr,D0         ; get error into D0
0020C   4E75                                          RTS                           ; return to caller
0020E
0020E
0020E
0020E
0020E
0020E                               ********************************************************************************************
0020E                               *     GpibCtl control call handler. 27 different calls:
0020E                               *
0020E                               *           (0)   ContInit;
0020E                               *           (1)   KillIo
0020E                               *           (2)   RemEnable
0020E                               *           (3)   Local
0020E                               *           (4)   Ifc
0020E                               *           (5)   SetEos(eosChar: CHAR);
0020E                               *           (6)   SetMyAddr(theAddr: CHAR);
0020E                               *           (7)   Trig(LisList: Pointer);
0020E                               *           (8)   DevClr(LisList: Pointer);
0020E                               *           (9)   PpEnable();
0020E                               *           (10)  PpDisable();
0020E                               *           (11)  PpUConfig();
0020E                               *           (12)  CParPoll(VAR pollByte: CHAR);
0020E                               *           (13)  CSerPoll(TalkList: Ptr; VAR pollBytes: Ptr);
0020E                               *           (14)  CRcv();
0020E                               *           (15)  CSend();
0020E                               *           (16)  SendCmd();
0020E                               *           (17)  NContInit;
0020E                               *           (18)  CXfer;
0020E                               *           (19)  CPassCntrl;
0020E                               *           (20)  CRcvCntrl;
0020E                               *           (21)  Rcv();
0020E                               *           (22)  Send();
0020E                               *           (23)  EnInter;
0020E                               *           (24)  SetOut;
0020E                               *           (25)  Read;
0020E                               *           (26)  Write;
0020E                               *           (27)  NewTimout;
0020E                               *
0020E                               *     Entry:      A0    - param blk pointer
0020E                               *                 A1    - DCE pointer
0020E                               *
0020E                               *     Uses:       A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0020E                               *                 A3    - scratch ( doesn't need to be preserved )
0020E                               *                 A4    - scratch ( must be preserved )
0020E                               *                 D0-D3 - scratch ( doesn't need to be preserved )
0020E                               *
0020E                               *     Exit:       D0    - error code
0020E                               *
0020E                               ********************************************************************************************
0020E                               ; decode the requested call
0020E   48E7 0888                   GpibCtl           MOVEM.L     A0/A4/D4,-(SP)                ; save work registers (A0 is saved
00212                                                                                           ; because it is used by ExitDrvr)
00212   3028 001A                                     MOVE.W      csCode(A0),D0                 ; get the opcode
00216 G 2468 001C                                     MOVE.L      csParam(A0),A2                ; A2 <- Ptr to control parameters
0021A
0021A G 0C40 001B                                     CMP.W       #27,D0                        ; IF csCode NOT IN [0..27] THEN
0021E   6242            00262                         BHI.S       CtlBad                        ; error, csCode out of bounds
00220   E348                                          LSL.W       #1,D0                         ; Adjust csCode to be an index into the table
00222   303B 0006       0022A                         MOVE.W      CtlJumpTbl(PC,D0.W),D0        ; Get the relative offset to the routine
00226   4EFB 0002       0022A                         JMP         CtlJumpTbl(PC,D0.W)           ; GOTO the proper routine
0022A
0022A   054A                        CtlJumpTbl        DC.W  ContInit-CtlJumpTbl                 ; init 9914 as controller
0022C   003C                                          DC.W  CtlGood-CtlJumpTbl                  ; KillIo
0022E   0888                                          DC.W  RemEnable-CtlJumpTbl                ; Remote enable
00230   0906                                          DC.W  Local-CtlJumpTbl                    ; Local
00232   0984                                          DC.W  Ifc-CtlJumpTbl                      ; Interface clear
00234   0A24                                          DC.W  SetEos-CtlJumpTbl                   ; set eos character
00236   0A7E                                          DC.W  SetMyAddr-CtlJumpTbl                ; set my address
00238   06F4                                          DC.W  Trig-CtlJumpTbl                     ; trigger
0023A   07BE                                          DC.W  DevClr-CtlJumpTbl                   ; Device clear
0023C   0AEA                                          DC.W  PpEnable-CtlJumpTbl                 ; Parallel Poll enable
0023E   0BDA                                          DC.W  PpDisable-CtlJumpTbl                ; Parallel Poll disable
00240   0CD8                                          DC.W  PpUConfig-CtlJumpTbl                ; Parallel Poll unconfigure
00242   0D76                                          DC.W  CParPoll-CtlJumpTbl                 ; Parallel Poll
00244   0E50                                          DC.W  CSerPoll-CtlJumpTbl                 ; Serial Poll
00246   0056                                          DC.W  CRcv-CtlJumpTbl                     ; Controller Receive routine
00248   0276                                          DC.W  CSend-CtlJumpTbl                    ; Controller Send routine
0024A   0456                                          DC.W  SendCmd-CtlJumpTbl                  ; Controller send command string
0024C   1012                                          DC.W  NContInit-CtlJumpTbl                ; Non-Controller initialization
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   6
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0024E   114A                                          DC.W  CXfer-CtlJumpTbl                    ; Controller transfer data
00250   1332                                          DC.W  CPassCntrl-CtlJumpTbl               ; Controller pass controll
00252   1870                                          DC.W  CRcvCntrl-CtlJumpTbl                ; Controller receive control
00254   1448                                          DC.W  Rcv-CtlJumpTbl                      ; Non-Controller receive data
00256   15D0                                          DC.W  Send-CtlJumpTbl                     ; Non-Controller send data
00258   1758                                          DC.W  EnInter-CtlJumpTbl                  ; enable/disable board interrupts
0025A   17DC                                          DC.W  SetOut-CtlJumpTbl                   ; set GPIB bus output buffers
0025C   19AE                                          DC.W  Read-CtlJumpTbl                     ; Read byte from board memory
0025E   1A1A                                          DC.W  Write-CtlJumpTbl                    ; Write byte from board memory
00260   1A80                                          DC.W  NewTimout-CtlJumpTbl                ; Set new timeout constant
00262
00262   70EF                        CtlBad            MOVEQ #controlErr,D0                      ; say we don't do this one
00264   6002            00268                         BRA.S CtlDone                             ; and return
00266
00266   7000                        CtlGood           MOVEQ #noErr,D0                           ; return no error
00268
00268   4CDF 1110                   CtlDone           MOVEM.L     (SP)+,A0/A4/D4                ; restore registers
0026C   6002            00270                         BRA.S ExitDrvr
0026E
0026E   4E71                                          NOP
00270
00270
00270
00270
00270
00270
00270
00270
00270
00270                               ********************************************************************************************
00270                               *     Exit from control calls
00270                               ********************************************************************************************
00270
00270   0828 0009 0006              ExitDrvr          BTST  #NoQueueBit,ioTrap(A0)              ; no queue bit set ?
00276   6702            0027A                         BEQ.S GoIODone                            ; => no, not immediate
00278   4E75                                          RTS                                       ; otherwise, it was an immediate call
0027A
0027A G 2078 08FC                   GoIODone          MOVE.L      JIODone,A0                    ; get IODone address
0027E   4ED0                                          JMP         (A0)                          ; invoke it
00280
00280
00280
00280
00280
00280
00280
00280
00280                               ********************************************************************************************
00280                               *     CRcv(params: GpibCtlBlkPtr); routine.  This routine is called when the driver control
00280                               *     function is invoked and the request was for a controller receive data operation.
00280                               *
00280                               *
00280                               *     Entry:      A0    - param blk pointer
00280                               *                 A1    - DCE pointer
00280                               *                 A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00280                               *
00280                               *     Uses:
00280                               *                 A0    - temporary, 9914 'Data In' register address
00280                               *                 A1    - Board base address
00280                               *                 A2    - Pointer to 'params'
00280                               *                 A3    - Buffer pointer
00280                               *                 A4    - 9914 int0 register address
00280                               *                 A5    - 9914 auxcmd register address
00280                               *
00280                               *                 D0    - temporary
00280                               *                 D1    - our GPIB address, timeout loop count
00280                               *                 D2    - eos character
00280                               *                 D3    - character count
00280                               *                 D4    - temporary
00280                               *                 D5    - max buffer size
00280                               *                 D6    - temporary
00280                               *                 D7    - EOS valid BOOLEAN
00280                               *
00280                               *     Exit: D0    - error code
00280                               *
00280                               ********************************************************************************************
00280                               CRcv
00280                               ; setup working registers
00280   48E7 FFFC                                     MOVEM.L     A0-A5/D0-D7,-(SP)       ; save local work registers
00284
00284   1029 0028                                     MOVE.B      dCtlSlot(A1),D0         ; get the slot address
00288   E188                                          LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
0028A   E188                                          LSL.L       #8,D0
0028C   E188                                          LSL.L       #8,D0
0028E   0080 F000 0003                                ORI.L       #$f0000003,D0           ; Slot space
00294   2240                                          MOVEA.L     D0,A1                   ; A1 = board base address
00296
00296   2009                                          MOVE.L      A1,D0
00298 G 0680 0002 0000                                ADD.L       #gpibint0,D0
0029E   2840                                          MOVEA.L     D0,A4                   ; A4 = 9914 int0 register address
002A0   2009                                          MOVE.L      A1,D0
002A2 G 0680 0002 0018                                ADD.L       #gpibauxcmd,D0
002A8   2A40                                          MOVEA.L     D0,A5                   ; A5 = 9914 auxcmd register address
002AA   266A 000C                                     MOVEA.L     csDataBuf(A2),A3        ; A3 = pointer to input buffer
002AE
002AE   6100 1AF2       01DA2                         BSR         GetGpibAddr
002B2   1200                                          MOVE.B      D0,D1                   ; D1 = our GPIB address
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   7
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

002B4   6100 1AD4       01D8A                                 BSR               GetEos
002B8   1400                                                  MOVE.B            D0,D2                         ; D2 = end-of-string character
002BA   2A2A 0008                                             MOVE.L            csCount(A2),D5                ; D5 = max input char count
002BE   3E2A 0002                                             MOVE.W            csFlag(A2),D7                 ; D7 = EOS Valid BOOLEAN
002C2 G 7600                                                  MOVE.L            #0,D3                         ; D3 = cleared character count
002C4
002C4                               ; set up the default return status.  Other bits will be filled in later.
002C4 G 426A 0004                                             MOVE.W            #stGood,csStatus(A2)          ; Default status
002C8
002C8   6100 1B48       01E12                                 BSR               AmIncharge                    ; are we the controller in charge?
002CC   6600 017C       0044A                                 BNE               CRcvNchg                      ; no ...
002D0
002D0                               ; get the talker address
002D0   206A 0010                                             MOVEA.L           csAddrList(A2),A0             ; get pointer to talker list
002D4   1010                                                  MOVE.B            (A0),D0                       ; get the talker
002D6
002D6                               ; check for valid talker
002D6 G 0C40 0040                                             CMP.W             #$40,D0
002DA   6D00 0182       0045E                                 BLT               CRcvBadAddr                   ; talker address too lo
002DE G 0C40 005E                                             CMP.W             #$5e,D0
002E2   6E00 017A       0045E                                 BGT               CRcvBadAddr                   ; talker address too hi
002E6
002E6                               ; output the talker address to GPIB
002E6   6100 1A72       01D5A                                 BSR               DataOut
002EA   6100 1A3C       01D28                                 BSR               WaitOut                       ; wait for GPIB bus free
002EE   6700 0144       00434                                 BEQ               CRcvTime1                     ; Bus timed out
002F2
002F2                               ; stop other listeners
002F2   103C 003F                                             MOVE.B            #unl,D0
002F6   6100 1A62       01D5A                                 BSR               DataOut
002FA   6100 1A2C       01D28                                 BSR               WaitOut                       ; wait for byte out
002FE   6700 0134       00434                                 BEQ               CRcvTime1                     ; Bus timed out
00302
00302                               ; make ourselves the listener
00302   1001                                                  MOVE.B            D1,D0                         ; our address
00304   0000 0020                                             ORI.B             #mla,D0
00308   6100 1A50       01D5A                                 BSR               DataOut
0030C   6100 1A1A       01D28                                 BSR               WaitOut                       ; wait for byte out
00310   6700 0122       00434                                 BEQ               CRcvTime1                     ; Bus timed out
00314
00314                               ; holdoff all data
00314                                                         MWrite            gpibauxcmd,hdfa
00314   48E7 8080             1                               MOVEM.L           D0/A0,-(SP)                   ; save work registers
00318                         1
00318   2009                  1                               MOVE.L            A1,D0                         ; from board base address
0031A G 0680 0002 0018        1                               ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00320   2040                  1                               MOVEA.L           D0,A0                         ; A0 has address
00322   103C 0083             1                               MOVE.B            #hdfa,D0                      ; set data
00326   6100 1B2A       01E52 1                               BSR               NbWrite
0032A                         1
0032A   4CDF 0101             1                               MOVEM.L           (SP)+,D0/A0                   ; restore registers
0032E
0032E                               ; listen only
0032E                                                         MWrite            gpibauxcmd,lon
0032E   48E7 8080             1                               MOVEM.L           D0/A0,-(SP)                   ; save work registers
00332                         1
00332   2009                  1                               MOVE.L            A1,D0                         ; from board base address
00334 G 0680 0002 0018        1                               ADD.L             #gpibauxcmd,D0                ; add to where byte will go
0033A   2040                  1                               MOVEA.L           D0,A0                         ; A0 has address
0033C   103C 0089             1                               MOVE.B            #lon,D0                       ; set data
00340   6100 1B10       01E52 1                               BSR               NbWrite
00344                         1
00344   4CDF 0101             1                               MOVEM.L           (SP)+,D0/A0                   ; restore registers
00348
00348                               ; go to standby
00348                                                         MWrite            gpibauxcmd,gts
00348   48E7 8080             1                               MOVEM.L           D0/A0,-(SP)                   ; save work registers
0034C                         1
0034C   2009                  1                               MOVE.L            A1,D0                         ; from board base address
0034E G 0680 0002 0018        1                               ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00354   2040                  1                               MOVEA.L           D0,A0                         ; A0 has address
00356   103C 000B             1                               MOVE.B            #gts,D0                       ; set data
0035A   6100 1AF6       01E52 1                               BSR               NbWrite
0035E                         1
0035E   4CDF 0101             1                               MOVEM.L           (SP)+,D0/A0                   ; restore registers
00362
00362   6100 1A56       01DBA                                 BSR               GetGpibTimot
00366   2200                                                  MOVE.L            D0,D1                         ; D1 = timeout loop count
00368
00368                               ; now start getting the data
00368   2809                                                  MOVE.L            A1,D4
0036A G 0684 0002 001C                                        ADD.L             #gpibdatain,D4
00370   2044                                                  MOVEA.L           D4,A0                         ; A0 = 9914 'Data In' register address
00372
00372   2038 0001                                             MOVE.L            true32b,D0                    ; set 32-bit mode
00376   A05D                                                  _SwapMMUMode
00378
00378   2C01                        CRcv1                     MOVE.L            D1,D6                         ; D6 = loop timeout pass count
0037A
0037A G 5386                        CRcv11                    SUBI.L            #1,D6                         ; decrement pass count
0037C   6700 00B0       0042E                                 BEQ               CRcvTime                      ; if bus not responding
00380   1014                                                  MOVE.B            (A4),D0                       ; get interrupt 0 status
00382   0200 0028                                             ANDI.B            #eoimk+bim,D0                 ; check for EOI or BI
00386   67F2            0037A                                 BEQ.S             CRcv11                        ; wait until set
00388
00388   0200 0008                                             ANDI.B            #eoimk,D0                     ; check for EOI
0038C   661A            003A8                                 BNE.S             CRcv2                         ; if EOI
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page   8
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0038E
0038E   1010                                          MOVE.B      (A0),D0                 ; if not EOI get data from GPIB
00390   16C0                                          MOVE.B      D0,(A3)+                ; store data away
00392 G 5283                                          ADD.L       #1,D3                   ; chalk up another character
00394
00394 G 0C47 0000                                     CMP.W       #0,D7                   ; should we check for EOS?
00398   6704            0039E                         BEQ.S       CRcv4                   ; NO...
0039A   B002                                          CMP.B       D2,D0                   ; YES, was it an 'eos' character
0039C   6720            003BE                         BEQ.S       CRcv3                   ; yes, finish up
0039E
0039E   BA83                        CRcv4             CMP.L       D3,D5                   ; max input buffer size hit ?
003A0   672A            003CC                         BEQ.S       CRcv31                  ; yes, finish up
003A2   1ABC 0002                                     MOVE.B            #rhdf,(A5)        ; release holdoff
003A6   60D0            00378                         BRA.S       CRcv1                   ; get more data
003A8
003A8                               ; byte had EOI with it
003A8   2038 0000                   CRcv2             MOVE.L      false32b,D0
003AC   A05D                                          _SwapMMUMode                        ; back to 24-bit mode
003AE   6100 19C2       01D72                         BSR         DataIn                  ; get data from GPIB
003B2   16C0                                          MOVE.B      D0,(A3)+                ; store data away
003B4 G 5283                                          ADD.L       #1,D3                   ; chalk up another character
003B6   006A 2000 0004                                ORI.W       #stEnd,csStatus(A2)     ; flag EOI received
003BC   601A            003D8                         BRA.S       CRcv5
003BE
003BE                               ; byte was EOS character
003BE   2038 0000                   CRcv3             MOVE.L      false32b,D0
003C2   A05D                                          _SwapMMUMode                        ; back to 24-bit mode
003C4   006A 2000 0004                                ORI.W       #stEnd,csStatus(A2)     ; flag EOS received
003CA   600C            003D8                         BRA.S       CRcv5
003CC
003CC                               ;max buffer size hit
003CC   2038 0000                   CRcv31            MOVE.L      false32b,D0
003D0   A05D                                          _SwapMMUMode                        ; back to 24-bit mode
003D2   006A 0200 0004                                ORI.W       #stCnt,csStatus(A2)     ; flag max buffer size reached
003D8
003D8                               ; finish up transaction
003D8                               CRcv5             MWrite      gpibauxcmd,tcs          ; take control synchronously
003D8   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
003DC                         1
003DC   2009                  1                       MOVE.L      A1,D0                   ; from board base address
003DE G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
003E4   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
003E6   103C 000D             1                       MOVE.B      #tcs,D0                 ; set data
003EA   6100 1A66       01E52 1                       BSR         NbWrite
003EE                         1
003EE   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
003F2   6100 1934       01D28                         BSR         WaitOut                 ; wait for byte out
003F6   673C            00434                         BEQ.S       CRcvTime1               ; Bus timed out
003F8                                                 MWrite      gpibauxcmd,rhdf         ; release holdoff
003F8   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
003FC                         1
003FC   2009                  1                       MOVE.L      A1,D0                   ; from board base address
003FE G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
00404   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
00406   103C 0002             1                       MOVE.B      #rhdf,D0                ; set data
0040A   6100 1A46       01E52 1                       BSR         NbWrite
0040E                         1
0040E   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
00412                                                 MWrite      gpibauxcmd,hdaclr       ; release holdoff on all
00412   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
00416                         1
00416   2009                  1                       MOVE.L      A1,D0                   ; from board base address
00418 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
0041E   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
00420   103C 0003             1                       MOVE.B      #hdaclr,D0              ; set data
00424   6100 1A2C       01E52 1                       BSR         NbWrite
00428                         1
00428   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
0042C   6040            0046E                         BRA.S       CRcvGood                ; return
0042E
0042E                               ; here if GPIB bus not responding
0042E   2038 0000                   CRcvTime          MOVE.L      false32b,D0
00432   A05D                                          _SwapMMUMode                        ; back to 24-bit mode
00434   7081                        CRcvTime1         MOVEQ       #gpibErr,D0             ; return error to O.S.
00436   357C 0001 0006                                MOVE.W      #ctlTime,csError(A2)    ; return error to application
0043C   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)     ; flag error
00442   006A 4000 0004                                ORI.W       #stTime,csStatus(A2)    ; flag timeout
00448   6030            0047A                         BRA.S       CRcvDone                ; and return
0044A
0044A                               ; here if interface not controller in charge
0044A   7081                        CRcvNchg          MOVEQ       #gpibErr,D0             ; return error to O.S.
0044C   357C 0004 0006                                MOVE.W      #ctlNinChg,csError(A2)  ; return error to application
00452 G 426A 0004                                     MOVE.W      #stGood,csStatus(A2)    ; Default status
00456   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)     ; flag error
0045C   601C            0047A                         BRA.S       CRcvDone                ; and return
0045E
0045E
0045E   7081                        CRcvBadAddr       MOVEQ       #gpibErr,D0             ; return error to O.S.
00460   357C 0002 0006                                MOVE.W      #ctlBaddr,csError(A2)   ; return error to application
00466   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)     ; flag error
0046C   600C            0047A                         BRA.S       CRcvDone                ; and return
0046E
0046E   7000                        CRcvGood          MOVEQ       #noErr,D0               ; return no error
00470 G 426A 0006                                     MOVE.W      #ctlNoErr,csError(A2)
00474   006A 0100 0004                                ORI.W       #stCmplt,csStatus(A2)   ; flag call complete
0047A
0047A   2543 0008                   CRcvDone          MOVE.L      D3,csCount(A2)          ; return # characters received
```

```
MC680xx Assembler - Ver 3.2b6                                                             18-May-91  Page   9
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0047E                                                 MSetCIC                                   ; set up status CIC bit
0047E                         1
0047E   6100 1992       01E12 1                       BSR         AmIncharge                    ; are we the controller in charge?
00482   6608            0048C 1                       BNE.S       @StCIC1                       ; no ...
00484   006A 0020 0004        1                       ORI.W       #stCic,csStatus(A2)           ; flag CIC
0048A   6006            00492 1                       BRA.S       @StCIC2
0048C                         1
0048C   026A FFDF 0004        1     @StCIC1           ANDI.W      #stNCic,csStatus(A2)          ; flag /CIC
00492                         1
00492   4E71                  1     @StCIC2           NOP
00494                         1
00494   4CDF 3FFF                                     MOVEM.L     (SP)+,A0-A5/D0-D7             ; restore local work registers
00498   4CDF 1110                                     MOVEM.L     (SP)+,A0/A4/D4                ; restore registers
0049C   6000 FDD2       00270                         BRA         ExitDrvr
004A0
004A0
004A0
004A0
004A0
004A0
004A0                               *******************************************************************************************
004A0                               *     CSend(params: GpibCtlBlkPtr); routine.  This routine is called when the driver Control
004A0                               *     function is invoked and the request was for a controller send data operation.
004A0                               *
004A0                               *
004A0                               *     Entry:      A0    - param blk pointer
004A0                               *                 A1    - DCE pointer
004A0                               *                 A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
004A0                               *
004A0                               *     Uses:
004A0                               *                 A0    - temporary, 9914 'Data Out' register address
004A0                               *                 A1    - Board base address
004A0                               *                 A2    - Pointer to 'params'
004A0                               *                 A3    - Buffer pointer
004A0                               *                 A4    - 9914 int0 register address
004A0                               *                 A5    - 9914 auxcmd register address
004A0                               *
004A0                               *                 D0    - temporary
004A0                               *                 D1    - timeout loop count
004A0                               *                 D2    - eos character
004A0                               *                 D3    - actual character count
004A0                               *                 D4    - temporary
004A0                               *                 D5    - # characters to transmit
004A0                               *                 D6    - temporary
004A0                               *                 D7    - EOS valid BOOLEAN
004A0                               *
004A0                               *     Exit:       D0    - error code
004A0                               *
004A0                               *******************************************************************************************
004A0                               CSend
004A0                               ; setup working registers
004A0   48E7 FFFC                                     MOVEM.L     A0-A5/D0-D7,-(SP)       ; save local work registers
004A4
004A4   1029 0028                                     MOVE.B      dCtlSlot(A1),D0         ; get the slot address
004A8   E188                                          LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
004AA   E188                                          LSL.L       #8,D0
004AC   E188                                          LSL.L       #8,D0
004AE   0080 F000 0003                                ORI.L       #$f0000003,D0           ; Slot space
004B4   2240                                          MOVEA.L     D0,A1                   ; A1 = board base address
004B6
004B6 G 426A 0004                                     MOVE.W      #stGood,csStatus(A2)    ; Default status, rest of bits set later
004BA
004BA   266A 000C                                     MOVEA.L     csDataBuf(A2),A3        ; A3 = pointer to data buffer
004BE   2009                                          MOVE.L      A1,D0
004C0 G 0680 0002 0000                                ADD.L       #gpibint0,D0
004C6   2840                                          MOVEA.L     D0,A4                   ; A4 = 9914 int0 register address
004C8
004C8   6100 18C0       01D8A                         BSR         GetEos
004CC   1400                                          MOVE.B      D0,D2                   ; D2 = end-of-string character
004CE G 7600                                          MOVE.L      #0,D3                   ; clear actual character count
004D0   2A2A 0008                                     MOVE.L      csCount(A2),D5          ; D5 = char count
004D4   6100 18E4       01DBA                         BSR         GetGpibTimot
004D8   2200                                          MOVE.L      D0,D1                   ; D1 = timeout loop count
004DA   3E12                                          MOVE.W      csVar(A2),D7            ; D7 = EOS Valid BOOLEAN
004DC
004DC   6100 1934       01E12                         BSR         AmIncharge              ; are we the controller in charge?
004E0   6600 014C       0062E                         BNE         CSendNchg               ; no ...
004E4
004E4                               ; Send our address as the talker
004E4   6100 18BC       01DA2                         BSR         GetGpibAddr             ; D0 = our address
004E8   0000 0040                                     ORI.B       #mta,D0                 ; send MTA to disable previous talkers
004EC   6100 186C       01D5A                         BSR         DataOut
004F0   6100 1836       01D28                         BSR         WaitOut                 ; wait for GPIB bus free
004F4   6700 0122       00618                         BEQ         CSendTime1              ; Bus timed out
004F8   103C 003F                                     MOVE.B      #unl,D0                 ; send universal unlisten
004FC   6100 185C       01D5A                         BSR         DataOut
00500
00500                               ; get the listener list
00500   206A 0010                                     MOVEA.L     csAddrList(A2),A0       ; get pointer to listener list
00504
00504                               ; Select all designated listeners
00504   1018                        CSend1            MOVE.B      (A0)+,D0                ; get listener
00506 G 0C40 0020                                     CMP.W       #$20,D0
0050A   6D14            00520                         BLT.S       CSend2                  ; listener address too lo
0050C G 0C40 003E                                     CMP.W       #$3e,D0
00510   6E0E            00520                         BGT.S       CSend2                  ; listener address too hi
00512   6100 1814       01D28                         BSR         WaitOut                 ; wait for byte out
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  10
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00516   6700 0100       00618                         BEQ         CSendTime1              ; Bus timed out
0051A   6100 183E       01D5A                         BSR         DataOut                 ; send over GPIB
0051E   60E4            00504                         BRA.S       CSend1                  ; next listener
00520
00520                               ; Set talk only mode
00520                               CSend2            MWrite      gpibauxcmd,ton          ; talk only
00520   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
00524                         1
00524   2009                  1                       MOVE.L      A1,D0                   ; from board base address
00526 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
0052C   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
0052E   103C 008A             1                       MOVE.B      #ton,D0                 ; set data
00532   6100 191E       01E52 1                       BSR         NbWrite
00536                         1
00536   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
0053A
0053A                               ; go to standby
0053A                                                 MWrite      gpibauxcmd,gts
0053A   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
0053E                         1
0053E   2009                  1                       MOVE.L      A1,D0                   ; from board base address
00540 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
00546   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
00548   103C 000B             1                       MOVE.B      #gts,D0                 ; set data
0054C   6100 1904       01E52 1                       BSR         NbWrite
00550                         1
00550   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
00554   6100 17D2       01D28                         BSR         WaitOut                 ; wait for byte out
00558   6700 00BE       00618                         BEQ         CSendTime1              ; Bus timed out
0055C
0055C G 0C85 0000 0000                                CMP.L       #0,D5                   ; Byte count zero?
00562   660A            0056E                         BNE.S       CSend8
00564   006A 0200 0004                                ORI.W       #stCnt,csStatus(A2)     ; yes, flag call count reached
0056A   6000 0084       005F0                         BRA         CSend5
0056E
0056E   2009                        CSend8            MOVE.L      A1,D0
00570 G 0680 0002 001C                                ADD.L       #gpibdataout,D0
00576   2040                                          MOVEA.L     D0,A0                   ; A0 has 9914 'Data Out' register address
00578
00578   2038 0001                                     MOVE.L      true32b,D0              ; set 32-bit mode
0057C   A05D                                          _SwapMMUMode
0057E
0057E                               ; now loop, sending the data
0057E   2C01                        CSend3            MOVE.L      D1,D6                   ; D6 = loop timeout pass count
00580   101B                                          MOVE.B      (A3)+,D0                ; get the data byte
00582 G 5385                                          SUBI.L      #1,D5                   ; decrement byte count
00584   671C            005A2                         BEQ.S       CSend4                  ; last byte to be output
00586 G 0C47 0000                                     CMP.W       #0,D7                   ; check for EOS Valid?
0058A   6704            00590                         BEQ.S       CSend6                  ; NO...
0058C   B400                                          CMP.B       D0,D2                   ; YES, is byte the EOS char?
0058E   671A            005AA                         BEQ.S       CSend41                 ; yes...
00590   1080                        CSend6            MOVE.B      D0,(A0)                 ; send data byte
00592 G 5283                                          ADDI.L      #1,D3                   ; Chalk up another byte sent
00594 G 5386                        CSend7            SUBI.L      #1,D6                   ; decrement pass count
00596   677A            00612                         BEQ.S       CSendTime               ; if bus not responding
00598   1014                                          MOVE.B      (A4),D0                 ; get interrupt 0 status
0059A   0200 0010                                     ANDI.B      #BOM,D0                 ; check for BO
0059E   67F4            00594                         BEQ.S       CSend7                  ; wait for GPIB bus free
005A0   60DC            0057E                         BRA.S       CSend3                  ; next byte
005A2
005A2                               ; last data byte to send because of byte count reached
005A2   006A 0200 0004              CSend4            ORI.W       #stCnt,csStatus(A2)     ; flag call count reached
005A8   6006            005B0                         BRA.S       CSend42
005AA
005AA                               ; last data byte to send because of EOS detected in data stream
005AA   006A 2000 0004              CSend41           ORI.W       #stEnd,csStatus(A2)     ; flag EOI sent
005B0   1200                        CSend42           MOVE.B      D0,D1                   ; save byte temporarily
005B2   2038 0000                                     MOVE.L      false32b,D0
005B6   A05D                                          _SwapMMUMode                        ; back to 24-bit mode
005B8
005B8   3E2A 0002                                     MOVE.W      csFlag(A2),D7           ; D7 = send EOI BOOLEAN
005BC G 0C47 0000                                     CMP.W       #0,D7                   ; send EOI ?
005C0   6720            005E2                         BEQ.S       CSend9                  ; NO...
005C2
005C2                                                 MWrite      gpibauxcmd,feoi         ; send EOI with last character
005C2   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
005C6                         1
005C6   2009                  1                       MOVE.L      A1,D0                   ; from board base address
005C8 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0          ; add to where byte will go
005CE   2040                  1                       MOVEA.L     D0,A0                   ; A0 has address
005D0   103C 0008             1                       MOVE.B      #feoi,D0                ; set data
005D4   6100 187C       01E52 1                       BSR         NbWrite
005D8                         1
005D8   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0             ; restore registers
005DC   006A 2000 0004                                ORI.W       #stEnd,csStatus(A2)     ; flag EOI sent
005E2
005E2   1001                        CSend9            MOVE.B      D1,D0                   ; restore data byte
005E4   6100 1774       01D5A                         BSR         DataOut                 ; send data byte
005E8 G 5283                                          ADDI.L      #1,D3                   ; Chalk up another byte sent
005EA   6100 173C       01D28                         BSR         WaitOut                 ; wait for GPIB bus free
005EE   6728            00618                         BEQ.S       CSendTime1              ; Bus timed out
005F0
005F0                               ; take control synchronously
005F0                               CSend5            MWrite      gpibauxcmd,tca          ; take control synchronously
005F0   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)             ; save work registers
005F4                         1
005F4   2009                  1                       MOVE.L      A1,D0                   ; from board base address
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  11
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

005F6 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0                ; add to where byte will go
005FC   2040                  1                       MOVEA.L     D0,A0                         ; A0 has address
005FE   103C 000C             1                       MOVE.B      #tca,D0                       ; set data
00602   6100 184E       01E52 1                       BSR         NbWrite
00606                         1
00606   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0                   ; restore registers
0060A   6100 171C       01D28                         BSR         WaitOut                       ; wait for GPIB bus free
0060E   6708            00618                         BEQ.S       CSendTime1                    ; Bus timed out
00610   603C            0064E                         BRA.S       CSendGood
00612
00612                               ; here if GPIB bus not responding
00612   2038 0000                   CSendTime         MOVE.L      false32b,D0
00616   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
00618   7081                        CSendTime1        MOVEQ       #gpibErr,D0                   ; return error to O.S.
0061A   357C 0001 0006                                MOVE.W      #ctlTime,csError(A2)          ; return error to application
00620   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
00626   006A 4000 0004                                ORI.W       #stTime,csStatus(A2)          ; flag timeout
0062C   602C            0065A                         BRA.S       CSendDone                     ; and return
0062E
0062E                               ; here if interface not controller in charge
0062E   7081                        CSendNchg         MOVEQ       #gpibErr,D0                   ; return error to O.S.
00630   357C 0004 0006                                MOVE.W      #ctlNinChg,csError(A2)        ; return error to application
00636   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
0063C   601C            0065A                         BRA.S       CSendDone                     ; and return
0063E
0063E   7081                        CSendBad          MOVEQ       #gpibErr,D0                   ; return error to O.S.
00640   357C 0003 0006                                MOVE.W      #ctlUnkErr,csError(A2)        ; return error to application
00646   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
0064C   600C            0065A                         BRA.S       CSendDone                     ; and return
0064E
0064E   7000                        CSendGood         MOVEQ       #noErr,D0                     ; return no error
00650 G 426A 0006                                     MOVE.W      #ctlNoErr,csError(A2)
00654   006A 0100 0004                                ORI.W       #stCmplt,csStatus(A2)         ; flag call complete
0065A
0065A   2543 0008                   CSendDone         MOVE.L      D3,csCount(A2)                ; return # characters sent
0065E                                                 MSetCIC                                         ; set up status CIC bit
0065E                         1
0065E   6100 17B2       01E12 1                       BSR         AmIncharge                    ; are we the controller in charge?
00662   6608            0066C 1                       BNE.S       @StCIC1                       ; no ...
00664   006A 0020 0004        1                       ORI.W       #stCic,csStatus(A2)           ; flag CIC
0066A   6006            00672 1                       BRA.S       @StCIC2
0066C                         1
0066C   026A FFDF 0004        1     @StCIC1           ANDI.W      #stNCic,csStatus(A2)          ; flag /CIC
00672                         1
00672   4E71                  1     @StCIC2           NOP
00674                         1
00674   4CDF 3FFF                                     MOVEM.L     (SP)+,A0-A5/D0-D7             ; restore local work registers
00678   4CDF 1110                                     MOVEM.L     (SP)+,A0/A4/D4                ; restore registers
0067C   6000 FBF2       00270                         BRA         ExitDrvr
00680
00680
00680
00680
00680
00680
00680
00680
00680
00680                               ********************************************************************************************
00680                               *     SendCmd(params: GpibCtlBlkPtr); routine.  This routine is called when the driver Control
00680                               *     function is invoked and the request was for the controller to send a command string.
00680                               *
00680                               *
00680                               *     Entry:      A0    - param blk pointer
00680                               *                 A1    - DCE pointer
00680                               *                 A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00680                               *
00680                               *     Uses:
00680                               *                 A0    - temporary, 9914 'Data Out' register address
00680                               *                 A1    - Board base address
00680                               *                 A2    - Pointer to 'params'
00680                               *                 A3    - Buffer pointer
00680                               *                 A4    - 9914 int0 register address
00680                               *
00680                               *                 D0    - temporary
00680                               *                 D1    - timeout loop count
00680                               *                 D3    - actual character count
00680                               *                 D4    - temporary
00680                               *                 D5    - # characters to transmit
00680                               *                 D6    - temporary
00680                               *
00680                               *     Exit:       D0    - error code
00680                               *
00680                               ********************************************************************************************
00680                               SendCmd
00680                               ; setup working registers
00680   48E7 FFFC                                     MOVEM.L     A0-A5/D0-D7,-(SP)             ; save local work registers
00684
00684   1029 0028                                     MOVE.B      dCtlSlot(A1),D0               ; get the slot address
00688   E188                                          LSL.L       #8,D0                         ; shift the 4 slot bits into proper position
0068A   E188                                          LSL.L       #8,D0
0068C   E188                                          LSL.L       #8,D0
0068E   0080 F000 0003                                ORI.L       #$f0000003,D0                 ; Slot space
00694   2240                                          MOVEA.L     D0,A1                         ; A1 = board base address
00696
00696 G 426A 0004                                     MOVE.W      #stGood,csStatus(A2)          ; Default status
0069A G 7600                                          MOVE.L      #0,D3                         ; clear actual character count
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  12
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0069C
0069C   6100 1774       01E12                         BSR         AmIncharge                    ; are we the controller in charge?
006A0   6600 0080       00722                         BNE         CSndCmdNchg                   ; no ...
006A4
006A4   266A 000C                                     MOVEA.L     csDataBuf(A2),A3              ; A3 = pointer to data buffer
006A8   2009                                          MOVE.L      A1,D0
006AA G 0680 0002 0000                                ADD.L       #gpibint0,D0
006B0   2840                                          MOVEA.L     D0,A4                         ; A4 has 9914 int0 register address
006B2   2009                                          MOVE.L      A1,D0
006B4 G 0680 0002 001C                                ADD.L       #gpibdataout,D0
006BA   2040                                          MOVEA.L     D0,A0                         ; A0 has 9914 'Data Out' register address
006BC
006BC   2A2A 0008                                     MOVE.L      csCount(A2),D5                ; D5 = char count
006C0   6100 16F8       01DBA                         BSR         GetGpibTimot
006C4   2200                                          MOVE.L      D0,D1                         ; D1 = timeout loop count
006C6
006C6 G 0C85 0000 0000                                CMP.L       #0,D5                         ; Byte count zero?
006CC   6774            00742                         BEQ.S       SndCmdGood                    ; just return
006CE
006CE   2038 0001                                     MOVE.L      true32b,D0                    ; set 32-bit mode
006D2   A05D                                          _SwapMMUMode
006D4
006D4                               ; now loop, sending the command bytes
006D4   2C01                        SndCmd3           MOVE.L      D1,D6                         ; D6 = loop timeout pass count
006D6   101B                                          MOVE.B      (A3)+,D0                      ; get the data byte
006D8 G 5385                                          SUBI.L      #1,D5                         ; decrement byte count
006DA   6712            006EE                         BEQ.S       SndCmd4                       ; last byte to be output
006DC   1080                                          MOVE.B      D0,(A0)                       ; send data byte
006DE G 5283                                          ADDI.L      #1,D3                         ; Chalk up another byte sent
006E0 G 5386                        SndCmd7           SUBI.L      #1,D6                         ; decrement pass count
006E2   6722            00706                         BEQ.S       SndCmdTime                    ; if bus not responding
006E4   1014                                          MOVE.B      (A4),D0                       ; get interrupt 0 status
006E6   0200 0010                                     ANDI.B      #BOM,D0                       ; check for BO
006EA   67F4            006E0                         BEQ.S       SndCmd7                       ; wait for GPIB bus free
006EC   60E6            006D4                         BRA.S       SndCmd3                       ; next byte
006EE
006EE                               ; last data byte to send
006EE   1080                        SndCmd4           MOVE.B      D0,(A0)                       ; send data byte
006F0 G 5283                                          ADDI.L      #1,D3                         ; Chalk up another byte sent
006F2   2038 0000                                     MOVE.L      false32b,D0
006F6   A05D                                          _SwapMMUMode                              ; back to 24-bit mode
006F8   006A 0200 0004                                ORI.W       #stCnt,csStatus(A2)           ; flag byte count hit
006FE   6100 1628       01D28                         BSR         WaitOut                       ; wait for GPIB bus free
00702   6708            0070C                         BEQ.S       SndCmdTime1                   ; Bus timed out
00704   603C            00742                         BRA.S       SndCmdGood                    ; return
00706
00706                               ; here if GPIB bus not responding
00706   2038 0000                   SndCmdTime        MOVE.L      false32b,D0
0070A   A05D                                          _SwapMMUMode                              ; back to 24-bit mode
0070C   7081                        SndCmdTime1       MOVEQ       #gpibErr,D0                   ; return error to O.S.
0070E   357C 0001 0006                                MOVE.W      #ctlTime,csError(A2)          ; return error to application
00714   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
0071A   006A 4000 0004                                ORI.W       #stTime,csStatus(A2)          ; flag timeout
00720   602C            0074E                         BRA.S       SndCmdDone                    ; and return
00722
00722                               ; here if interface not controller in charge
00722   7081                        CSndCmdNchg       MOVEQ       #gpibErr,D0                   ; return error to O.S.
00724   357C 0004 0006                                MOVE.W      #ctlNinChg,csError(A2)        ; return error to application
0072A   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
00730   601C            0074E                         BRA.S       SndCmdDone                    ; and return
00732
00732   7081                        SndCmdBad         MOVEQ       #gpibErr,D0                   ; return error to O.S.
00734   357C 0003 0006                                MOVE.W      #ctlUnkErr,csError(A2)        ; return error to application
0073A   006A 8000 0004                                ORI.W       #stErr,csStatus(A2)           ; flag error
00740   600C            0074E                         BRA.S       SndCmdDone                    ; and return
00742
00742   7000                        SndCmdGood        MOVEQ       #noErr,D0                     ; return no error
00744 G 426A 0006                                     MOVE.W      #ctlNoErr,csError(A2)
00748   006A 0100 0004                                ORI.W       #stCmplt,csStatus(A2)         ; flag call complete
0074E
0074E   2543 0008                   SndCmdDone        MOVE.L      D3,csCount(A2)                ; return # characters sent
00752                                                 MSetCIC                                   ; set up status CIC bit
00752                         1
00752   6100 16BE       01E12 1                       BSR         AmIncharge                    ; are we the controller in charge?
00756   6608            00760 1                       BNE.S       @StCIC1                       ; no ...
00758   006A 0020 0004        1                       ORI.W       #stCic,csStatus(A2)           ; flag CIC
0075E   6006            00766 1                       BRA.S       @StCIC2
00760                         1
00760   026A FFDF 0004        1     @StCIC1           ANDI.W      #stNCic,csStatus(A2)          ; flag /CIC
00766                         1
00766   4E71                  1     @StCIC2           NOP
00768                         1
00768   4CDF 3FFF                                     MOVEM.L     (SP)+,A0-A5/D0-D7             ; restore local work registers
0076C   4CDF 1110                                     MOVEM.L     (SP)+,A0/A4/D4                ; restore registers
00770   6000 FAFE       00270                         BRA         ExitDrvr
00774
00774
00774
00774
00774
00774
00774
00774
00774
00774                               ********************************************************************************************
00774                               *     ContInit - driver control routine to init 9914 as controller
00774                               *
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  13
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00774                               *     Entry:      A0 - param blk pointer
00774                               *                 A1 - DCE pointer
00774                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00774                               *
00774                               ********************************************************************************************
00774                               ContInit
00774   6148            007BE                         BSR.S       CInit                         ; init 9914 as controller
00776
00776   48E7 0040                                     MOVEM.L     A1,-(SP)                      ; save local work registers
0077A
0077A                               ; get base address of board
0077A   1029 0028                                     MOVE.B      dCtlSlot(A1),D0               ; get the slot address
0077E   E188                                          LSL.L       #8,D0                         ; shift the 4 slot bits into proper position
00780   E188                                          LSL.L       #8,D0
00782   E188                                          LSL.L       #8,D0
00784   0080 F000 0003                                ORI.L       #$f0000003,D0                 ; Slot space
0078A   2240                                          MOVEA.L     D0,A1                         ; A1 = board base address
0078C
0078C   7000                        ContGood          MOVEQ       #noErr,D0                     ; return no error to O.S.
0078E G 426A 0006                                     MOVE.W      #ctlNoErr,csError(A2)
00792 G 426A 0004                                     MOVE.W      #stGood,csStatus(A2)          ; Default status
00796   006A 0100 0004                                ORI.W       #stCmplt,csStatus(A2)         ; flag call complete
0079C                                                 MSetCIC                                   ; set up status CIC bit
0079C                         1
0079C   6100 1674       01E12 1                       BSR         AmIncharge                    ; are we the controller in charge?
007A0   6608            007AA 1                       BNE.S       @StCIC1                       ; no ...
007A2   006A 0020 0004        1                       ORI.W       #stCic,csStatus(A2)           ; flag CIC
007A8   6006            007B0 1                       BRA.S       @StCIC2
007AA                         1
007AA   026A FFDF 0004        1     @StCIC1           ANDI.W      #stNCic,csStatus(A2)          ; flag /CIC
007B0                         1
007B0   4E71                  1     @StCIC2           NOP
007B2                         1
007B2
007B2   4CDF 0200                   ContDone          MOVEM.L     (SP)+,A1                      ; restore local work registers
007B6   4CDF 1110                                     MOVEM.L     (SP)+,A0/A4/D4                ; restore registers
007BA   6000 FAB4       00270                         BRA         ExitDrvr
007BE
007BE
007BE
007BE                               ********************************************************************************************
007BE                               *     initialize the 9914 chip as a controller
007BE                               *
007BE                               *     Entry:      A0 - param blk pointer
007BE                               *                 A1 - DCE pointer
007BE                               *
007BE                               *     Uses:       D0 - temporary
007BE                               *                 D1 - temporary
007BE                               *
007BE                               ********************************************************************************************
007BE                               ; save registers
007BE                               CInit
007BE   48E7 C0F8                                     MOVEM.L     D0-D1/A0-A4,-(SP)             ; save work registers
007C2
007C2 G 2448                                          MOVE.L      A0,A2                         ; A2 <- param block pointer
007C4 G 2649                                          MOVE.L      A1,A3                         ; A3 <- DCE pointer
007C6
007C6                               ; get base address of board
007C6   102B 0028                                     MOVE.B      dCtlSlot(A3),D0               ; get the slot address
007CA   E188                                          LSL.L       #8,D0                         ; shift the 4 slot bits into proper position
007CC   E188                                          LSL.L       #8,D0
007CE   E188                                          LSL.L       #8,D0
007D0   0080 F000 0003                                ORI.L       #$f0000003,D0                 ; Slot space
007D6   2240                                          MOVEA.L     D0,A1                         ; A1 = board base address
007D8
007D8                               ; set flag as controller in local storage
007D8                                                 MWrite      amController,$ff
007D8   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)                   ; save work registers
007DC                         1
007DC   2009                  1                       MOVE.L      A1,D0                         ; from board base address
007DE G 0680 0000 001C        1                       ADD.L       #amController,D0              ; add to where byte will go
007E4   2040                  1                       MOVEA.L     D0,A0                         ; A0 has address
007E6   103C 00FF             1                       MOVE.B      #$ff,D0                       ; set data
007EA   6100 1666       01E52 1                       BSR         NbWrite
007EE                         1
007EE   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0                   ; restore registers
007F2
007F2                               ; issue software reset to 9914
007F2                                                 MWrite      gpibauxcmd,swrst
007F2   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)                   ; save work registers
007F6                         1
007F6   2009                  1                       MOVE.L      A1,D0                         ; from board base address
007F8 G 0680 0002 0018        1                       ADD.L       #gpibauxcmd,D0                ; add to where byte will go
007FE   2040                  1                       MOVEA.L     D0,A0                         ; A0 has address
00800   103C 0080             1                       MOVE.B      #swrst,D0                     ; set data
00804   6100 164C       01E52 1                       BSR         NbWrite
00808                         1
00808   4CDF 0101             1                       MOVEM.L     (SP)+,D0/A0                   ; restore registers
0080C
0080C                               ; disable all interrupt mask bits
0080C                                                 MWrite      gpibintm0,0                   ; reset int mask 0 register
0080C   48E7 8080             1                       MOVEM.L     D0/A0,-(SP)                   ; save work registers
00810                         1
00810   2009                  1                       MOVE.L      A1,D0                         ; from board base address
00812 G 0680 0002 0000        1                       ADD.L       #gpibintm0,D0                 ; add to where byte will go
00818   2040                  1                       MOVEA.L     D0,A0                         ; A0 has address
0081A G 4200                  1                       MOVE.B      #0,D0                         ; set data
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  14
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0081C   6100 1634       01E52 1                       BSR               NbWrite
00820                         1
00820   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
00824                                                 MWrite            gpibintm1,0             ; reset int mask 1 register
00824   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
00828                         1
00828   2009                  1                       MOVE.L            A1,D0                   ; from board base address
0082A G 0680 0002 0010        1                       ADD.L             #gpibintm1,D0           ; add to where byte will go
00830   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
00832 G 4200                  1                       MOVE.B            #0,D0                   ; set data
00834   6100 161C       01E52 1                       BSR               NbWrite
00838                         1
00838   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
0083C
0083C                               ; write address to 9914
0083C   2009                                          MOVE.L            A1,D0                   ; from board base address
0083E G 0680 0000 0014                                ADD.L             #gpibAddrSt,D0          ; add to where address is stored
00844   2040                                          MOVEA.L           D0,A0                   ; A0 has address
00846   6100 15F0       01E38                         BSR               NbRead
0084A   1200                                          MOVE.B            D0,D1                   ; save address in D1
0084C   2009                                          MOVE.L            A1,D0                   ; from board base address
0084E G 0680 0002 0004                                ADD.L             #gpibaddr,D0            ; add to where byte will go
00854   2040                                          MOVEA.L           D0,A0                   ; A0 has address
00856   1001                                          MOVE.B            D1,D0                   ; get address
00858   6100 15F8       01E52                         BSR               NbWrite
0085C
0085C                               ; Set fast T1 mode
0085C                                                 MWrite            gpibauxcmd,vstdl        ; Very short T1 delay
0085C   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
00860                         1
00860   2009                  1                       MOVE.L            A1,D0                   ; from board base address
00862 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0          ; add to where byte will go
00868   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
0086A   103C 0017             1                       MOVE.B            #vstdl,D0               ; set data
0086E   6100 15E2       01E52 1                       BSR               NbWrite
00872                         1
00872   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
00876
00876                               ; set the driver buffers to 3-state control and select 'system controller'
00876                                                 MWrite            swaddr,$06              ; write to configuration register
00876   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
0087A                         1
0087A   2009                  1                       MOVE.L            A1,D0                   ; from board base address
0087C G 0680 0008 0000        1                       ADD.L             #swaddr,D0              ; add to where byte will go
00882   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
00884   103C 0006             1                       MOVE.B            #$06,D0                 ; set data
00888   6100 15C8       01E52 1                       BSR               NbWrite
0088C                         1
0088C   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
00890                                                 MWrite            swImage,$06             ; store memory image of configuration register
00890   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
00894                         1
00894   2009                  1                       MOVE.L            A1,D0                   ; from board base address
00896 G 0680 0000 0030        1                       ADD.L             #swImage,D0             ; add to where byte will go
0089C   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
0089E   103C 0006             1                       MOVE.B            #$06,D0                 ; set data
008A2   6100 15AE       01E52 1                       BSR               NbWrite
008A6                         1
008A6   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
008AA
008AA                               ; clear software reset to 9914
008AA                                                 MWrite            gpibauxcmd,swrstclr
008AA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
008AE                         1
008AE   2009                  1                       MOVE.L            A1,D0                   ; from board base address
008B0 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0          ; add to where byte will go
008B6   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
008B8 G 4200                  1                       MOVE.B            #swrstclr,D0            ; set data
008BA   6100 1596       01E52 1                       BSR               NbWrite
008BE                         1
008BE   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
008C2
008C2                               ; send IFC and take control
008C2                                                 MWrite            gpibauxcmd,sic          ; send interface clear cmd
008C2   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
008C6                         1
008C6   2009                  1                       MOVE.L            A1,D0                   ; from board base address
008C8 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0          ; add to where byte will go
008CE   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
008D0   103C 008F             1                       MOVE.B            #sic,D0                 ; set data
008D4   6100 157C       01E52 1                       BSR               NbWrite
008D8                         1
008D8   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
008DC                                                                                           ; delay a bit
008DC   3038 0D00                                     MOVE.W            TimeDBRA,D0             ; # iterations per millisecond
008E0 G 51C8 FFFE       008E0       CInit1            DBRA              D0,CInit1               ; wait
008E4
008E4                                                 MWrite            gpibauxcmd,siclr        ; reset interface clear cmd
008E4   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
008E8                         1
008E8   2009                  1                       MOVE.L            A1,D0                   ; from board base address
008EA G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0          ; add to where byte will go
008F0   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
008F2   103C 000F             1                       MOVE.B            #siclr,D0               ; set data
008F6   6100 155A       01E52 1                       BSR               NbWrite
008FA                         1
008FA   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  15
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

008FE
008FE                               ; turn on 'rem'
008FE                                                 MWrite            gpibauxcmd,sre
008FE   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)             ; save work registers
00902                         1
00902   2009                  1                       MOVE.L            A1,D0                   ; from board base address
00904 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0          ; add to where byte will go
0090A   2040                  1                       MOVEA.L           D0,A0                   ; A0 has address
0090C   103C 0090             1                       MOVE.B            #sre,D0                 ; set data
00910   6100 1540       01E52 1                       BSR               NbWrite
00914                         1
00914   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0             ; restore registers
00918
00918   4CDF 1F03                   CInit2            MOVEM.L           (SP)+,D0-D1/A0-A4       ; restore registers
0091C   4E75                                          RTS
0091E
0091E
0091E
0091E
0091E
0091E
0091E                               ********************************************************************************************
0091E                               *     Trig - control call to send 'group execute trigger' to the listener list
0091E                               *
0091E                               *     Entry:      A0 - param blk pointer
0091E                               *                 A1 - DCE pointer
0091E                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0091E                               *
0091E                               ********************************************************************************************
0091E                               Trig
0091E   48E7 0050                                     MOVEM.L           A1/A3,-(SP)                   ; save local work registers
00922
00922                               ; get base address of board
00922   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00926   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00928   E188                                          LSL.L             #8,D0
0092A   E188                                          LSL.L             #8,D0
0092C   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00932   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00934
00934   6100 14DC       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00938   6668            009A2                         BNE.S             TrigNchg                      ; no ...
0093A
0093A                               ; send 'universal unlisten' command over bus
0093A                                                 MWrite            gpibdataout,unl               ; send 'universal unlisten'
0093A   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
0093E                         1
0093E   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00940 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00946   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00948   103C 003F             1                       MOVE.B            #unl,D0                       ; set data
0094C   6100 1504       01E52 1                       BSR               NbWrite
00950                         1
00950   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00954
00954                               ; get pointer to listener list
00954   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to listener list
00958
00958                               ; loop sending listeners
00958   101B                        Trig1             MOVE.B            (A3)+,D0                      ; get the next listener
0095A G 0C00 0020                                     CMP.B             #$20,D0
0095E   6D12            00972                         BLT.S             Trig2
00960 G 0C00 003E                                     CMP.B             #$3e,D0
00964   6E0C            00972                         BGT.S             Trig2
00966   6100 13C0       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
0096A   671C            00988                         BEQ.S             Trig3                         ; Bus timed out
0096C   6100 13EC       01D5A                         BSR               DataOut                       ; send listener over GPIB
00970   60E6            00958                         BRA.S             Trig1                         ; until done
00972
00972                               ; send 'GET' command over bus
00972   6100 13B4       01D28       Trig2             BSR               WaitOut                       ; wait for GPIB bus free
00976   6710            00988                         BEQ.S             Trig3                         ; Bus timed out
00978   103C 0008                                     MOVE.B            #get,D0                       ; 'group execute trigger' command
0097C   6100 13DC       01D5A                         BSR               DataOut
00980   6100 13A6       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00984   6702            00988                         BEQ.S             Trig3                         ; Bus timed out
00986   602E            009B6                         BRA.S             TrigGood                      ; and return...
00988
00988                               ; bus timed out
00988   7081                        Trig3             MOVEQ             #gpibErr,D0                   ; return error to O.S.
0098A   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
00990 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00994   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0099A   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
009A0   6024            009C6                         BRA.S             TrigDone
009A2
009A2                               ; here if interface not controller in charge
009A2   7081                        TrigNchg          MOVEQ             #gpibErr,D0                   ; return error to O.S.
009A4   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
009AA G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
009AE   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
009B4   6010            009C6                         BRA.S             TrigDone                      ; and return
009B6
009B6   7000                        TrigGood          MOVEQ             #noErr,D0                     ; return no error
009B8 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
009BC G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
009C0   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  16
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

009C6
009C6                               TrigDone          MSetCIC                                 ; set up status CIC bit
009C6                         1
009C6   6100 144A       01E12 1                       BSR               AmIncharge            ; are we the controller in charge?
009CA   6608            009D4 1                       BNE.S             @StCIC1               ; no ...
009CC   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)   ; flag CIC
009D2   6006            009DA 1                       BRA.S             @StCIC2
009D4                         1
009D4   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)  ; flag /CIC
009DA                         1
009DA   4E71                  1     @StCIC2           NOP
009DC                         1
009DC   4CDF 0A00                                     MOVEM.L           (SP)+,A1/A3           ; restore local registers
009E0   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4        ; restore registers
009E4   6000 F88A       00270                         BRA               ExitDrvr
009E8
009E8
009E8
009E8
009E8                               ********************************************************************************************
009E8                               *     DevClr - control call to send 'device clear' to the listener list
009E8                               *
009E8                               *     Entry:      A0 - param blk pointer
009E8                               *                 A1 - DCE pointer
009E8                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
009E8                               *
009E8                               ********************************************************************************************
009E8                               DevClr
009E8   48E7 0050                                     MOVEM.L           A1/A3,-(SP)           ; save local work registers
009EC
009EC                               ; get base address of board
009EC   1029 0028                                     MOVE.B            dCtlSlot(A1),D0       ; get the slot address
009F0   E188                                          LSL.L             #8,D0                 ; shift the 4 slot bits into proper position
009F2   E188                                          LSL.L             #8,D0
009F4   E188                                          LSL.L             #8,D0
009F6   0080 F000 0003                                ORI.L             #$f0000003,D0         ; Slot space
009FC   2240                                          MOVEA.L           D0,A1                 ; A1 = board base address
009FE
009FE   6100 1412       01E12                         BSR               AmIncharge            ; are we the controller in charge?
00A02   6668            00A6C                         BNE.S             DevClrNchg            ; no ...
00A04
00A04                               ; send 'universal unlisten' command over bus
00A04                                                 MWrite            gpibdataout,unl       ; send 'universal unlisten'
00A04   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)           ; save work registers
00A08                         1
00A08   2009                  1                       MOVE.L            A1,D0                 ; from board base address
00A0A G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0       ; add to where byte will go
00A10   2040                  1                       MOVEA.L           D0,A0                 ; A0 has address
00A12   103C 003F             1                       MOVE.B            #unl,D0               ; set data
00A16   6100 143A       01E52 1                       BSR               NbWrite
00A1A                         1
00A1A   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0           ; restore registers
00A1E
00A1E                               ; get pointer to listener list
00A1E   266A 0010                                     MOVEA.L           csAddrList(A2),A3     ; A3 <- pointer to listener list
00A22
00A22                               ; loop sending listeners
00A22   101B                        DevClr1           MOVE.B            (A3)+,D0              ; get the next listener
00A24 G 0C00 0020                                     CMP.B             #$20,D0
00A28   6D12            00A3C                         BLT.S             DevClr2
00A2A G 0C00 003E                                     CMP.B             #$3e,D0
00A2E   6E0C            00A3C                         BGT.S             DevClr2
00A30   6100 12F6       01D28                         BSR               WaitOut               ; wait for GPIB bus free
00A34   671C            00A52                         BEQ.S             DevClr3               ; Bus timed out
00A36   6100 1322       01D5A                         BSR               DataOut               ; send listener over GPIB
00A3A   60E6            00A22                         BRA.S             DevClr1               ; until done
00A3C
00A3C                               ; send 'device clear' command over bus
00A3C   6100 12EA       01D28       DevClr2           BSR               WaitOut               ; wait for GPIB bus free
00A40   6710            00A52                         BEQ.S             DevClr3               ; Bus timed out
00A42   103C 0004                                     MOVE.B            #sdc,D0               ; 'selected device clear' command
00A46   6100 1312       01D5A                         BSR               DataOut
00A4A   6100 12DC       01D28                         BSR               WaitOut               ; wait for GPIB bus free
00A4E   6702            00A52                         BEQ.S             DevClr3               ; Bus timed out
00A50   602E            00A80                         BRA.S             DvClrGood             ; return
00A52
00A52                               ; bus timed out
00A52   7081                        DevClr3           MOVEQ             #gpibErr,D0           ; return error to O.S.
00A54   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)  ; return error to application
00A5A G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)  ; Default status
00A5E   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)   ; flag error
00A64   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)  ; flag timeout
00A6A   6024            00A90                         BRA.S             DvClrDone
00A6C
00A6C                               ; here if interface not controller in charge
00A6C   7081                        DevClrNchg        MOVEQ             #gpibErr,D0           ; return error to O.S.
00A6E   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2) ; return error to application
00A74 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)  ; Default status
00A78   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)   ; flag error
00A7E   6010            00A90                         BRA.S             DvClrDone             ; and return
00A80
00A80   7000                        DvClrGood         MOVEQ             #noErr,D0             ; return no error
00A82 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00A86 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)  ; Default status
00A8A   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2) ; flag call complete
00A90
00A90                               DvClrDone         MSetCIC                                 ; set up status CIC bit
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  17
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00A90                         1
00A90   6100 1380       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00A94   6608            00A9E 1                       BNE.S             @StCIC1                       ; no ...
00A96   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00A9C   6006            00AA4 1                       BRA.S             @StCIC2
00A9E                         1
00A9E   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00AA4                         1
00AA4   4E71                  1     @StCIC2           NOP
00AA6                         1
00AA6   4CDF 0A00                                     MOVEM.L           (SP)+,A1/A3                   ; restore local registers
00AAA   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00AAE   6000 F7C0       00270                         BRA               ExitDrvr
00AB2
00AB2
00AB2
00AB2
00AB2                               ********************************************************************************************
00AB2                               *     RemEnable - control call to command remote enable
00AB2                               *
00AB2                               *     Entry:      A0 - param blk pointer
00AB2                               *                 A1 - DCE pointer
00AB2                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00AB2                               *
00AB2                               ********************************************************************************************
00AB2                               RemEnable
00AB2   48E7 0040                                     MOVEM.L           A1,-(SP)                      ; save local work registers
00AB6
00AB6                               ; get base address of board
00AB6   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00ABA   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00ABC   E188                                          LSL.L             #8,D0
00ABE   E188                                          LSL.L             #8,D0
00AC0   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00AC6   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00AC8
00AC8   6100 1348       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00ACC   661C            00AEA                         BNE.S             RemENchg                      ; no ...
00ACE
00ACE                               ; send remote enable command
00ACE                                                 MWrite            gpibauxcmd,sre
00ACE   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00AD2                         1
00AD2   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00AD4 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00ADA   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00ADC   103C 0090             1                       MOVE.B            #sre,D0                       ; set data
00AE0   6100 1370       01E52 1                       BSR               NbWrite
00AE4                         1
00AE4   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00AE8   6014            00AFE                         BRA.S             RemGood
00AEA
00AEA                               ; here if interface not controller in charge
00AEA   7081                        RemENchg          MOVEQ             #gpibErr,D0                   ; return error to O.S.
00AEC   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00AF2 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00AF6   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00AFC   6010            00B0E                         BRA.S             RemDone
00AFE
00AFE   7000                        RemGood           MOVEQ             #noErr,D0                     ; return no error
00B00 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00B04 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00B08   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00B0E
00B0E                               RemDone           MSetCIC                                         ; set up status CIC bit
00B0E                         1
00B0E   6100 1302       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00B12   6608            00B1C 1                       BNE.S             @StCIC1                       ; no ...
00B14   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00B1A   6006            00B22 1                       BRA.S             @StCIC2
00B1C                         1
00B1C   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00B22                         1
00B22   4E71                  1     @StCIC2           NOP
00B24                         1
00B24   4CDF 0200                                     MOVEM.L           (SP)+,A1                      ; restore local registers
00B28   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00B2C   6000 F742       00270                         BRA               ExitDrvr
00B30
00B30
00B30
00B30
00B30                               ********************************************************************************************
00B30                               *     Local - control call to command 'local'
00B30                               *
00B30                               *     Entry:      A0 - param blk pointer
00B30                               *                 A1 - DCE pointer
00B30                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00B30                               *
00B30                               ********************************************************************************************
00B30                               Local
00B30   48E7 0040                                     MOVEM.L           A1,-(SP)                      ; save local work registers
00B34
00B34                               ; get base address of board
00B34   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00B38   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00B3A   E188                                          LSL.L             #8,D0
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  18
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00B3C   E188                                          LSL.L             #8,D0
00B3E   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00B44   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00B46
00B46   6100 12CA       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00B4A   661C            00B68                         BNE.S             LclNchg                       ; no ...
00B4C
00B4C                               ; reset remote enable command
00B4C                                                 MWrite            gpibauxcmd,sreclr
00B4C   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00B50                         1
00B50   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00B52 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00B58   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00B5A   103C 0010             1                       MOVE.B            #sreclr,D0                    ; set data
00B5E   6100 12F2       01E52 1                       BSR               NbWrite
00B62                         1
00B62   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00B66   6014            00B7C                         BRA.S             LclGood
00B68
00B68                               ; here if interface not controller in charge
00B68   7081                        LclNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
00B6A   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00B70 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00B74   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00B7A   6010            00B8C                         BRA.S             LclDone
00B7C
00B7C   7000                        LclGood           MOVEQ             #noErr,D0                     ; return no error
00B7E G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00B82 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00B86   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00B8C
00B8C                               LclDone           MSetCIC                                         ; set up status CIC bit
00B8C                         1
00B8C   6100 1284       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00B90   6608            00B9A 1                       BNE.S             @StCIC1                       ; no ...
00B92   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00B98   6006            00BA0 1                       BRA.S             @StCIC2
00B9A                         1
00B9A   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00BA0                         1
00BA0   4E71                  1     @StCIC2           NOP
00BA2                         1
00BA2   4CDF 0200                                     MOVEM.L           (SP)+,A1                      ; restore local registers
00BA6   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00BAA   6000 F6C4       00270                         BRA               ExitDrvr
00BAE
00BAE
00BAE
00BAE
00BAE                               ********************************************************************************************
00BAE                               *     Ifc - control call to command 'interface clear'
00BAE                               *
00BAE                               *     Entry:      A0 - param blk pointer
00BAE                               *                 A1 - DCE pointer
00BAE                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00BAE                               *
00BAE                               ********************************************************************************************
00BAE                               Ifc
00BAE   48E7 0040                                     MOVEM.L           A1,-(SP)                      ; save local work registers
00BB2
00BB2                               ; get base address of board
00BB2   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00BB6   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00BB8   E188                                          LSL.L             #8,D0
00BBA   E188                                          LSL.L             #8,D0
00BBC   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00BC2   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00BC4
00BC4   6100 124C       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00BC8   663E            00C08                         BNE.S             IfcNchg                       ; no ...
00BCA
00BCA                               ; send 'interface clear' command
00BCA                                                 MWrite            gpibauxcmd,sic
00BCA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00BCE                         1
00BCE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00BD0 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00BD6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00BD8   103C 008F             1                       MOVE.B            #sic,D0                       ; set data
00BDC   6100 1274       01E52 1                       BSR               NbWrite
00BE0                         1
00BE0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00BE4
00BE4   3038 0D00                                     MOVE.W            TimeDBRA,D0                   ; # iterations per millisecond
00BE8 G 51C8 FFFE       00BE8       Ifc1              DBRA              D0,Ifc1                       ; wait
00BEC
00BEC                                                 MWrite            gpibauxcmd,siclr              ; reset interface clear cmd
00BEC   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00BF0                         1
00BF0   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00BF2 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00BF8   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00BFA   103C 000F             1                       MOVE.B            #siclr,D0                     ; set data
00BFE   6100 1252       01E52 1                       BSR               NbWrite
00C02                         1
00C02   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  19
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00C06   6014            00C1C                         BRA.S             IfcGood
00C08
00C08                               ; here if interface not controller in charge
00C08   7081                        IfcNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
00C0A   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00C10 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00C14   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00C1A   6010            00C2C                         BRA.S             IfcDone
00C1C
00C1C   7000                        IfcGood           MOVEQ             #noErr,D0                     ; return no error
00C1E G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00C22 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00C26   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00C2C
00C2C                               IfcDone           MSetCIC                                         ; set up status CIC bit
00C2C                         1
00C2C   6100 11E4       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00C30   6608            00C3A 1                       BNE.S             @StCIC1                       ; no ...
00C32   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00C38   6006            00C40 1                       BRA.S             @StCIC2
00C3A                         1
00C3A   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00C40                         1
00C40   4E71                  1     @StCIC2           NOP
00C42                         1
00C42   4CDF 0200                                     MOVEM.L           (SP)+,A1                      ; restore local registers
00C46   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00C4A   6000 F624       00270                         BRA               ExitDrvr
00C4E
00C4E
00C4E
00C4E
00C4E                               ********************************************************************************************
00C4E                               *     SetEos - control call to define the EOS character.  The character will be
00C4E                               *                  passed in the 'csVar' field.
00C4E                               *
00C4E                               *     Entry:      A0 - param blk pointer
00C4E                               *                 A1 - DCE pointer
00C4E                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00C4E                               *
00C4E                               ********************************************************************************************
00C4E                               SetEos
00C4E   48E7 00E0                                     MOVEM.L           A0/A1/A2,-(SP)                ; save local work registers
00C52
00C52                               ; get base address of board
00C52   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00C56   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00C58   E188                                          LSL.L             #8,D0
00C5A   E188                                          LSL.L             #8,D0
00C5C   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00C62   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00C64
00C64   2009                                          MOVE.L            A1,D0                         ; from board base address
00C66 G 0680 0000 0010                                ADD.L             #eos,D0                       ; add to where byte will go
00C6C   2040                                          MOVEA.L           D0,A0                         ; A0 has address
00C6E   102A 0001                                     MOVE.B            csVar+1(A2),D0                ; get the new EOS character
00C72   6100 11DE       01E52                         BSR               NbWrite
00C76
00C76   7000                        StEosGood         MOVEQ             #noErr,D0                     ; return no error
00C78 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00C7C G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00C80   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00C86
00C86                               StEosDone         MSetCIC                                         ; set up status CIC bit
00C86                         1
00C86   6100 118A       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00C8A   6608            00C94 1                       BNE.S             @StCIC1                       ; no ...
00C8C   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00C92   6006            00C9A 1                       BRA.S             @StCIC2
00C94                         1
00C94   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00C9A                         1
00C9A   4E71                  1     @StCIC2           NOP
00C9C                         1
00C9C   4CDF 0700                                     MOVEM.L           (SP)+,A0/A1/A2                ; restore local registers
00CA0   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00CA4   6000 F5CA       00270                         BRA               ExitDrvr
00CA8
00CA8
00CA8
00CA8
00CA8                               ********************************************************************************************
00CA8                               *     SetMyAddr - control call to define a new GPIB address for the interface.
00CA8                               *                       The character ill be passed in the 'csVar' location.
00CA8                               *
00CA8                               *     Entry:      A0 - param blk pointer
00CA8                               *                 A1 - DCE pointer
00CA8                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00CA8                               *
00CA8                               ********************************************************************************************
00CA8                               SetMyAddr
00CA8   48E7 00E0                                     MOVEM.L           A0/A1/A2,-(SP)                ; save local work registers
00CAC
00CAC                               ; get base address of board
00CAC   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00CB0   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00CB2   E188                                          LSL.L             #8,D0
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  20
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00CB4   E188                                          LSL.L             #8,D0
00CB6   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00CBC   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00CBE
00CBE                               ; write address to local storage
00CBE   2009                                          MOVE.L            A1,D0                         ; from board base address
00CC0 G 0680 0000 0014                                ADD.L             #gpibAddrSt,D0                ; add to where byte will go
00CC6   2040                                          MOVEA.L           D0,A0                         ; A0 has address
00CC8   102A 0001                                     MOVE.B            csVar+1(A2),D0                ; get the new GPIB address
00CCC   6100 1184       01E52                         BSR               NbWrite
00CD0
00CD0                               ; write address to 9914
00CD0   2009                                          MOVE.L            A1,D0                         ; from board base address
00CD2 G 0680 0002 0004                                ADD.L             #gpibaddr,D0                  ; add to where byte will go
00CD8   2040                                          MOVEA.L           D0,A0                         ; A0 has address
00CDA   102A 0001                                     MOVE.B            csVar+1(A2),D0                ; get the new GPIB address
00CDE   6100 1172       01E52                         BSR               NbWrite
00CE2
00CE2   7000                        AddrGood          MOVEQ             #noErr,D0                     ; return no error
00CE4 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00CE8 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00CEC   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00CF2
00CF2                               AddrDone          MSetCIC                                         ; set up status CIC bit
00CF2                         1
00CF2   6100 111E       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00CF6   6608            00D00 1                       BNE.S             @StCIC1                       ; no ...
00CF8   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00CFE   6006            00D06 1                       BRA.S             @StCIC2
00D00                         1
00D00   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00D06                         1
00D06   4E71                  1     @StCIC2           NOP
00D08                         1
00D08   4CDF 0700                                     MOVEM.L           (SP)+,A0/A1/A2                ; restore local registers
00D0C   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00D10   6000 F55E       00270                         BRA               ExitDrvr
00D14
00D14
00D14
00D14
00D14
00D14                               ********************************************************************************************
00D14                               *     PpEnable - control call to configure certain listeners on the bus to respond to
00D14                               *                       a parallel poll.
00D14                               *
00D14                               *     Entry:      A0 - param blk pointer
00D14                               *                 A1 - DCE pointer
00D14                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00D14                               *
00D14                               ********************************************************************************************
00D14                               PpEnable
00D14   48E7 00F0                                     MOVEM.L           A0-A3,-(SP)                   ; save local work registers
00D18
00D18                               ; get base address of board
00D18   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00D1C   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00D1E   E188                                          LSL.L             #8,D0
00D20   E188                                          LSL.L             #8,D0
00D22   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00D28   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00D2A
00D2A   6100 10E6       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00D2E   6600 008E       00DBE                         BNE               PpENchg                       ; no ...
00D32
00D32                               ; get pointer to listener list
00D32   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to listener list
00D36
00D36                               ; get pointer to configuration list
00D36   206A 000C                                     MOVEA.L           csDataBuf(A2),A0              ; A0 <- pointer to configuration bytes
00D3A
00D3A                               ; loop sending configuration data to listeners
00D3A                               Ppe1              MWrite            gpibdataout,unl               ; send 'universal unlisten'
00D3A   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00D3E                         1
00D3E   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00D40 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00D46   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00D48   103C 003F             1                       MOVE.B            #unl,D0                       ; set data
00D4C   6100 1104       01E52 1                       BSR               NbWrite
00D50                         1
00D50   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00D54   6100 0FD2       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00D58   674A            00DA4                         BEQ.S             PpeTimeout                    ; Bus timed out
00D5A
00D5A                               ; now address the next listener
00D5A   101B                                          MOVE.B            (A3)+,D0                      ; get the next listener
00D5C G 0C00 0020                                     CMP.B             #$20,D0
00D60   6D70            00DD2                         BLT.S             PpeGood
00D62 G 0C00 003E                                     CMP.B             #$3e,D0
00D66   6E6A            00DD2                         BGT.S             PpeGood
00D68   6100 0FF0       01D5A                         BSR               DataOut                       ; send listener over GPIB
00D6C   6100 0FBA       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00D70   6732            00DA4                         BEQ.S             PpeTimeout                    ; Bus timed out
00D72
00D72                               ; send parallel poll configure
00D72                                                 MWrite            gpibdataout,ppc
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  21
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00D72   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00D76                         1
00D76   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00D78 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00D7E   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00D80   103C 0005             1                       MOVE.B            #ppc,D0                       ; set data
00D84   6100 10CC       01E52 1                       BSR               NbWrite
00D88                         1
00D88   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00D8C   6100 0F9A       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00D90   6712            00DA4                         BEQ.S             PpeTimeout                    ; Bus timed out
00D92
00D92                               ; send configuration byte
00D92   1018                                          MOVE.B            (A0)+,D0                      ; get byte
00D94   0000 0060                                     ORI.B             #ppe,D0                       ; OR with PPE command
00D98   6100 0FC0       01D5A                         BSR               DataOut                       ; send to listener
00D9C   6100 0F8A       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00DA0   6702            00DA4                         BEQ.S             PpeTimeout                    ; Bus timed out
00DA2
00DA2   6096            00D3A                         BRA.S             Ppe1                          ; until done
00DA4
00DA4
00DA4                               ; bus timed out
00DA4   7081                        PpeTimeout        MOVEQ             #gpibErr,D0                   ; return error to O.S.
00DA6   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
00DAC G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00DB0   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00DB6   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
00DBC   6024            00DE2                         BRA.S             PpeDone
00DBE
00DBE                               ; here if interface not controller in charge
00DBE   7081                        PpENchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
00DC0   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00DC6 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00DCA   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00DD0   6010            00DE2                         BRA.S             PpeDone
00DD2
00DD2   7000                        PpeGood           MOVEQ             #noErr,D0                     ; return no error
00DD4 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00DD8 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00DDC   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00DE2
00DE2                               PpeDone           MSetCIC                                         ; set up status CIC bit
00DE2                         1
00DE2   6100 102E       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00DE6   6608            00DF0 1                       BNE.S             @StCIC1                       ; no ...
00DE8   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00DEE   6006            00DF6 1                       BRA.S             @StCIC2
00DF0                         1
00DF0   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00DF6                         1
00DF6   4E71                  1     @StCIC2           NOP
00DF8                         1
00DF8   4CDF 0F00                                     MOVEM.L           (SP)+,A0-A3                   ; restore local registers
00DFC   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00E00   6000 F46E       00270                         BRA               ExitDrvr
00E04
00E04
00E04
00E04
00E04
00E04                               *******************************************************************************************
00E04                               *     PpDisable - control call to disable parallel polling.
00E04                               *
00E04                               *     Entry:      A0 - param blk pointer
00E04                               *                 A1 - DCE pointer
00E04                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00E04                               *
00E04                               *******************************************************************************************
00E04                               PpDisable
00E04   48E7 00F0                                     MOVEM.L           A0-A3,-(SP)                   ; save local work registers
00E08
00E08                               ; get base address of board
00E08   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00E0C   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00E0E   E188                                          LSL.L             #8,D0
00E10   E188                                          LSL.L             #8,D0
00E12   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00E18   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00E1A
00E1A   6100 0FF6       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00E1E   6600 009C       00EBC                         BNE               PpDNchg                       ; no ...
00E22
00E22                               ; get pointer to listener list
00E22   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to listener list
00E26
00E26                               ; send 'universal unlisten'
00E26                                                 MWrite            gpibdataout,unl               ; send 'universal unlisten'
00E26   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00E2A                         1
00E2A   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00E2C G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00E32   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00E34   103C 003F             1                       MOVE.B            #unl,D0                       ; set data
00E38   6100 1018       01E52 1                       BSR               NbWrite
00E3C                         1
00E3C   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  22
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00E40   6100 0EE6       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00E44   675C            00EA2                         BEQ.S             PpDisTimeout                  ; Bus timed out
00E46
00E46                               ; loop addressing the listeners
00E46   101B                        PpDis1            MOVE.B            (A3)+,D0                      ; get the next listener
00E48 G 0C00 0020                                     CMP.B             #$20,D0
00E4C   6D12            00E60                         BLT.S             PpDis2
00E4E G 0C00 003E                                     CMP.B             #$3e,D0
00E52   6E0C            00E60                         BGT.S             PpDis2
00E54   6100 0F04       01D5A                         BSR               DataOut                       ; send listener over GPIB
00E58   6100 0ECE       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00E5C   6744            00EA2                         BEQ.S             PpDisTimeout                  ; Bus timed out
00E5E   60E6            00E46                         BRA.S             PpDis1                        ; until done
00E60
00E60                               ; send parallel poll configure
00E60                               PpDis2            MWrite            gpibdataout,ppc
00E60   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00E64                         1
00E64   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00E66 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00E6C   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00E6E   103C 0005             1                       MOVE.B            #ppc,D0                       ; set data
00E72   6100 0FDE       01E52 1                       BSR               NbWrite
00E76                         1
00E76   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00E7A   6100 0EAC       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00E7E   6722            00EA2                         BEQ.S             PpDisTimeout                  ; Bus timed out
00E80
00E80                               ; send parallel poll disable
00E80                                                 MWrite            gpibdataout,ppd
00E80   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00E84                         1
00E84   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00E86 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00E8C   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00E8E   103C 0070             1                       MOVE.B            #ppd,D0                       ; set data
00E92   6100 0FBE       01E52 1                       BSR               NbWrite
00E96                         1
00E96   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00E9A   6100 0E8C       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00E9E   6702            00EA2                         BEQ.S             PpDisTimeout                  ; Bus timed out
00EA0   602E            00ED0                         BRA.S             PpDisGood
00EA2
00EA2                               ; bus timed out
00EA2   7081                        PpDisTimeout      MOVEQ             #gpibErr,D0                   ; return error to O.S.
00EA4   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
00EAA G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00EAE   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00EB4   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
00EBA   6024            00EE0                         BRA.S             PpDisDone
00EBC
00EBC                               ; here if interface not controller in charge
00EBC   7081                        PpDNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
00EBE   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00EC4 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00EC8   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00ECE   6010            00EE0                         BRA.S             PpDisDone
00ED0
00ED0   7000                        PpDisGood         MOVEQ             #noErr,D0                     ; return no error
00ED2 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00ED6 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00EDA   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00EE0
00EE0                               PpDisDone         MSetCIC                                         ; set up status CIC bit
00EE0                         1
00EE0   6100 0F30       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00EE4   6608            00EEE 1                       BNE.S             @StCIC1                       ; no ...
00EE6   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00EEC   6006            00EF4 1                       BRA.S             @StCIC2
00EEE                         1
00EEE   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00EF4                         1
00EF4   4E71                  1     @StCIC2           NOP
00EF6                         1
00EF6   4CDF 0F00                                     MOVEM.L           (SP)+,A0-A3                   ; restore local registers
00EFA   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00EFE   6000 F370       00270                         BRA               ExitDrvr
00F02
00F02
00F02
00F02
00F02
00F02                               ********************************************************************************************
00F02                               *     PpUConfig - control call to command 'Parallel Poll Unconfigure'.  An unaddressed
00F02                               *                       command.
00F02                               *
00F02                               *     Entry:      A0 - param blk pointer
00F02                               *                 A1 - DCE pointer
00F02                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00F02                               *
00F02                               ********************************************************************************************
00F02                               PpUConfig
00F02   48E7 00F0                                     MOVEM.L           A0-A3,-(SP)                   ; save local work registers
00F06
00F06                               ; get base address of board
00F06   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00F0A   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  23
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00F0C   E188                                          LSL.L             #8,D0
00F0E   E188                                          LSL.L             #8,D0
00F10   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00F16   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00F18
00F18   6100 0EF8       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00F1C   663C            00F5A                         BNE.S             PpUNchg                       ; no ...
00F1E
00F1E                               ; send 'PPU'
00F1E                                                 MWrite            gpibdataout,ppu               ; send 'Parallel Poll Unconfigure'
00F1E   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00F22                         1
00F22   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00F24 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
00F2A   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00F2C   103C 0015             1                       MOVE.B            #ppu,D0                       ; set data
00F30   6100 0F20       01E52 1                       BSR               NbWrite
00F34                         1
00F34   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00F38   6100 0DEE       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
00F3C   6702            00F40                         BEQ.S             PpUConTimeout                 ; Bus timed out
00F3E   602E            00F6E                         BRA.S             PpUConGood
00F40
00F40                               ; bus timed out
00F40   7081                        PpUConTimeout     MOVEQ             #gpibErr,D0                   ; return error to O.S.
00F42   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
00F48 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00F4C   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00F52   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
00F58   6024            00F7E                         BRA.S             PpUConDone
00F5A
00F5A                               ; here if interface not controller in charge
00F5A   7081                        PpUNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
00F5C   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
00F62 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00F66   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
00F6C   6010            00F7E                         BRA.S             PpUConDone
00F6E
00F6E   7000                        PpUConGood        MOVEQ             #noErr,D0                     ; return no error
00F70 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
00F74 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
00F78   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
00F7E
00F7E                               PpUConDone  MSetCIC                                               ; set up status CIC bit
00F7E                         1
00F7E   6100 0E92       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
00F82   6608            00F8C 1                       BNE.S             @StCIC1                       ; no ...
00F84   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
00F8A   6006            00F92 1                       BRA.S             @StCIC2
00F8C                         1
00F8C   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
00F92                         1
00F92   4E71                  1     @StCIC2           NOP
00F94                         1
00F94   4CDF 0F00                                     MOVEM.L           (SP)+,A0-A3                   ; restore local registers
00F98   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
00F9C   6000 F2D2       00270                         BRA               ExitDrvr
00FA0
00FA0
00FA0
00FA0
00FA0
00FA0
00FA0                               ********************************************************************************************
00FA0                               *     CParPoll
00FA0                               *
00FA0                               *     Entry:      A0 - param blk pointer
00FA0                               *                 A1 - DCE pointer
00FA0                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00FA0                               *
00FA0                               ********************************************************************************************
00FA0                               CParPoll
00FA0   48E7 00C0                                     MOVEM.L           A0-A1,-(SP)                   ; save local work registers
00FA4
00FA4                               ; get base address of board
00FA4   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
00FA8   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
00FAA   E188                                          LSL.L             #8,D0
00FAC   E188                                          LSL.L             #8,D0
00FAE   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
00FB4   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
00FB6
00FB6   6100 0E5A       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
00FBA   6678            01034                         BNE.S             CPpNchg                       ; no ...
00FBC
00FBC                               ; execute parallel poll
00FBC                                                 MWrite            gpibauxcmd,rpp
00FBC   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
00FC0                         1
00FC0   2009                  1                       MOVE.L            A1,D0                         ; from board base address
00FC2 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
00FC8   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
00FCA   103C 008E             1                       MOVE.B            #rpp,D0                       ; set data
00FCE   6100 0E82       01E52 1                       BSR               NbWrite
00FD2                         1
00FD2   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
00FD6
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  24
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

00FD6                               ;delay 125 microseconds
00FD6   3038 0D00                                     MOVE.W            TimeDBRA,D0                   ; # iterations per millisecond
00FDA   0280 0000 FFFF                                ANDI.L            #$0000ffff,D0
00FE0   E088                                          LSR.L             #8,D0                         ; divide by 8
00FE2   6602            00FE6                         BNE.S             CParPoll1                     ; minimum #
00FE4   7001                                          MOVEQ             #1,D0
00FE6 G 51C8 FFFE       00FE6       CParPoll1         DBRA              D0,CParPoll1                  ; wait
00FEA
00FEA                               ; get poll status
00FEA   2009                                          MOVE.L            A1,D0                         ; from board base address
00FEC G 0680 0002 000C                                ADD.L             #gpibcmd,D0                   ; add offset to command pass thru register
00FF2   2040                                          MOVEA.L           D0,A0                         ; A0 has address
00FF4   6100 0E42       01E38                         BSR               NbRead
00FF8
00FF8                               ; store return result
00FF8   0240 00FF                                     ANDI.W            #$ff,D0                       ; mask significant bits
00FFC   3480                                          MOVE.W            D0,csVar(A2)
00FFE
00FFE                               ; clear parallel poll
00FFE                                                 MWrite            gpibauxcmd,rppclr
00FFE   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01002                         1
01002   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01004 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
0100A   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
0100C   103C 000E             1                       MOVE.B            #rppclr,D0                    ; set data
01010   6100 0E40       01E52 1                       BSR               NbWrite
01014                         1
01014   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01018   602E            01048                         BRA.S             PPollGood
0101A
0101A                               ; bus timed out
0101A   7081                        PPollTimeout      MOVEQ             #gpibErr,D0                   ; return error to O.S.
0101C   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
01022 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01026   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0102C   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
01032   6024            01058                         BRA.S             PPollDone
01034
01034                               ; here if interface not controller in charge
01034   7081                        CPpNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
01036   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
0103C G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01040   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01046   6010            01058                         BRA.S             PPollDone
01048
01048   7000                        PPollGood         MOVEQ             #noErr,D0                     ; return no error
0104A G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
0104E G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01052   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01058
01058                               PPollDone         MSetCIC                                         ; set up status CIC bit
01058                         1
01058   6100 0DB8       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
0105C   6608            01066 1                       BNE.S             @StCIC1                       ; no ...
0105E   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01064   6006            0106C 1                       BRA.S             @StCIC2
01066                         1
01066   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
0106C                         1
0106C   4E71                  1     @StCIC2           NOP
0106E                         1
0106E   4CDF 0300                                     MOVEM.L           (SP)+,A0-A1                   ; restore local registers
01072   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01076   6000 F1F8       00270                         BRA               ExitDrvr
0107A
0107A
0107A
0107A
0107A
0107A
0107A
0107A                               *********************************************************************************************
0107A                               *     CSerPoll    - conduct a serial poll of a designated device(s).  Routine
0107A                               *                         will fill the pointed to buffer with the response byte(s)
0107A                               *                         from the specified talker(s).
0107A                               *
0107A                               *     Entry:      A0 - param blk pointer
0107A                               *                 A1 - DCE pointer
0107A                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0107A                               *
0107A                               *********************************************************************************************
0107A                               CSerPoll
0107A   48E7 E0FC                                     MOVEM.L           A0-A5/D0-D2,-(SP)             ; save local work registers
0107E
0107E                               ; get base address of board
0107E   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01082   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01084   E188                                          LSL.L             #8,D0
01086   E188                                          LSL.L             #8,D0
01088   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
0108E   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01090
01090   6100 0D80       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
01094   6600 0160       011F6                         BNE               CSpNchg                       ; no ...
01098
01098                               ; set up local registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  25
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01098   2009                                          MOVE.L            A1,D0
0109A G 0680 0002 0000                                ADD.L             #gpibint0,D0
010A0   2840                                          MOVEA.L           D0,A4                         ; A4 = 9914 int0 register address
010A2   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to talker list
010A6   2A6A 000C                                     MOVEA.L           csDataBuf(A2),A5              ; A5 <- pointer to status buffer
010AA   6100 0D0E       01DBA                         BSR               GetGpibTimot
010AE   2200                                          MOVE.L            D0,D1                         ; D1 = timeout loop count
010B0
010B0                               ; serial poll enable
010B0                                                 MWrite            gpibdataout,spe               ; send 'serial poll enable'
010B0   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
010B4                         1
010B4   2009                  1                       MOVE.L            A1,D0                         ; from board base address
010B6 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
010BC   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
010BE   103C 0018             1                       MOVE.B            #spe,D0                       ; set data
010C2   6100 0D8E       01E52 1                       BSR               NbWrite
010C6                         1
010C6   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
010CA
010CA                                                 MWrite            gpibauxcmd,hdfa               ; holdoff all data
010CA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
010CE                         1
010CE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
010D0 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
010D6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
010D8   103C 0083             1                       MOVE.B            #hdfa,D0                      ; set data
010DC   6100 0D74       01E52 1                       BSR               NbWrite
010E0                         1
010E0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
010E4   6100 0C42       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
010E8   6700 00F2       011DC                         BEQ               SPTimeout                     ; Bus timed out
010EC
010EC                               ; loop addressing each talker
010EC   101B                        CSerPoll1         MOVE.B            (A3)+,D0                      ; get the next talker
010EE G 0C00 0040                                     CMP.B             #$40,D0
010F2   6D00 00AC       011A0                         BLT               CSerPoll4
010F6 G 0C00 005E                                     CMP.B             #$5e,D0
010FA   6E00 00A4       011A0                         BGT               CSerPoll4
010FE   6100 0C5A       01D5A                         BSR               DataOut                       ; send talker over GPIB
01102   6100 0C24       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
01106   6700 00D4       011DC                         BEQ               SPTimeout                     ; Bus timed out
0110A
0110A                                                 MWrite            gpibauxcmd,lon                ; listen only
0110A   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
0110E                         1
0110E   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01110 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01116   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01118   103C 0089             1                       MOVE.B            #lon,D0                       ; set data
0111C   6100 0D34       01E52 1                       BSR               NbWrite
01120                         1
01120   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01124
01124                                                 MWrite            gpibauxcmd,gts                ; go to standby
01124   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01128                         1
01128   2009                  1                       MOVE.L            A1,D0                         ; from board base address
0112A G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01130   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01132   103C 000B             1                       MOVE.B            #gts,D0                       ; set data
01136   6100 0D1A       01E52 1                       BSR               NbWrite
0113A                         1
0113A   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0113E
0113E                               ; wait for byte-in
0113E   204C                                          MOVEA.L           A4,A0                         ; A0 = 9914 int0 register address
01140   2401                        CSerPoll2         MOVE.L            D1,D2                         ; D2 = loop timeout pass count
01142 G 5382                        CSerPoll3         SUBI.L            #1,D2                         ; decrement pass count
01144   6700 0096       011DC                         BEQ               SPTimeout                     ; if bus not responding
01148   6100 0CEE       01E38                         BSR               NbRead                        ; get interrupt 0 status
0114C   0200 0020                                     ANDI.B            #bim,D0                       ; check for BI
01150   67F0            01142                         BEQ.S             CSerPoll3                     ; wait until set
01152
01152                                                 MWrite            gpibauxcmd,tcs                ; take control synchronously
01152   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01156                         1
01156   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01158 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
0115E   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01160   103C 000D             1                       MOVE.B            #tcs,D0                       ; set data
01164   6100 0CEC       01E52 1                       BSR               NbWrite
01168                         1
01168   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0116C   6100 0BBA       01D28                         BSR               WaitOut                       ; wait for byte out
01170   676A            011DC                         BEQ.S             SPTimeout                     ; Bus timed out
01172
01172   2409                                          MOVE.L            A1,D2
01174 G 0682 0002 001C                                ADD.L             #gpibdatain,D2
0117A   2042                                          MOVEA.L           D2,A0                         ; A0 = 9914 'Data In' register address
0117C   6100 0CBA       01E38                         BSR               NbRead                        ; get serial poll response byte
01180   1AC0                                          MOVE.B            D0,(A5)+                      ; store byte away
01182
01182                                                 MWrite            gpibauxcmd,rhdf               ; release holdoff
01182   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01186                         1
01186   2009                  1                       MOVE.L            A1,D0                         ; from board base address
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  26
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01188 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
0118E   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01190   103C 0002             1                       MOVE.B            #rhdf,D0                      ; set data
01194   6100 0CBC       01E52 1                       BSR               NbWrite
01198                         1
01198   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0119C   6000 FF4E       010EC                         BRA               CSerPoll1
011A0
011A0                               ; done getting poll responses
011A0                               CSerPoll4         MWrite            gpibdataout,spd               ; send 'serial poll disable'
011A0   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
011A4                         1
011A4   2009                  1                       MOVE.L            A1,D0                         ; from board base address
011A6 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
011AC   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
011AE   103C 0019             1                       MOVE.B            #spd,D0                       ; set data
011B2   6100 0C9E       01E52 1                       BSR               NbWrite
011B6                         1
011B6   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
011BA                                                 MWrite            gpibauxcmd,hdaclr             ; release holdoff on all
011BA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
011BE                         1
011BE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
011C0 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
011C6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
011C8   103C 0003             1                       MOVE.B            #hdaclr,D0                    ; set data
011CC   6100 0C84       01E52 1                       BSR               NbWrite
011D0                         1
011D0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
011D4   6100 0B52       01D28                         BSR               WaitOut                       ; wait for byte out
011D8   6702            011DC                         BEQ.S             SPTimeout                     ; Bus timed out
011DA   602E            0120A                         BRA.S             SPollGood
011DC
011DC                               ; bus timed out
011DC   7081                        SPTimeout         MOVEQ             #gpibErr,D0                   ; return error to O.S.
011DE   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
011E4 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
011E8   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
011EE   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
011F4   6024            0121A                         BRA.S             SPollDone
011F6
011F6                               ; here if interface not controller in charge
011F6   7081                        CSpNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
011F8   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
011FE G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01202   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01208   6010            0121A                         BRA.S             SPollDone
0120A
0120A   7000                        SPollGood         MOVEQ             #noErr,D0                     ; return no error
0120C G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01210 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01214   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
0121A
0121A                               SPollDone   MSetCIC                                               ; set up status CIC bit
0121A                         1
0121A   6100 0BF6       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
0121E   6608            01228 1                       BNE.S             @StCIC1                       ; no ...
01220   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01226   6006            0122E 1                       BRA.S             @StCIC2
01228                         1
01228   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
0122E                         1
0122E   4E71                  1     @StCIC2           NOP
01230                         1
01230   4CDF 3F07                                     MOVEM.L           (SP)+,A0-A5/D0-D2             ; restore local registers
01234   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01238   6000 F036       00270                         BRA               ExitDrvr
0123C
0123C
0123C
0123C
0123C
0123C
0123C
0123C
0123C                               ********************************************************************************************
0123C                               *     NContInit - driver control routine to init 9914 as non-controller
0123C                               *
0123C                               *     Entry:      A0 - param blk pointer
0123C                               *                 A1 - DCE pointer
0123C                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0123C                               *
0123C                               ********************************************************************************************
0123C                               NContInit
0123C   6148            01286                         BSR.S             NCInit                        ; init 9914 as non-controller
0123E
0123E   48E7 0040                                     MOVEM.L           A1,-(SP)                      ; save work registers
01242
01242                               ; get base address of board
01242   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01246   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01248   E188                                          LSL.L             #8,D0
0124A   E188                                          LSL.L             #8,D0
0124C   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01252   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01254
01254   7000                        NContGood         MOVEQ             #noErr,D0                     ; return no error to O.S.
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  27
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01256 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
0125A G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
0125E   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01264
01264
01264                               NContDone         MSetCIC                                         ; set up status CIC bit
01264                         1
01264   6100 0BAC       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01268   6608            01272 1                       BNE.S             @StCIC1                       ; no ...
0126A   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01270   6006            01278 1                       BRA.S             @StCIC2
01272                         1
01272   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01278                         1
01278   4E71                  1     @StCIC2           NOP
0127A                         1
0127A   4CDF 0200                                     MOVEM.L           (SP)+,A1                      ; restore local work registers
0127E   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01282   6000 EFEC       00270                         BRA               ExitDrvr
01286
01286
01286
01286
01286
01286
01286
01286                               ********************************************************************************************
01286                               *     initialize the 9914 chip as a non-controller
01286                               *
01286                               *     Entry:      A0 - param blk pointer
01286                               *                 A1 - DCE pointer
01286                               *
01286                               *     Uses: D0 - temporary
01286                               *                 D1 - temporary
01286                               *
01286                               ********************************************************************************************
01286                               ; save registers
01286                               NCInit
01286   48E7 C0F8                                     MOVEM.L           A0-A4/D0-D1,-(SP)             ; save work registers
0128A
0128A G 2448                                          MOVE.L            A0,A2                         ; A2 <- param block pointer
0128C G 2649                                          MOVE.L            A1,A3                         ; A3 <- DCE pointer
0128E
0128E                               ; get base address of board
0128E   102B 0028                                     MOVE.B            dCtlSlot(A3),D0               ; get the slot address
01292   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01294   E188                                          LSL.L             #8,D0
01296   E188                                          LSL.L             #8,D0
01298   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
0129E   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
012A0
012A0                               ; set flag as non-controller in local storage
012A0                                                 MWrite            amController,$00
012A0   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
012A4                         1
012A4   2009                  1                       MOVE.L            A1,D0                         ; from board base address
012A6 G 0680 0000 001C        1                       ADD.L             #amController,D0              ; add to where byte will go
012AC   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
012AE G 4200                  1                       MOVE.B            #$00,D0                       ; set data
012B0   6100 0BA0       01E52 1                       BSR               NbWrite
012B4                         1
012B4   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
012B8
012B8                               ; issue software reset to 9914
012B8                                                 MWrite            gpibauxcmd,swrst
012B8   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
012BC                         1
012BC   2009                  1                       MOVE.L            A1,D0                         ; from board base address
012BE G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
012C4   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
012C6   103C 0080             1                       MOVE.B            #swrst,D0                     ; set data
012CA   6100 0B86       01E52 1                       BSR               NbWrite
012CE                         1
012CE   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
012D2
012D2                               ; disable all interrupt mask bits
012D2                                                 MWrite            gpibintm0,0                   ; reset int mask 0 register
012D2   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
012D6                         1
012D6   2009                  1                       MOVE.L            A1,D0                         ; from board base address
012D8 G 0680 0002 0000        1                       ADD.L             #gpibintm0,D0                 ; add to where byte will go
012DE   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
012E0 G 4200                  1                       MOVE.B            #0,D0                         ; set data
012E2   6100 0B6E       01E52 1                       BSR               NbWrite
012E6                         1
012E6   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
012EA                                                 MWrite            gpibintm1,0                   ; reset int mask 1 register
012EA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
012EE                         1
012EE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
012F0 G 0680 0002 0010        1                       ADD.L             #gpibintm1,D0                 ; add to where byte will go
012F6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
012F8 G 4200                  1                       MOVE.B            #0,D0                         ; set data
012FA   6100 0B56       01E52 1                       BSR               NbWrite
012FE                         1
012FE   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01302
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  28
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01302                               ; write address to 9914
01302   2009                                          MOVE.L            A1,D0                         ; from board base address
01304 G 0680 0000 0014                                ADD.L             #gpibAddrSt,D0                ; add to where address is stored
0130A   2040                                          MOVEA.L           D0,A0                         ; A0 has address
0130C   6100 0B2A       01E38                         BSR               NbRead
01310   1200                                          MOVE.B            D0,D1                         ; save address in D1
01312   2009                                          MOVE.L            A1,D0                         ; from board base address
01314 G 0680 0002 0004                                ADD.L             #gpibaddr,D0                  ; add to where byte will go
0131A   2040                                          MOVEA.L           D0,A0                         ; A0 has address
0131C   1001                                          MOVE.B            D1,D0                         ; get address
0131E   6100 0B32       01E52                         BSR               NbWrite
01322
01322                               ; set the driver buffers to 3-state control and select 'non-system controller'
01322                                                 MWrite            swaddr,$02                    ; write to configuration register
01322   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01326                         1
01326   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01328 G 0680 0008 0000        1                       ADD.L             #swaddr,D0                    ; add to where byte will go
0132E   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01330   103C 0002             1                       MOVE.B            #$02,D0                       ; set data
01334   6100 0B1C       01E52 1                       BSR               NbWrite
01338                         1
01338   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0133C                                                 MWrite            swImage,$02                   ; store memory image of configuration register
0133C   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01340                         1
01340   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01342 G 0680 0000 0030        1                       ADD.L             #swImage,D0                   ; add to where byte will go
01348   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
0134A   103C 0002             1                       MOVE.B            #$02,D0                       ; set data
0134E   6100 0B02       01E52 1                       BSR               NbWrite
01352                         1
01352   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01356
01356                               ; clear software reset to 9914
01356                                                 MWrite            gpibauxcmd,swrstclr
01356   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
0135A                         1
0135A   2009                  1                       MOVE.L            A1,D0                         ; from board base address
0135C G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01362   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01364 G 4200                  1                       MOVE.B            #swrstclr,D0                  ; set data
01366   6100 0AEA       01E52 1                       BSR               NbWrite
0136A                         1
0136A   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0136E
0136E   4CDF 1F03                                     MOVEM.L           (SP)+,A0-A4/D0-D1             ; restore registers
01372   4E75                                          RTS
01374
01374
01374
01374
01374
01374
01374
01374
01374
01374
01374
01374                               ********************************************************************************************
01374                               *     CXfer - controller sets up a data transfer but does not participate in it.
01374                               *
01374                               *     Entry:      A0 - param blk pointer
01374                               *                 A1 - DCE pointer
01374                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01374                               *
01374                               ********************************************************************************************
01374                               CXfer
01374   48E7 E050                                     MOVEM.L           A1/A3/D0-D2,-(SP)             ; save local work registers
01378
01378                               ; get base address of board
01378   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
0137C   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
0137E   E188                                          LSL.L             #8,D0
01380   E188                                          LSL.L             #8,D0
01382   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01388   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
0138A
0138A   6100 0A86       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
0138E   6600 0172       01502                         BNE               CXfrNchg                      ; no ...
01392
01392                               ; get pointer to address list
01392   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to address address
01396
01396                               ; Send talker address over bus
01396   101B                                          MOVE.B            (A3)+,D0                      ; get talker
01398 G 0C00 0040                                     CMP.B             #$40,D0
0139C   6D00 0178       01516                         BLT               CXferBadAddr
013A0 G 0C00 005E                                     CMP.B             #$5e,D0
013A4   6E00 0170       01516                         BGT               CXferBadAddr
013A8   6100 09B0       01D5A                         BSR               DataOut                       ; send talker address over GPIB
013AC   6100 097A       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
013B0   6700 0136       014E8                         BEQ               CXfer5                        ; Bus timed out
013B4
013B4                               ; send 'universal unlisten' command over bus
013B4                                                 MWrite            gpibdataout,unl               ; send 'universal unlisten'
013B4   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                               18-May-91  Page  29
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

013B8                         1
013B8   2009                  1                 MOVE.L            A1,D0                          ; from board base address
013BA G 0680 0002 001C        1                 ADD.L             #gpibdataout,D0                ; add to where byte will go
013C0   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
013C2   103C 003F             1                 MOVE.B            #unl,D0                        ; set data
013C6   6100 0A8A       01E52 1                 BSR               NbWrite
013CA                         1
013CA   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
013CE   6100 0958       01D28                   BSR               WaitOut                        ; wait for GPIB bus free
013D2   6700 0114       014E8                   BEQ               CXfer5                         ; Bus timed out
013D6
013D6                               ; loop sending listeners
013D6   101B                        CXfer1      MOVE.B            (A3)+,D0                       ; get the next listener
013D8 G 0C00 0020                               CMP.B             #$20,D0
013DC   6D14            013F2                   BLT.S             CXfer2
013DE G 0C00 003E                               CMP.B             #$3e,D0
013E2   6E0E            013F2                   BGT.S             CXfer2
013E4   6100 0974       01D5A                   BSR               DataOut                        ; send listener over GPIB
013E8   6100 093E       01D28                   BSR               WaitOut                        ; wait for GPIB bus free
013EC   6700 00FA       014E8                   BEQ               CXfer5                         ; Bus timed out
013F0   60E4            013D6                   BRA.S             CXfer1                         ; until done
013F2
013F2                               CXfer2      MWrite            gpibauxcmd,shdw                ; activate shadow handshake
013F2   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
013F6                         1
013F6   2009                  1                 MOVE.L            A1,D0                          ; from board base address
013F8 G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
013FE   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
01400   103C 0096             1                 MOVE.B            #shdw,D0                       ; set data
01404   6100 0A4C       01E52 1                 BSR               NbWrite
01408                         1
01408   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
0140C                                           MWrite            gpibauxcmd,hdfe                ; hold off on EOI only
0140C   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
01410                         1
01410   2009                  1                 MOVE.L            A1,D0                          ; from board base address
01412 G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
01418   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
0141A   103C 0084             1                 MOVE.B            #hdfe,D0                       ; set data
0141E   6100 0A32       01E52 1                 BSR               NbWrite
01422                         1
01422   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
01426                                           MWrite            gpibauxcmd,lon                 ; listen only
01426   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
0142A                         1
0142A   2009                  1                 MOVE.L            A1,D0                          ; from board base address
0142C G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
01432   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
01434   103C 0089             1                 MOVE.B            #lon,D0                        ; set data
01438   6100 0A18       01E52 1                 BSR               NbWrite
0143C                         1
0143C   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
01440                                           MWrite            gpibauxcmd,gts                 ; go to standby
01440   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
01444                         1
01444   2009                  1                 MOVE.L            A1,D0                          ; from board base address
01446 G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
0144C   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
0144E   103C 000B             1                 MOVE.B            #gts,D0                        ; set data
01452   6100 09FE       01E52 1                 BSR               NbWrite
01456                         1
01456   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
0145A
0145A                               ; loop getting EOI status
0145A   6100 095E       01DBA       CXfer3      BSR               GetGpibTimot
0145E   2400                                    MOVE.L            D0,D2                          ; D2 = timeout loop count
01460
01460   2209                                    MOVE.L            A1,D1
01462 G 0681 0002 0000                          ADD.L             #gpibint0,D1
01468   2041                                    MOVEA.L           D1,A0                          ; A0 has 9914 int0 register address
0146A
0146A G 5382                        CXfer4      SUBI.L            #1,D2                          ; decrement loop pass count
0146C   677A            014E8                   BEQ.S             CXfer5                         ; bus timed out
0146E   6100 09C8       01E38                   BSR               NbRead                         ; get interrupt 0 status
01472   0200 0008                               ANDI.B            #EOIMK,D0                      ; check for EOI
01476   67F2            0146A                   BEQ.S             CXfer4                         ; wait until set
01478
01478                               ; get control back
01478                                           MWrite            gpibauxcmd,tcs                 ; take control synchronously
01478   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
0147C                         1
0147C   2009                  1                 MOVE.L            A1,D0                          ; from board base address
0147E G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
01484   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
01486   103C 000D             1                 MOVE.B            #tcs,D0                        ; set data
0148A   6100 09C6       01E52 1                 BSR               NbWrite
0148E                         1
0148E   4CDF 0101             1                 MOVEM.L           (SP)+,D0/A0                    ; restore registers
01492   6100 0894       01D28                   BSR               WaitOut                        ; wait for GPIB bus free
01496   6750            014E8                   BEQ.S             CXfer5                         ; Bus timed out
01498                                           MWrite            gpibauxcmd,rhdf                ; release holdoff
01498   48E7 8080             1                 MOVEM.L           D0/A0,-(SP)                    ; save work registers
0149C                         1
0149C   2009                  1                 MOVE.L            A1,D0                          ; from board base address
0149E G 0680 0002 0018        1                 ADD.L             #gpibauxcmd,D0                 ; add to where byte will go
014A4   2040                  1                 MOVEA.L           D0,A0                          ; A0 has address
014A6   103C 0002             1                 MOVE.B            #rhdf,D0                       ; set data
```

```
MC680xx Assembler - Ver 3.2b6                                                             18-May-91  Page  30
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

014AA   6100 09A6       01E52 1                       BSR               NbWrite
014AE                         1
014AE   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
014B2                                                 MWrite            gpibauxcmd,hdeclr             ; release holdoff on EOI
014B2   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
014B6                         1
014B6   2009                  1                       MOVE.L            A1,D0                         ; from board base address
014B8 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
014BE   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
014C0   103C 0004             1                       MOVE.B            #hdeclr,D0                    ; set data
014C4   6100 098C       01E52 1                       BSR               NbWrite
014C8                         1
014C8   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
014CC                                                 MWrite            gpibauxcmd,shdclr             ; reset shadow handshake
014CC   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
014D0                         1
014D0   2009                  1                       MOVE.L            A1,D0                         ; from board base address
014D2 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
014D8   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
014DA   103C 0016             1                       MOVE.B            #shdclr,D0                    ; set data
014DE   6100 0972       01E52 1                       BSR               NbWrite
014E2                         1
014E2   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
014E6   6042            0152A                         BRA.S             CXferGood                     ; return
014E8
014E8                               ; bus timed out
014E8   7081                        CXfer5            MOVEQ             #gpibErr,D0                   ; return error to O.S.
014EA   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
014F0 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
014F4   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
014FA   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
01500   6038            0153A                         BRA.S             CXferDone
01502
01502                               ; here if interface not controller in charge
01502   7081                        CXfrNchg          MOVEQ             #gpibErr,D0                   ; return error to O.S.
01504   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
0150A G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
0150E   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01514   6024            0153A                         BRA.S             CXferDone                     ; and return
01516
01516                               ; here if the talker address was bad
01516   7081                        CXferBadAddr      MOVEQ             #gpibErr,D0                   ; return error to O.S.
01518   357C 0002 0006                                MOVE.W            #ctlBaddr,csError(A2)         ; return bad device address
0151E G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01522   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01528   6010            0153A                         BRA.S             CXferDone                     ; and return
0152A
0152A   7000                        CXferGood         MOVEQ             #noErr,D0                     ; return no error
0152C G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01530 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01534   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
0153A
0153A                               CXferDone         MSetCIC                                         ; set up status CIC bit
0153A                         1
0153A   6100 08D6       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
0153E   6608            01548 1                       BNE.S             @StCIC1                       ; no ...
01540   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01546   6006            0154E 1                       BRA.S             @StCIC2
01548                         1
01548   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
0154E                         1
0154E   4E71                  1     @StCIC2           NOP
01550                         1
01550   4CDF 0A07                                     MOVEM.L           (SP)+,A1/A3/D0-D2             ; restore local registers
01554   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01558   6000 ED16       00270                         BRA               ExitDrvr
0155C
0155C
0155C
0155C
0155C
0155C
0155C
0155C                               ********************************************************************************************
0155C                               *     CPassCntrl - controller passes control to the specified device
0155C                               *
0155C                               *     Entry:      A0 - param blk pointer
0155C                               *                 A1 - DCE pointer
0155C                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0155C                               *
0155C                               ********************************************************************************************
0155C                               CPassCntrl
0155C   48E7 E050                                     MOVEM.L           A1/A3/D0-D2,-(SP)             ; save local work registers
01560
01560                               ; get base address of board
01560   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01564   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01566   E188                                          LSL.L             #8,D0
01568   E188                                          LSL.L             #8,D0
0156A   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01570   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01572
01572   6100 089E       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
01576   6600 00A0       01618                         BNE               CPassNchg                     ; no ...
0157A
0157A   6100 0826       01DA2                         BSR               GetGpibAddr                   ; get our address
0157E   0040 0040                                     ORI               #mta,D0                       ; our talker address
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  31
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01582   1400                                          MOVE.B            D0,D2                         ; D2 = our talk address
01584
01584                               ; get talker address
01584   266A 0010                                     MOVEA.L           csAddrList(A2),A3             ; A3 <- pointer to talker address
01588
01588   101B                                          MOVE.B            (A3)+,D0                      ; get specified talker
0158A   B400                                          CMP.B             D0,D2                         ; is it our address ?
0158C   6700 009E       0162C                         BEQ               CPassBadAddr                  ; yes ...
01590
01590 G 0C00 0040                                     CMP.B             #$40,D0                       ; valid talker address ?
01594   6D00 0096       0162C                         BLT               CPassBadAddr
01598 G 0C00 005E                                     CMP.B             #$5e,D0
0159C   6E00 008E       0162C                         BGT               CPassBadAddr
015A0   6100 07B8       01D5A                         BSR               DataOut                       ; send talker address over GPIB
015A4   6100 0782       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
015A8   6754            015FE                         BEQ.S             CPassTim                      ; Bus timed out
015AA
015AA                               ; command 'take control'
015AA                                                 MWrite            gpibdataout,tct               ; send 'take control'
015AA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
015AE                         1
015AE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
015B0 G 0680 0002 001C        1                       ADD.L             #gpibdataout,D0               ; add to where byte will go
015B6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
015B8   103C 0009             1                       MOVE.B            #tct,D0                       ; set data
015BC   6100 0894       01E52 1                       BSR               NbWrite
015C0                         1
015C0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
015C4   6100 0762       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
015C8   6734            015FE                         BEQ.S             CPassTim                      ; Bus timed out
015CA
015CA                               ; new controller has accepted control, so release control to it.
015CA                                                 MWrite            gpibauxcmd,rlc                ; release control
015CA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
015CE                         1
015CE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
015D0 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
015D6   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
015D8   103C 0012             1                       MOVE.B            #rlc,D0                       ; set data
015DC   6100 0874       01E52 1                       BSR               NbWrite
015E0                         1
015E0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
015E4
015E4                               ; set flag as non-controller in local storage
015E4                                                 MWrite            amController,$00
015E4   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
015E8                         1
015E8   2009                  1                       MOVE.L            A1,D0                         ; from board base address
015EA G 0680 0000 001C        1                       ADD.L             #amController,D0              ; add to where byte will go
015F0   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
015F2 G 4200                  1                       MOVE.B            #$00,D0                       ; set data
015F4   6100 085C       01E52 1                       BSR               NbWrite
015F8                         1
015F8   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
015FC
015FC   6042            01640                         BRA.S             CPassGood                     ; return
015FE
015FE
015FE                               ; bus timed out
015FE   7081                        CPassTim          MOVEQ             #gpibErr,D0                   ; return error to O.S.
01600   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
01606 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
0160A   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01610   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
01616   6038            01650                         BRA.S             CPassDone
01618
01618                               ; here if interface not controller in charge
01618   7081                        CPassNchg         MOVEQ             #gpibErr,D0                   ; return error to O.S.
0161A   357C 0004 0006                                MOVE.W            #ctlNinChg,csError(A2)        ; return error to application
01620 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01624   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0162A   6024            01650                         BRA.S             CPassDone                     ; and return
0162C
0162C                               ; here if the talker address was bad
0162C   7081                        CPassBadAddr      MOVEQ             #gpibErr,D0                   ; return error to O.S.
0162E   357C 0002 0006                                MOVE.W            #ctlBaddr,csError(A2)         ; return bad device address
01634 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01638   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0163E   6010            01650                         BRA.S             CPassDone                     ; and return
01640
01640   7000                        CPassGood         MOVEQ             #noErr,D0                     ; return no error
01642 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01646 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
0164A   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01650
01650                               CPassDone         MSetCIC                                         ; set up status CIC bit
01650                         1
01650   6100 07C0       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01654   6608            0165E 1                       BNE.S             @StCIC1                       ; no ...
01656   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
0165C   6006            01664 1                       BRA.S             @StCIC2
0165E                         1
0165E   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01664                         1
01664   4E71                  1     @StCIC2           NOP
01666                         1
01666   4CDF 0A07                                     MOVEM.L           (SP)+,A1/A3/D0-D2             ; restore local registers
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  32
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0166A   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4          ; restore registers
0166E   6000 EC00       00270                         BRA               ExitDrvr
01672
01672
01672
01672
01672
01672
01672
01672
01672
01672                               ********************************************************************************************
01672                               *     Rcv(params: GpibCtlBlkPtr); routine.  This routine is called when the driver control
01672                               *     function is invoked and the request was for a device receive data operation.
01672                               *
01672                               *
01672                               *     Entry:      A0    - param blk pointer
01672                               *                 A1    - DCE pointer
01672                               *                 A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01672                               *
01672                               *     Uses:
01672                               *                 A0    - temporary, 9914 'Data In' register address
01672                               *                 A1    - Board base address
01672                               *                 A2    - Pointer to 'params'
01672                               *                 A3    - Buffer pointer
01672                               *                 A4    - 9914 int0 register address
01672                               *                 A5    - 9914 auxcmd register address
01672                               *
01672                               *                 D0    - temporary
01672                               *                 D1    - timeout loop count
01672                               *                 D2    - eos character
01672                               *                 D3    - character count
01672                               *                 D4    - temporary
01672                               *                 D5    - max buffer size
01672                               *                 D6    - temporary
01672                               *                 D7    - EOS valid BOOLEAN
01672                               *
01672                               *     Exit: D0    - error code
01672                               *
01672                               ********************************************************************************************
01672                               Rcv
01672                               ; setup working registers
01672   48E7 FFFC                                     MOVEM.L           A0-A5/D0-D7,-(SP)             ; save local work registers
01676
01676   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
0167A   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
0167C   E188                                          LSL.L             #8,D0
0167E   E188                                          LSL.L             #8,D0
01680   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01686   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01688
01688   6100 0788       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
0168C   6700 0116       017A4                         BEQ               RcvNchg                       ; yes ...
01690
01690   2009                                          MOVE.L            A1,D0
01692 G 0680 0002 0000                                ADD.L             #gpibint0,D0
01698   2840                                          MOVEA.L           D0,A4                         ; A4 = 9914 int0 register address
0169A   2009                                          MOVE.L            A1,D0
0169C G 0680 0002 0018                                ADD.L             #gpibauxcmd,D0
016A2   2A40                                          MOVEA.L           D0,A5                         ; A5 = 9914 auxcmd register address
016A4   266A 000C                                     MOVEA.L           csDataBuf(A2),A3              ; A3 = pointer to input buffer
016A8
016A8   6100 0710       01DBA                         BSR               GetGpibTimot
016AC   2200                                          MOVE.L            D0,D1                         ; D1 = timeout loop count
016AE   6100 06DA       01D8A                         BSR               GetEos
016B2   1400                                          MOVE.B            D0,D2                         ; D2 = end-of-string character
016B4   2A2A 0008                                     MOVE.L            csCount(A2),D5                ; D5 = max input char count
016B8   3E2A 0002                                     MOVE.W            csFlag(A2),D7                 ; D7 = EOS Valid BOOLEAN
016BC G 7600                                          MOVE.L            #0,D3                         ; D3 = cleared character count
016BE
016BE                               ; set up the default return status.  Other bits will be filled in later.
016BE G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
016C2
016C2                               ; holdoff all data
016C2                                                 MWrite            gpibauxcmd,hdfa
016C2   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
016C6                         1
016C6   2009                  1                       MOVE.L            A1,D0                         ; from board base address
016C8 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
016CE   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
016D0   103C 0083             1                       MOVE.B            #hdfa,D0                      ; set data
016D4   6100 077C       01E52 1                       BSR               NbWrite
016D8                         1
016D8   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
016DC
016DC                               ; check to see if user requested zero characters
016DC   BA83                                          CMP.L             D3,D5                         ; max input buffer size hit ?
016DE   676C            0174C                         BEQ.S             Rcv7                          ; yes, finish up
016E0
016E0                               ; now start getting the data
016E0   2809                                          MOVE.L            A1,D4
016E2 G 0684 0002 001C                                ADD.L             #gpibdatain,D4
016E8   2044                                          MOVEA.L           D4,A0                         ; A0 = 9914 'Data In' register address
016EA
016EA   2038 0001                                     MOVE.L            true32b,D0                    ; set 32-bit mode
016EE   A05D                                          _SwapMMUMode
016F0
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  33
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

016F0   2C01                        Rcv1              MOVE.L            D1,D6                         ; D6 = loop timeout pass count
016F2 G 5386                        Rcv11             SUBI.L            #1,D6                         ; decrement pass count
016F4   6700 0092       01788                         BEQ               RcvTime                       ; if bus not responding
016F8   1014                                          MOVE.B            (A4),D0                       ; get interrupt 0 status
016FA   0200 0028                                     ANDI.B            #eoimk+bim,D0                 ; check for EOI or BI
016FE   67F2            016F2                         BEQ.S             Rcv11                         ; wait until set
01700
01700   1C00                                          MOVE.B            D0,D6                         ; save status
01702   0200 0008                                     ANDI.B            #eoimk,D0                     ; check for EOI
01706   661A            01722                         BNE.S             Rcv2                          ; if EOI
01708
01708   1010                                          MOVE.B            (A0),D0                       ; if not EOI get data from GPIB
0170A   16C0                                          MOVE.B            D0,(A3)+                      ; store data away
0170C G 5283                                          ADD.L             #1,D3                         ; chalk up another character
0170E
0170E G 0C47 0000                                     CMP.W             #0,D7                         ; should we check for EOS?
01712   6704            01718                         BEQ.S             Rcv4                          ; NO...
01714   B002                                          CMP.B             D2,D0                         ; YES, was it an 'eos' character
01716   6720            01738                         BEQ.S             Rcv3                          ; yes, finish up
01718
01718   BA83                        Rcv4              CMP.L             D3,D5                         ; max input buffer size hit ?
0171A   672A            01746                         BEQ.S             Rcv6                          ; yes, finish up
0171C   1ABC 0002                                     MOVE.B            #rhdf,(A5)                    ; release holdoff
01720   60CE            016F0                         BRA.S             Rcv1                          ; get more data
01722
01722                               ; byte had EOI with it
01722   2038 0000                   Rcv2              MOVE.L            false32b,D0
01726   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
01728   6100 0648       01D72                         BSR               DataIn                        ; get data from GPIB
0172C   16C0                                          MOVE.B            D0,(A3)+                      ; store data away
0172E G 5283                                          ADD.L             #1,D3                         ; chalk up another character
01730   006A 2000 0004                                ORI.W             #stEnd,csStatus(A2)           ; flag EOI received
01736   601A            01752                         BRA.S             Rcv5
01738
01738                               ; byte was EOS character
01738   2038 0000                   Rcv3              MOVE.L            false32b,D0
0173C   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
0173E   006A 2000 0004                                ORI.W             #stEnd,csStatus(A2)           ; flag EOS received
01744   600C            01752                         BRA.S             Rcv5
01746
01746                               ; max buffer size hit
01746   2038 0000                   Rcv6              MOVE.L            false32b,D0
0174A   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
0174C   006A 0200 0004              Rcv7              ORI.W             #stCnt,csStatus(A2)           ; flag buffer size hit
01752
01752                               ; finish up transaction
01752                               Rcv5              MWrite            gpibauxcmd,rhdf               ; release holdoff
01752   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01756                         1
01756   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01758 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
0175E   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01760   103C 0002             1                       MOVE.B            #rhdf,D0                      ; set data
01764   6100 06EC       01E52 1                       BSR               NbWrite
01768                         1
01768   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
0176C                                                 MWrite            gpibauxcmd,hdaclr             ; release holdoff on all
0176C   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01770                         1
01770   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01772 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01778   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
0177A   103C 0003             1                       MOVE.B            #hdaclr,D0                    ; set data
0177E   6100 06D2       01E52 1                       BSR               NbWrite
01782                         1
01782   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01786   6040            017C8                         BRA.S             RcvGood                       ; return
01788
01788                               ; here if GPIB bus not responding
01788   2038 0000                   RcvTime           MOVE.L            false32b,D0
0178C   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
0178E   7081                        RcvTime1          MOVEQ             #gpibErr,D0                   ; return error to O.S.
01790   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
01796   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0179C   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
017A2   6030            017D4                         BRA.S             RcvDone                       ; and return
017A4
017A4                               ; here if interface was the controller in charge
017A4   7081                        RcvNchg           MOVEQ             #gpibErr,D0                   ; return error to O.S.
017A6   357C 0005 0006                                MOVE.W            #ctlInChg,csError(A2)         ; return error to application
017AC G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
017B0   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
017B6   601C            017D4                         BRA.S             RcvDone                       ; and return
017B8
017B8
017B8   7081                        RcvBadAddr        MOVEQ             #gpibErr,D0                   ; return error to O.S.
017BA   357C 0002 0006                                MOVE.W            #ctlBaddr,csError(A2)         ; return error to application
017C0   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
017C6   600C            017D4                         BRA.S             RcvDone                       ; and return
017C8
017C8   7000                        RcvGood           MOVEQ             #noErr,D0                     ; return no error
017CA G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
017CE   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
017D4
017D4   2543 0008                   RcvDone           MOVE.L            D3,csCount(A2)                ; return # characters received
017D8                                                 MSetCIC                                         ; set up status CIC bit
017D8                         1
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  34
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

017D8   6100 0638       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
017DC   6608            017E6 1                       BNE.S             @StCIC1                       ; no ...
017DE   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
017E4   6006            017EC 1                       BRA.S             @StCIC2
017E6                         1
017E6   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
017EC                         1
017EC   4E71                  1     @StCIC2           NOP
017EE                         1
017EE   4CDF 3FFF                                     MOVEM.L           (SP)+,A0-A5/D0-D7             ; restore local work registers
017F2   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
017F6   6000 EA78       00270                         BRA               ExitDrvr
017FA
017FA
017FA
017FA
017FA
017FA
017FA
017FA
017FA                               ********************************************************************************************
017FA                               *     Send(params: GpibCtlBlkPtr); routine.  This routine is called when the driver Control
017FA                               *     function is invoked and the request was for a device send data operation.
017FA                               *
017FA                               *
017FA                               *     Entry:      A0    - param blk pointer
017FA                               *                 A1    - DCE pointer
017FA                               *                 A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
017FA                               *
017FA                               *     Uses:
017FA                               *                 A0    - temporary, 9914 'Data Out' register address
017FA                               *                 A1    - Board base address
017FA                               *                 A2    - Pointer to 'params'
017FA                               *                 A3    - Buffer pointer
017FA                               *                 A4    - 9914 int0 register address
017FA                               *                 A5    - 9914 auxcmd register address
017FA                               *
017FA                               *                 D0    - temporary
017FA                               *                 D1    - timeout loop count
017FA                               *                 D2    - eos character
017FA                               *                 D3    - actual character count
017FA                               *                 D4    - send EOI BOOLEAN
017FA                               *                 D5    - # characters to transmit
017FA                               *                 D6    - temporary
017FA                               *                 D7    - EOS valid BOOLEAN
017FA                               *
017FA                               *     Exit:       D0    - error code
017FA                               *
017FA                               ********************************************************************************************
017FA                               Send
017FA                               ; setup working registers
017FA   48E7 FFFC                                     MOVEM.L           A0-A5/D0-D7,-(SP)             ; save local work registers
017FE
017FE   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01802   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01804   E188                                          LSL.L             #8,D0
01806   E188                                          LSL.L             #8,D0
01808   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
0180E   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01810   266A 000C                                     MOVEA.L           csDataBuf(A2),A3              ; A3 = pointer to data buffer
01814   2009                                          MOVE.L            A1,D0
01816 G 0680 0002 0000                                ADD.L             #gpibint0,D0
0181C   2840                                          MOVEA.L           D0,A4                         ; A4 = 9914 int0 register address
0181E
0181E   6100 056A       01D8A                         BSR               GetEos
01822   1400                                          MOVE.B            D0,D2                         ; D2 = end-of-string character
01824 G 7600                                          MOVE.L            #0,D3                         ; D3 = cleared actual character count
01826   2A2A 0008                                     MOVE.L            csCount(A2),D5                ; D5 = char count
0182A   6100 058E       01DBA                         BSR               GetGpibTimot
0182E   2200                                          MOVE.L            D0,D1                         ; D1 = timeout loop count
01830   3E12                                          MOVE.W            csVar(A2),D7                  ; D7 = EOS Valid BOOLEAN
01832   382A 0002                                     MOVE.W            csFlag(A2),D4                 ; D4 = send EOI BOOLEAN
01836
01836                               ; set up the default return status.  Other bits will be filled in later.
01836 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
0183A
0183A   6100 05D6       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
0183E   6700 00F0       01930                         BEQ               SendNchg                      ; yes ...
01842
01842 G 0C85 0000 0000                                CMP.L             #0,D5                         ; Byte count zero?
01848   6700 00C2       0190C                         BEQ               Send9
0184C
0184C   2009                                          MOVE.L            A1,D0
0184E G 0680 0002 001C                                ADD.L             #gpibdataout,D0
01854   2040                                          MOVEA.L           D0,A0                         ; A0 has 9914 'Data Out' register address
01856
01856   2038 0001                                     MOVE.L            true32b,D0                    ; set 32-bit mode
0185A   A05D                                          _SwapMMUMode
0185C
0185C                               ; now loop, sending the data
0185C   2C01                        Send3             MOVE.L            D1,D6                         ; D6 = loop timeout pass count
0185E   101B                                          MOVE.B            (A3)+,D0                      ; get the data byte
01860 G 5385                                          SUBI.L            #1,D5                         ; decrement byte count
01862   671E            01882                         BEQ.S             Send4                         ; last byte to be output
01864 G 0C47 0000                                     CMP.W             #0,D7                         ; check for EOS Valid?
01868   6704            0186E                         BEQ.S             Send6                         ; NO...
0186A   B400                                          CMP.B             D0,D2                         ; YES, is byte the EOS char?
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  35
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

0186C   675A            018C8                         BEQ.S             Send8                         ; yes...
0186E   1080                        Send6             MOVE.B            D0,(A0)                       ; send data byte
01870 G 5283                                          ADDI.L            #1,D3                         ; Chalk up another byte sent
01872 G 5386                        Send7             SUBI.L            #1,D6                         ; decrement pass count
01874   6700 009E       01914                         BEQ               SendTime                      ; if bus not responding
01878   1014                                          MOVE.B            (A4),D0                       ; get interrupt 0 status
0187A   0200 0010                                     ANDI.B            #BOM,D0                       ; check for BO
0187E   67F2            01872                         BEQ.S             Send7                         ; wait for GPIB bus free
01880   60DA            0185C                         BRA.S             Send3                         ; next byte
01882
01882                               ; last data byte to send because max buffer size hit
01882   1200                        Send4             MOVE.B            D0,D1                         ; save byte temporarily
01884   2038 0000                                     MOVE.L            false32b,D0
01888   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
0188A   006A 0200 0004                                ORI.W             #stCnt,csStatus(A2)           ; flag buffer size hit
01890
01890 G 0C44 0000                                     CMP.W             #0,D4                         ; send EOI ?
01894   6720            018B6                         BEQ.S             Send41                        ; NO...
01896
01896                                                 MWrite            gpibauxcmd,feoi               ; send EOI with last character
01896   48E7 8080              1                      MOVEM.L           D0/A0,-(SP)                   ; save work registers
0189A                         1
0189A   2009                   1                      MOVE.L            A1,D0                         ; from board base address
0189C G 0680 0002 0018         1                      ADD.L             #gpibauxcmd,D0                ; add to where byte will go
018A2   2040                   1                      MOVEA.L           D0,A0                         ; A0 has address
018A4   103C 0008              1                      MOVE.B            #feoi,D0                      ; set data
018A8   6100 05A8       01E52 1                      BSR               NbWrite
018AC                         1
018AC   4CDF 0101              1                      MOVEM.L           (SP)+,D0/A0                   ; restore registers
018B0   006A 2000 0004                                ORI.W             #stEnd,csStatus(A2)           ; flag EOI
018B6
018B6   1001                        Send41            MOVE.B            D1,D0                         ; restore data byte
018B8   6100 04A0       01D5A                         BSR               DataOut                       ; send data byte
018BC G 5283                                          ADDI.L            #1,D3                         ; Chalk up another byte sent
018BE   6100 0468       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
018C2   6756            0191A                         BEQ.S             SendTime1                     ; Bus timed out
018C4   6000 008A       01950                         BRA               SendGood
018C8
018C8                               ; last data byte to send EOS character detected
018C8   1200                        Send8             MOVE.B            D0,D1                         ; save byte temporarily
018CA   2038 0000                                     MOVE.L            false32b,D0
018CE   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
018D0   006A 2000 0004                                ORI.W             #stEnd,csStatus(A2)           ; flag EOS
018D6
018D6 G 0C44 0000                                     CMP.W             #0,D4                         ; send EOI ?
018DA   6720            018FC                         BEQ.S             Send81                        ; NO...
018DC
018DC                                                 MWrite            gpibauxcmd,feoi               ; send EOI with last character
018DC   48E7 8080              1                      MOVEM.L           D0/A0,-(SP)                   ; save work registers
018E0                         1
018E0   2009                   1                      MOVE.L            A1,D0                         ; from board base address
018E2 G 0680 0002 0018         1                      ADD.L             #gpibauxcmd,D0                ; add to where byte will go
018E8   2040                   1                      MOVEA.L           D0,A0                         ; A0 has address
018EA   103C 0008              1                      MOVE.B            #feoi,D0                      ; set data
018EE   6100 0562       01E52 1                      BSR               NbWrite
018F2                         1
018F2   4CDF 0101              1                      MOVEM.L           (SP)+,D0/A0                   ; restore registers
018F6   006A 2000 0004                                ORI.W             #stEnd,csStatus(A2)           ; flag EOI
018FC
018FC   1001                        Send81            MOVE.B            D1,D0                         ; restore data byte
018FE   6100 045A       01D5A                         BSR               DataOut                       ; send data byte
01902 G 5283                                          ADDI.L            #1,D3                         ; Chalk up another byte sent
01904   6100 0422       01D28                         BSR               WaitOut                       ; wait for GPIB bus free
01908   6710            0191A                         BEQ.S             SendTime1                     ; Bus timed out
0190A   6044            01950                         BRA.S             SendGood
0190C
0190C                               ; zero characters specified to send
0190C   006A 0200 0004              Send9             ORI.W             #stCnt,csStatus(A2)           ; flag buffer size hit
01912   603C            01950                         BRA.S             SendGood
01914
01914                               ; here if GPIB bus not responding
01914   2038 0000                   SendTime          MOVE.L            false32b,D0
01918   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
0191A   7081                        SendTime1         MOVEQ             #gpibErr,D0                   ; return error to O.S.
0191C   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
01922   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01928   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
0192E   602C            0195C                         BRA.S             SendDone                      ; and return
01930
01930                               ; here if interface controller in charge
01930   7081                        SendNchg          MOVEQ             #gpibErr,D0                   ; return error to O.S.
01932   357C 0005 0006                                MOVE.W            #ctlInChg,csError(A2)         ; return error to application
01938   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0193E   601C            0195C                         BRA.S             SendDone                      ; and return
01940
01940   7081                        SendBad           MOVEQ             #gpibErr,D0                   ; return error to O.S.
01942   357C 0003 0006                                MOVE.W            #ctlUnkErr,csError(A2)        ; return error to application
01948   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
0194E   600C            0195C                         BRA.S             SendDone                      ; and return
01950
01950   7000                        SendGood          MOVEQ             #noErr,D0                     ; return no error
01952 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01956   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
0195C
0195C   2543 0008                   SendDone          MOVE.L            D3,csCount(A2)                ; return # characters sent
01960                                                 MSetCIC                                         ; set up status CIC bit
01960                         1
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  36
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01960   6100 04B0       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01964   6608            0196E 1                       BNE.S             @StCIC1                       ; no ...
01966   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
0196C   6006            01974 1                       BRA.S             @StCIC2
0196E                         1
0196E   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01974                         1
01974   4E71                  1     @StCIC2           NOP
01976                         1
01976   4CDF 3FFF                                     MOVEM.L           (SP)+,A0-A5/D0-D7             ; restore local work registers
0197A   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
0197E   6000 E8F0       00270                         BRA               ExitDrvr
01982
01982
01982
01982
01982
01982
01982
01982                               ********************************************************************************************
01982                               *     EnInter - control call to enable/disable board interrupts
01982                               *
01982                               *     Entry:      A0 - param blk pointer
01982                               *                 A1 - DCE pointer
01982                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01982                               *
01982                               ********************************************************************************************
01982                               EnInter
01982   48E7 8040                                     MOVEM.L           A1/D0,-(SP)                   ; save local work registers
01986
01986                               ; get base address of board
01986   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
0198A   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
0198C   E188                                          LSL.L             #8,D0
0198E   E188                                          LSL.L             #8,D0
01990   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01996   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01998
01998   302A 0002                                     MOVE.W            csFlag(A2),D0                 ; get desired operation
0199C G 0C40 0000                                     CMP.W             #0,D0                         ; enable or disable ?
019A0   671A            019BC                         BEQ.S             EnInter1                      ; to diasable
019A2                                                 MWrite            intenaddr,0                   ; enable interrupts
019A2   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
019A6                         1
019A6   2009                  1                       MOVE.L            A1,D0                         ; from board base address
019A8 G 0680 0004 0000        1                       ADD.L             #intenaddr,D0                 ; add to where byte will go
019AE   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
019B0 G 4200                  1                       MOVE.B            #0,D0                         ; set data
019B2   6100 049E       01E52 1                       BSR               NbWrite
019B6                         1
019B6   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
019BA   6018            019D4                         BRA.S             EnInterGood
019BC
019BC                               EnInter1          MWrite            intdisaddr,0                  ; disable interrupts
019BC   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
019C0                         1
019C0   2009                  1                       MOVE.L            A1,D0                         ; from board base address
019C2 G 0680 0006 0000        1                       ADD.L             #intdisaddr,D0                ; add to where byte will go
019C8   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
019CA G 4200                  1                       MOVE.B            #0,D0                         ; set data
019CC   6100 0484       01E52 1                       BSR               NbWrite
019D0                         1
019D0   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
019D4
019D4   7000                        EnInterGood       MOVEQ             #noErr,D0                     ; return no error
019D6 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
019DA G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
019DE   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
019E4
019E4                               EnInterDone       MSetCIC                                         ; set up status CIC bit
019E4                         1
019E4   6100 042C       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
019E8   6608            019F2 1                       BNE.S             @StCIC1                       ; no ...
019EA   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
019F0   6006            019F8 1                       BRA.S             @StCIC2
019F2                         1
019F2   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
019F8                         1
019F8   4E71                  1     @StCIC2           NOP
019FA                         1
019FA   4CDF 0201                                     MOVEM.L           (SP)+,A1/D0                   ; restore local registers
019FE   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01A02   6000 E86C       00270                         BRA               ExitDrvr
01A06
01A06
01A06
01A06
01A06                               ********************************************************************************************
01A06                               *     SetOut - setup GPIB buffer outputs
01A06                               *
01A06                               *     Entry:      A0 - param blk pointer
01A06                               *                 A1 - DCE pointer
01A06                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01A06                               *
01A06                               ********************************************************************************************
01A06                               SetOut
01A06   48E7 E0F0                                     MOVEM.L           A0-A3/D0-D2,-(SP)             ; save local work registers
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  37
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01A0A
01A0A                               ; get base address of board
01A0A   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01A0E   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01A10   E188                                          LSL.L             #8,D0
01A12   E188                                          LSL.L             #8,D0
01A14   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01A1A   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01A1C
01A1C                               ; get current configuration
01A1C   2009                                          MOVE.L            A1,D0                         ; from board base address
01A1E G 0680 0000 0030                                ADD.L             #swImage,D0                   ; add to where configuration is stored
01A24   2640                                          MOVEA.L           D0,A3                         ; A3 has address
01A26   2040                                          MOVEA.L           D0,A0
01A28   6100 040E       01E38                         BSR               NbRead
01A2C   1400                                          MOVE.B            D0,D2                         ; D2 has current configuration
01A2E   0202 00FC                                     ANDI.B            #$0fc,D2                      ; mask off 'system' bit
01A32
01A32                               ; get new configuration
01A32   3012                                          MOVE.W            csVar(A2),D0                  ; get desired operation
01A34   0200 0003                                     ANDI.B            #$3,D0                        ; only two LSB's valid
01A38   8002                                          OR.B              D2,D0                         ; add desired output type to current config
01A3A
01A3A                               ; write new configuration
01A3A   2209                                          MOVE.L            A1,D1
01A3C G 0681 0008 0000                                ADD.L             #swaddr,D1
01A42   2041                                          MOVEA.L           D1,A0                         ; A0 has configuration register address
01A44   6100 040C       01E52                         BSR               NbWrite                       ; set configuration
01A48
01A48                               ; store image of configuration
01A48   204B                                          MOVEA.L           A3,A0                         ; A0 has address of 'swimage'
01A4A   6100 0406       01E52                         BSR               NbWrite                       ; set configuration
01A4E
01A4E   7000                        SetOutGood        MOVEQ             #noErr,D0                     ; return no error
01A50 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01A54 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01A58   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01A5E
01A5E                               SetOutDone        MSetCIC                                         ; set up status CIC bit
01A5E                         1
01A5E   6100 03B2       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01A62   6608            01A6C 1                       BNE.S             @StCIC1                       ; no ...
01A64   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01A6A   6006            01A72 1                       BRA.S             @StCIC2
01A6C                         1
01A6C   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01A72                         1
01A72   4E71                  1     @StCIC2           NOP
01A74                         1
01A74   4CDF 0F07                                     MOVEM.L           (SP)+,A0-A3/D0-D2             ; restore local registers
01A78   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01A7C   6000 E7F2       00270                         BRA               ExitDrvr
01A80
01A80
01A80
01A80
01A80                                                 MWrite            swImage,$06                   ; store memory image of configuration register
01A80   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01A84                         1
01A84   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01A86 G 0680 0000 0030        1                       ADD.L             #swImage,D0                   ; add to where byte will go
01A8C   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01A8E   103C 0006             1                       MOVE.B            #$06,D0                       ; set data
01A92   6100 03BE       01E52 1                       BSR               NbWrite
01A96                         1
01A96   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01A9A
01A9A
01A9A
01A9A
01A9A
01A9A                               ********************************************************************************************
01A9A                               *     CRcvCntrl - receive controll from the currently active controller
01A9A                               *
01A9A                               *     Entry:      A0 - param blk pointer
01A9A                               *                 A1 - DCE pointer
01A9A                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01A9A                               *
01A9A                               ********************************************************************************************
01A9A                               CRcvCntrl
01A9A   48E7 E058                                     MOVEM.L           A1/A3/A4/D0-D2,-(SP)          ; save local work registers
01A9E
01A9E                               ; get base address of board
01A9E   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01AA2   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01AA4   E188                                          LSL.L             #8,D0
01AA6   E188                                          LSL.L             #8,D0
01AA8   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01AAE   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01AB0
01AB0   6100 0360       01E12                         BSR               AmIncharge                    ; are we the controller in charge?
01AB4   6700 00AE       01B64                         BEQ               CRcvCntrlNchg                 ; yes ...
01AB8
01AB8   2009                                          MOVE.L            A1,D0
01ABA G 0680 0002 0010                                ADD.L             #gpibint1,D0
01AC0   2040                                          MOVEA.L           D0,A0                         ; A0 = 9914 int1 register address
01AC2   6100 0374       01E38                         BSR               NbRead                        ; get int1 status register
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  38
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01AC6   0200 0020                                     ANDI.B            #ucgm,D0                      ; is it 'unidentified command' ?
01ACA   6700 00C6       01B92                         BEQ               CRcvCB1                       ; no ...
01ACE
01ACE   2009                                          MOVE.L            A1,D0
01AD0 G 0680 0002 000C                                ADD.L             #gpibcmd,D0
01AD6   2040                                          MOVEA.L           D0,A0                         ; A0 = 9914 'gpibcmd' register address
01AD8   6100 035E       01E38                         BSR               NbRead                        ; get command pass thru register contents
01ADC G 0C00 0009                                     CMP.B             #tct,D0                       ; is it 'take control' ?
01AE0   6600 0096       01B78                         BNE               CRcvCntrlBad                  ; no ...
01AE4
01AE4   2009                                          MOVE.L            A1,D0
01AE6 G 0680 0002 0008                                ADD.L             #gpibadst,D0
01AEC   2040                                          MOVEA.L           D0,A0                         ; A0 = 9914 'gpibadst' register address
01AEE   6100 0348       01E38                         BSR               NbRead                        ; get address status
01AF2   0200 0002                                     ANDI.B            #tadsm,D0                     ; are we addressed to talk?
01AF6   6700 0080       01B78                         BEQ               CRcvCntrlBad                  ; no ...
01AFA
01AFA                                                 MWrite            gpibauxcmd,rqc                ; request control
01AFA   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01AFE                         1
01AFE   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01B00 G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01B06   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01B08   103C 0011             1                       MOVE.B            #rqc,D0                       ; set data
01B0C   6100 0344       01E52 1                       BSR               NbWrite
01B10                         1
01B10   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01B14                                                 MWrite            gpibauxcmd,dacr               ; release dac holdoff
01B14   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01B18                         1
01B18   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01B1A G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01B20   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01B22   103C 0001             1                       MOVE.B            #dacr,D0                      ; set data
01B26   6100 032A       01E52 1                       BSR               NbWrite
01B2A                         1
01B2A   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01B2E
01B2E                               ; set flag as controller in local storage
01B2E                                                 MWrite            amController,$ff
01B2E   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01B32                         1
01B32   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01B34 G 0680 0000 001C        1                       ADD.L             #amController,D0              ; add to where byte will go
01B3A   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01B3C   103C 00FF             1                       MOVE.B            #$ff,D0                       ; set data
01B40   6100 0310       01E52 1                       BSR               NbWrite
01B44                         1
01B44   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01B48
01B48   605C            01BA6                         BRA.S             CRcvCntrlGood                 ; return
01B4A
01B4A
01B4A                               ; bus timed out
01B4A   7081                        CRcvCntrlTim      MOVEQ             #gpibErr,D0                   ; return error to O.S.
01B4C   357C 0001 0006                                MOVE.W            #ctlTime,csError(A2)          ; return error to application
01B52 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01B56   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01B5C   006A 4000 0004                                ORI.W             #stTime,csStatus(A2)          ; flag timeout
01B62   6052            01BB6                         BRA.S             CRcvCntrlDone
01B64
01B64                               ; here if interface controller in charge
01B64   7081                        CRcvCntrlNchg     MOVEQ             #gpibErr,D0                   ; return error to O.S.
01B66   357C 0005 0006                                MOVE.W            #ctlInChg,csError(A2)         ; return error to application
01B6C G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01B70   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01B76   603E            01BB6                         BRA.S             CRcvCntrlDone                 ; and return
01B78
01B78                               CRcvCntrlBad      MWrite            gpibauxcmd,dacr               ; release dac holdoff
01B78   48E7 8080             1                       MOVEM.L           D0/A0,-(SP)                   ; save work registers
01B7C                         1
01B7C   2009                  1                       MOVE.L            A1,D0                         ; from board base address
01B7E G 0680 0002 0018        1                       ADD.L             #gpibauxcmd,D0                ; add to where byte will go
01B84   2040                  1                       MOVEA.L           D0,A0                         ; A0 has address
01B86   103C 0001             1                       MOVE.B            #dacr,D0                      ; set data
01B8A   6100 02C6       01E52 1                       BSR               NbWrite
01B8E                         1
01B8E   4CDF 0101             1                       MOVEM.L           (SP)+,D0/A0                   ; restore registers
01B92   7081                        CRcvCB1           MOVEQ             #gpibErr,D0                   ; return error
01B94   357C 0003 0006                                MOVE.W            #ctlUnkErr,csError(A2)
01B9A G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01B9E   006A 8000 0004                                ORI.W             #stErr,csStatus(A2)           ; flag error
01BA4   6010            01BB6                         BRA.S             CRcvCntrlDone                 ; and return
01BA6
01BA6   7000                        CRcvCntrlGood     MOVEQ             #noErr,D0                     ; return no error
01BA8 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01BAC G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01BB0   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01BB6
01BB6                               CRcvCntrlDone     MSetCIC                                         ; set up status CIC bit
01BB6                         1
01BB6   6100 025A       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01BBA   6608            01BC4 1                       BNE.S             @StCIC1                       ; no ...
01BBC   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01BC2   6006            01BCA 1                       BRA.S             @StCIC2
01BC4                         1
01BC4   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  39
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01BCA                         1
01BCA   4E71                  1     @StCIC2           NOP
01BCC                         1
01BCC   4CDF 1A07                                     MOVEM.L           (SP)+,A1/A3/A4/D0-D2          ; restore local registers
01BD0   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01BD4   6000 E69A       00270                         BRA               ExitDrvr
01BD8
01BD8
01BD8
01BD8
01BD8                               ********************************************************************************************
01BD8                               *     Read - read byte from board memory
01BD8                               *
01BD8                               *     Entry:      A0 - param blk pointer
01BD8                               *                 A1 - DCE pointer
01BD8                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01BD8                               *
01BD8                               ********************************************************************************************
01BD8                               Read
01BD8   48E7 C0C0                                     MOVEM.L           A0/A1/D0/D1,-(SP)             ; save local work registers
01BDC
01BDC                               ; get base address of board
01BDC   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01BE0   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01BE2   E188                                          LSL.L             #8,D0
01BE4   E188                                          LSL.L             #8,D0
01BE6   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01BEC   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01BEE
01BEE                               ; get address
01BEE   200A                                          MOVE.L            A2,D0
01BF0 G 0680 0000 0010                                ADD.L             #csAddrList,D0
01BF6   2040                                          MOVEA.L           D0,A0
01BF8   2010                                          MOVE.L            (A0),D0                       ; D0 = address on board
01BFA   0280 00FF FFFF                                ANDI.L            #$00ffffff,D0                 ; mask to 24 bits
01C00   2209                                          MOVE.L            A1,D1                         ; get board base address
01C02   D081                                          ADD.L             D1,D0                         ; add it to the requested address
01C04   2040                                          MOVEA.L           D0,A0                         ; A0 = requested address on board
01C06   6100 0230       01E38                         BSR               NbRead                        ; get the byte
01C0A   0280 0000 00FF                                ANDI.L            #$000000ff,D0                 ; mask off lower byte
01C10   3480                                          MOVE.W            D0,csVar(A2)                  ; Return to caller the requested byte.
01C12
01C12   7000                        ReadGood          MOVEQ             #noErr,D0                     ; return no error
01C14 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01C18 G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01C1C   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01C22
01C22                               ReadDone          MSetCIC                                         ; set up status CIC bit
01C22                         1
01C22   6100 01EE       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01C26   6608            01C30 1                       BNE.S             @StCIC1                       ; no ...
01C28   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01C2E   6006            01C36 1                       BRA.S             @StCIC2
01C30                         1
01C30   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01C36                         1
01C36   4E71                  1     @StCIC2           NOP
01C38                         1
01C38   4CDF 0303                                     MOVEM.L           (SP)+,A0/A1/D0/D1             ; restore local registers
01C3C   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01C40   6000 E62E       00270                         BRA               ExitDrvr
01C44
01C44
01C44
01C44
01C44                               ********************************************************************************************
01C44                               *     Write - write byte to board memory
01C44                               *
01C44                               *     Entry:      A0 - param blk pointer
01C44                               *                 A1 - DCE pointer
01C44                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01C44                               *
01C44                               ********************************************************************************************
01C44                               Write
01C44   48E7 C0C0                                     MOVEM.L           A0/A1/D0/D1,-(SP)             ; save local work registers
01C48
01C48                               ; get base address of board
01C48   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01C4C   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01C4E   E188                                          LSL.L             #8,D0
01C50   E188                                          LSL.L             #8,D0
01C52   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01C58   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01C5A
01C5A                               ; get address
01C5A   200A                                          MOVE.L            A2,D0
01C5C G 0680 0000 0010                                ADD.L             #csAddrList,D0
01C62   2040                                          MOVEA.L           D0,A0
01C64   2010                                          MOVE.L            (A0),D0                       ; D0 = address on board
01C66   0280 00FF FFFF                                ANDI.L            #$00ffffff,D0                 ; mask to 24 bits
01C6C   2209                                          MOVE.L            A1,D1                         ; get board base address
01C6E   D081                                          ADD.L             D1,D0                         ; add it to the requested address
01C70   2040                                          MOVEA.L           D0,A0                         ; A0 = requested address on board
01C72   3012                                          MOVE.W            csVar(A2),D0                  ; D0 contains the byte to be written
01C74   6100 01DC       01E52                         BSR               NbWrite                       ; write the byte
01C78
01C78   7000                        WriteGood         MOVEQ             #noErr,D0                     ; return no error
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  40
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01C7A G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01C7E G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01C82   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01C88
01C88                               WriteDone         MSetCIC                                         ; set up status CIC bit
01C88                         1
01C88   6100 0188       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01C8C   6608            01C96 1                       BNE.S             @StCIC1                       ; no ...
01C8E   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01C94   6006            01C9C 1                       BRA.S             @StCIC2
01C96                         1
01C96   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01C9C                         1
01C9C   4E71                  1     @StCIC2           NOP
01C9E                         1
01C9E   4CDF 0303                                     MOVEM.L           (SP)+,A0/A1/D0/D1             ; restore local registers
01CA2   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01CA6   6000 E5C8       00270                         BRA               ExitDrvr
01CAA
01CAA
01CAA
01CAA
01CAA
01CAA                               ********************************************************************************************
01CAA                               *     NewTimout - control call to define a new timeout constant.  The value will be
01CAA                               *                        passed in the 'csCount' field.
01CAA                               *
01CAA                               *     Entry:      A0 - param blk pointer
01CAA                               *                 A1 - DCE pointer
01CAA                               *                 A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
01CAA                               *
01CAA                               ********************************************************************************************
01CAA                               NewTimout
01CAA   48E7 C0E0                                     MOVEM.L           A0/A1/A2/D0/D1,-(SP)          ; save local work registers
01CAE
01CAE                               ; get base address of board
01CAE   1029 0028                                     MOVE.B            dCtlSlot(A1),D0               ; get the slot address
01CB2   E188                                          LSL.L             #8,D0                         ; shift the 4 slot bits into proper position
01CB4   E188                                          LSL.L             #8,D0
01CB6   E188                                          LSL.L             #8,D0
01CB8   0080 F000 0003                                ORI.L             #$f0000003,D0                 ; Slot space
01CBE   2240                                          MOVEA.L           D0,A1                         ; A1 = board base address
01CC0
01CC0                               ; set timout constant
01CC0   2009                                          MOVE.L            A1,D0                         ; from board base address
01CC2 G 0680 0000 002C                                ADD.L             #(timot+12),D0                ; add to where LSB byte will go
01CC8   2040                                          MOVEA.L           D0,A0                         ; A0 has address
01CCA   202A 0008                                     MOVE.L            csCount(A2),D0                ; get the new timeout value
01CCE   6100 0182       01E52                         BSR               NbWrite                       ; write low byte
01CD2   E088                                          LSR.L             #8,D0                         ; position next byte
01CD4   2208                                          MOVE.L            A0,D1
01CD6 G 5981                                          SUB.L             #4,D1
01CD8   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01CDA   6100 0176       01E52                         BSR               NbWrite                       ; write 2nd byte
01CDE   E088                                          LSR.L             #8,D0                         ; position next byte
01CE0   2208                                          MOVE.L            A0,D1
01CE2 G 5981                                          SUB.L             #4,D1
01CE4   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01CE6   6100 016A       01E52                         BSR               NbWrite                       ; write 3rd byte
01CEA   E088                                          LSR.L             #8,D0                         ; position next byte
01CEC   2208                                          MOVE.L            A0,D1
01CEE G 5981                                          SUB.L             #4,D1
01CF0   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01CF2   6100 015E       01E52                         BSR               NbWrite                       ; write high byte
01CF6
01CF6   7000                        NwTmotGood        MOVEQ             #noErr,D0                     ; return no error
01CF8 G 426A 0006                                     MOVE.W            #ctlNoErr,csError(A2)
01CFC G 426A 0004                                     MOVE.W            #stGood,csStatus(A2)          ; Default status
01D00   006A 0100 0004                                ORI.W             #stCmplt,csStatus(A2)         ; flag call complete
01D06
01D06                               NwTmotDone        MSetCIC                                               ; set up status CIC bit
01D06                         1
01D06   6100 010A       01E12 1                       BSR               AmIncharge                    ; are we the controller in charge?
01D0A   6608            01D14 1                       BNE.S             @StCIC1                       ; no ...
01D0C   006A 0020 0004        1                       ORI.W             #stCic,csStatus(A2)           ; flag CIC
01D12   6006            01D1A 1                       BRA.S             @StCIC2
01D14                         1
01D14   026A FFDF 0004        1     @StCIC1           ANDI.W            #stNCic,csStatus(A2)          ; flag /CIC
01D1A                         1
01D1A   4E71                  1     @StCIC2           NOP
01D1C                         1
01D1C   4CDF 0703                                     MOVEM.L           (SP)+,A0/A1/A2/D0/D1          ; restore local registers
01D20   4CDF 1110                                     MOVEM.L           (SP)+,A0/A4/D4                ; restore registers
01D24   6000 E54A       00270                         BRA               ExitDrvr
01D28
01D28
01D28
01D28
01D28
01D28
01D28                               ********************************************************************************************
01D28                               *     WaitOut - waits for a byte to be output to the GPIB
01D28                               *
01D28                               *     Enter:
01D28                               *                 A1    - NUBUS board base address
01D28                               *
01D28                               *     Registers affected:           None
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  41
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01D28                               *
01D28                               *     Return:
01D28                               *                 Z bit set in condition code register if bus not responding
01D28                               *
01D28                               ********************************************************************************************
01D28                               WaitOut
01D28   48E7 E080                                     MOVEM.L           A0/D0-D2,-(SP)                ; save work registers
01D2C
01D2C   6100 008C       01DBA                         BSR               GetGpibTimot
01D30   2400                                          MOVE.L            D0,D2                         ; D2 = timeout loop count
01D32
01D32   2209                                          MOVE.L            A1,D1
01D34 G 0681 0002 0000                                ADD.L             #gpibint0,D1
01D3A   2041                                          MOVEA.L           D1,A0                         ; A0 has 9914 int0 register address
01D3C
01D3C G 5382                        WaitOut1          SUBI.L            #1,D2                         ; decrement loop pass count
01D3E   6710            01D50                         BEQ.S             WaitOut3                      ; bus timed out
01D40   6100 00F6       01E38                         BSR               NbRead                        ; get interrupt 0 status
01D44   0200 0010                                     ANDI.B            #BOM,D0                       ; check for BO
01D48   67F2            01D3C                         BEQ.S             WaitOut1                      ; wait until set
01D4A
01D4A   023C 001B                   WaitOut2          ANDI              #$1B,CCR                      ; All good, clear 'Z' bit
01D4E   6004            01D54                         BRA.S             WaitOut4                      ; exit
01D50
01D50   003C 0004                   WaitOut3          ORI               #$4,CCR                       ; timed out, set 'Z' bit
01D54
01D54   4CDF 0107                   WaitOut4          MOVEM.L           (SP)+,A0/D0-D2                ; restore work register
01D58   4E75                                          RTS
01D5A
01D5A
01D5A
01D5A
01D5A
01D5A
01D5A
01D5A                               ********************************************************************************************
01D5A                               *     DataOut - sends byte out GPIB bus
01D5A                               *
01D5A                               *     Enter:
01D5A                               *                 A1    - NUBUS board base address
01D5A                               *                 D0    - Byte to be output
01D5A                               *
01D5A                               *     Registers affected:           None
01D5A                               *
01D5A                               ********************************************************************************************
01D5A                               DataOut
01D5A   48E7 4080                                     MOVEM.L           A0/D1,-(SP)                   ; save work registers
01D5E
01D5E   2209                                          MOVE.L            A1,D1
01D60 G 0681 0002 001C                                ADD.L             #gpibdataout,D1
01D66   2041                                          MOVEA.L           D1,A0                         ; A0 has 9914 'Data Out' register address
01D68
01D68   6100 00E8       01E52                         BSR               NbWrite                       ; send byte
01D6C
01D6C   4CDF 0102                                     MOVEM.L           (SP)+,A0/D1                   ; restore work register
01D70   4E75                                          RTS
01D72
01D72
01D72
01D72
01D72                               ********************************************************************************************
01D72                               *     DataIn - gets byte from GPIB bus
01D72                               *
01D72                               *     Enter:
01D72                               *                 A1    - NUBUS board base address
01D72                               *
01D72                               *     Exit:
01D72                               *                 D0    - Byte received
01D72                               *
01D72                               *     Other registers affected:           None
01D72                               *
01D72                               ********************************************************************************************
01D72                               DataIn
01D72   48E7 4080                                     MOVEM.L           A0/D1,-(SP)             ; save work registers
01D76
01D76   2209                                          MOVE.L            A1,D1
01D78 G 0681 0002 001C                                ADD.L             #gpibdatain,D1
01D7E   2041                                          MOVEA.L           D1,A0                   ; A0 has 9914 'Data In' register address
01D80
01D80   6100 00B6       01E38                         BSR               NbRead                  ; get data from GPIB
01D84
01D84   4CDF 0102                                     MOVEM.L           (SP)+,A0/D1             ; restore work register
01D88   4E75                                          RTS
01D8A
01D8A
01D8A
01D8A
01D8A                               ********************************************************************************************
01D8A                               *     GetEos - return in D0 the value store in the 'eos' location on our board's local RAM.
01D8A                               *
01D8A                               *     Enter:
01D8A                               *                 A1 - NUBUS board base address
01D8A                               *
01D8A                               *     Exit:
01D8A                               *                 D0 - 'eos' byte
01D8A                               *
01D8A                               *     Other registers affected:           None
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  42
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01D8A                               *
01D8A                               ********************************************************************************************
01D8A                               GetEos
01D8A   48E7 0080                                     MOVEM.L           A0,-(SP)                      ; save work registers
01D8E
01D8E   2009                                          MOVE.L            A1,D0                         ; from board base address
01D90 G 0680 0000 0010                                ADD.L             #eos,D0                       ; add to where byte stored
01D96   2040                                          MOVEA.L           D0,A0                         ; A0 has address
01D98   6100 009E       01E38                         BSR               NbRead
01D9C
01D9C   4CDF 0100                                     MOVEM.L           (SP)+,A0                      ; restore registers
01DA0   4E75                                          RTS
01DA2
01DA2
01DA2
01DA2
01DA2
01DA2
01DA2                               ********************************************************************************************
01DA2                               *     GetGpibAddr - return in D0 the value store in the 'gpibAddrSt' location on our board's
01DA2                               *                             local RAM.
01DA2                               *
01DA2                               *     Enter:
01DA2                               *                 A1 - NUBUS board base address
01DA2                               *
01DA2                               *     Exit:
01DA2                               *                 D0 - 'gpibAddrSt' byte
01DA2                               *
01DA2                               *     Other registers affected:           None
01DA2                               *
01DA2                               ********************************************************************************************
01DA2                               GetGpibAddr
01DA2   48E7 0080                                     MOVEM.L           A0,-(SP)                      ; save work registers
01DA6
01DA6   2009                                          MOVE.L            A1,D0                         ; from board base address
01DA8 G 0680 0000 0014                                ADD.L             #gpibAddrSt,D0                ; add to where byte stored
01DAE   2040                                          MOVEA.L           D0,A0                         ; A0 has address
01DB0   6100 0086       01E38                         BSR               NbRead
01DB4
01DB4   4CDF 0100                                     MOVEM.L           (SP)+,A0                      ; restore registers
01DB8   4E75                                          RTS
01DBA
01DBA
01DBA
01DBA                               ********************************************************************************************
01DBA                               *     GetGpibTimot - return in D0 the value store in the 'timot' location on our board's
01DBA                               *                             local RAM.
01DBA                               *
01DBA                               *     Enter:
01DBA                               *                 A1 - NUBUS board base address
01DBA                               *
01DBA                               *     Exit:
01DBA                               *                 D0 - 'timot' longword
01DBA                               *
01DBA                               *     Other registers affected:           None
01DBA                               *
01DBA                               ********************************************************************************************
01DBA                               GetGpibTimot
01DBA   48E7 6080                                     MOVEM.L           A0/D1/D2,-(SP)                ; save work registers
01DBE
01DBE G 7400                                          MOVE.L            #0,D2                         ; clear temp storage
01DC0
01DC0   2009                                          MOVE.L            A1,D0                         ; from board base address
01DC2 G 0680 0000 002C                                ADD.L             #(timot+12),D0                ; add to where LSB byte stored
01DC8   2040                                          MOVEA.L           D0,A0                         ; A0 has address
01DCA   616C            01E38                         BSR.S             NbRead
01DCC   1400                                          MOVE.B            D0,D2                         ; save first byte
01DCE   2208                                          MOVE.L            A0,D1
01DD0 G 5981                                          SUB.L             #4,D1
01DD2   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01DD4   6162            01E38                         BSR.S             NbRead
01DD6   E188                                          LSL.L             #8,D0                         ; shift byte to proper location
01DD8   0280 0000 FF00                                ANDI.L            #$0000FF00,D0                 ; mask unused bits
01DDE   8480                                          OR.L              D0,D2                         ; save second byte
01DE0   2208                                          MOVE.L            A0,D1
01DE2 G 5981                                          SUB.L             #4,D1
01DE4   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01DE6   6150            01E38                         BSR.S             NbRead
01DE8   E188                                          LSL.L             #8,D0                         ; shift byte to proper location
01DEA   E188                                          LSL.L             #8,D0
01DEC   0280 00FF 0000                                ANDI.L            #$00FF0000,D0                 ; mask unused bits
01DF2   8480                                          OR.L              D0,D2                         ; save third byte
01DF4   2208                                          MOVE.L            A0,D1
01DF6 G 5981                                          SUB.L             #4,D1
01DF8   2041                                          MOVEA.L           D1,A0                         ; point to address of next byte
01DFA   613C            01E38                         BSR.S             NbRead
01DFC   E188                                          LSL.L             #8,D0                         ; shift byte to proper location
01DFE   E188                                          LSL.L             #8,D0
01E00   E188                                          LSL.L             #8,D0
01E02   0280 FF00 0000                                ANDI.L            #$FF000000,D0                 ; mask unused bits
01E08   8480                                          OR.L              D0,D2                         ; save fourth byte
01E0A
01E0A   2002                                          MOVE.L            D2,D0                         ; set return value
01E0C
01E0C   4CDF 0106                                     MOVEM.L           (SP)+,A0/D1/D2                ; restore registers
01E10   4E75                                          RTS
01E12
```

```
MC680xx Assembler - Ver 3.2b6                                                              18-May-91  Page  43
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01E12
01E12
01E12
01E12
01E12                               ********************************************************************************************
01E12                               *     AmIncharge - checks to see if the interface is the controller in charge of the bus
01E12                               *
01E12                               *     Enter:
01E12                               *                 A1    - NUBUS board base address
01E12                               *
01E12                               *     Registers affected:          None
01E12                               *
01E12                               *     Return:
01E12                               *                 Z bit set in condition code register if we are the controller in charge
01E12                               *
01E12                               ********************************************************************************************
01E12                               AmIncharge
01E12   48E7 C080                                     MOVEM.L           A0/D0-D1,-(SP)                ; save work registers
01E16
01E16   2209                                          MOVE.L            A1,D1
01E18 G 0681 0000 001C                                ADD.L             #amController,D1
01E1E   2041                                          MOVEA.L           D1,A0                         ; A0 has flag address
01E20
01E20   6116            01E38       AmIncharge1       BSR.S             NbRead                        ; get flag
01E22   0200 00FF                                     ANDI.B            #$ff,D0                       ; check the flag
01E26   6606            01E2E                         BNE.S             AmIncharge2
01E28   023C 001B                                     ANDI              #$1B,CCR                      ; not in charge, clear 'Z' bit
01E2C   6004            01E32                         BRA.S             AmIncharge3                   ; exit
01E2E
01E2E   003C 0004                   AmIncharge2       ORI               #$4,CCR                       ; am in charge, set 'Z' bit
01E32
01E32   4CDF 0103                   AmIncharge3       MOVEM.L           (SP)+,A0/D0-D1                ; restore work register
01E36   4E75                                          RTS
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38
01E38                               ********************************************************************************************
01E38                               *     NbRead - reads a byte from a NUBUS card
01E38                               *
01E38                               *     Enter:      A0 - pointer to address in 32-bit address space
01E38                               *
01E38                               *     Uses: no other registers
01E38                               *
01E38                               *     Exit: D0 - the byte in low 8 bits
01E38                               *
01E38                               ********************************************************************************************
01E38                               NbRead
01E38   48E7 4000                                     MOVEM.L           D1,-(SP)                      ; save work register
01E3C
01E3C   2038 0001                                     MOVE.L            true32b,D0                    ; set 32-bit mode
01E40   A05D                                          _SwapMMUMode
01E42
01E42   1210                                          MOVE.B            (A0),D1                       ; Get value specified
01E44
01E44   2038 0000                                     MOVE.L            false32b,D0
01E48   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
01E4A
01E4A   1001                                          MOVE.B            D1,D0                         ; return value
01E4C   4CDF 0002                                     MOVEM.L           (SP)+,D1                      ; restore work register
01E50   4E75                                          RTS
01E52
01E52
01E52
01E52                               ********************************************************************************************
01E52                               *     NbWrite - writes a byte to a NUBUS card
01E52                               *
01E52                               *     Enter:      A0 - pointer to address in 32-bit address space
01E52                               *                 D0 - the byte in low 8 bits
01E52                               *
01E52                               *     Uses: no other registers
01E52                               *
01E52                               ********************************************************************************************
01E52                               NbWrite
01E52   48E7 4000                                     MOVEM.L           D1,-(SP)                      ; save work registers
01E56
01E56   2200                                          MOVE.L            D0,D1                         ; save value in D1
01E58
01E58   2038 0001                                     MOVE.L            true32b,D0                    ; set 32-bit mode
01E5C   A05D                                          _SwapMMUMode
01E5E
01E5E   1081                                          MOVE.B            D1,(A0)                       ; write value specified
01E60
01E60   2038 0000                                     MOVE.L            false32b,D0
01E64   A05D                                          _SwapMMUMode                                    ; back to 24-bit mode
01E66
01E66   2001                                          MOVE.L            D1,D0                         ; restore entry value
01E68   4CDF 0002                                     MOVEM.L           (SP)+,D1                      ; restore work register
01E6C   4E75                                          RTS
```

```
MC680xx Assembler - Ver 3.2b6                                                                              18-May-91  Page  44
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M     Source Statement

01E6E
01E6E
01E6E
01E6E
01E6E   0000 1E6E                   _End020Drvr EQU               *                              ; the end of the driver
01E6E                                                 STRING            C
01E6E
01E6E
01E6E
01E6E
01E6E
01E6E
01E6E   0000 1FEC                                     ORG               ROMSize-fhBlockSize
01FEC
01FEC                               **********************************************************************************************
01FEC                               *                 format/header block
01FEC                               **********************************************************************************************
01FEC   00FF E014                                     DC.L        (_sRsrcDir-*)**$00ffffff      ; offset to sResource directory
01FF0   0000 2000                                     DC.L        ROMSize                       ; length of declaration data
01FF4   0000 0000                                     DC.L        0                             ; CRC (Patched by crcPatch, an MPW tool
01FF8   02                                            DC.B        Rev2                          ; revision level
01FF9   01                                            DC.B        AppleFormat                   ; format
01FFA   5A93 2BC7                                     DC.L        TestPattern                   ; test pattern
01FFE   00                                            DC.B        0                             ; Reserved byte (must be zero)
01FFF   78                                            DC.B        $78                           ; Byte lanes: 0111 1000 (bytelane 3)
02000
02000
02000                                                 ENDMAIN

                                                      END

Elapsed time: 18.46 seconds.

Assembly complete - no errors found.  10494 lines.
```